Universidad Complutense de Madrid

Facultad de Informática

Tesis doctoral

Tratamiento Computacional de Lengua de Signos Española y SignoEscritura

Memoria para optar al grado de Doctor

presentada por

Antonio F. G. Sevilla

Director

Alberto Díaz Esteban

Este documento es la versión personal del autor, a 20 de noviembre de 2023, de la tesis "Tratamiento Computacional de Lengua de Signos Española y SignoEscritura". Incluye los siguientes cambios:

- Se utiliza la versión aceptada para publicación del artículo "Building the VisSE Corpus of Spanish SignWriting".
- Se integra el contenido de las publicaciones de la parte II, en lugar del PDF original, para permitir referencias cruzadas y unificar índices y bibliografía.
- Cuestiones menores de formato y presentación que no alteran el contenido.

La versión oficial presentada, más información, y otros formatos como HTML, PDF o libro electrónico se pueden encontrar en: `https://garciasevilla.com/tesis`.

"Study hard what interests you the most in the most undisciplined, irreverent and original manner possible."

—Richard Feynman

El doctorado y la dirección del proyecto VisSE han supuesto una gran experiencia formativa para mí. La gestión de un proyecto financiado, contratar colaboradores, realizar la planificación e implementación de un proyecto concreto de investigación, me han otorgado un mayor entendimiento del proceso científico. Las publicaciones realizadas y el software desarrollado han supuesto un gran aprendizaje, pero también otras cosas que quizá no quedan reflejadas en otros lugares, como la docencia, la vida académica, y el proceso silencioso pero fundamental de la investigación básica.

Todo ello ha sido posible gracias a mis compañeros de doctorado, como Jorro, Isma, Marta y Rome, y compañeros más "senior" como Raquel y Vir y el resto del grupo NIL. Y por supuesto José María "Ach", compañero en la investigación y el estudio de la lengua de signos todos estos años y confidente en ciencia, burocracia e incluso filosofía si hace falta. Uno de los pocos oponentes dialécticos eficaces y dispuestos que me encontrado, y ciertamente no puede haber σ si no hay λ.

No habría podido descubrir las lenguas de signos sin la ayuda de las profesoras de Idiomas Complutense, Mamen, Gloria, Sarah... que no sólo nos han enseñado la gramática y vocabulario de la Lengua de Signos Española, sino también su belleza, su expresividad, y junto a ella la cultura Sorda, propia de una comunidad que ha resultado siempre receptiva, cálida y generosa.

Mención especial merece sin duda Alberto, que me ha aguantado, animado, y encauzado cuando hacía falta, sin quien esta tesis no se habría podido completar, y al que espero haber causado más alegrías que dolores de cabeza.

Por último, no quiero olvidar a mis padres y mi hermana, que con su ejemplo y ánimos me han ayudado durante el camino. Espero poder defender esta tesis con éxito y que, en las comidas familiares, ¡ya todos nos tengamos que saludar unos a otros como "doctor"!

Índice general

II. Compendium Scriptorum

III. Appendices

Índice de figuras

Índice de tablas

Índice de códigos fuente

Resumen

Las lenguas de signos son lenguas naturales propias de las comunidades de personas sordas o con déficit auditivo. No son meras transposiciones de la lengua oral, sino que tienen origen y evolución propias, así como su propio léxico, gramática e idiosincrasia.

Esto las convierte en un objeto de estudio legítimo para los campos de la Lingüística Computacional y el Procesamiento del Lenguaje Natural. En esta tesis se describe una investigación enmarcada en este estudio, con énfasis en la Lengua de Signos Española (LSE) y su SignoEscritura. La investigación doctoral realizada se articula en seis publicaciones, elaboradas en el marco de un proyecto de investigación financiado, dirigido por el autor y de título "Visualizando la SignoEscritura" (VisSE).

Se incluye una discusión integradora que establece y justifica los objetivos de la investigación. Se realiza además un análisis crítico del estado de la cuestión, en el que se consideran aspectos fundamentales para la investigación en lenguas de signos, como los métodos para su reconocimiento automático, su gramática y estructura, y su representación escrita. Es precisamente este último tema el que motiva el principal objetivo técnico de la tesis: mejorar el procesamiento automático de la SignoEscritura.

En este marco, se ha recolectado un corpus original de muestras de SignoEscritura de LSE, para el que se ha desarrollado un esquema de anotación novedoso y complejo basado en el conoci-

miento de su lingüística. En base a este corpus se han entrenado algoritmos de Inteligencia Artificial, en concreto redes neuronales de aprendizaje profundo, para reconocer e interpretar la SignoEscritura. La eficacia de estos algoritmos se ha potenciado mediante la creación de un sistema experto que los combina con reglas lógicas, deducidas del análisis teórico realizado. Estos desarrollos han sido posibles gracias a una librería Python creada para esta investigación, Quevedo, que permite gestionar corpus de datos, visualizarlos, anotarlos y procesarlos. También se ha creado una aplicación web para transferir a la sociedad los resultados obtenidos que convierte la SignoEscritura en explicaciones textuales y en un modelo tridimensional de la mano.

Adicionalmente, esta tesis no sólo ha aportado avances técnicos en el procesamiento de la SignoEscritura, sino que también ha generado un conocimiento profundo sobre la LSE, materializado en un modelo subyacente y en diversas publicaciones y aplicaciones computacionales no incluidas en el compendio pero referenciadas en este documento.

El software y los datos generados durante la investigación han sido liberados bajo licencias de código abierto, permitiendo así que futuros investigadores puedan continuar avanzando en este ámbito sin tener que partir desde cero.

Abstract

Sign languages are natural languages native to the Deaf or Hard-of-Hearing communities. They are not mere transpositions of an oral language, but have their own origin and evolution, as well as their own lexicon, grammar, and idiosyncrasy.

This makes them a legitimate object of study for the fields of Computational Linguistics and Natural Language Processing. This thesis describes research framed within this study, with an emphasis on Spanish Sign Language and its SignWriting. The doctoral research conducted is articulated in six publications, developed within the framework of a competitively funded research project led by the author, "Visualizing SignWriting" (VisSE).

An unifying discussion is included that establishes and justifies the objectives of the research. A critical analysis of the current state of the question is also conducted, considering essential aspects for sign language research, such as methods for their automatic recognition, their grammar and structure, and their written representation. It is precisely this last topic that motivates the main technical objective of the thesis: improving the automatic processing of SignWriting.

In this context, an original corpus of SignWriting samples has been collected, for which a new and complex annotation scheme based on its linguistics has been developed. Based on this corpus, Artificial Intelligence algorithms, specifically deep learning neural networks, have been trained to recognize and interpret

SignWriting. The effectiveness of these algorithms has been enhanced by creating an expert system that combines them with logical rules, derived from the theoretical analysis performed. These developments have been made possible thanks to a Python library built for this research, Quevedo, which allows managing data corpora as well as visualizing them, annotating them and processing them. A web application has also been created in order to transfer the research results to society. This application converts SignWriting into textual explanations and includes a three-dimensional model of the hand.

Additionally, this thesis has not only contributed technical advances in the processing of SignWriting but has also generated in-depth knowledge about Spanish Sign Language, materialized in an underlying model and in various publications and computational applications not included in the compendium but referenced in this document.

The software and data generated during the research have been released under open source licenses, thus allowing future researchers to continue advancing in this area without having to start from scratch.

I

Dissertatio

1. Introducción

Las Lenguas de Signos (LS) constituyen un conjunto diverso y complejo de sistemas lingüísticos que desempeñan un papel crucial en las comunidades de personas con discapacidades auditivas a nivel global (Stokoe 1960; Parkhurst y Parkhurst 2001). Contrariamente a los idiomas orales, que se transmiten a través del canal auditivo-vocal, las lenguas de signos se comunican mediante un canal viso-gestual. En este contexto, el emisor, también conocido como signante, efectúa una serie de gestos codificados con las manos y el cuerpo, que en su conjunto se denominan "signos". Estos signos se realizan dentro de un dominio espacial específico, comúnmente llamado "espacio de signado". El receptor u observador, a su vez, se vale del sentido de la vista para captar, interpretar y procesar estos signos (Quer, Pfau y Herrmann 2021).

Estos sistemas de signos exhiben una rica variedad de articulaciones gestuales que abarcan desde configuraciones manuales estáticas hasta movimientos dinámicos de las manos, e incluso inclinaciones sutiles del cuerpo. En una analogía con los idiomas orales, estas articulaciones gestuales pueden considerarse equivalentes a los fonemas en las lenguas habladas, y se organizan en patrones específicos para formar signos individuales, que a su vez se combinan para crear frases y oraciones complejas (Brentari 2019).

En años recientes, las lenguas de signos han experimentado un notable incremento en su visibilidad e interés por parte de la sociedad en general. Diversos medios de comunicación, incluyendo cadenas de televisión públicas, han empezado a incorporar intérpretes de lengua de signos en sus transmisiones, lo cual ha contribuido significativamente al reconocimiento público de la complejidad y la riqueza de estas lenguas (Utray y Gil 2014). Paralelamente, su expresividad inherente y atractivo visual las han convertido en un fenómeno recurrente en la difusión de la cultura popular, como en el caso de los memes en internet o en la interpretación de música en LS (Peñalba Acitores, Moriyón Mojica y Luque Perea 2018).

1.1. Objetivo general: tratamiento computacional de LSE

Al mismo tiempo que aumenta la visibilidad de las lenguas de signos, estas se están convirtiendo en un foco de atención en el ámbito de la Inteligencia Artificial (IA), especialmente en el contexto de nuevos paradigmas como el Aprendizaje Profundo (Deep Learning) y la Analítica de Datos Masivos (Big Data). Dada su naturaleza intrínsecamente multimodal y su potencial para mejorar la accesibilidad de las personas con discapacidades auditivas, las lenguas de signos ofrecen un campo de estudio especialmente atractivo para los investigadores en IA, quienes buscan desarrollar algoritmos y sistemas que puedan tratar estas lenguas de manera automática y eficaz.

Además del innegable atractivo que representan las Lenguas de Signos (LS) para las tecnologías computacionales emergentes, es imperativo reconocerlas como lenguajes naturales que son susceptibles de análisis en el ámbito de la lingüística computacional (LC) y el procesamiento del lenguaje natural (PLN). La LC se encarga del estudio de las lenguas y la facultad humana del lenguaje a través de métodos computacionales, incluyendo pero no limitado a la lógica formal y las matemáticas. En contraposición, el PLN se centra en el tratamiento de la lengua humana desde la perspectiva de la ingeniería informática, ya sea como datos de entrada, salida o para procesos de análisis y síntesis del lenguaje.

Este trabajo se inserta en la intersección disciplinar entre informática y lingüística, ámbito en el que he focalizado mi investigación. En este sentido, el objetivo principal de la presente tesis es explorar y avanzar en el tratamiento computacional de las lenguas de signos, específicamente la Lengua de Signos Española (LSE), usando técnicas modernas de IA, LC y PLN.

Cuando se contempla la posibilidad de tratar computacionalmente las LS, las primeras ideas que suelen surgir son la generación de avatares signantes o el reconocimiento automático de la LS (Elliott et al. 2008; Lombardo et al. 2011; Ludeña 2014). Estos son igualmente los puntos de partida con los que inicié mi propia investigación académica y son comúnmente propuestos por estudiantes para sus trabajos de fin de grado o máster (Sánchez Jiménez, López Prieto y Garrido Montoya 2019; Vegas Cañas, Rodríguez Cuesta y Torralbo Fuentes 2020), así como en enfoques empresariales que abordan esta temática. Mi trayectoria inicial también se centró en abordar estos problemas,

a través del diseño de avatares animados y la utilización de cámaras de video y de profundidad para el reconocimiento de signos.

No obstante, se debe subrayar que tanto la creación de avatares como el reconocimiento automático son tareas de gran envergadura que no admiten una aplicación directa de tecnologías preexistentes. Esta complejidad se torna evidente al considerar las LS como lenguas en pleno derecho. Crear un avatar animado que signe de manera natural es un problema computacional comparable a desarrollar un software que pueda hablar en una lengua oral de forma natural. Del mismo modo, el reconocimiento automático de la LS representa un reto en el campo de la traducción automática. El estado del arte en traducción automática ha alcanzado niveles de sofisticación considerables, pero esto se ha logrado tras años de investigaciones dedicadas y la asignación de recursos significativos.

1.2. Objetivos específicos

Para lograr el objetivo general planteado, establecemos una serie de objetivos específicos que facilitarán su consecución. Iniciamos con un objetivo teórico centrado en el conocimiento profundo del dominio en cuestión. Abordar el tratamiento computacional de una lengua natural como la LSE sin un entendimiento sólido de sus particularidades sería imprudente y podría conducirnos a errores significativos. Este aspecto se torna aún más crucial dado que la LSE carece de la abundancia de recursos e información que suelen estar disponibles para otras lenguas orales.

El objetivo teórico descubre el principal obstáculo para la consecución de nuestro objetivo general, que radica en la representación adecuada de la LSE. Este desafío nos lleva a formular objetivos técnicos con el fin de superar tal obstáculo. Estos objetivos técnicos abarcan esfuerzos de diversa índole enmarcados en la informática, desde la recolección y el procesamiento de datos, su tratamiento algorítmico mediante técnicas avanzadas de Inteligencia Artificial, hasta el desarrollo de software específico.

Finalmente, se incorpora un objetivo complementario que orienta nuestra metodología de trabajo. Este objetivo consiste en publicar todos los datos recabados, el conocimiento generado y los desarrollos de software en for-

matos de acceso abierto, con el fin de fomentar una mayor difusión de la información y permitir la reproducibilidad de nuestros resultados.

1.2.1. Objetivo teórico

En el caso de las Lenguas de Signos, sus características intrínsecas, que las diferencian notablemente de las lenguas orales, suponen que no se pueda realizar una aplicabilidad directa de las técnicas empleadas para las lenguas orales. Las limitaciones de datos y la singularidad de su estructura gramatical y sintáctica exigen que los investigadores adopten un enfoque adaptado específicamente para su tratamiento computacional. Para que las investigaciones realizadas sean efectivas y útiles realmente en el contexto de la LSE, un objetivo preliminar y fundamental de mi investigación doctoral es adquirir una sólida base de conocimiento científico-técnico en relación con la LSE.

Esto incluye no sólo un análisis teórico y bibliográfico de la estructura y grámatica de la lengua de signos, sino también el estudio de la LSE en sí. Para poder tratar con éxito una lengua, sus peculiaridades y características únicas, es fundamental, en mi opinión, comprenderla y poder articularla.

En el proceso de explorar la LSE y evaluar el estado actual de su tratamiento computacional, identificamos que uno de los desafíos más imponentes es la cuestión de su representación (Miller 2001). A diferencia de las lenguas orales occidentales, que generalmente comparten un sistema de escritura basado en el alfabeto latino, las lenguas de signos carecen de un sistema de escritura estándar y universalmente aceptado. Esto contrasta con idiomas como el ruso, árabe o japonés, donde uno de los primeros pasos para el aprendiz es familiarizarse con un sistema de escritura específico.

Para las LS, sin embargo, la ausencia de un sistema de escritura estandarizado y ampliamente reconocido constituye un obstáculo notorio en la pedagogía y el aprendizaje. Frecuentemente, los estudiantes optan por desarrollar sus propios sistemas idiosincrásicos de notación o, en su defecto, prescinden de cualquier registro escrito, abordando el aprendizaje del idioma desde una perspectiva exclusivamente oral-visual.

La ausencia de un sistema de escritura formalizado para las LS presenta una complejidad adicional para los enfoques de Lingüística Computacional (LC) y Procesamiento del Lenguaje Natural (PLN). En estos campos, la repre-

sentación textual es primordial para la modelización y el análisis. Por ende, se genera una necesidad imperante de desarrollar o adaptar sistemas de representación lingüística que sean coherentes y eficaces para la transcripción y el procesamiento computacional de la LSE. Sin una representación adecuada, los esfuerzos en LC y PLN quedan considerablemente obstaculizados, limitando las posibilidades de avance tanto en la teoría como en la aplicación práctica de tecnologías relacionadas con la lengua de signos.

Afortunadamente, sí que existen distintos sistemas de escritura para la LS, aunque no sean estándar ni muy extendidos. Entre ellos, se encuentra la SignoEscritura, inventado en norteamérica por Valerie Sutton para la Lengua de Signos Americana (Sutton 1995). La SignoEscritura es especialmente relevante para nosotros ya que se utiliza en el Centro de Idiomas Complutense por las profesoras de LSE como apoyo lingüístico, si bien no se enseña en sí en su total complejidad. La SignoEscritura constituye una representación abstracta pero visual de las LS, y utiliza iconografía específica para representar las manos y el cuerpo. Estas representaciones se disponen en un plano bidimensional para capturar el espacio y movimiento de la lengua de signos. Aunque de origen estadounidense, es un sistema fonético y, por lo tanto, adaptable a cualquier lengua de signos, lo que lo convierte en un candidato óptimo para su uso como representación, tanto desde una perspectiva científica como ingenieril, de las LS.

1.2.2. Objetivos técnicos

No obstante, la SignoEscritura presenta un desafío crucial: su naturaleza gráfica. A diferencia de las escrituras de lenguas orales, que son esencialmente secuencias lineales de símbolos individuales, la disposición de los símbolos en la SignoEscritura es bidimensional y gráfica. Esta característica plantea una problemática específica en su implementación y uso en entornos digitales. Esto da lugar al objetivo técnico de esta tesis: desarrollar métodos para el procesamiento computacional efectivo de la SignoEscritura.

Este objetivo a su vez se desglosa en una serie de sub-objetivos interrelacionados que deben alcanzarse con éxito. En primer lugar, se plantea la necesidad de recopilar una muestra significativa de ejemplos de SignoEscritura (sub-objetivo técnico 1). Este paso es fundamental para disponer de

datos empíricos auténticos sobre los cuales basar los avances y para poder llevar a cabo una evaluación concreta de las implementaciones realizadas. Paralelamente, se requiere investigar en el campo de la inteligencia artificial, en concreto en la visión artificial, para poder desarrollar algoritmos de IA capaces de reconocer y comprender la SignoEscritura (sub-objetivo técnico 2). Por último, resulta esencial plasmar estos algoritmos y métodos en forma de software (sub-objetivo técnico 3), con el fin de poder evaluarlos objetivamente utilizando los datos reunidos, al mismo tiempo que se garantiza que la funcionalidad resultante esté disponible para otros programas o para los usuarios finales.

Estos objetivos técnicos convergieron en un proyecto financiado en convocatoria competitiva a nivel nacional, y que se describe en detalle en el capítulo 3. Este proyecto, titulado "Visualizando la SignoEscritura" (VisSE) desempeña un papel central en la integración de la tesis, al proporcionar un marco unificador para los diversos objetivos planteados.

1.2.3. Objetivo complementario

Uno de los pilares fundamentales de la ciencia reside en la puesta en común del conocimiento y de los métodos empleados para adquirirlo. Esta práctica no solo permite el escrutinio y la validación del trabajo científico por parte de la comunidad, sino que también facilita la colaboración y la construcción colectiva de un corpus de sabiduría científica. Sin embargo, el ritmo acelerado y la complejidad creciente de la ciencia moderna han generado obstáculos para una compartición efectiva.

En el ámbito de la IA, la generación de algoritmos, scripts y grandes conjuntos de datos es constante. Afortunadamente, además, en la actualidad existe un fuerte movimiento hacia la ciencia abierta. No obstante, la mera compartición de datos y scripts suele ser insuficiente para replicar o extender investigaciones previas. Los entornos de investigación varían, y a menudo faltan detalles sobre configuraciones específicas, versiones de software, o incluso métodos de preprocesamiento de datos, lo que hace que la reproducibilidad sea una tarea ardua.

En este contexto, me planteo un objetivo complementario a la investigación desarrollada en mi tesis. Mi meta no es solo publicar el máximo número

posible de datos y software creados durante mi investigación, sino hacerlo de una forma que permita su fácil replicabilidad y extensión. Esto incluye el uso de software libre, el uso de prácticas modernas en la compartición de recursos científicos, la creación de documentación exhaustiva y detallada, y la estructuración cuidadosa de los datos.

Además, propongo no sólo utilizar scripts ad-hoc, sino crear software distribuible y modular, que pueda ser útil para terceros y que encapsule los resultados teóricos y técnicos obtenidos en la investigación. Este software, acorde con los principios de la cienca abierta, será publicado bajo licencias de software libre.

Este enfoque no solo aumenta la transparencia y robustez de mi propio trabajo, sino que también facilita el avance del campo en general y la construcción de futuras investigaciones basándose en él.

1.3. Estructura del documento

Esta tesis recoge los resultados de mi investigación y desarrollo, en relación a los objetivos planteados, y en el marco del Programa de Doctorado en Ingeniería Informática de la Facultad de Informática de la Universidad Complutense. Es una tesis por "compendio de publicaciones", es decir, la contribución principal son las publicaciones científicas y técnicas realizadas, contenidas en la segunda parte de este documento. Esta primera parte del documento hace una discusión integradora de la investigación doctoral, sus objetivos y resultados, desde un punto de vista más divulgador y menos técnico que las publicaciones propiamente dichas. Puesto que la mayoría de éstas se han realizado en inglés, además, decidí escribir la discusión integradora en español, no sólo para reconocer la importancia de nuestra lengua y su uso en el contexto nacional e internacional, sino también para permitir que mis resultados y pensamientos sean accesibles a otras comunidades que quizá no estén tan acostumbradas al inglés y al lenguaje técnico.

La primera parte de la tesis incluye una revisión crítica del estado de la cuestión (capítulo 2), diferente de la incluida en los artículos publicados y orientada a poner al lector en situación y motivar el resto de la investigación. El proyecto VisSE, núcleo de la tesis, se narra a continuación, en el capítulo

3, y el capítulo 4 concluye la discusión integradora contenida en la primera parte de la tesis, para dar paso al compendio de publicaciones.

Las publicaciones recogidas en la segunda parte de esta tesis incluyen un artículo de revista revisado por pares y un ePrint UCM, además de dos artículos de conferencia, también revisados por pares y presentados conjuntamente en Marsella en 2022 debido a la pandemia de 2020. Las dos publicaciones restantes son manuales técnicos. El orden en el que se presentan estas publicaciones intenta seguir una estructura narrativa y lógica, más que cronológica o de importancia.

La primera publicación (captítulo 5) se corresponde con la propuesta del proyecto VisSE, enviada al taller internacional sobre lenguas de signos *9th Workshop on the Representation and Processing of Sign Languages: Sign Language Resources in the Service of the Language Community, Technological Challenges and Application Perspectives*. La segunda publación (capítulo 6) detalla el proceso de diseño y recolección del corpus de datos subyacente a la investigación, y la tercera (capítulo 7) es una reproducción del manual de anotación elaborado, que sirve también como documento de diseño. La cuarta publicación (capítulo 8, publicada en la revista IEEE Access) abarca todo el proyecto VisSE, e incluye los detalles y experimentos más relevantes desde el punto de vista del aprendizaje automático, así como nuestra propuesta fundamental de un "pipeline" combinado de aprendizaje profundo y conocimiento experto. La quinta publicación (capítulo 9) se corresponde a la presentación de nuestro software, Quevedo, en la conferencia *Language Resources and Evaluation Conference* (LREC). El capítulo 10 reproduce el manual técnico de uso de éste.

Por último, el apéndice A reúne otras publicaciones y desarrollos realizados durante el doctorado, pero que no encajan en este documento por distintos motivos.

2. Análisis crítico del estado de la cuestión

El primer objetivo específico de esta tesis es la adquisición de una sólida base de conocimiento científico-técnico en relación con la Lengua de Signos Española. Este objetivo no sólo implica el aprendizaje del idioma en sí, sino que también requiere una revisión exhaustiva de la literatura científica relevante al estado actual de la cuestión. En este capítulo se ofrece no solamente una narración descriptiva de la literatura existente, sino también un análisis crítico que evalúa tanto los obstáculos como las ventajas en el campo. De esta manera, se construye paulatinamente una visión integral del problema del tratamiento computacional de la LSE en su estado actual, culminando en la identificación del obstáculo científico-técnico fundamental que será abordado en el siguiente capítulo.

La seccion 2.1 inicia el capítulo con un examen de las diversas aproximaciones metodológicas al reconocimiento de la lengua de signos, dado que este constituye el primer paso hacia la traducción automática. Este último es, por lo general, el objetivo más inmediato que los investigadores establecen cuando se abordan proyectos relacionados con las LS.

Las deficiencias de una aproximación puramente informática nos llevarán a analizar la literatura lingüística en 2.2 para obtener una comprensión más profunda de las características idiosincráticas de las lenguas de signos, tanto en aspectos gramaticales como en su realización física. Esta sección concluye con la observación crítica de que el desafío se intensifica en ausencia de un sistema de representación y captura de la lengua eficaz.

En el contexto de esta problemática, la sección 2.3 presenta diferentes métodos para la transcripción de las lenguas de signos. En particular, se incluye una breve introducción a la SignoEscritura, que constituye el objeto

de estudio del proyecto VisSE, núcleo técnico de la tesis descrito en el capítulo siguiente.

2.1. Reconocimiento automático de lenguas de signos

El primer paso para traducir la lengua de signos consiste, como es de esperar, en su captura. En este ámbito, es posible identificar dos enfoques principales: el reconocimiento de signos a partir de imágenes o vídeos, y la captura directa de los movimientos del signante mediante hardware de captura de movimiento ("motion capture", Lu et al. 1998).

La aproximación mediante tecnologías de motion capture representa un desafío debido a la complejidad del hardware necesario. Este enfoque requiere un montaje sofisticado que incluye la instalación de cámaras en un espacio especialmente designado, como un laboratorio, así como la presencia física de los informantes en dicho espacio. Adicionalmente, los sujetos deben colocarse marcadores en el cuerpo, lo cual demanda tiempo para su preparación y calibración (Parvini et al. 2009; Benbasat y Paradiso 2002). Los guantes de captura de movimiento actuales son costosos y, según nuestra experiencia, carecen de la precisión requerida para registrar los intrincados movimientos internos de la mano que son comunes en las lenguas de signos. Sin embargo, pueden ser útiles para analizar movimientos más amplios y la velocidad del signado (Crespo Vidal 2020; Krebs et al. 2021).

Por otro lado, el enfoque basado en visión artificial se vale de cámaras bidimensionales para capturar las articulaciones efectuadas por el signante. Diversos algoritmos de procesamiento buscan ya sea reconstruir la acción tridimensional y luego identificar el signo correspondiente (Trettenbrein y Zaccarella 2021; Fragkiadakis y Putten 2021; Li et al. 2022), o reconocer el signo directamente en el vídeo (Starner, Weaver y Pentland 1998; Cooper, Holt y Bowden 2011).

Antes de la irrupción de las técnicas de aprendizaje profundo, se requerían estrategias variadas para facilitar el reconocimiento automático. Entre estas, la segmentación desempeña un papel crucial al separar elementos relevantes, en este caso las manos, dentro de una imagen. Para ello, se podían utilizar guantes coloreados, técnicas de croma o la basadas en el color de la piel, pero

estas técnica necesitan de calibración constante (Starner y Pentland 1995). Además, dada la naturaleza cinética de los signos, el procesamiento posterior requiere técnicas avanzadas, como por ejemplos Modelos Ocultos de Márkov (Fang et al. 2002).

Las redes neuronales y el aprendizaje profundo han simplificado significativamente estos procesos, ofreciendo además resultados más precisos; Carneiro, Silva y Salvadeo (2021) ofrece una revisión de las posibilidades y limitaciones actuales. El aprendizaje automático emplea grandes volúmenes de datos (corpus, datasets) para ajustar progresivamente una función predictora, minimizando el error de manera iterativa. Esta naturaleza empírica les otorga una ventaja considerable frente a otros paradigmas de la inteligencia artificial, como el procesamiento lógico o basado en reglas. En este sentido, no es necesario investigar y desarrollar las características específicas del dominio de aplicación; estas "emergen" del proceso empírico de ensayo y error.

Las redes neuronales constituyen un tipo de algoritmo dentro de la familia del aprendizaje automático que ha demostrado ser extremadamente eficaz para una amplia gama de tareas. Aunque existen múltiples arquitecturas de red neuronal, todas comparten la esencia de emplear funciones predictoras no lineales. Estas funciones, conectadas en capas sucesivas, permiten que la predicción resultante haga uso de una estructura interna jerárquica, aunque desafortunadamente, esta estructura interna es estocástica y a menudo opera como una "caja negra", complicando su interpretación. En cualquier caso, el aprendizaje profundo o Deep Learning explota la enorme capacidad de cómputo de la que disponemos actualmente para crear y entrenar arquitecturas de redes neuronales de gran escala, pero sobre todo, de múltiples capas, de donde procede el sobrenombre de "profundo". Zhang et al. (2021) dan una exploración técnica de las posibles causas del éxito de generalización de estas redes, o ver Moulton (2020) para una respuesta menos formal pero plausible.

En el contexto del aprendizaje profundo, existe una diversidad notable de arquitecturas. Estas redes han demostrado ser particularmente efectivas en el ámbito de la visión artificial, donde las reglas lógicas tradicionales habían alcanzado limitaciones significativas. Interesantemente, el cortex visual humano está organizado anatómicamente de una manera no muy diferente, con múltiples capas de procesamiento interconectadas que reconocen

jerárquicamente distintas características de la imagen visual (Kruger et al. 2013).

El desarrollo y éxito de la visión artificial, tanto previo como posterior a la llegada del aprendizaje profundo, ha incentivado la búsqueda de nuevas áreas de aplicación, incluido el reconocimiento de gestos manuales. Las primeras aproximaciones se centraron en identificar gestos estáticos, como el "pulgar hacia arriba" o el signo de "OK" (Freeman y Roth 1995; Jimoh, Ajayi y Ogundoyin 2020). Estos métodos se pueden extender de manera directa para reconocer el alfabeto manual de diversas LS (Makarov et al. 2019; Rumi et al. 2019; Aung et al. 2020), aunque este enfoque no captura la riqueza dinámica y multimodal de las lenguas de signos: la dactilología o alfabeto manual es una técnica útil para representar palabras de la lengua oral deletreándolas, pero no es el mecanismo principal con el que se crean los signos nativos de la lengua.

Enfoques más sofisticados buscan reconocer signos completos a partir de fuentes de vídeo (Starner, Weaver y Pentland 1998; Matsuo et al. 1998; Miyazaki, Morita y Sano 2020; Vázquez-Enríquez et al. 2021). Para ello, se requieren grandes volúmenes de datos anotados, que a pesar de estar disponibles en algunos corpus (Hanke et al. 2020; Hassan et al. 2020), tienen sus propias limitaciones y desafíos, como la privacidad de los informantes y los derechos de imagen. En Bragg et al. (2020) se puede leer una extensa discusión sobre la recolección de corpus de LS y los distintos desafíos involucrados, y en nuestro país existen múltiples iniciativas específicas para la LSE (Santiago et al. 2019; Carmen Cabeza-Pereiro et al. 2016; Shterionov et al. 2021). En general, la literatura sobre reconocimiento o traducción de lengua de signos aborda diversos retos relacionados con la multimodalidad, la escasez relativa de datos y el uso de vídeo, pero a menudo presupone un modelo subyacente que asigna a cada signo una palabra correspondiente de la lengua oral. En este contexto, se suele llamar a esta palabra "glosa".

No obstante, como señalan Ong y Ranganath (2005), este modelo de glosas no es suficiente para abordar el reconocimiento o traducción automática de LS. Por un lado, las frases en lengua de signos tienen un orden distinto de la oración, y una gramática propia distinta que la de la lengua oral. Una posible aproximación es, una vez obtenidas las glosas, reordenarlas para obtener una frase en lengua oral (Camgoz et al. 2018). Esta tarea, que se conoce como

"alineamiento", es de por sí una tarea crítica en la traducción automática de lenguas orales que, a pesar de su relevancia, no ha sido aún completamente resuelta, como refleja el informe de los propios ingenieros de Google sobre el asunto: Isaac Caswell y Bowen Liang (2020). Se puede leer una discusión informal pero entretenida y recomendable sobre este tema en Hofstadter (2018).

Más allá de estas complicaciones compartidas con la traducción de lenguas orales, la LS presenta desafíos específicos. En particular, la relevancia sintáctica del espacio y del movimiento, como apuntan investigaciones recientes como la de Rodríguez y Martínez (2021), así como la morfología interna de los signos, constituyen aspectos fundamentales para comprender su significado. Elementos como la posición de la mano, la dirección hacia la que apunta o el número de repeticiones del signo son variaciones morfológicas que influyen en el significado y que resultan irrepresentables mediante palabras en lengua oral. Además, la expresión facial emerge como un articulador adicional crucial para captar el sentido completo del signo, tal como se evidencia en Porta-Lorenzo et al. (2022).

Por lo tanto, para efectuar un reconocimiento preciso de la LS, no es suficiente con aplicar métodos preexistentes de visión artificial o de traducción automática. Se requiere un estudio exhaustivo de la LS desde una perspectiva de lingüística computacional, lo cual permitiría diseñar modelos más ajustados a las particularidades de esta lengua. En la siguiente sección abordamos este estudio, adquiriendo el conocimiento necesario sobre la estructura propia de las LS que nos permitirá avanzar en su tratamiento computacional efectivo.

2.2. Lingüística de las lenguas de signos

En esta sección hacemos una revisión de las características lingüísticas de la LS, con especial atención a la LSE. Empezamos con unas breves notas generales, para luego sumergirnos en un repaso rápido de su gramática y sintaxis, con especial referencia al uso distintivo del espacio. Finalmente, nos detenemos un poco más en la descripción fonológica, donde la divergencia respecto a las lenguas orales es más notable.

Como punto de partida, es esencial destacar que las lenguas de signos no constituyen una invención artificial reciente ni una simple codificación en gestos de la lengua oral. Son, en efecto, lenguas naturales que han surgido en las comunidades de personas sordas, tal como lo propulsó William Stokoe, el "padre" de la lingüística de las lenguas de signos (Stokoe 1960). Es más, la comunidad signante incluye no solo a individuos con deficiencias auditivas de diverso grado, sino también a sus familiares y amigos cercanos. Es relevante mencionar que, en ocasiones, los hijos de personas sordas no presentan deficiencias auditivas pero son, no obstante, hablantes nativos de la lengua de signos: los Coda (Children of Deaf Adults).

Tampoco son las lenguas de signos una mera mímica o teatralización de la realidad, sino que poseen una complejidad estructural que incluye léxico, gramática y una estructura interna bien definida (ver por ejemplo Hulst 2022). La figura 2.1 ofrece una transcripción de dos oraciones en Lengua de Signos Española. Acompañan a la transcripción explicaciones y observaciones sobre distintos rasgos lingüísticos, con el objetivo de presentar algunas de sus características más notables.

Que las LS sean lenguas naturales no puede sorprendernos. La necesidad de comunicación es intrínseca al ser humano, y las lenguas de signos representan una manifestación esperable de esta necesidad. No obstante, cabe destacar que el medio viso-gestual no es solo patrimonio de las comunidades sordas; Deacon (1997) llega a plantear que la comunicación gestual podría ser el origen de todo sistema lingüístico humano, y su presencia habitual en el contexto comunicativo de las comunidades oyentes refuerza este planteamiento.

En cuanto a su aparición y desarrollo, los estudios son concluyentes al demostrar su naturaleza igualmente "orgánica" o "natural" comparada con las lenguas orales. Senghas y Coppola (2001) ofrece un relato fascinante de la emergencia de una lengua de signos en Nicaragua, en el que niños sordos escolarizados de forma separada desarrollaron un sistema lingüístico desde cero. Esta observación directa confirma que las lenguas de signos son lenguas humanas legítimas, con la misma naturalidad que sus contrapartes orales. Sandler et al. (2014) amplía este cuerpo de evidencia con ejemplos adicionales.

Como lenguas naturales, las LS también tienen una genealogía propia, en la que la Lengua de Signos Francesa (*Langue des Signes Française*, LSF) ostenta

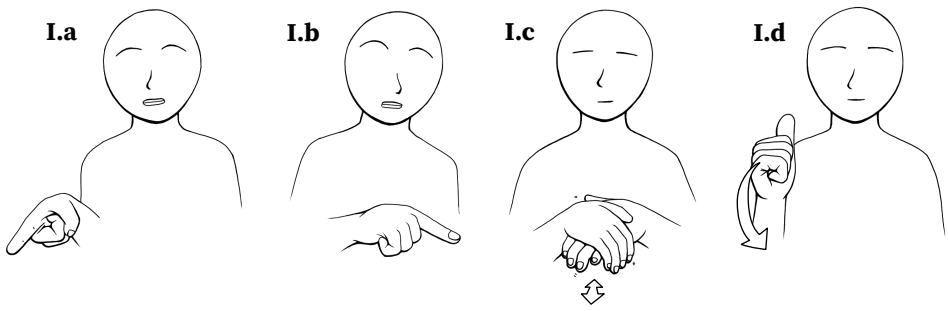
un papel prominente. A través de la influencia de educadores franceses para sordos, la LSF ha dado origen a diversas lenguas de signos en todo el mundo, incluida la Lengua de Signos Americana (*American Sign Language*, ASL), que hoy en día tiene una presencia significativa en los medios audiovisuales. Como dato interesante, la ASL no está emparentada con la Lengua de Signos Británica (*British Sign Language*, BSL), demostrando la independencia de las LS del contexto oral. En España, la LSE es ampliamente utilizada en el territorio nacional y goza de reconocimiento oficial[1], al igual que la Lengua de Signos Catalana (LSC). Plann (1997) nos remite a una de las referencias más antiguas en el ámbito de la educación para personas sordas, localizada precisamente en nuestro país, y Parkhurst y Parkhurst (2001) ofrece un análisis comparativo de las variaciones dialectales en España.

2.2.1. Visión de conjunto gramatical

Aunque no se deriven directamente de las lenguas orales, las lenguas de signos comparten con ellas numerosas características que posibilitan su análisis mediante el aparato teórico de la lingüística. La descripción subsiguiente es una síntesis propia que se nutre de múltiples fuentes académicas, entre las que destacan Brentari (2019), Branchini y Mantovan (2020) y Quer, Pfau y Herrmann (2021), y sobre todo Herrero Blanco (2009), cuyo trabajo constituye una descripción gramatical exhaustiva y meticulosa de la Lengua de Signos Española. Se han incorporado referencias adicionales donde se han estimado pertinentes para enriquecer el argumento.

Las diferencias fundamentales entre las lenguas de signos y las lenguas orales vienen sobrevenidas principalmente del uso único del espacio y el movimiento por las LS (Rodríguez Redondo et al. 2008). Adicionalmente, es crucial señalar la variabilidad y grados de libertad de los articuladores utilizados—tales como manos, cuerpo y cara—en comparación con la predominancia del aparato fonador, menos multifacético, en las lenguas orales. Estas diferencias son particularmente evidentes a nivel de expresión superficial pero tienden a atenuarse a medida que ascendemos en los niveles de abstracción del lenguaje. De hecho, a nivel cognitivo, las diferencias son

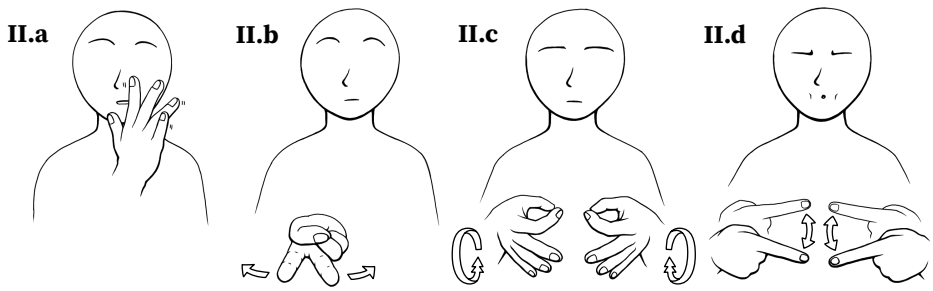
[1] *Ley 27/2007, de 23 de octubre, por la que se reconocen las lenguas de signos españolas y se regulan los medios de apoyo a la comunicación oral de las personas sordas, con discapacidad auditiva y sordociegas* (2007)



CEJAS-ABIERTAS

DEM.3SG\A DEM.3SG\B amigo de_toda_la_vida

«Él y ella son amigos de toda la vida.»



CEJAS-ABIERTAS *CEJAS-FRUNCIDAS*

ahora DEM.3PL\AB siempre discutir

«Ahora están siempre discutiendo.»

Fig. 2.1. – (página opuesta)
Transcripción ilustrada de dos oraciones en LSE. Bajo las ilustraciones de los signos, glosas para cada uno de ellos, utilizando bien palabras del castellano que aproximan el significado, bien términos lingüísticos. Bajo las glosas, traducción más o menos libre al castellano.
I.a, I.b – El signo deíctico, realizado señalando con un dedo, actúa en este caso como demostrativo de tercera persona. El gesto bucal con los labios abiertos pero los dientes cerrados es parte léxica del signo. Las distintas terceras personas se ubican en el espacio, señalándolas con el dedo e inclinando ligeramente el cuerpo. En español se han traducido con género para distinguirlos, pero en LSE no existe el género gramatical. En una situación real, lógicamente, los referentes estarían identificados en el contexto del discurso.
I.a, I.b, II.a, II.b – las cejas se alzan para marcar el tópico principal de la oración (no exactamente coincidente con el sujeto, nótese el tópico contrastivo "ahora/actualmente" en II.a).
I.c – El número gramatical es un fenómeno complejo en LSE debido a las interacciones con el espacio de signado, y no funciona como en otras lenguas a las que podamos estar acostumbrados. En este caso, por ejemplo, el signo "amigo" no tiene información de número, por lo que no tiene que concordar con el sujeto.
I – La oración copulativa en LSE no necesita de verbo principal, sino que se forma con la yuxtaposición del tópico principal (las dos terceras personas de I.a y I.b) y el atributo (I.c.), como ocurre también en lenguas orales como el ruso o el japonés.
II – En castellano, el sujeto de la segunda oración se puede omitir, al incluir el verbo principal información de persona con la que reconstruirlo. En LSE, en cambio, el sujeto es obligatorio (II.b), como en francés, inglés, o muchas otras lenguas.
II.b – el pronombre plural "ambos/los dos" incluye con su movimiento a los referentes que han sido colocados anteriormente en el espacio. Si los referentes estuvieran en otras posiciones (por ejemplo, incluyeran la primera o segunda persona) el movimiento y la orientación de la mano cambiaría para incluir los lugares sintácticos correspondientes. Si la cantidad de elementos del sintagma plural también fuera distinta, cambiaría también la configuración, usándose tres dedos para tres elementos y así sucesivamente.
II.c – la entrada léxica subyacente para el adverbio "siempre" incluye movimientos circulares consecutivos. No obstante, se pueden realizar menos o más repeticiones, o repeticiones incompletas, para ajustar el signo a la velocidad y ritmo del discurso, o para añadir matices al significado.
II.d – la expresión facial intensifica la acción. Combinado con "siempre" de II.c, la traducción al castellano se ha elegido con el aspecto verbal continuo para reflejar el matiz correcto. En ocasiones la expresión facial es parte léxica del signo, en otros casos es información sintáctica (como en el tópico principal, o en las oraciones interrogativas); también puede usarse para expresar la emoción del signante. En este sentido, la expresión facial y corporal actúan de manera muy similar a la entonación en la lengua oral.

mínimas o quizás inexistentes, más allá de la variación interlingüística que podría esperarse.

Desde un enfoque macroestructural, las lenguas de signos no muestran divergencias significativas respecto a las lenguas orales en términos de organización discursiva. El uso del espacio se manifiesta principalmente para representar lugares o conceptos de manera icónica, así como el transcurso del tiempo (Stamp, Dachkovsky y Sandler 2020). El discurso se articula de forma secuencial, mediante oraciones independientes que estructuran la información según un patrón de tópico-foco. Los marcadores no manuales, como el alzamiento de cejas o la inclinación de la cabeza, sirven para demarcar el tópico principal de la oración (Kimmelman 2015; Aarons 1996). Aunque este fenómeno puede parecer exótico, muy distinto al español o el inglés, es en cambio similar a la estructura informativa del japonés, que inicia también las oraciones con el tópico principal, y lo marca explícitamente con la partícula は (pronunciada en este caso como ~*wa*).

También como en japonés (y otras lenguas), en las lenguas de signos el verbo suele ocupar la posición final de la oración. Sin embargo, es relevante señalar que el orden de los complementos se presenta de manera inversa a lo que se suele encontrar en lenguas de este tipo. A pesar de estas particularidades, la sintaxis y las categorías gramaticales de las LS son plenamente compatibles con los paradigmas de análisis establecidos en la lingüística de las lenguas orales, aunque el uso del espacio sigue estando presente, y es fundamental incorporarlo como un rasgo morfológico y sintáctico crucial (Barberà Altimira 2015).

Los distintos lugares del espacio de signado se pueden utilizar para "ubicar" referentes, lo que permite hacer referencia a ellos posteriormente al señalar su posición correspondiente (Tkachman 2016). En este contexto, el espacio funciona casi como una especie de tercera persona, pero con posibilidades expresivas inexistentes en la lengua oral. Por ejemplo, el signante tiene la opción de mover el torso para "situarse" en uno de estos roles de tercera persona, lo que da lugar al habla en estilo directo o "cita". Este movimiento sutil, además, tiene un impacto fonológico en todas las formas presentes en el texto citado, modificando su estructura espacial para adecuarse al nuevo "signante".

Adicionalmente, numerosos verbos pueden "conjugarse" espacialmente, mediante la modificación o adición de un desplazamiento al signo para que comience, pase por o termine en diferentes espacios del espacio de signado. Estos espacios señalados se convierten en los argumentos del verbo, llenando las casillas correspondientes de sujeto, objeto, etc., de acuerdo con el orden en que aparecen (Holler y Steinbach 2018). Es también viable formar sintagmas nominales múltiples que son más expresivos que los permitidos por las conjunciones de la lengua oral, simplemente mediante la distribución y agrupación significativa de referentes en el espacio. Aunque estos aspectos no representan desafíos insuperables para la lingüística, lo cierto es que aún no existe un marco teórico consolidado que aborde de manera integral el uso del espacio en las lenguas de signos[2].

En términos fonológicos y fonéticos, las lenguas de signos divergen considerablemente de las lenguas orales. De hecho, hay quienes argumentan en contra del uso del término "fonología" debido a sus connotaciones oralistas (Lucas y Valli 1989; Sutton-Spence y Woll 1999) y prefieren emplear el término "quirología" (contrastando la etimología griega para mano, χείρ *~kheír*, con sonido, φωνή *~foné*). Independientemente de la nomenclatura, existe un nivel de análisis en las lenguas de signos, vinculado al medio físico, que encuentra su equivalente en la fonología y la fonética de las lenguas orales. En estas últimas, el canal de transmisión principal es el auditivo, basado en una onda de presión unidimensional, lo que configura su naturaleza lineal. En cambio, en las lenguas de signos el canal de transmisión es viso-gestual, y se apoya en la visión y el movimiento tridimensionales. Los articuladores en las LS son también más numerosos que en las lenguas orales (ambas manos, el cuerpo, la cara, etc.) y operan de manera paralela y relativamente independiente. Esta multiplicidad confiere a las lenguas de signos una calidad de simultaneidad que no es inherente a la lengua oral, además de un gran rango de posibilidades expresivas. Es por ello que la descripción fonológica resulta frecuentemente el tema más prominente en la investigación lingüística de las LS, y también por ello le dedicamos la siguiente sección.

[2] En mi opinión, este es un tema de investigación fascinante no sólo en términos de lengua de signos sino para descubrir las posibilidades y limitaciones de la capacidad del lenguaje humano.

2.2.2. Modelo fonológico QOLDF

Los gestos lingüísticos paralelos y simultáneos que mencionábamos, y que conforman los signos en las LS, son organizados por la mayoría de los autores en torno a varios rasgos fundamentales. Tanto la mano dominante como la no dominante (si se utiliza) adquieren valores específicos para cada uno de los rasgos de configuración (Q), orientación (O) y lugar (L). La **configuración** es el rasgo que describe la disposición de los dedos, ofreciendo distintas "formas" o configuraciones de la mano. Este rasgo muestra una variabilidad significativa entre las diferentes lenguas de signos, tanto en términos del inventario de configuraciones utilizadas como en los rasgos que se consideran distintivos. En el caso de la LSE, el repertorio de configuraciones es algo superior al medio centenar, aunque hay cierta discrepancia en el listado entre diferentes autores.

Además de la configuración, la **orientación** de la mano en el espacio constituye otro rasgo de relevancia lingüística (ver figura 2.2). Algunos investigadores, como Herrero Blanco (2009), abordan este rasgo centrándose en la rotación de la muñeca. Sin embargo, la dirección en la que la mano apunta puede tener relevancia sintáctica, lo cual nos llevó a proponer una perspectiva alternativa en Sevilla y Lahoz-Bengoechea (2019). El tercer rasgo "espacial", denominado **lugar**, especifica la ubicación de la mano, ya sea en relación al cuerpo del signante o dentro del espacio de signado. Es también crucial determinar si existe o no contacto, ya sea con el cuerpo del signante o con la mano no dominante.

Adicionalmente a estos rasgos "estáticos" (QOL), los signos suelen incorporar algún elemento de movimiento. Autores prominentes, incluyendo a Herrero Blanco, distinguen entre movimientos formales (F) o internos de la mano (como giros o cambios en la flexión de los dedos) y movimientos "externos" o desplazamientos (D), que implican cambios en la ubicación de las manos. Es fundamental además definir rasgos que determinen qué manos se utilizan, cuál es la relación entre ellas (por ejemplo, si son simétricas o si una sirve de apoyo a la otra) y si el movimiento se repite y de qué manera. Herrero Blanco, sin embargo, agrupa todos estos elementos bajo la categoría general de movimientos (M).

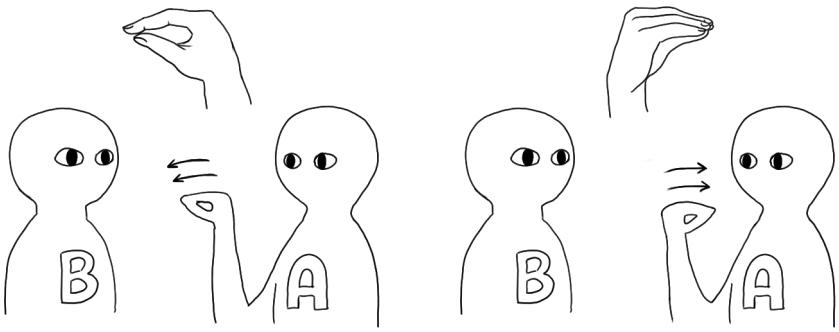


Fig. 2.2. – Distintas conjugaciones del signo Preguntar. Dependiendo de la orientación de la mano y el movimiento, el signo "Preguntar" puede tomar distintos valores argumentales. En la izquierda, el signante "A" pregunta a "B" (*yo te pregunto*), mientras que en la derecha "B" es el sujeto del verbo, y "A" el objeto directo (*tú me preguntas*). Original: Sevilla y Lahoz-Bengoechea (2019).

En la figura 2.3 se ilustran sobre un ejemplo los diversos parámetros que hemos discutido. No obstante, es importante no olvidar las expresiones faciales y corporales. Aunque suelen ser obviadas por muchos autores debido a la dificultad de su análisis y su impacto relativamente menor, tienen la capacidad de portar significado tanto a nivel léxico como prosódico y paralingüístico. Kocab, Senghas y Pyers (2022), por ejemplo, describen un caso de estudio sobre la emergencia y fijación gramatical de los marcadores no manuales en las oraciones interrogativas de la lengua de signos emergente en Nicaragua.

Este enfoque basado en rasgos que hemos visto está bastante extendido, pero existen otras aproximaciones a la descripción lingüística que plantean propuestas alternativas. Liddell y Johnson (1989) sugiere un modelo fonológico renovado para el movimiento de las manos, centrado en un patrón alternante de gestos estáticos y dinámicos, y Tyrone et al. (2010) ofrece otro enfoque basado en el paradigma contemporáneo de la fonología articulatoria.

Como hemos visto, los múltiples rasgos fonológicos y la utilización del espacio constituyen características lingüísticas singulares de las lenguas de signos. Aunque algunas veces es posible describirlas con terminología lingüística ya existente, a menudo presentan notables diferencias con respecto a sus homólogos orales. No obstante, estos rasgos son esenciales para capturar

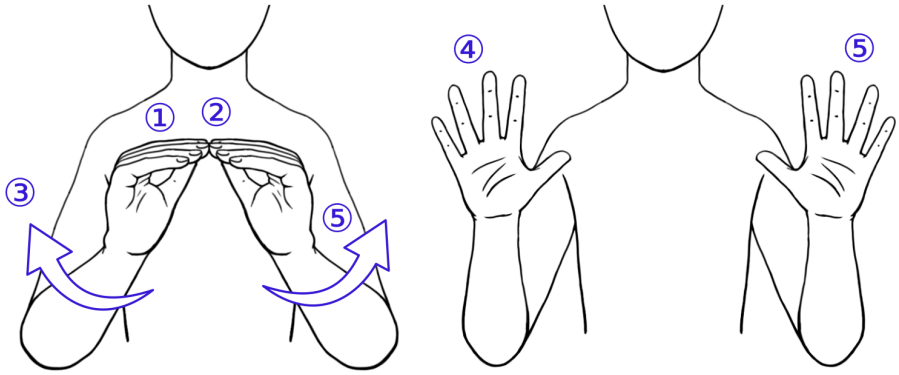


Fig. 2.3. – Ilustración del signo idiomático "No hacer nada". La configuración (Q) inicial es de "pinza recta", y la orientación (O) es contralateral. Estos son los rasgos "internos" de la mano, marcados con ①. El lugar (L) es neutro, centrado, y con contacto entre las manos ②. El signo incluye dinamicidad, capturada en ③, en la que las manos rotan sobre la muñeca y se desplazan ipsilateralmente. Además, la configuración también cambia, extendiéndose los dedos. El estado final de la mano (Q, O, y L de nuevo) se representa en ④. Finalmente, ⑤ nos llama la atención al hecho de que la mano no dominante realiza, de manera completamente simétrica, los mismos gestos articulatorios que la mano dominante. Este rasgo se suele recoger bajo la denominación de "bimanualidad". Figura original de Lahoz-Bengoechea y Sevilla (2022b).

la totalidad de su gramática y significado, lo cual resulta indispensable si se aspira a un tratamiento computacional efectivo de las lenguas de signos.

Para ello, se requiere de una representación digital más robusta que la que proporcionan las glosas. Estas son palabras de una lengua oral que traducen, con mayor o menor exactitud, el significado de los signos y que se pueden emplear para representar una oración signada como una secuencia de "palabras". Sin embargo, esta estrategia resulta insuficiente para capturar los rasgos espaciales y dinámicos que son fundamentales tanto a nivel sintáctico como semántico en las lenguas de signos.

Adicionalmente, las glosas son incapaces de capturar la fonología inherente a los signos. Si nuestro objetivo es procesar la lengua a nivel del medio visogestual —como sería en el caso de reconocimiento de vídeo o la generación de avatares signantes— es imprescindible representar de manera fidedigna los

distintos rasgos que conforman los signos. Para esto, no podemos depender de la lengua oral como sustituto, sino que se hace necesario recurrir a una representación específica que respete las particularidades de las lenguas de signos, y en la siguiente sección veremos distintas opciones para ello.

2.3. Representación de las lenguas de signos

Dentro del ámbito de las ciencias de la computación y la lingüística, existe una marcada inclinación hacia la búsqueda de mecanismos estructurados para representar la información. En el contexto de las lenguas orales, modelos como los árboles de análisis sintáctico y las gramáticas formales gozan de amplia aceptación y uso. Estos enfoques, por lo general, ponen un especial énfasis en aspectos como la sintaxis y la semántica. Sin embargo, su aplicabilidad no se limita a estos niveles, sino que puede extenderse para abarcar también la morfología (Aronoff 1993) y la fonología, tal como se manifiesta en modelos como la fonología autosegmental (Goldsmith 1976). Este enfoque es también válido para las LS, y en la siguiente sección veremos varias representaciones emmarcadas en este paradigma. En 2.3.2, sin embargo, pasaremos a un modelo completamente distinto: la SignoEscritura.

2.3.1. Representaciones fonológicas estructuradas

Dada la intrincada naturaleza fonológica de las lenguas de signos, los modelos estructurales diseñados para describirlas tienden a enfocarse predominantemente en aspectos fonológicos. Uno de estos sistemas, denominado "AZee" (Filhol y Braffort 2007; Hadjadj, Filhol y Braffort 2018) y diseñado específicamente para la LSF, propone un formalismo altamente computacional que se centra en la ejecución precisa de los signos. Gracias a esto, permite un amplio procesamiento computacional, incluyendo la generación de avatares signantes que ostentan un alto nivel de expresividad y corrección (Filhol, McDonald y Wolfe 2017).

Otros modelos adoptan un enfoque más parecido a la descripción QOLDF detallada en la sección anterior, concentrándose en los parámetros fonológicos del signo. Stokoe (1960), en su descripción lingüística fundacional de la ASL, diseñó un sistema de notación en el cual diferentes símbolos re-

presentan distintos valores para cada rasgo fonológico. Estos símbolos se concatenan en un orden específico, generando aparentemente una estructura lineal similar a la escritura en las lenguas orales. No obstante, es crucial reconocer que los parámetros de los signos ocurren de forma simultánea en la realidad, haciendo que su ordenación secuencial sea simplemente arbitraria y aparente.

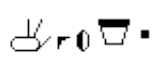
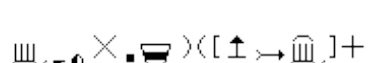
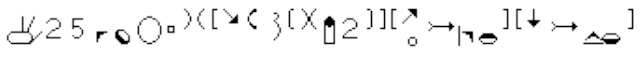
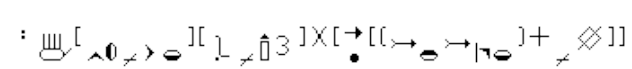
Construyendo sobre este mismo principio de secuenciación, el sistema de notación de Hamburgo (HamNoSys, Hanke 2004) aumenta el inventario de símbolos disponibles y refina la descripción lingüística subyacente. HamNoSys aspira a incorporar una gama más amplia de características de las lenguas de signos, incluidas aquellas que son específicas de lenguas de signos distintas de la ASL, particularmente la Lengua de Signos Alemana (*Deutsche Gebärdensprache*, DGS). Diseñado con la era digital en mente, HamNoSys suele codificarse en XML, reflejando claramente su naturaleza estructurada. La tabla 2.1 ofrece una comparativa entre HamNoSys, la notación de Stokoe y la SignoEscritura, de la cual hablaremos más adelante.

En España, Herrero Blanco (2003) introdujo un sistema de escritura alfabética (SEA) que emplea los caracteres latinos convencionales, asignando a cada uno o a combinaciones de ellos los distintos posibles valores de los rasgos de la LSE.

Similares al SEA, han ido surgiendo otros sistemas que utilizan caracteres latinos, facilitando su entrada a través de un teclado convencional y haciéndolos más accesibles para el público en general. Este es el caso del modelo propuesto por Eccarius y Brentari (2008), que se enfoca exclusivamente en representar la configuración de la mano, y de nuestro propio sistema, denominado "Signotación," que fue presentado en la Sociedad Española de Lingüística (Lahoz-Bengoechea y Sevilla 2022b). Además de los caracteres latinos, la "Signotación" incorpora también números y símbolos ASCII, fácilmente accesibles en un teclado habitual, y se fundamenta en un modelo fonológico de nueva creación. Una implementación práctica de este sistema se puede ver en el Signario[3] (Lahoz-Bengoechea y Sevilla 2022a), donde se utiliza como representación subyacente para almacenar y recuperar signos de la LSE.

[3] https://griffos.filol.ucm.es/signario

Tab. 2.1. – Comparativa de sistemas de notación para las lenguas de signos, usando palabras de una traducción a ASL de la historia de Rizitos de Oro[a]. Original: tabla 5.1.

	Notación de Stokoe	HamNoSys	SignoEscritura
Tres	$3^{\perp}$		
Osos	$[]\checkmark C^{\mp}\checkmark C_{X}^{v\cdot}$		
Rizitos de oro	$\lrcorner Y_{v}^{\circlearrowleft}$		
Bosque profundo	$\overline{B}_{a}\checkmark B_{\wedge}^{\circlearrowleft}$		

[a] http://www.signwriting.org/forums/linguistics/ling001.html

La ventaja intrínseca de los sistemas estructurados radica en su capacidad para capturar la información semántica y sintáctica que está implícita en la lengua, más allá de su manifestación superficial. Estos sistemas habilitan un procesamiento computacional altamente eficiente y robusto, dado que la estructura de la lengua se encuentra ya codificada en la representación.

Sin embargo, las representaciones estructuradas también albergan diversas desventajas. En primer lugar, están basadas en un paradigma lingüístico concreto, que podría resultar inexacto o insuficientemente preciso para capturar todas las complejidades de la lengua. Esto es especialmente problemático en el contexto de las lenguas de signos, en el que no existe un marco teórico unánimemente aceptado o suficientemente exhaustivo. En segundo lugar, tales representaciones tienden a ser altamente académicas y técnicas, lo que dificulta su uso y comprensión por parte de la población general. Esta limitación reduce su aplicabilidad a contextos y circunstancias específicos, restringiendo así la diversidad de datos potencialmente disponibles y su representatividad en los distintos registros de la lengua. Además, cualquier signante que desee contribuir como informante tendría primero que ser instruido tanto en el paradigma lingüístico subyacente como en el uso específico del sistema de transcripción.

2.3.2. SignoEscritura

Contrastando con las representaciones estructuradas, en las lenguas orales la escritura constituye la modalidad más convencional que los hablantes emplean para representar su idioma. Esta tecnología, que frecuentemente se da por sentada pero que representa uno de los inventos más significativos de la historia humana, consiste en transcribir la manifestación superficial (sonidos) del lenguaje en símbolos abstractos, como letras, jeroglíficos o ideogramas[4].

Los sistemas de escritura descansan en gran medida sobre la intuición lingüística del lector, dado que no codifican de forma explícita ni el significado ni la gramática inherente a la lengua. Aunque esto puede considerarse una desventaja desde la perspectiva del procesamiento computacional, ofrece el

[4]Los ideogramas, a pesar de poseer una carga conceptual aparentemente más elevada, son en esencia representaciones de los sonidos del habla.

beneficio de capturar la lengua en su estado más puro y natural. Al no estar vinculada a un modelo lingüístico específico, la escritura permite una mayor flexibilidad y variabilidad en la representación del lenguaje. Para el hablante nativo, utilizar su propia representación mental del lenguaje no constituye una dificultad, y por otro lado no requiere aprender un modelo formal más complicado y académico.

En el caso de las lenguas de signos, existe un modelo que comparte estas características: la SignoEscritura (*SignWriting*, Sutton 1995).

La SignoEscritura ofrece una transcripción gráfica y altamente visual de las LS, capturando las configuraciones y movimientos tal y como "se ven". Se utilizan símbolos abstractos pero icónicos para representar las manos y otras partes del cuerpo, mientras que las flechas sirven para indicar la dirección del movimiento. Cada uno de estos símbolos, que denominaremos "grafemas", se dispone bidimensionalmente sobre la página para simular el espacio de signado. Designamos como "logograma" a la imagen conformada por los grafemas, que habitualmente se corresponde con un signo individual, palabra o morfema. En la figura 2.4 se pueden observar varios ejemplos, así como sus grafemas constituyentes.

Para representar la orientación de la mano, los grafemas correspondientes pueden rotarse y colorearse de diversas maneras, tal como se muestra en la figura 2.5. El lugar de articulación se indica situando los grafemas en posiciones relativas bidimensionales que equivalen a sus posiciones tridimensionales relativas en el espacio. Asimismo, se emplean distintas formas icónicas para representar la configuración de los dedos, y otros grafemas adicionales que pueden marcar contactos, relaciones dinámicas entre ambas manos, y otros aspectos. La representación de la cara y el cuerpo también se realiza mediante grafemas que son tanto etimológicamente transparentes como fáciles de comprender, aunque permanecen abstractos y arbitrarios.

La SignoEscritura es intrínsecamente fonética, pues transcribe la lengua tal y como se presenta, sin codificar ningún modelo fonológico o gramatical subyacente. Es un sistema independiente de cualquier LS concreta (aunque para transcribirlas todas se ha necesitado realizar algunas extensiones). Su carácter visual facilita tanto su producción como su interpretación, incluso para aquellos únicamente con formación básica en el uso del sistema (Di Renzo et al. 2006). Por estas razones, la SignoEscritura se erige como un

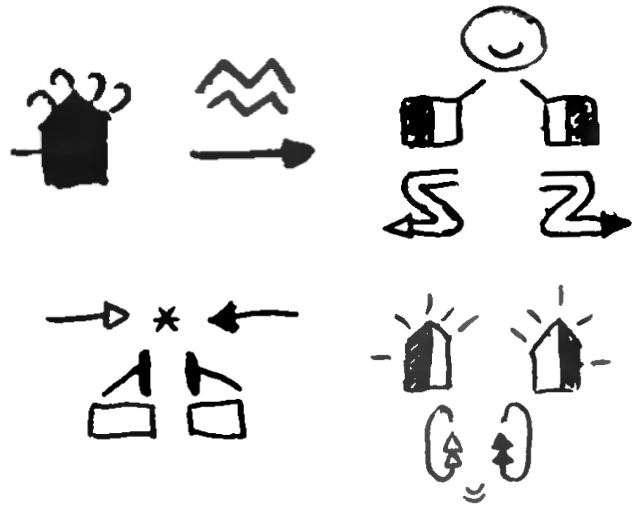


Fig. 2.4. – Algunos logogramas de SignoEscritura para la LSE, provinientes del Corpus VisSE de SignoEscritura del Español (capítulo 6). Arriba a la izquierda ("dactilológico") la mano con la palma hacia el frente se mueve hacia la derecha mientras los dedos se alternan. Arriba a la derecha ("feliz") las manos con el canto hacia el frente y meñique extendido, describen una trayectoria de zigzag simultánea hacia abajo. En la esquina inferior izquierda ("junto") las manos en configuración de "pinza" y con la palma hacia arriba se mueven hasta tocarse. Finalmente, abajo a la derecha ("lengua de signos") las manos hacen movimientos circulares antisimétricos en el plano vertical. Original: figura 6.5.

sistema ideal para la creación de corpus de datos, recursos léxicos y como una forma efectiva de transcribir la LS para su registro y preservación (Sutton y Frost 2008; Galea 2014).

Sin embargo, dada la naturaleza compleja y expresiva de las LS y la aspiración internacional de la SignoEscritura, este sistema es intrincado y extenso. Para una comprensión más detallada, se pueden consultar los capítulos 6 y 7, o las Lecciones de SignoEscritura disponibles de manera gratuita en múltiples idiomas, incluido el español (Sutton 1995). Como ilustración de su uso, la figura 2.6 ofrece una transcripción en SignoEscritura de las oraciones de la figura 2.1.

Y a pesar de las ventajas mencionadas, el aspecto gráfico y visual de la SignoEscritura presenta desafíos para su tratamiento computacional. En la lingüística computacional, una de las piedras angulares es la representación de la lengua como "cadena de caracteres", un formato lineal e intrínseca-

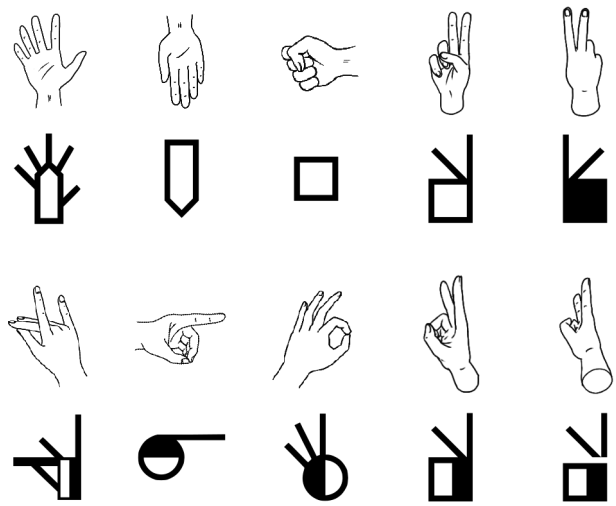


Fig. 2.5. – Algunos grafemas para las manos, incluyendo distintas configuraciones y orientaciones. Original: 8.2, a su vez modificado de Sevilla y Lahoz-Bengoechea (2019).

mente incompatible con la naturaleza bidimensional de la SignoEscritura. Existen esfuerzos para codificar esta como texto plano (Sutton y Slevinski 2010; Slevinski 2016), pero no resuelven el problema fundamental de su bidimensionalidad y simultaneidad. Además, difieren significativamente de las nociones convencionales de "carácter" y "palabra" en las lenguas orales, lo que impide en cualquier caso su uso directo en algoritmos convencionales de procesamiento del lenguaje.

Por lo tanto, si queremos explotar plenamente el potencial de la SignoEscritura como sistema para representar y procesar las LS, resulta imprescindible optimizar su tratamiento computacional. Abordar este obstáculo es la meta del objetivo técnico de esta tesis, que es abordado en profundidad en el capítulo siguiente así como en las publicaciones relacionadas.

2.4. Conclusiones

En este capítulo, hemos efectuado una revisión del estado actual del tratamiento computacional de las lengua de signos. En la sección 2.1 hemos comenzado analizando distintos métodos de reconocimiento de LS, ya que la captura precisa del contenido lingüístico de vídeos o imágenes representa el

punto de partida esencial para cualquier tipo de traducción o procesamiento subsiguiente. Al respecto, hemos explorado diversas técnicas de IA y visión artificial, tales como el aprendizaje profundo, que allanan el terreno para un procesamiento que sin ellas se torna impracticable.

Sin embargo, para realizar un procesamiento efectivo de la LS es necesario primero entender sus matices y características intrínsecas. Solamente mediante una comprensión profunda podremos abordar una traducción que capture realmente el significado deseado. En este sentido, en la sección 2.2, hemos llevado a cabo un análisis lingüístico de las LS que nos provee del marco conceptual requerido para trabajar con ella de manera efectiva. Como conclusión de este examen teórico, hemos podido constatar la importancia crucial para nuestro objetivo de contar con sistemas de representación adecuados para las lenguas de signos.

En la sección 2.3 hemos evaluado varios de estos sistemas de representación, culminando con un enfoque en la SignoEscritura. Este sistema se presenta como icónico y visual, siendo intuitivo para los usuarios de las LS y poseyendo la capacidad de capturar, dada su naturaleza altamente fonética, todos los matices de la lengua sin estar circunscrito a un paradigma lingüístico concreto. No obstante, la naturaleza gráfica de la SignoEscritura plantea desafíos para su procesamiento computacional, y superar este obstáculo se convirtió en la meta de los objetivos técnicos de esta tesis, que convergieron en la realización del proyecto VisSE descrito en el siguiente capítulo.

Como resultado adicional del análisis teórico realizado, hemos delineado una descripción sofisticada y basada en rasgos de la LS. Aunque este análisis ha sido esbozado de forma muy superficial en este capítulo, sienta las bases para los distintos desarrollos posteriores, y planeamos publicarlo de manera más completa en una etapa posterior. También podemos ver reflejado este conocimiento indirectamente en el esquema de anotación de SignoEscritura utilizado en el corpus VisSE, como se detalla en el capítulo 7. Este esquema de anotación complejo se deriva directamente del conocimiento lingüístico adquirido, y es lo que posibilita en secciones posteriores la consecución del objetivo técnico de procesamiento mediante IA que nos habíamos planteado.

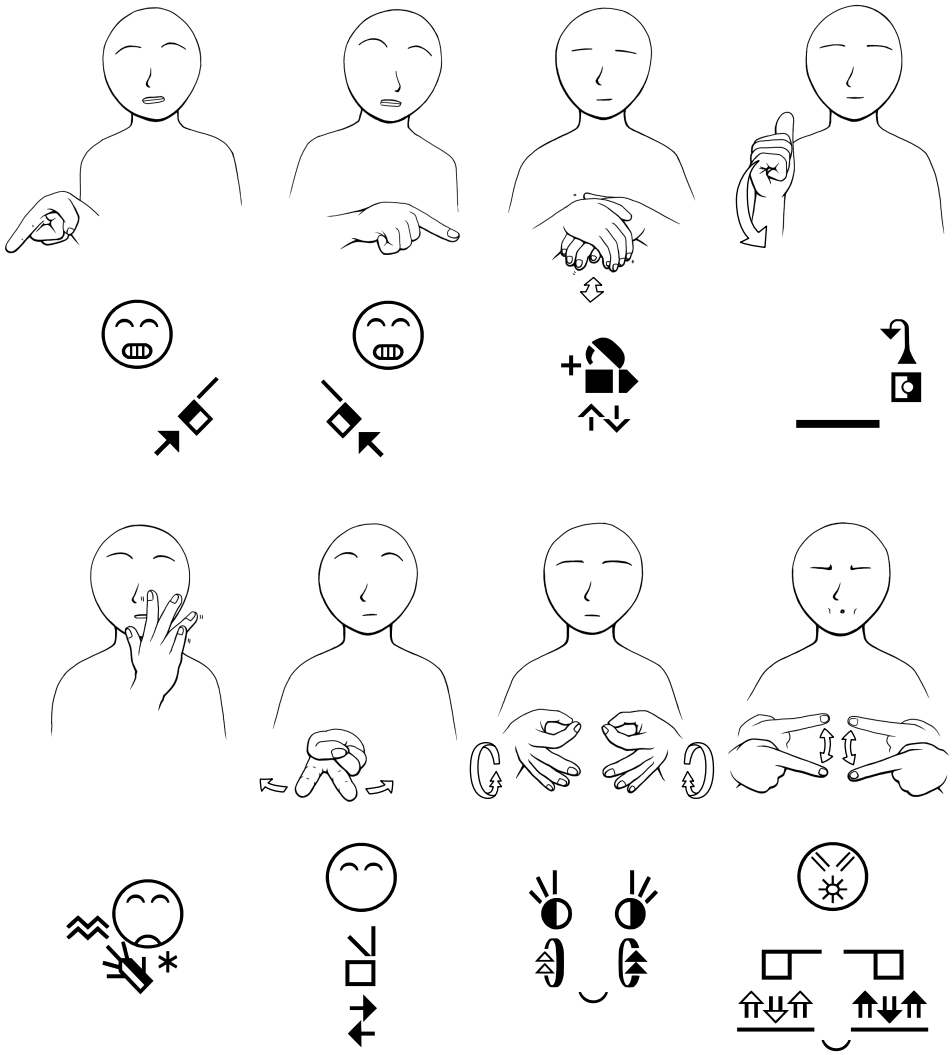


Fig. 2.6. – Transcripción en SignoEscritura de la figura 2.1. Obsérvese que las figuras están dibujadas como las vería un observador, pero la SignoEscritura se escribe desde el punto de vista del signante, por lo que la derecha y la izquierda se intercambian. SignoEscritura adaptada ligeramente de la Fábrica de Señas: `https://www.signbank.org/signmaker/#?ui=es&dictionary=ssp`.

3. Visualizando la SignoEscritura

La investigación que constituye el objeto de esta tesis, con los objetivos delineados en la introducción, se ha prolongado durante varios años, y ha involucrado una variedad de investigaciones y desarrollos. Sin embargo, estos objetivos y elementos diversos se coaligaron en el proyecto de investigación "VisSE" (Visualizando la SignoEscritura), llevado a cabo entre los años 2021 y 2022. Este proyecto proporcionó los recursos y una estructura sólida sobre la que abordar la investigación de manera rigurosa, resultando en la publicación de la mayoría de los artículos que se encuentran en la parte II.

En este capítulo, describimos el desarrollo del proyecto "VisSE" desde una perspectiva tanto narrativa como divulgativa, para proporcionar una visión clara y detallada de cómo los diferentes componentes y objetivos convergieron en un esfuerzo cohesivo, enlazando a las publicaciones correspondientes para la visión más técnica y académica.

3.1. Génesis y evolución del proyecto

En el año 2018, algunos estudiantes de grado estaban interesados en desarrollar un trabajo de fin de grado en el ámbito del deep learning y contactaron con el director de mi tesis. Su interés se centraba en la temática de la visión artificial aplicada a la lengua de signos. Durante aquel período, tanto la visión artificial como la lengua de signos eran temas prominentes, y la fusión de técnicas vanguardistas de inteligencia artificial con la aplicabilidad social de asistir a un colectivo en desventaja resultaba (y sigue resultando) altamente atractiva. Dada nuestra familiaridad con la complejidad del problema, los estudiantes fueron redirigidos a un problema similar pero más acotado: el reconocimiento de SignoEscritura usando aprendizaje profundo. Esto culminó en un trabajo de fin de grado (Sánchez Jiménez, López Prieto y Garrido

Fig. 3.1. – Logo del proyecto Visualizando la SignoEscritura.

Fig. 3.2. – Signo del proyecto VisSE, con fotografías y en SignoEscritura.

Montoya 2019), dando inicio a la investigación en el objetivo previamente identificado: mejorar el tratamiento computacional de la SignoEscritura.

Al tratarse de un trabajo de fin de grado, el tiempo disponible para resolver el problema era limitado, y los estudiantes enfrentaban múltiples desafíos nuevos, como la lengua de signos, la visión artificial, y el trabajo en Linux, entre otros. A pesar de estos obstáculos, el trabajo fue crucial para identificar los principales problemas a superar en esta tarea.

Posteriormente, en el año 2019, Indra y Fundación Universia, con financiación del Banco Santander, anunciaron la IV Convocatoria de ayudas a proyectos de investigación en Tecnologías Accesibles, una convocatoria competitiva a nivel nacional. Ese año, como novedad, se permitió que los estudiantes presentaran propuestas, lo que me llevó a enviar la mía: “Visualizando la SignoEscritura” (VisSE), ver el logo en la figura 3.1 y el signo propio del proyecto en la figura 3.2. El objetivo era resolver el problema del uso de la SignoEscri-

tura en el mundo digital, y tuve la suerte de ser seleccionado como ganador de la convocatoria[1].

El plan previo del proyecto, enviado a la comunidad científica para su discusión en el Noveno Taller sobre la Representación y Procesado de Lenguas de Signos puede ser consultado en el capítulo 5. La pandemia de Covid-19 impidió la presentación presencial de la comunicación, pero gracias a los organizadores fue posible realizarla durante el siguiente congreso, en 2022, presentando además los resultados ya obtenidos. El póster presentado se puede encontrar también en las aportaciones del apéndice A.

El objetivo de VisSE fue desarrollar la infraestructura y algoritmos para abordar el problema que nos ocupa, brindando además una utilidad final para los usuarios en forma de aplicaciones basadas en SignoEscritura. Aunque inicialmente el proyecto era quizá demasiado ambicioso, lo que llevó a reducir el alcance de las aplicaciones finales, los objetivos se alcanzaron, y la investigación básica se completó exitosamente, permitiendo futuros desarrollos.

Una parte complicada del proyecto fue la gestión económica y administrativa. Dado que era estudiante, no personal permanente de la universidad, y primerizo como investigador principal, los trámites burocráticos y la gestión fueron lentos y complejos. Además, la pandemia en 2020 causó un retraso casi de un año en el inicio del proyecto y dificultó el trabajo en persona y los contactos con asociaciones e investigadores. A pesar de ello, se logró llevar a cabo el trabajo, y los resultados se han convertido en el núcleo de esta tesis. A continuación, se presenta una descripción más detallada del desarrollo del proyecto enlazando a los diferentes resultados publicados. En la figura 3.3 se puede observar un esquema de alto nivel de las distintas partes.

3.2. Visión general

El desarrollo de VisSE, cuya meta es cumplir el objetivo técnico de la tesis de procesar la SignoEscritura, puede entenderse a través de tres esfuerzos paralelos, que comprenden distintas áreas relacionadas de la informática

[1] https://www.indracompany.com/es/noticia/indra-fundacion-universia-apoyan-tecnologias-personas-discapacidad-auditiva-visual-tea

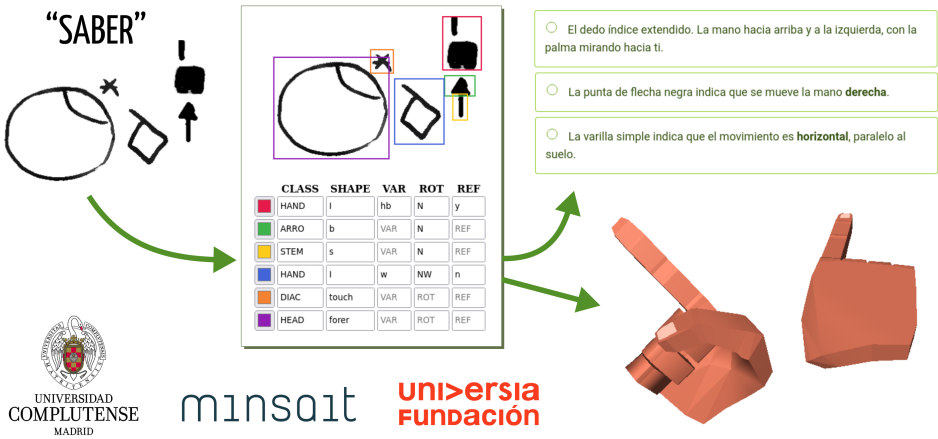


Fig. 3.3. – Esquema general del proyecto VisSE. A la izquierda, la fuente de datos: muestras de SignoEscritura. En el centro, la anotación computacional de los grafemas y su significado. A la derecha, las aplicaciones del procesamiento automático: descripción textual del logograma y modelo tridimensional de la mano.

y que se corresponden con los sub-objetivos técnicos que planteamos en la introducción.

El primer esfuerzo se enfoca en la ciencia de datos, y es crucial para disponer de un corpus anotado sobre el cual trabajar. La dimensión de este corpus debe ser adecuada para obtener resultados significativos, y su anotación debe realizarse con un nivel de detalle que permita su posterior análisis. Este esfuerzo incluye la recogida de datos, el diseño del esquema de anotación, y la estructuración del corpus resultante, como se detalla en la sección 3.3. Dicho corpus ha sido publicado en línea (Sevilla, Lahoz-Bengoechea y Díaz 2022), y el esquema de anotación se incluye en el capítulo 7. Adicionalmente, se ha redactado un artículo para la revista *Language Resources and Evaluation*, que se encuentra en el capítulo 6 y que ha sido aceptado para publicación con cambios menores.

El segundo esfuerzo se orienta hacia el procesamiento y la algoritmia, aplicando los datos anotados del corpus para entrenar y evaluar algoritmos de Inteligencia Artificial en la tarea de reconocer la SignoEscritura. Se utiliza la técnica de las redes neuronales profundas (*Deep learning*). También se integra el conocimiento experto sobre la SignoEscritura y las observaciones obtenidas del primer esfuerzo, permitiendo aumentar los algoritmos de aprendizaje con

reglas lógicas y superando algunos de los obstáculos que presenta un enfoque directo o naïve. El proceso está descrito en la sección 3.4, y los detalles pueden encontrarse en el artículo "Automatic SignWriting Recognition" (Sevilla, Díaz y Lahoz-Bengoechea 2023), publicado en la revista *IEEE Access*, y en el capítulo 8 de este documento.

El tercer esfuerzo en esta investigación se centra en el desarrollo de software. Mediante la aplicación de técnicas de ingeniería de software, las investigaciones resultantes de los otros esfuerzos han quedado codificadas de una manera reproducible y formal. Este proceso ha involucrado la incorporación de las mejores prácticas en el desarrollo de software, así como el uso de herramientas modernas y de código abierto. El trabajo realizado también ha quedado publicado libremente en abierto.

Dentro de este esfuerzo, descrito en la sección 3.5, se encuentra Quevedo, una librería que encapsula los algoritmos, la gestión de corpus y la inteligencia artificial aplicada en este proyecto. Quevedo está publicado como software libre en GitHub[2], y en la segunda parte de esta tesis se incluye la publicación académica que lo describe (capítulo 9) así como su documentación técnica (capítulo 10).

Pero, además de esto, he desarrollado software dirigido al usuario final, asegurando que los resultados de la investigación y de la tesis estén disponibles no sólo para investigadores y científicos, sino también para el público en general. Esta aplicación permite reconocer instancias nuevas de SignoEscritura, identificar sus elementos constituyentes y extraer su significado. Incorpora, además, un modelo interactivo 3D de la mano que facilita la comprensión de los grafemas manuales. Se presenta en forma de una aplicación web, publicada también como código abierto en Github[3], y una demostración de su funcionamiento está disponible en línea[4]. El modelo 3D fue creado también por mí como parte de esta investigación, constituyendo otro resultado enmarcado dentro de los objetivos tanto de análisis teórico como de diseminación de los resultados en forma de software y artefactos computacionales.

Como es habitual, aunque hay una cierta linealidad en las dependencias de los distintos esfuerzos, no se desarrollaron secuencialmente, sino en paralelo,

[2] https://github.com/agarsev/quevedo
[3] https://github.com/agarsev/visse-app
[4] https://garciasevilla.com/visse

adaptándose a las necesidades y circunstancias del proyecto. Avances en un área posibilitaban o requerían trabajo en las demás, o incluso proporcionaban nuevo conocimiento que llevaba a replantear decisiones ya tomadas. Aun así, plantearemos los tres esfuerzos por separado y uno detrás de otro en las siguientes secciones para facilitar una comprensión clara y detallada.

3.3. Diseño del corpus y esquema de anotación

El primer esfuerzo del proyecto, enmarcado en la "ciencia de datos", consistió en la recolección, tratamiento y anotación de un corpus de datos sobre el cual basar el procesamiento posterior. Teníamos disponible una gran cantidad de vocabulario de la Lengua de Signos Española (LSE), adquirido durante las clases en el Centro de Idiomas Complutense, y registrado en forma de SignoEscritura[5] escrita a mano sobre papel. Para convertirlo en "datos" utilizables, sin embargo, era necesario un proceso previo. Las distintas hojas de papel debían ser escaneadas, procesadas para mejorar el contraste y eliminar artefactos y ruido, y los distintos logogramas debían ser extraídos como muestras individuales. Afortunadamente, gracias al proyecto, pudimos ofrecer una gratificación a un colaborador externo para que realizara la mayoría de la parte mecánica del proceso. También recopilamos una serie de grafemas de manos, dibujados por el equipo investigador del proyecto en varias modalidades. Estos grafemas de manos finalmente no fueron utilizados para el proyecto, pero están disponibles en el corpus para aumentar la diversidad y exhaustividad de los datos.

Una vez recogidas todas las muestras "crudas", las imágenes escaneadas de los logogramas, fue necesario anotarlas para capturar su estructura y significado de manera computacional. La anotación de datos es un proceso tan importante y delicado como su correcta digitalización, ya que los modelos computacionales solo podrán trabajar con el conocimiento que haya sido adecuadamente capturado. De nada le sirve a un ordenador que los humanos sepamos cosas sobre un dominio de conocimiento, si esa información no está representada de manera lógica y formal. A menudo los humanos no somos conscientes de la cantidad de conocimiento que asumimos de manera

[5] Ver sección 2.3.2.

implícita, incluso en áreas del conocimiento tan estrictas como la ciencia o la ingeniería.

En el pasado he trabajado en el dominio médico, y sorprende tanto a los informáticos como a los propios médicos la cantidad de hipótesis, suposiciones y "sentido común" que no son explícitos y, sin embargo, subyacen al proceso formal. Es además fundamental que los datos estén anotados de forma precisa y correcta, ya que los errores se van acumulando con cada paso de procesado o capa de abstracción, hasta llegar en ocasiones a modelos teóricos completamente incorrectos por basarse en datos pobremente anotados (Sambasivan et al. 2021).

En el caso del PLN o la LC, el problema del conocimiento implícito se multiplica. Cuando decimos "gato", a nosotros ya nos vale como representación última de la información; o si somos biólogos, quizá preferimos decir *Felis domesticus*, y añadir algunas precisiones más. Como humanos, compartimos una realidad en la que existen los gatos, por lo que compartimos ese concepto en nuestra mente, y con crear un signo (la palabra) para el significado, hemos terminado. El ordenador, al ser una máquina puramente lógica y no tener el contexto o la experiencia humana, necesita que todo el conocimiento que se quiera utilizar esté explícitamente codificado.

Este fenómeno se puede observar también con la SignoEscritura, ya que a pesar de estar formalmente definida en términos humanos, el formalismo no es suficientemente detallado para su procesamiento automático. La SignoEscritura asume, por ejemplo, que el humano será capaz de trasladar el espacio bidimensional al espacio 3D de manera intuitiva, o entender qué manos son derechas y cuáles son izquierdas. Era por tanto necesario desarrollar un modelo de la SignoEscritura computacionalmente adecuado y que además permitiera el procesamiento que teníamos en mente. Afortunadamente, llevábamos ya varios años investigando sobre la fonética y fonología de la LSE, y pudimos aprovechar nuestro conocimiento e hipótesis sobre ellas para desarrollar el modelo de representación para la SignoEscritura. Además, gracias a mi investigación inicial sobre el reconocimiento automático de lengua de signos, también tenía claro los requisitos de la anotación para que fueran aprovechables por los algoritmos de visión artificial. El resultado fue un modelo de anotación exhaustivo y estructurado, que sirve para capturar tanto las propiedades físicas de la SignoEscritura como el significado de los distintos

grafemas y sus propiedades. Este esquema de anotación es clave para el éxito del resto del proyecto, y representa bien la convergencia de análisis teórico y conocimiento experto con técnicas informáticas y de aprendizaje automático que es la metodología resultado de esta tesis.

La primera clave de esta representación radica en la definición de los términos logograma y grafema, asignando las propiedades léxicas y morfológicas a los grafemas, y concebiendo el logograma como una composición sintáctica de los mismos. Para los grafemas, se definió un conjunto de cinco etiquetas. La primera, denominada "clase", representa una clasificación de grano grueso que tiene por objetivo dividir a los grafemas en grupos con características tanto gráficas como semánticas afines. Esta clasificación es fundamental para el procesamiento posterior y, además, está justificada lingüísticamente, pudiendo ser interpretada como un análogo de la parte de la oración ("Part of speech", POS). La segunda etiqueta, denominada "forma" (shape), determina el grafema propiamente dicho tal y como sería comprendido por los seres humanos. Esto incluye cada una de las manos, cada uno de los símbolos de cabeza, las partes de los movimientos, etc. Estas formas están implícitas en la SignoEscritura y en los materiales didácticos, pero no están clasificadas de una manera sistemática ni con una nomenclatura significativa. En la representación digital existente de la SignoEscritura, denominada "Formal SignWriting" (FSW, Slevinski 2016), los grafemas se identifican mediante un número, su código de carácter Unicode, pero estos números son poco prácticos, ya que es difícil para un humano memorizarlos todos, y además, no proporcionan información sobre el significado (fonético) del grafema que representan.

Por tanto, fue necesario desarrollar una nomenclatura sistemática para la forma de los grafemas. De particular interés es la nomenclatura utilizada para las manos. Existen más de 60 configuraciones de la mano en la LSE, que no corresponden a letras ni palabras del español, por lo que su representación no es trivial. Para clasificarlas, recurrimos a nuestro sistema, la Signotación (Lahoz-Bengoechea y Sevilla 2022b), mencionada anteriormente y que codifica las distintas configuraciones con caracteres ASCII. Esta notación se basa en nuestra investigación de la fonología de la LSE, y su uso en el corpus demuestra su utilidad tanto para una descripción lingüísticamente correcta y exhaustiva de la LSE como para su procesamiento automático.

Además de la clase y la forma, que conjuntamente conforman la clasificación de los grafemas, estos tienen propiedades gráficas que también requieren anotación. Los grafemas pueden aparecer rotados, reflejados, y presentar distintos colores y rellenos. Aunque los seres humanos, gracias a nuestra percepción visual, somos capaces de reconocer estas transformaciones y entender que la forma del grafema es la misma, en la imagen tal como es interpretada por el ordenador, son completamente distintas. En la FSW y las fuentes tipográficas digitales de SignoEscritura, se asignan números distintos a cada una de estas variaciones, incrementando así la cantidad total de símbolos. Aunque estos números están asignados de manera consecutiva y con cierta lógica, terminan siendo meramente un índice, que no ofrece información sobre las propiedades gráficas (la transformación) del grafema.

Como dato curioso, en la versión oficial de Unicode para SignoEscritura (The Unicode Consortium 2021, pp. 831–832), en lugar de esta explosión combinatoria, se designan secuencias de caracteres o clústeres de grafemas, en los que el primer carácter representa la forma, el segundo una variación tipográfica, el tercero la rotación, etc. Sin embargo, esta representación, que es análoga a la nuestra (desarrollada de forma independiente) y más o menos compatible, no parece estar en uso en FSW ni en ninguna otra herramienta oficial de SignoEscritura, probablemente debido a la complejidad de su uso sin herramientas especializadas.

En nuestra anotación sistemática, la información gráfica tal como la variación (color, relleno), la rotación y la reflexión (espejo) de los grafemas constituyen el resto de etiquetas que completan la información atómica proporcionada por cada grafema. Sin embargo, aún falta la información sintáctica, es decir, la manera en la que los grafemas se combinan dentro del logograma. A diferencia del lenguaje oral, que es lineal y unidimensional y donde las palabras ocupan una posición ordenada en la oración, en la SignoEscritura los grafemas se disponen de manera dispersa en un plano bidimensional. Resulta necesario, por tanto, anotar la ubicación de los grafemas, para lo cual almacenamos el centro de cada uno de ellos, con coordenadas (x, y). Para evitar distorsiones artificales por el tamaño de cada logograma, en lugar de la posición en píxeles empleamos una fracción que varía de 0 a 1, indicando la posición del grafema relativa al "lienzo" del logograma.

Sin embargo, grabar únicamente la posición con dos coordenadas no es suficiente, ya que no se puede ignorar el tamaño de los grafemas. Aunque no sea explícitamente significativo, y pueda haber variación en la escritura de distintas personas, es indispensable que el ordenador conozca este aspecto para realizar el reconocimiento automático, ya que es necesario delimitar físicamente el área de cada grafema. Además, el tamaño es sintácticamente relevante, puesto que muchas construcciones en SignoEscritura se basan en la cercanía de los grafemas. Esta cercanía, sin embargo, no es la distancia entre sus centros, calculable con las posiciones puntuales, sino la distancia entre sus bordes, y para calcularla se necesita conocer extensión gráfica de los grafemas. Por lo tanto, almacenamos otro par de coordenadas, el ancho y alto del grafema (de nuevo relativos al tamaño del logograma). Es decir, en total, para cada grafema anotamos su la información "léxica" (clase y forma) y "morfológica" (relleno, rotación, reflexión), pero también su posición y tamaño como atributos numéricos adicionales.

Para una exposición completa sobre la anotación, incluyendo los distintos valores de las etiquetas, se recomienda consultar la guía de anotación del corpus, disponible en inglés en el capítulo 7 y en español junto a los datos publicados en línea[6]. Se puede ver un ejemplo también en la figura 3.4. Este esquema de anotación del corpus VisSE es una de las innovaciones de esta tesis, ya que captura una información para cada muestra más sofisticada que la habitual en el campo de la visión artificial, y está además lingüísticamente motivada, habilitando el uso efectivo de los datos para las aplicaciones posteriores.

Es preciso señalar que, además de definir el esquema de anotación, también es necesario realizar la anotación en sí, es decir, el proceso manual de identificar cada grafema presente en un logograma y asignar los valores para sus respectivas etiquetas. A pesar de ser una labor ardua, meticulosa y mecánica, requiere un conocimiento profundo de la anotación y del dominio, por lo que su externalización no es tarea sencilla. El proceso de anotación tampoco es directo, ya que, a medida que se anotan más datos y se realizan experimentos, se identifican problemas y mejoras en el esquema de anotación, lo que exige revisar las anotaciones previamente efectuadas. Este

[6] https://zenodo.org/record/6337885

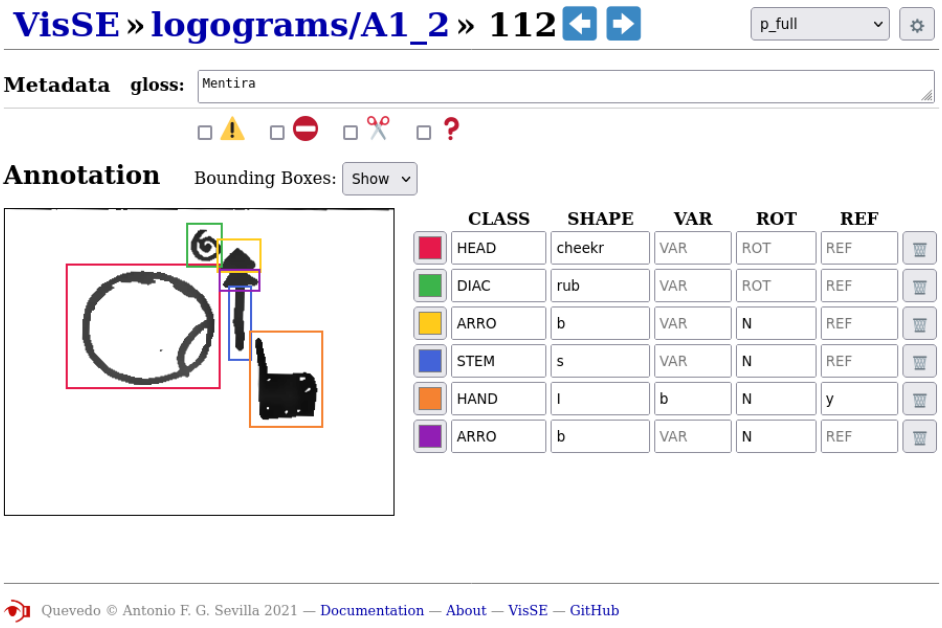


Fig. 3.4. – Anotación visual del signo “mentira/mentir”. Las cajas delimitadoras de los grafemas se dibujan superpuestas al logograma, y las etiquetas de los grafemas se ven en la tabla a la derecha. Los colores sirven para identificar correctamente los grafemas en la tabla con las marcas visuales. Original: figura 6.2.

proceso consume una gran cantidad de tiempo, y resultó en la anotación de 982 logogramas. Esta cifra es significativa y fue útil para el resto del proyecto, pero representa menos del 40 % de las muestras recogidas.

Un ejemplo de este proceso de revisión se puede ver con la aparición de los signos “bimorfémicos”. En el vocabulario original recogido, la mayoría de los signos estaban transcritos con un solo logograma. Sin embargo, algunos signos presentaban dos “partes”, comúnmente denominadas sílabas en la literatura, aunque en nuestro estudio las consideramos más próximas a los morfemas. En cualquier caso, cada una de estas partes constituye un logograma independiente, situación que generaba numerosos problemas durante el reconocimiento automático. Optamos por dividirlos en dos, pero ello implicaba repetir la anotación para muchos signos. Afortunadamente, la herramienta informática que habíamos desarrollado para crear y gestionar el

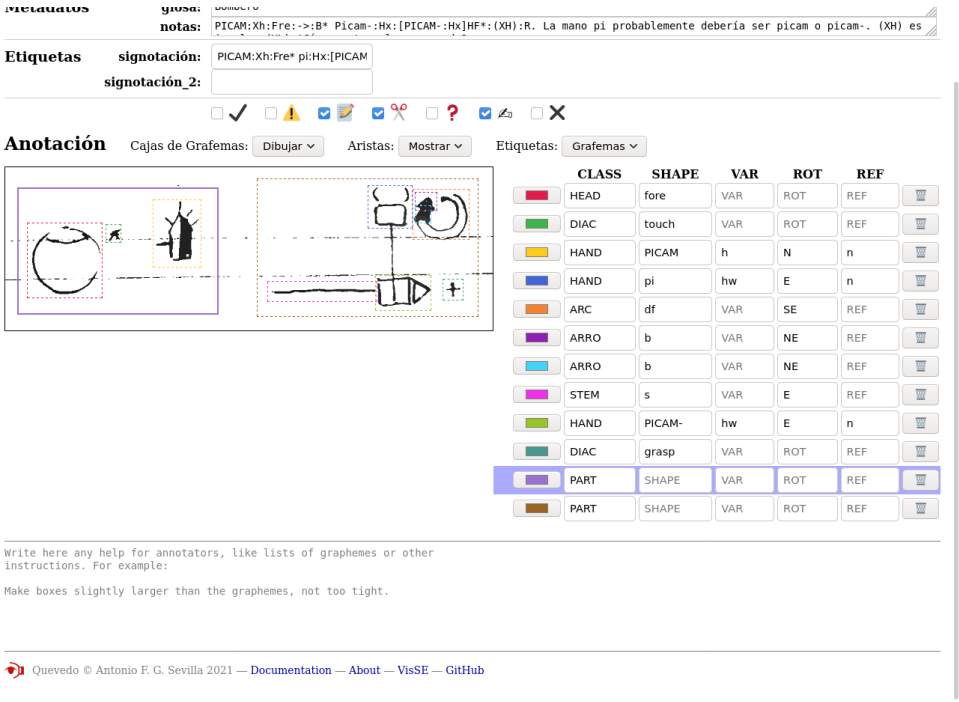


Fig. 3.5. – Proceso de corte de transcripciones dobles, en este caso el signo "Bombero". Algunos signos tienen dos (o más) partes, lo que supone un problema para los algoritmos de procesamiento. Cuando tomamos conciencia de ello, ya habían sido anotados, pero utilizando Quevedo, pudimos emplear la anotación visual existente para demarcar los dos logogramas contenidos. Una vez demarcados manualmente, un "script" de usuario (sección 10.8.2) permitió extraer las distintas partes en ficheros independientes, manteniendo y corrigiendo las posiciones y el etiquetado de los grafemas de manera automática.

corpus (Quevedo) nos facilitó llevar a cabo esta tarea de manera más o menos automatizada (ver figura 3.5).

El uso de Quevedo como infraestructura, codificando de manera formal y automática todos los procesos, fue una ayuda constante en la anotación, ya fuera realizando chequeos o correcciones deterministas que podían ser ejecutadas por la máquina, o efectuando una pre-anotación automática que posteriormente era revisada y corregida por el humano. Este proceso, conocido como *bootstrapping*, es una técnica habitual en el estado del arte, y se basa en el uso de resultados preliminares de los algoritmos de aprendizaje automático para agilizar el proceso de anotación. Primero, los humanos ano-

tan manualmente y con detalle un subconjunto de datos. Este subconjunto se emplea luego para entrenar los algoritmos, que, aunque no alcanzan una gran precisión, ofrecen ya una anotación preliminar que puede ser corregida por el humano. Con estos nuevos datos anotados y corregidos, se vuelve a entrenar el algoritmo, resultando ahora en una mejor aproximación, por lo que la corrección manual se agiliza. Esto acelera la anotación, al tiempo que permite comprobar que los algoritmos están funcionando correctamente, y realizar las mejoras y adaptaciones que se observen necesarias sobre la marcha.

En la siguiente sección, abordaremos estos algoritmos en detalle, pero la herramienta Quevedo se describe con más profundidad en la sección 3.5.1 y los capítulos 9 y 10. Como ya se ha mencionado, los datos anotados han sido publicados en línea en Zenodo (Sevilla, Lahoz-Bengoechea y Díaz 2022), junto con la guía de anotación en español e inglés y reproducida en el capítulo 7. El artículo "Building the VisSE corpus of Spanish SignWriting", reproducido en el capítulo 6 y aceptado para publicación en la revista *Language Resources and Evaluation*, amplía lo expuesto en esta sección y ofrece algunas estadísticas preliminares sobre el corpus.

Al estar Quevedo disponible libremente en el repositorio de paquetes de Python (PyPI), cualquier usuario puede examinar visualmente el corpus con los comandos de terminal enunciados en el listado de código 3.1.

List. 3.1. – Cómo explorar el corpus VisSE con Quevedo. Original: listado 8.1.

```
$ wget https://zenodo.org/record/6337885/
  files/visse-corpus-2.0.0.tgz?download=1
$ tar xzf visse-corpus-2.0.0.tgz
$ cd visse-corpus
$ pip install quevedo[web]
$ quevedo web
```

3.4. Procesamiento computacional de SignoEscritura

Como se mencionaba anteriormente, los seres humanos a menudo trabajamos con conceptos e hipótesis que, tras un examen minucioso, no se encuentran definidos de manera exhaustiva ni formal. Esto resulta especialmente evidente al abordar el trabajo con imágenes. Si se inquiere a un especialista en radiología sobre la probabilidad de que una masa en la radiografía sea un tumor, o a un paleontólogo sobre por qué un determinado diente pertenece a un tiburón y no a un ratón, será complicado obtener una descripción matemática precisa, como la que requieren las máquinas. Los expertos pueden ofrecer una multiplicidad de razones y argumentos, pero la formalización mediante una lógica susceptible de ser computarizada resulta un desafío extremo, en ocasiones casi imposible. Las excepciones y el "sentido común" se acumulan, y conceptos aparentemente sencillos como "circular", "proporción" o "textura" no se prestan fácilmente a una definición matemática inmediata.

En el ámbito de la imagen, uno de los obstáculos principales radica en su representación digital. Como en una cámara, o en la retina humana, podemos entender el medio visual como un campo plano y bidimensional de luz. Aunque biológica y neuronalmente nuestra visión es más compleja, esta es una aproximación suficiente en la que se fundamentan pantallas, impresoras y los formatos de imagen convencionales en el ordenador. La versión discreta de este campo físico de intensidades luminosas es una matriz bidimensional de puntos (píxeles), a los que asignamos una serie de valores para representar la luz correspondiente a esa posición en la imagen. No vamos a entrar aquí en teoría del color, pero debido a cómo lo percibimos los humanos, con tres tipos de células receptoras que responden a distintas longitudes de onda, la manera más habitual de representar la luz en una imagen es con tres intensidades: "rojo", "verde" y "azul", para cada uno de los píxeles (RGB).

Esta representación ilustra la brecha significativa que hay entre la *imagen* y la *visión*. Si los datos que poseemos son matrices bidimensionales de números, definir en base a esta representación lo que constituye un círculo, una línea, un polígono, más aún un diente, una letra o, en nuestro caso, un grafema de la SignoEscritura, no resulta trivial. Incluso el caso más sencillo, un segmento de recta, se torna en algo de complejidad asombrosa. Determinar qué píxeles

corresponden a otros píxeles, dónde termina el segmento y comienza el fondo, cuál es el ángulo, en qué posición de la imagen se encuentra, todo ello requeriría ser codificado manualmente con reglas y una gran cantidad de excepciones. Además, dada la naturaleza imperfecta de la realidad, habría que incorporar modelos probabilísticos de cálculo complejo y definición desafiante.

Esta problemática forma parte de los desafíos de la inteligencia artificial clásica, pero, afortunadamente, es precisamente una de las cuestiones que resuelve la revolución del aprendizaje automático y, en particular, su versión más avanzada: el aprendizaje profundo o *deep learning*.

Este paradigma de la IA se fundamenta en inferir el modelo computacional directamente desde los datos, extrayendo patrones heurísticos e "intuiciones" que son difíciles de explicar, mediante el examen repetido de datos adecuadamente preparados. La preparación de datos o anotación, en nuestro caso descrita como parte del primer esfuerzo de ciencia de datos, es esencial, permitiendo que los algoritmos descubran las realidades subyacientes a los datos y las interpretaciones que se buscan.

El aprendizaje automático se fundamenta, desde una perspectiva algorítmica, en un principio esencial: la minimización del error. Este enfoque se basa en la disposición de una serie de datos y una predicción preliminar, corrigiéndose progresivamente el método de inferencia en función del error medido en las predicciones efectuadas. En muchos casos, se permite al algoritmo explorar de manera aleatoria durante un período determinado, e incluso avanzar en una "dirección incorrecta", de modo que se puedan explorar distintas vías y soluciones hasta alcanzar una óptima. Esta solución óptima no suele ser única, y de hecho, estos algoritmos frecuentemente no son deterministas, llegando a una conclusión diferente cada vez que se ejecutan. La tarea del ingeniero consiste en manipular los distintos parámetros del algoritmo, las condiciones de contexto, y en ocasiones los propios datos y su pre-procesamiento, para lograr que el algoritmo encuentre con suficiente fiabilidad una solución óptima en términos de error de predicción.

La solución identificada se convierte en sí en un modelo del problema, una función que, al recibir datos conocidos o nuevos, ofrece una predicción sobre dichos datos. Esta predicción puede constituir información sobre el futuro, como el siguiente evento en una serie temporal o la previsión meteorológica

para el día siguiente. También puede tratarse de información de "alto nivel", que no está explícita en los datos brutos, como determinar si la matriz de píxeles se corresponde a un diente de tiburón, murciélago o si representa un perro. La predicción puede incluso consistir en reconstruir parte de la información original, que ha sido ocultada por los ingenieros a la red. En este modelo de aprendizaje "no supervisado", no es necesario definir una tarea o etiquetar los datos, y se basa en entrenar a la red para que reconstruya diferentes partes de la entrada. Con suficientes datos de entrenamiento, este procedimiento permite al algoritmo aprender de forma íntima la estructura y apariencia de los datos, y es el procedimiento en el que se basan redes de última generación como las que crean los *Deep Fakes* o generan texto en lenguaje natural, como *GPT*.

La clave para todos estos avances, el *Deep Learning*, reside precisamente en lo que sugiere su nombre: aprendizaje profundo. Pero esta profundidad no es metafórica o conceptual, sino literal: las redes neuronales utilizadas en el *Deep Learning* constan de múltiples capas de procesamiento, donde cada capa recibe como entrada la salida de las capas anteriores. Esto les permite organizarse en niveles de abstracción: la primera capa simplemente detecta grupos de píxeles del mismo color, por ejemplo; la siguiente capa detecta segmentos de líneas rectas o arcos de curva; la capa subsiguiente compone estos pequeños patrones en formas más grandes, y así sucesivamente hasta llegar a una capa de clasificación o salida, que decide si la imagen es un perro o una bicicleta. Este funcionamiento es análogo a cómo opera el córtex visual humano; sin embargo, de la misma manera que no comprendemos en detalle cómo funciona este último, tampoco podemos afirmar con certeza qué hace cada capa o neurona de una red, limitándonos a observar patrones generales de funcionamiento. A pesar de que todo pueda parecer muy impreciso y poco determinista, y sea difícil creer incluso que pueda funcionar, el éxito de estos algoritmos es evidente y palpable incluso en nuestro entorno cotidiano.

3.4.1. El algoritmo YOLO de detección

Para la tarea concreta de reconocimiento de SignoEscritura, necesitamos algoritmos específicos en el dominio de la Visión Artificial. El primer algoritmo requerido es conocido como "You Only Look Once" (YOLO, Redmon,

Divvala et al. 2016). Este algoritmo es capaz de localizar los diferentes elementos constituyentes en una imagen (una tarea identificada como "detección"), además de asignarles una etiqueta ("clasificación") dentro de las categorías previamente entrenadas. A tal fin, fracciona la imagen en pequeñas áreas y, para cada una, calcula la probabilidad de que dicha área represente un borde, esquina o zona interior de un objeto de interés, identificando también la clase del objeto. Posteriormente, estas probabilidades son ensambladas en regiones cuadrangulares, conocidas como "bounding boxes" (cajas delimitadoras), que constituyen el resultado del algoritmo: una lista de áreas en la imagen donde se encuentran los objetos identificados, junto con su etiqueta correspondiente.

Este algoritmo, en principio, resultaría adecuado para resolver la tarea propuesta: permite detectar los diferentes grafemas que forman un logograma, identificando cada uno de ellos y, en consecuencia, discerniendo su significado. Así comenzaron los estudiantes del trabajo de fin de grado aludido (Sánchez Jiménez, López Prieto y Garrido Montoya 2019), pero pronto enfrentaron la dificultad de no contar con suficientes datos. Una debilidad inherente de los algoritmos de aprendizaje profundo radica en que, al operar de manera tan indiscriminada, requieren una considerable cantidad de datos para el aprendizaje efectivo. Si los datos y la capacidad de cómputo son limitados, no será viable alcanzar una solución con un error lo suficientemente reducido para ser útil. Por el contrario, incrementar la capacidad de cómputo sin aumentar los datos conduce al "overfitting" (sobreajuste), donde el algoritmo aprende a resolver exactamente la tarea con los datos presentados, sin ser capaz de generalizar a nuevos datos.

Además, es fundamental que los datos abarquen una proporción significativa, cuanto mayor mejor, del dominio de estudio, ya que las redes neurales tienen problemas para trabajar con datos fuera del dominio aprendido: interpolan mejor que extrapolan. Para que la red sea capaz de distinguir características comunes como rasgos independientes, estos deben manifestarse con suficiente frecuencia en los datos de entrenamiento. Por ejemplo, si solo existe una muestra de una mano en configuración "X" y con la palma hacia abajo, la red no distinguirá dos rasgos separados, configuración "X" y "palma hacia abajo", sino que los interpretará como características indivisibles.

La gran cantidad de rasgos simultáneos existentes la fonología signada, reproducidos en la SignoEscritura, implica que este fenómeno ocurra con frecuencia en nuestro corpus de datos. Si se aplica una aproximación naïve, utilizando una única clase para cada grafema completamente distinto, se encontrarán combinaciones que solo aparecen una vez en el corpus, o incluso ninguna, obstaculizando el aprendizaje adecuado del algoritmo. Matemáticamente, podríamos decir, nos enfrentamos a un espacio altamente dimensional, con una (hiper) superficie por identificar pero de la que conocemos un conjunto de puntos notablemente poco denso.

3.4.2. Procesamiento aumentado con conocimiento experto

Para abordar este problema, es necesario desrarificar (hacer más denso) el espacio de búsqueda del problema, y para ello podemos utilizar las múltiples etiquetas codificadas en el corpus a las que hemos hecho referencia en la sección previa. En un procedimiento reconocido comúnmente en ingeniería como "divide y vencerás", separamos el proceso de reconocimiento en varias etapas consecutivas, cada una de ellas más sencilla de resolver que la tarea en su totalidad.

En primer lugar, solicitamos al detector (YOLO) que identifique los grafemas, aunque no se requiere que los clasifique completamente, sino únicamente que determine la clase de grano grueso. Esto implica que el algoritmo puede generalizar y aprender que, por ejemplo, los grafemas para la cabeza suelen tener un tamaño similar y más grande que los de las manos, etc. Una vez identificados los grafemas, podemos utilizar una red especializada en clasificación, en nuestro caso "Alexnet" (Krizhevsky, Sutskever e Hinton 2017), que no necesita aprender a localizar los diferentes grafemas dentro del logograma. De hecho, es posible utilizar distintas redes para las manos, las cabezas, etc., permitiendo que estas redes aprendan los detalles más finos de esas clases sin tener que distinguirlas de las demás.

Para las manos, no obstante, esta idea no es suficiente. Existen más de medio centenar de configuraciones posibles de la mano, representadas cada una por formas de grafema diferentes. Además, como hemos visto anteriormente, cada una de ellas puede aparecer rotada o con distinto relleno de color, multiplicando de manera exponencial los grafemas a reconocer.

Por tanto, debemos codificar de alguna manera el conocimiento que tenemos sobre la rotación de los grafemas en nuestro algoritmo. Es fundamental aclarar que esto es algo que las redes pueden aprender por sí mismas, pero requiere una mayor cantidad de datos de la que disponemos. Experimentos preliminares para aumentar artificialmente la cantidad de datos ("data augmentation") no resultaron exitosos, por lo que se necesitó otra aproximación.

Afortunadamente, ni la rotación ni la simetría son procesos difusos, como la definición de una línea, un diente o un perro. Son, en cambio, procesos deterministas, matemáticamente bien definidos y que podemos calcular en el ordenador. Si utilizamos esta lógica de manera externa al algoritmo, éste no tiene por qué aprenderla, y se ve liberado para aprender otros detalles. Podemos ver esta técnica como una de densificación de datos o compresión de características, ya que al eliminar los grados de libertad que crean las rotaciones reducimos la dimensionalidad del espacio del problema.

En una primera aproximación, se intentó rotar los grafemas de mano detectados, generando todas las posibilidades posibles y clasificando cada una de ellas. La elección se basaría en la rotación que la red clasificase con mayor confianza, descartando las demás. No obstante, este enfoque resultó problemático debido a la elevada cantidad de falsos positivos, en los que la red seleccionaba la rotación incorrecta o varias de ellas con una alta confianza.

La solución encontrada finalmente consistió en dividir de nuevo el problema para conquistarlo. Tras detectar las manos en el logograma, se entrenó una red clasificadora con el propósito exclusivo de determinar la transformación (rotación y simetría) del grafema. Una vez identificada, esta transformación puede ser revertida fácilmente mediante un proceso determinista, de manera que se obtiene un grafema siempre en orientación "erguida". Así, es posible entrenar redes clasificadoras para decidir las etiquetas subsiguientes (forma y variación) que no tienen que tener en cuenta la posible transformación geométrica del grafema.

Todo el proceso fue programado en un "pipeline", representado en la figura 3.6. Este "pipeline" de decisiones, el entrenamiento de las diversas redes neuronales y, en general, todo el desarrollo programático, forman parte nuevamente de Quevedo, que se describe con mayor detalle en la sección 3.5.1 y en los capítulos 9 y 10.

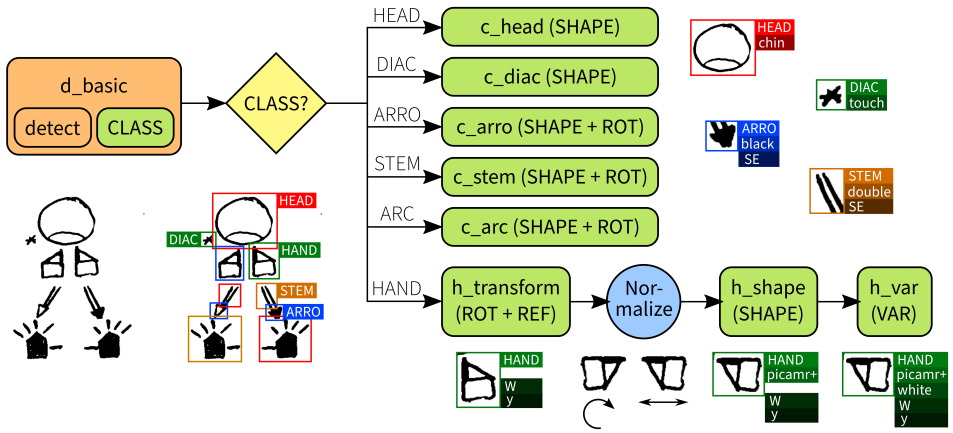


Fig. 3.6. – Arquitectura completa del pipeline de procesamiento. Incluye las distintas redes neuronales, decisiones y procesos dinámicos. Se muestra como ejemplo un logograma, correspondiente al signo "Historia", así como los grafemas detectados y su progreso por el pipeline. Original: figura 8.7.

Para evaluar los resultados de nuestra algoritmia, recurrimos al "acierto" (accuracy), una métrica de evaluación equilibrada y que tiene la ventaja añadida de permitir una comparación directa entre nuestra solución y el enfoque base directo. Se formula como la proporción directa de predicciones correctas en el total de predicciones realizadas y potenciales. El aspecto clave a evaluar son los grafemas pronosticados: cuántos de ellos pueden ser encontrados y qué tan precisas son sus características pronosticadas. Medimos este acierto con tres cálculos: el acierto en la detección de los grafemas, el acierto en su clasificación (es decir, la predicción de sus etiquetas) y una medida combinada que puntúa cada solución para la tarea integral de reconocer la SignoEscritura.

Tras evaluar el acierto de los distintos algoritmos, concluimos que nuestra aproximación experta, basada en el pipeline adaptativo, mejora el algoritmo YOLO directo en un 17 %, alcanzando un acierto global del 68 %. En concreto, este enfoque mejora sustancialmente la detección de grafemas. Al detectar mayor número de ellos, incrementa la detección de instancias que son "difíciles de predecir", lo cual provoca una leve disminución en el acierto de clasificación. No obstante, globalmente se observa la mejora cuantitativa del 17 % antes mencionada. Adicionalmente, desde un análisis cualitativo, las

nuevas predicciones resultan mucho más aplicables. Detectar la mayoría de los grafemas presentes, incluso si algunas de sus etiquetas son incorrectas, incrementa significativamente la utilidad y aplicabilidad práctica de los resultados.

El objetivo principal de nuestro desarrollo no consistía en la evaluación de las capacidades inherentes a las redes neuronales o los algoritmos individuales, sino en explorar la viabilidad de aplicar nuestro conocimiento experto, adquirido a través del análisis teórico, para facilitar un procesamiento más eficiente en un dominios como la SignoEscritura o la LS, caracterizados por la escasez de datos. Los resultados obtenidos, que muestran una mejora significativa en la tasa de acierto, corroboran la validez de nuestra enfoque metodológico. Este hallazgo sugiere que, incluso en ausencia de grandes volúmenes de datos —tan comunes en los enfoques computacionales contemporáneos—, la selección meticulosa de los datos y la aplicación de conocimientos teóricos específicos al dominio en cuestión pueden no sólo mejorar, sino incluso habilitar, el procesamiento computacional eficaz.

Para obtener información más detallada sobre la evaluación, incluida la formulación matemática y los datos numéricos, se puede consultar el capítulo 8, en el que se ofrece un análisis más detallado de la algoritmia empleada.

3.5. Software desarrollado

Como parte fundamental del desarrollo del proyecto VisSE, tanto para los esfuerzos descritos anteriormente de ciencia de datos como de algoritmia, se hizo imperativo el desarrollo de distintos tipos de software. Por un lado, se requirió software especializado para la administración de datos y la gestión de experimentos, así como para la implementación de los algoritmos de análisis y procesamiento. Por otro lado, se elaboró software de aplicación con el objetivo de brindar visibilidad y aplicabilidad práctica a los resultados obtenidos.

Retomando el objetivo complementario de esta tesis, consistente en facilitar la utilización del desarrollo realizado por terceros, opté por diseñar el software siguiendo arquitecturas estándar, e intentando abstraer los detalles concretos del dominio y corpus de datos propio. A fin de promover una mayor

comprensión y aplicabilidad, se ha documentado tanto el funcionamiento como la interfaz del software, que ha sido publicado de forma abierta en la plataforma Github. Este enfoque cobra particular importancia en el caso del software de infraestructura; en lugar de recurrir a una colección de scripts ad-hoc, decidí compilar toda la funcionalidad en una librería y solución software unificada, a la que se ha denominado Quevedo.

3.5.1. Quevedo

Como hemos visto, la creación de un corpus de datos robusto constituyó uno de los esfuerzos primordiales en el desarrollo de VisSE. En este contexto, Quevedo (ver figura 3.7) emergió como un elemento crucial, habilitando una gestión eficaz del corpus mediante herramientas específicas para organizar muestras de SignoEscritura de forma estructurada.

Quevedo abarca diversas funcionalidades, que van desde la gestión de datos almacenados en disco, pasando por la administración de ficheros binarios y metadatos, hasta el procesamiento avanzado de datos. Este último incluye técnicas de "data augmentation" y el tratamiento de imágenes para la extracción de grafemas. Todos estos procesos son accesibles a través de una interfaz de línea de comandos, lo cual posibilita la incorporación de Quevedo en pipelines de datos administrados con herramientas como "Makefiles" o software especializado en ciencia de datos como DVC. Este diseño facilita la iteración y experimentación en el proceso de análisis de datos.

Un componente esencial de este corpus son sus anotaciones, las cuales, en gran medida, consisten en información visual. Quevedo brinda una interfaz web para acceder al conjunto de datos, permitiendo anotaciones intuitivas mediante acciones de clic y arrastre con el ratón. Esta interfaz no solo sirve como un medio para realizar anotaciones, sino también como una herramienta invaluable para la visualización y divulgación de los resultados del corpus. Al usar tecnología web, dicha interfaz puede compartirse con facilidad entre múltiples colaboradores, sin la necesidad de que descarguen el conjunto de datos ni instalen software adicional.

Cabe destacar que las capacidades de anotación en Quevedo son más sofisticadas que las ofrecidas por otras herramientas similares. Mientras que otras aplicaciones se limitan a permitir la asignación de etiquetas individuales a

Fig. 3.7. – Logo de Quevedo. Es una estilización de la SignoEscritura del signo propio del proyecto, ejecutado con los dedos índice y pulgar posicionados sobre el ojo. Este signo, cuyo significado es "gafas"[7], es altamente icónico, por lo que el logo también busca evocar dicho objeto.

objetos, Quevedo admite un esquema multi-etiqueta, como el que se requirió para VisSE. Además, se encuentra en desarrollo una extensión que permite añadir información sintáctica, como las relaciones entre grafemas, abriendo así nuevas vías para futuras investigaciones.

Quevedo no solo resultó útil para la gestión y análisis de datos, sino que también desempeñó un papel relevante en el ámbito de la inteligencia artificial. Quevedo permite la incorporación de parametrizaciones para redes neuronales profundas como parte del conjunto de datos. Dichas parametrizaciones, definidas por el usuario en el lenguaje de configuración TOML, abarcan desde el tipo de tarea (ya sea detección o clasificación de grafemas) hasta las etiquetas y subconjuntos de datos que se emplearán para entrenar y evaluar las redes.

Este diseño posibilita la realización de experimentos de manera reproducible y transparente, manteniendo el control sobre los datos y los hiperparámetros utilizados. Asimismo, gestiona eficientemente las distintas fases del proceso de entrenamiento y evaluación de las redes neuronales. Quevedo también permite que las redes sean incorporadas en un pipeline dinámico, junto con procesos deterministas, facilitando la implementación y la iteración de la solución diseñada en el esfuerzo de IA.

[7] ver `https://griffos.filol.ucm.es/signario/signo/11768`

En resumen, al encapsular las diversas fases de la investigación en el software Quevedo, se logró simplificar de manera significativa la exploración y experimentación iterativas. Este enfoque fortalece además la trazabilidad y reproducibilidad de los experimentos, ayudando a cumplir con nuestro objetivo complementario de difusión y utilidad para la comunidad científica.

A continuación haremos una breve descripción del diseño arquitectónico de la librería, pero para mayor detalle se puede consultar la documentación (capítulo 10) o directamente el código fuente en `https://github.com/agarsev/quevedo`.

Arquitectura de Quevedo

Quevedo está diseñado como una librería, programada en Python, que ofrece las distintas funcionalidades, así como una interfaz de comandos para acceder a las mismas. Utiliza la librería 'Click' para la creación de la interfaz de comandos, 'Pillow' para el procesamiento de imagen, y 'Flask' para la construcción de la interfaz web. La parte dinámica de dicha interfaz web se ha desarrollado en JavaScript, haciendo uso de 'preact'. Las redes neuronales se gestionan con el software 'Darknet', desarrollado por el propio inventor del algoritmo YOLO.

El elemento central de Quevedo es el objeto denominado "Dataset", que representa un conjunto de datos almacenados en el disco duro. Este objeto permite al usuario acceder y manipular anotaciones sin la necesidad de gestionar archivos de forma manual. El objeto "anotación" (*Annotation*), que puede constituir un grafema o un logograma, se obtiene a través de las funciones de búsqueda que ofrece el dataset. Este objeto otorga acceso a metadatos, anotaciones y datos de imagen, permitiendo su modificación si se considera necesario. Adicionalmente, el objeto Dataset ofrece acceso a las redes neuronales incluidas en el conjunto de datos, las cuales pueden ser empleadas tanto para inferencia como para entrenamiento.

Dado que Quevedo está diseñado como librería, sus funcionalidades pueden ser incorporadas en otros programas. Esto posibilita la construcción de software que se erige sobre los resultados obtenidos con Quevedo, facilitando así el desarrollo de aplicaciones de usuario como la que se describirá más adelante.

Otra característica importante de Quevedo es su naturaleza agnóstica de dominio. En su codificación, intenté separar lo máximo posible los detalles correspondientes a la SignoEscritura y las funcionalidades más genéricas y abstractas. Esto otorga a Quevedo una versatilidad que trasciende su aplicación original; es decir, la herramienta puede ser útil en otros dominios en los cuales sea necesario identificar y clasificar componentes con significado independiente pero sintácticamente relacionado dentro de imágenes con un significado composicional.

Si bien esta utilidad podría considerarse altamente especializada a primera vista, lo cierto es que hay una variedad de "lenguajes gráficos" que podrían beneficiarse de su arquitectura. Ejemplos de esto son la aritmética elemental, los diagramas UML (Lenguaje Unificado de Modelado, por sus siglas en inglés), o los diagramas de Feynman. Sin embargo, la adaptación de Quevedo para estos otros dominios representa una línea de investigación futura.

Para más información sobre Quevedo, se puede consultar el capítulo 9, la publicación científica donde describimos el software, o directamente la documentación técnica, publicada en línea e incluida en este documento también en el capítulo 10. Una demostración que incluye el corpus de VisSE se halla disponible en la dirección `https://holstein.fdi.ucm.es/visse/quevedo/`[8].

3.5.2. Aplicación web

Uno de los aspectos destacados de la investigación en lingüística es su cercanía con la sociedad. Dado que todos empleamos el lenguaje en nuestra vida diaria, resulta sencillo encontrar nexos entre la investigación académica y el público en general.

En lo referente a las lenguas de signos, este vínculo con la comunidad de personas sordas y signantes es sumamente potente y palpable. Todos los esfuerzos emprendidos en la investigación y el desarrollo de las lenguas de signos conllevan a un mayor reconocimiento, expansión y aceptación de las mismas, razón por la cual la comunidad signante acoge con entusiasmo cualquier forma de participación.

[8] Para solicitar acceso a esta demostración dirigirse por correo electrónico a `antonio@garciasevilla.com`

Sin embargo, la lingüística computacional presenta ciertos desafíos, principalmente debido a la necesidad de dispositivos específicos y a menudo, requisitos de software que no resultan fácilmente accesibles. En nuestro caso, el empleo de algoritmos avanzados de aprendizaje automático representa una dificultad considerable. El obstáculo no reside únicamente en el hecho de que no podemos solicitar a los usuarios que ejecuten nuestros algoritmos, sino que comúnmente ni siquiera poseen el hardware requerido.

Afortunadamente, los avances recientes en tecnología web y en los navegadores de internet posibilitan el uso de arquitecturas cliente-servidor que eluden estos inconvenientes. Los navegadores web modernos constituyen entornos de ejecución potentes que incluyen interfaces de usuario, acceso a recursos del dispositivo y una capacidad de cómputo significativa. Pero, lo más relevante, es que todo ello se logra mediante estándares ampliamente aceptados y una arquitectura subyacente distribuida. Si se evita la utilización de tecnologías de vanguardia, se garantiza un entorno de ejecución, el navegador, compatible con casi todos los dispositivos que el usuario pueda tener: ordenadores de sobremesa, dispositivos móviles, tabletas e incluso consolas de videojuegos.

Adicionalmente, el navegador se encuentra conectado de forma nativa a la red, lo que nos permite ejecutar en el servidor las partes de la aplicación que requieren hardware y configuración específicos, delegando el resto de las tareas al dispositivo cliente.

Por consiguiente, como culminación del proyecto VisSE, y con el objetivo de brindar un acceso sencillo y accesible a los resultados de nuestra investigación, desarrollé una aplicación web progresiva (*progressive web app*, PWA) que está disponible en la URL `https://garciasevilla.com/visse`.

Explorador didáctico de SignoEscritura

Para crear una aplicación final que culmine nuestro desarrollo de reconocimiento de SignoEscritura, necesitamos plantearnos cuál es la funcionalidad que puede resultar útil a un usuario general, no científico. Desafortunadamente, nuestro desarrollo no ha alcanzado el nivel de poder realizar una traducción automática, que sería el objetivo último de este tipo de investigación. Tampoco existen recursos de LSE con los cuales conectar de manera

sencilla, que pudieran servirnos para enlazar la SignoEscritura reconocida a un diccionario o similar. No obstante, estamos trabajando en ello en el proyecto Signario[9], ya mencionado anteriormente (Lahoz-Bengoechea y Sevilla 2022a), y con un poco de suerte en el futuro próximo se pueda ampliar esta funcionalidad.

Lo que sí podemos hacer con el trabajo realizado, y que resulte útil para un usuario final, es tomar el conocimiento implícito en nuestra algoritmia de reconocimiento y convertirlo en conocimiento explícito y accesible para el usuario. Esto es, transformar los grafemas detectados, junto con sus parámetros (`CLASS`, `SHAPE`, etc.), en explicaciones en español, y utilizarlos para ayudar a personas que no conozcan la SignoEscritura. Por ejemplo, esto puede ser útil para estudiantes de LSE, usuarios de un diccionario escrito en SignoEscritura, o para los propios signantes que, a pesar de su conocimiento nativo de la lengua, no conozcan la SignoEscritura y por lo tanto no dispongan de un sistema de escritura efectivo. A través de nuestra aplicación, pueden ver el significado "fonético" de los logogramas y grafemas desconocidos, reproduciendo las configuraciones y gestos física o mentalmente, y reconocer con ello qué signo o elemento gramatical está codificado en la imagen.

La aplicación se compone de dos partes principales. En primer lugar, el cliente, que se ejecuta en el dispositivo del usuario. Está programado con el paradigma de interfaz de usuario (UI) "reactiva", uno de los más extendidos en la actualidad en la programación web. El software por excelencia de este paradigma es React, una librería JavaScript creada por los ingenieros de Meta (anteriormente Facebook), y que llevo utilizando desde la versión 0.14 (Sevilla 2015). Para la aplicación de VisSE, empleamos una versión más ligera de React conocida como Preact. El estilo visual está diseñado con Tailwind CSS, una librería de estilos moderna que encaja muy bien en el paradigma y en la forma de desarrollo de las PWAs.

La aplicación del cliente se comunica con el servidor (backend), creado utilizando FastAPI. FastAPI es una librería Python destinada a la creación eficiente de servidores con APIs, manteniendo al mismo tiempo un alto nivel de buenas prácticas y una alta compatibilidad mediante el uso de estándares extendidos. La función de FastAPI es principalmente de mediador, y la mayor

[9] `https://griffos.filol.ucm.es/signario`

parte del trabajo del backend recae en la propia librería Quevedo. FastAPI recoge las peticiones del usuario, las procesa con la funcionalidad de Quevedo y envía la respuesta al frontend en formato JSON.

Para implementar esto, incorporamos al backend un sistema de generación de lenguaje natural que crea explicaciones, empleando plantillas para convertir el conocimiento codificado en las etiquetas de los grafemas en explicaciones accesibles al gran público.

El frontend, por su parte, permite al usuario seleccionar o capturar una imagen de SignoEscritura, que es enviada al servidor. Para poder demostrar la aplicación a usuarios que no dispongan de muestras de SignoEscritura, el frontend proporcionan también algunos ejemplos sin necesidad de que el usuario busque o dibuje ningún logograma. Una vez recibida la imagen elegida, el backend utiliza la funcionalidad de Quevedo para ejecutar el pipeline de reconocimiento entrenado con el corpus VisSE y descrito en secciones anteriores. El resultado es, como hemos visto, una lista de los grafemas detectados, junto con sus atributos portadores de significado. El backend convierte este resultado en texto natural, y lo envía de vuelta al frontend, junto con algunos metadatos adicionales.

El frontend recoge esta respuesta y la muestra al usuario en una interfaz adaptada al tamaño de pantalla disponible, como se puede apreciar en la figura 3.8. El usuario puede navegar por las explicaciones, observando a qué grafema corresponden, o hacer clic directamente en los grafemas para ver qué significa cada uno.

Modelo 3D de la Mano

Adicionalmente, en lo que respecta a los grafemas manuales, se proporciona un modelo tridimensional interactivo que puede facilitar la comprensión de la posición de los dedos y la orientación de la mano, sirviendo de complemento a la descripción textual.

Este modelo tridimensional, creado personalmente utilizando la herramienta Blender[10], emplea el conocimiento fonológico adquirido a lo largo de estos años de investigación. La intención original residía en la creación de un avatar 3D completo, capaz de reproducir todos los rasgos de la LSE, pero

[10] `https://www.blender.org`

Fig. 3.8. – Captura de pantalla de la aplicación web de VisSE.

este objetivo resultó ser excesivamente ambicioso, por lo que se optó por una aproximación más limitada.

No obstante, el modelo de la mano es considerablemente completo y captura con precisión la fonología configuracional de la mano en la LSE, de acuerdo con nuestra teoría actual. Dicho modelo ha sido publicado en acceso abierto bajo una licencia de código abierto, lo cual permite su utilización por otros investigadores o en diferentes aplicaciones. Ejemplos de estas aplicaciones, desarrollados durante la tesis, se pueden encontrar en la sección A.3, aunque utilizando un modelo 3D de una versión anterior.

En términos técnicos, el código del modelo se ha implementado en JavaScript, aprovechando la librería THREE.js. De esta manera, su uso no requiere de la instalación de ningún software específico, más allá del navegador, permitiendo su utilización en la mayoría de ordenadores y dispositivos móviles ampliamente difundidos.

Este modelo 3D (figura 3.9), así como la funcionalidad fonológica implementada, es otro resultado de desarrollo software dentro del proyecto VisSE, quizá secundario ante la funcionalidad de reconocimiento de SignoEscritura pero que también abre la vía a futuros desarrollos y aplicaciones.

Fig. 3.9. – Modelo 3D de la mano, mostrando el esqueleto o "rigging" en la configuración de "T" del alfabeto dactilológico de la LSE. Se muestra el modelo mediante una captura de pantalla de Blender, la aplicación utilizada para su elaboración.

3.6. Conclusiones

El planteamiento del proyecto VisSE era crear herramientas para mejorar el uso de la SignoEscritura en el contexto digital. Como resultado más destacado, se ha desarrollado una aplicación web, accesible en línea través en `https://holstein.fdi.ucm.es/visse/`[11]. Esta aplicación sirve como una plataforma educativa que instruye a los usuarios en la SignoEscritura, permitiéndoles seleccionar sus diversos elementos y generando descripciones ya sea en formato textual o mediante un modelo 3D de la mano.

No obstante, esta aplicación web representa meramente la punta del iceberg. Su concepción y desarrollo exigieron distintos esfuerzos intensivos durante un período de aproximadamente año y medio. En el sitio web del proyecto, `https://www.ucm.es/visse`, se ofrece un resumen más extenso de los resultados obtenidos, incluyendo un vídeo en LSE que explica las metas y logros del proyecto. El resultado del proyecto fue también cubierto

[11] Alternativamente en `https://garciasevilla.com/visse`

por la prensa («Una herramienta para traducir la SignoEscritura, la lengua de signos escrita» 2022)[12].

Los esfuerzos realizados para la consecución del proyecto, descritos en este capítulo, se alinean con los objetivos técnicos de la tesis, permitiéndome por tanto avanzar con mi investigación y cumplir con los objetivos más académicos, y al mismo tiempo crear el resultado final de transferencia a la sociedad que se materializa en la aplicación web mencionada.

Adicionalmente, la gestión del proyecto ha contribuido a mi desarrollo como científico. He tenido que emprender diversas actividades que, si bien no se enmarcan en los objetivos específicos de la tesis, forman parte integral de la formación que he adquirido durante el período de doctorado. Aspectos como la gestión presupuestaria, la interacción con diferentes niveles de burocracia (universitaria y empresarial), la planificación temporal y la coordinación de colaboradores, constituyen también un componente esencial de mi formación como investigador. Estos elementos complementan la investigación y desarrollo que constituyen el núcleo de esta tesis, contenida en el compendio de publicaciones, pero no resultan menos importantes para mí al haber supuesto una experiencia de aprendizaje valiosa y muy extensiva.

[12] https://www.lavanguardia.com/vida/20220502/8236972/herramienta-traducir-signoescritura-lengua-signos-escrita.html

4. Conclusiones

El objetivo general de esta tesis consistía en explorar y desarrollar avances en el tratamiento computacional de las lenguas de signos, con un enfoque particular en la Lengua de Signos Española, mediante la aplicación de técnicas modernas de IA, LC y PLN.

En aras de alcanzar este objetivo, se planteó inicialmente una meta teórica consistente en el análisis profundo de la lengua de signos. Esto implicaba no solo un estudio del estado actual del campo, sino también un entendimiento exhaustivo de la LSE y su uso como lengua natural. A este respecto, realicé estudios formales de LSE en el Centro Superior de Idiomas Modernos (CSIM) de la Universidad Complutense, alcanzando el nivel C2 dentro del Marco Común Europeo de Referencia para las Lenguas.

El estudio riguroso de la lengua y la interacción con la comunidad sorda han contribuido de manera invaluable no solo a mi investigación y a la elaboración de esta tesis, sino también a mi desarrollo integral como profesional en la intersección de la informática, la lingüística computacional y como individuo. Mi experiencia en el aprendizaje y análisis de la LSE ha reconfigurado significativamente mi perspectiva sobre la naturaleza del lenguaje, el pensamiento humano y la inteligencia artificial. Mi profundización en la gramática y en las características únicas de la LSE no solo ha ampliado mi conocimiento lingüístico, sino que también ha establecido un sólido fundamento para futuros desarrollos en el campo, permitiéndome adaptar las técnicas de LC y PLN a las particularidades de las lenguas de signos.

Entre los logros específicos derivados del análisis teórico, cabe destacar la publicación de varios artículos académicos más allá de los incluidos en el compendio. En Sevilla y Lahoz-Bengoechea (2019), publicado en la revista de la Sociedad Española de Procesamiento del Lenguaje Natural, abordamos un estudio sobre la característica fonológica de la orientación en las lenguas de signos, resaltando sus implicaciones sintácticas. Otro trabajo re-

levante es Lahoz-Bengoechea y Sevilla (2022b), presentado en el congreso de la Asociación Española de Lingüística Aplicada, en el que introducimos la "Signotación", una representación fonológica de la LSE que no solo ofrece utilidad lingüística, sino que también sirve como base para numerosos enfoques computacionales para el procesamiento de la LSE.

Adicionalmente, diseñé el esquema de anotación del corpus VisSE, que está lingüísticamente fundamentado y emplea en algunos casos la Signotación para capturar características específicas de la LSE. Este esquema de anotación, resultado del análisis teórico, contribuye también de manera significativa a la realización de los objetivos posteriores.

Un hallazgo crucial del análisis teórico fue la identificación del desafío asociado a la representación de la LSE, que representa un obstáculo fundamental para su tratamiento computacional. Este descubrimiento condujo al objetivo técnico de la tesis: avanzar en el procesamiento de la SignoEscritura. Para abordar este desafío, se inició el proyecto VisSE[1], cuyo propósito era desarrollar herramientas que facilitasen el uso de SignoEscritura en entornos digitales.

Para abordar el objetivo técnico principal, se establecieron tres sub-objetivos específicos. El primero de estos sub-objetivos respondía a la necesidad imperante de trabajar con un volumen suficiente de datos. Esta necesidad era doble: por un lado, se pretendía adoptar una aproximación empírica al estudio, y por otro, se quería emplear técnicas de IA modernas, que requieren de conjuntos de datos significativos para su entrenamiento efectivo. En respuesta a este primer sub-objetivo, se recopilaron más de 2000 muestras de SignoEscritura, de las cuales 982 fueron cuidadosamente anotadas y organizadas en el corpus VisSE. Aunque la anotación manual de todas las muestras recopiladas resultó ser una tarea demasiado ardua para completar en el marco temporal del proyecto, se han sentado las bases para futuras anotaciones que permitirán ampliar el corpus en futuras revisiones.

El segundo sub-objetivo técnico se centraba en el uso de técnicas de IA para el reconocimiento y la comprensión de la SignoEscritura. Gracias al conjunto de datos previamente recopilado, se entrenaron algoritmos de aprendizaje automático que lograron reconocer los grafemas en nuevas instancias de

[1] https://www.ucm.es/visse

SignoEscritura con un acierto del 68 %. Este nivel de acierto supera al que se podría haber alcanzado mediante el uso de un único algoritmo de aprendizaje automático. Para lograrlo, se diseñó un sistema experto que combina diversas redes neuronales junto con reglas lógicas de procesamiento, aumentando así tanto la precisión como la robustez de nuestro sistema, en una metodología novedosa que constituye otra de las aportaciones principales de la tesis. A pesar de estos avances, aún queda margen para mejorar. Por una parte, la tasa de acierto global es susceptible de mejora, aunque la sección 8.6 presenta una discusión más optimista al respecto. Por otra parte, la comprensión completa de la SignoEscritura también implica entender el significado composicional de los logogramas, emergente de su combinación sintáctica más allá de sus propiedades léxicas. Esta tarea, no obstante, se presenta como un reto para futuras investigaciones.

El tercer y último sub-objetivo técnico se orientaba hacia la materialización de los resultados obtenidos a través del desarrollo de software. El principal logro en este ámbito es la creación de Quevedo, una librería y aplicación en Python que codifica de forma reproducible y utilizable el conocimiento experto adquirido. Esta herramienta no solo posibilita el procesamiento de nuestra colección de SignoEscritura o de nuevas muestras que puedan ser recopiladas, sino que también permite replicar nuestra investigación con otros conjuntos de datos o incluso adaptarla a otros "lenguajes gráficos", una posibilidad que descubrimos gracias a la metodología usada de abstracción y generalización de procesos y teorías.

Además, el proyecto VisSE tenía como meta la creación de herramientas que pudieran ser transferidas a aplicaciones útiles para los usuarios. Este objetivo responde no solo al tercer sub-objetivo técnico relativo al desarrollo de software, sino también al objetivo social más amplio del proyecto. La principal herramienta desarrollada en este sentido es una aplicación web que permite a los usuarios entender los distintos grafemas de la SignoEscritura y su significado en nuevas muestras enviadas. Asimismo, se creó un modelo de mano en 3D que es capaz de articular la fonología configuracional de la LSE, y que ha sido integrado en la aplicación para proporcionar una representación visual, además de textual, del significado de los grafemas. Este proyecto, por tanto, no solo ha avanzado en el ámbito técnico del procesamiento lin-

güístico de SignoEscritura, cumpliendo con nuestros objetivos académicos y científicos, sino que también ha hecho contribuciones tangibles a la sociedad.

Finalmente, me había planteado como objetivo complementario el preparar tanto los datos como el software desarrollado de una forma estandarizada y formalizada. Estos han sido publicados en línea y bajo licencias abiertas con el fin de facilitar su utilización por otros investigadores y desarrolladores. Todo el software desarrollado se encuentra accesible en GitHub, incluyendo el sistema experto para el reconocimiento de SignoEscritura. Los datos recopilados se han organizado meticulosamente en un corpus que, adicionalmente, incluye una exhaustiva guía de anotación tanto en inglés como en español. Todo este conjunto de recursos se halla disponible en línea, así como los pesos de las redes neuronales que hemos entrenado. Los distintos resultados obtenidos a lo largo de esta investigación, así como otros desarrollos de menor envergadura que no son objeto de esta discusión para evitar dilatarnos innecesariamente, se detallan en el apéndice A.

4.1. Trabajo futuro

A pesar de que no he logrado desarrollar un algoritmo que comprenda y traduzca de manera íntegra la SignoEscritura, he realizado avances significativos en su procesamiento que sientan las bases para avanzar hacia el reconocimiento total de la SignoEscritura en futuras investigaciones. El elemento que aún falta por desarrollar, la composición sintáctica de grafemas, se encuentra actualmente en desarrollo, y Quevedo permite ya la anotación visual de dependencias sintácticas que podrán ser posteriormente analizadas y aprendidas por algoritmos de Inteligencia Artificial.

La metodología que he diseñado en esta tesis ofrece además potencial de aplicabilidad a otras lenguas de signos. Esta adaptabilidad permite ajustar los detalles particulares que la SignoEscritura requiere para cada lengua de signos específica. Dado que el software está publicado y debidamente documentado, los futuros trabajos en esta línea de investigación pueden beneficiarse de un punto de partida sólido, basado en los resultados que he obtenido, sin tener que empezar desde cero.

En lo que respecta al conocimiento científico-técnico adquirido sobre la Lengua de Signos Española, este resultado, emanado del objetivo teórico, abre diversas vías de trabajo más allá de la SignoEscritura. El sistema que venimos desarrollando estos años, la Signotación, ofrece una codificación paramétrica de la LSE, complementaria a la SignoEscritura y que permite mayor y más fácil procesamiento computacional. Es relevante señalar que este trabajo futuro ya está en curso, y que las bases establecidas en esta tesis contribuyen también al desarrollo del Signario, un diccionario paramétrico de la LSE[2] (Lahoz-Bengoechea y Sevilla 2022a).

Finalmente, otra dirección posible de investigación es el estudio de los lenguajes gráficos, un paradigma lingüístico que emerge del trabajo realizado para VisSE. Este enfoque podría mejorar el uso digital de otras formas de comunicación, así como quizá proporcionar nuevas perspectivas acerca de la capacidad lingüística humana.

[2] `https://www.ucm.es/signariolse`

II

Compendium Scriptorum

5. Tools for the use of SignWriting as a Language Resource

Antonio F. G. Sevilla, Alberto Díaz Esteban, José María Lahoz-Bengoechea

This article was submitted to the 9th Workshop on the Representation and Processing of Sign Languages, May 2020. The proceedings, as well as the article excerpt and the poster presented (at the 2022 edition, due to the Covid-19 lockdowns) are linked from my personal page: `https://garciasevilla.com/2020/05/31/Tools-for-the-use-of-SignWriting-as-a-Language-Resource/`.

The text is reproduced in the following as one of the contributions to this thesis.

Abstract

Representation of linguistic data is an issue of utmost importance when developing language resources, but the lack of a standard written form in sign languages presents a challenge. Different notation systems exist, but only SignWriting seems to have some use in the native signer community. It is, however, a difficult system to use computationally, not based on a linear sequence of characters. We present the project "VisSE", which aims to develop tools for the effective use of SignWriting in the computer. The first of these is an application which uses computer vision to interpret SignWriting, understanding the meaning of new or existing transcriptions, or even handwritten images. Two additional tools will be able to consume the result of this recognizer: first, a textual description of the features of the transcription will make it understandable for non-signers. Second, a three-dimensional avatar will be able to reproduce the configurations and movements contained

within the transcription, making it understandable for signers even if not familiar with SignWriting. Additionally, the project will result in a corpus of annotated SignWriting data which will also be of use to the computational linguistics community.

5.1. Introduction

One of the challenges in the study of sign language linguistics is the collection and representation of linguistic data. In computational linguistics, this problem is even more crippling, since data are the basis of any computational approach to a subject.

There is an increasing interest both in society and the scientific community in sign languages, and corpora have been created for many different sign languages and with varying schemes of annotation. However, most corpora are video-based, which is equivalent to the hypothetical case of corpora of oral languages being mostly based on audio recordings.

Recordings of real utterances, both of oral or signed languages, are difficult to process computationally, whether it is for searching or managing the data, or for linguistically analyzing it and finding its structure and meaning. Video is especially difficult, since the human visual system is highly sophisticated, and emulating its processes with artificial intelligence is not a solved problem yet.

In oral languages, writing poses a useful alternative to recordings, and is indeed (and maybe to a fault) the basis on which computational linguistics have been built. However, there does not exist an equivalent in signed languages. There is not a widely accepted written form for these languages, even less a literature or a corpus of real world linguistic data that can be exploited.

There exist some candidates for this, the most promising being SignWriting. SignWriting is a system that can act as a written form of sign language, or at least as a transcription system for it. It is iconic and very in-line with the visual nature of sign languages, so it is easy to understand and accept by native signers. The problem is that it is not as easy to use in the digital world, not being formed by linear strings of characters that can be quickly input with

a keyboard and consumed by the many tools developed by the computational linguistics community.

We present an early-stage project for developing tools and resources that aim to facilitate the effective use of SignWriting in computers. With these tools, input of SignWriting can be as quick as writing it on paper, and no further processing by the user is necessary. Other tools will also use this input to generate related output, such as a textual description of the signer's actions or an animated avatar, which means that SignWriting will be useful as a digital representation of sign language even for users not familiar with it. This can help in the teaching of sign language, by facilitating the use of this language in computers, and also increase accessibility and inclusion of the Deaf community in the digital world.

In the next section, we give a brief overview of the problems of sign language notation, and quickly explain SignWriting and computer vision, the artificial intelligence tool to be used for its processing. Section 5.3 explicates the architecture of the project and its different components, and in Section 5.4 some conclusions are drawn.

5.2. Background

Sign languages are natural languages which use the visual-gestual modality instead of the oral-acoustic one. This means that instead of performing gestures with the vocal organs, which are transmitted to the receiver via sound, sign languages utilize gestures of the body, especially of the hands, to transmit information visually.

While oral languages have developed writing systems that represent the sounds (or sometimes ideas) of the language in a visual, abstract, and standard way, none such system has organically appeared for sign languages. Writing systems have many advantages, both to users of the language in helping them analyze it, and making structure explicit, and to linguists. To linguists, one advantage of writing systems of great relevance lately, and especially to us in the computational linguistics community, is the ease of computational treatment.

A number of systems have been developed for the transcription of sign language into written form (Stokoe 1960; Herrero Blanco 2003; Hanke 2004). Most of them are intended for linguistic research and transcription of fine linguistic detail, and none of them seem to have seen universal use or the kind of standardization seen in the writing systems of oral languages.

This presents a challenge for the development of language resources. Systems which are alien to native informants of sign language require training for these users, and in limited time frames inevitably pose the question of whether the information transcribed with them really is what the signer intended. Additionally, we have found that computational tools for the management of the different notation systems are not very mature or wide-spread.

However, there is another proposed transcription system for sign languages: SignWriting (Sutton 1995). SignWriting is a system developed by Valerie Sutton, a non-linguist, in 1974, designed specifically to write sign languages. There is much information on its use and practicalities on the website[1], and especially interesting is the comparison between some notation systems[2]. We reproduce a slightly modified excerpt in Table 5.1.

We give a short introduction to SignWriting in the following, but Di Renzo et al. (2006) give an informative discussion of the use of this system in linguistic research, along with some notes on the challenges that notation systems present. More on this topic and on the differences between notation and writing systems can be found in Van der Hulst and Channon (2010).

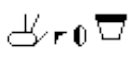
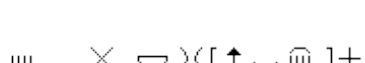
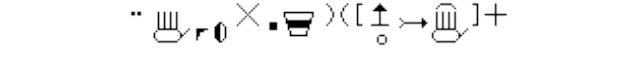
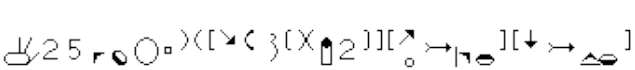
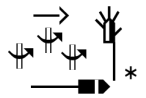
5.2.1. SignWriting

As mentioned before, SignWriting is a system intended for the writing of sign languages. It is made up of symbols, many of which are highly iconic, that represent different linguistic or paralinguistic aspects. See for example Sutton and Frost (2008).

Different handshapes (such as a closed fist, an open palm, etc.) are depicted by figures like a square, or a pentagon, respectively. Conventional strokes can be added to these basic shapes to represent the thumb or the different fingers. The spatial orientation of the hand is symbolized by a black and

[1] http://www.signwriting.org
[2] http://www.signwriting.org/forums/linguistics/ling001.html

Tab. 5.1. – Comparison of notation systems for sign languages, using words from an American Sign Language text for the story of Goldilocks and the Three Bears[a]. The systems compared are Stokoe Notation (Stokoe 1960), HamNoSys (Hanke 2004) and SignWriting (Sutton 1995).

	Stokoe Notation	HamNoSys	SignWriting
Three	$3^{\perp}$		
Bears	$[]\checkmark C^{\ddagger}\checkmark C^{\vee\bullet}_{X}$		
Goldilocks	$\lambda Y^{\circlearrowleft}_{V}$		
Deep Forest	$\overline{B}_{a}\checkmark B_{\wedge}^{\lambda}$		

[a] http://www.signwriting.org/forums/linguistics/ling001.html

white color code, among other possibilities. There are also icons for different locations on the body (mainly, parts of the head and face). Other symbols stand for changes in the handshape or the orientation, for different kinds of movements and trajectories, for contacts, for variations in the speed, and for facial expressions, including eyebrow intonation and other paralinguistic realizations. Finally, there are symbols that represent pauses and prosodic grouping, thus allowing to write full sentences.

All these symbols combine non-linearly in space to transcribe signs in a visually intuitive way. This is a most welcome characteristic for the Deaf community, inasmuch as they give preeminence to anything visual, and it makes it easier to learn for students of sign languages or any interested person.

Furthermore, its iconicity, together with its flexibility, allow to transcribe any newly-coined sign, making it advantageous for treating sign languages not only for daily use but also in technical, scientific, and educational environments.

The fact that symbols are not interpreted linearly, but according to their position relative to other symbols, poses a challenge to the computational treatment of these bundles. It is necessary to decompose the fully transcribed signs into their components and parametrize them in linguistically relevant subunits or features.

If SignWriting annotations are created with computer tools, this information may be readily available. However, the non-linearity of SignWriting, along with the large amount of symbols that can be used, make computational input cumbersome and far slower than hand-drawing of transcriptions. Additionally, existing transcriptions, even if computer made, may not be available in their decomposed form, but rather as a plain image with no annotation. Therefore, there exists the need for tooling that can interpret images containing SignWriting transcriptions in an automatic way.

5.2.2. Computer Vision

Broadly speaking, computer vision is the field of artificial intelligence where meaning is to be extracted from images using automatic procedures. What this meaning is depends on the context, the available data, and the

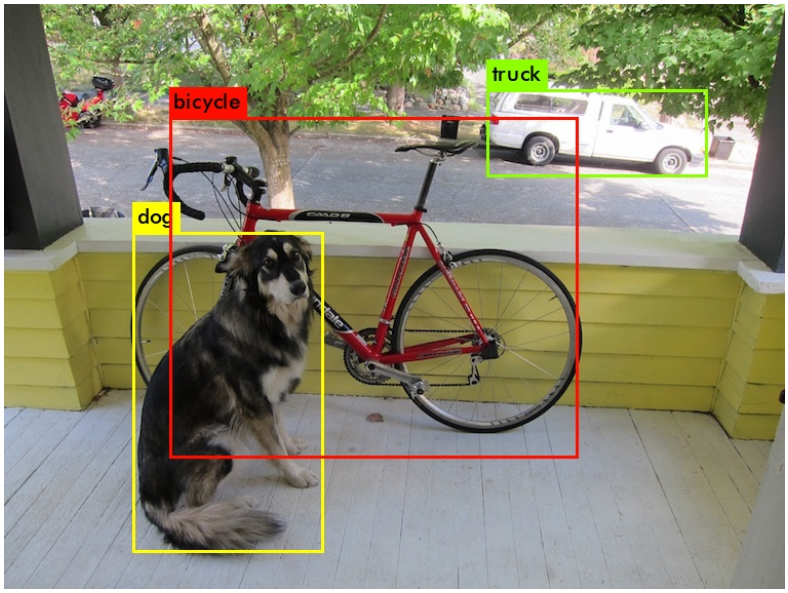


Fig. 5.1. – Object detection task, where objects in an image are located and classified (Redmon, Divvala, et al. 2016).

desired result. As in other fields of artificial intelligence, classification is the task of assigning a label to an image, for example the type of object found in a photograph, or the name of the person a face belongs to.

Object detection is a step beyond, in which it is to be found in an image not only what object it represents, but also where in the image it is. In the most common case, there can be many objects in an image, or none, and it is necessary to find how many there are, where, and what their labels are.

This is a difficult task, but it is very well suited to machine learning approaches, especially neural networks and deep learning. These techniques work by presenting a large amount of annotated data to the algorithm, which is able to extract from them features and patterns from which to decide the result of the procedure. Often, this means bounding boxes: rectangles that contain the object in the image, along with labels for what the detected object is. In Figure 5.1 an example of this task can be seen.

YOLO (You Only Look Once) is an algorithm for object detection that works by applying a single neural network to the full image (Redmon and Farhadi 2018). Other algorithms work in multiple steps, for example by first performing detection of possible candidates and then classifying them, but YOLO

works in a single pass, making it faster and easier to use. It works by dividing the image into regions, predicting bounding boxes and label probabilities for each region, and then collating these regions and possibilities into the final list of results. Its implementation in Darknet (Redmon 2013) is very easy to configure and utilize, while retaining precision at the state of the art.

This task of object detection is exactly what we need for understanding SignWriting transcriptions. They are formed by different symbols, placed relative to each other in a way that is meaningful and significant. By using YOLO, we can automatically find these symbols and their positions in SignWriting images, which allows us to further work with the meaning of the transcription instead of with the pixels of the image[3].

5.3. The VisSE project

During the authors' research in Spanish Sign Language, the problems outlined in the introduction regarding its digital treatment were patent. As students of this language as well as researchers and engineers, ideas for solutions started to come to our minds. At some point, previous expertise in image recognition, a very salient topic in sign language research, joined the knowledge of SignWriting as a useful tool for these languages, used by our educators and many in the Deaf community.

Some of the ideas for both tools and processes were combined into a single effort for which funding was requested, and granted by Indra and Fundación Universia as a grant for research on Accessible Technologies. This effort resulted in the VisSE project ("Visualizando la SignoEscritura", Spanish for "Visualizing SignWriting") aimed at developing tools for the effective use of SignWriting in computers. These tools can help with the integration of Hard of Hearing people in the digital society, and will also help accelerate sign language research by providing another methodology for its research.

A general architecture of the project can be seen in Figure 5.2. There, the sign in Spanish Sign Language for "teacher" is used as an example. Its transcription in SignWriting is decomposed and processed by an artificial

[3]When we say meaning of the transcription, we mean the codification it contains of sign language utterances, not the meaning of the represented signs in a linguistic semantics way.

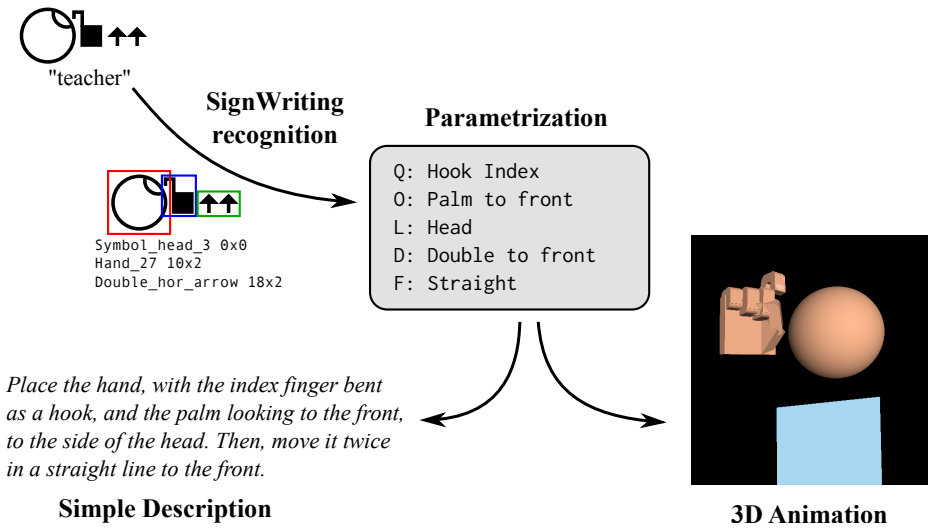


Fig. 5.2. – Architecture of the different components of the VisSE project.

vision algorithm, which finds the different symbols and classifies them. The labels and relative positions of the symbols are then transformed into their linguistic meaning, called here "parametrization". In the example, the usual features of sign language analysis are used, but this representation is yet to be decided, and has to follow closely the information encoded in the SignWriting transcription. The parametrization is then turned into a textual description, which allows a non-signer to realize the sign, and a 3D animation which can be understood by a signer.

5.3.1. Corpus of SignWriting Transcriptions

While the goal of the project is to develop the tools mentioned before, which will help with the use of SignWriting in the digital world, there will be an additional language resource result. Data are of paramount importance when doing computational linguistics, and the artificial vision algorithms to be used rely on these data for their successful training and use.

Therefore, one of the products of the project will be a corpus of linguistic annotations. Entries in the corpus will be, as far as possible, input by informants who are native signers of Spanish Sign Language. For this purpose, a custom computer interface will be developed. This interface needs

only be a simple front-end to the database, with roles for informants and for corpus managers, and with some tool to facilitate SignWriting input, either by a point-and-click interface or by a hand-drawing or scanner technology. Annotation, however, will not consist of grammatical information, but rather of the locations and meanings of the different symbols in the transcription.

Even if less interesting to our users, this result will probably be of use to other researchers, so it too will be publicly released. Similar to other such projects, the main object of annotation will be lexical entries, words of sign language and their realization, the main difference being that the data recorded will be in the form of SignWriting. The meaning of the annotated sign will be transcribed using an appropriate translation in Spanish.

Corpora that peruse SignWriting already exist (Forster et al. 2014), and there is also SignBank[4], a collection of tools and resources related to SignWriting, including dictionaries for many sign languages around the world, and SignMaker, an interface for the creation of SignWriting images. While useful, the data available in the dictionaries are limited, especially for languages other than American Sign Language, and its interface is more oriented toward small-scale, manual research rather than large-scale, automated computation.

5.3.2. Transcription Recognizer

At first, the annotations in the corpus will have to be performed by humans, but they will immediately serve as training data for the YOLO algorithm explained in Section 5.2.2. As annotation advances, so will increase the performance of the automatic recognition, which will be used to help annotators in their process by providing them with the prediction from the algorithm as a draft. This will accelerate data collection, which will in turn increase training effectiveness until at some point the algorithm will be able to recognize most input on its own.

The use of YOLO for recognition of SignWriting has already been successfully prototyped by students of ours (Sánchez Jiménez, López Prieto, and Garrido Montoya 2019). The located and classified symbols found by the algorithm will then be transformed into the representation used in the corpus,

[4] http://www.signbank.org/

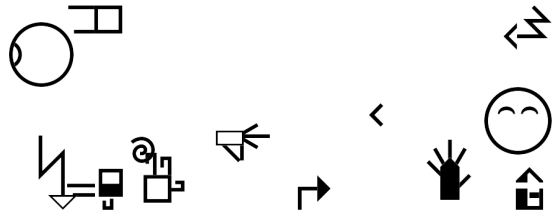


Fig. 5.3. – Automatically generated training samples for YOLO.

which will include the linguistically relevant parameters (for example, it is relevant that the location is "at head level", but not whether the transcription is drawn 7 pixels to the right).

This process of finding out sign language parameters using computer vision is akin to that of automatic sign language recognition in video, which is often performed for video-based corpora. However, it is much simpler, both for the human annotator and the computer vision algorithm, since images are black and white, standardized and far less noisy.

Transcriptions, being composed out of a discrete (even if large) amount of symbols, present an additional advantage: training data can be immensely augmented by automatic means. An algorithm, already implemented, mixes the possible symbols to create random images which contain the data and annotations the YOLO algorithm needs to be trained. An example of this can be seen in Figure 5.3. Even if these are meaningless as SignWriting transcriptions, they contain the features and patterns that the algorithm needs to learn. This will help bootstrap the system, which will make the recognizer useful to annotators earlier rather than later during the project.

5.3.3. Description of the Sign

Once the algorithm is able to understand transcriptions, its parametric result will be used in tools that can help integrate the Deaf community in Spain into the digital world.

The first such tool will be a generator of textual descriptions from SignWriting parametrizations. This description will explain, in a language easy to understand, the articulation and movements codified in the SignWriting symbols.

This will be a useful aid in communication between signers and non-signers. For example, it can be explained to non-Deaf people how to sign basic vocabulary, like pleasantries, or maybe important words in a particular domain (an office, a factory). This will allow non-Deaf people communicate to Deaf people basic information, useful for the daily routine or maybe an emergency, without the need to learn sign language proper. This may help broaden the employability landscape for Deaf people, increasing their inclusion in the office community and with their non-signing peers.

The use of text instead of video has some advantages. While observation of real video and images is necessary for proper understanding of the rhythm and cadence of sign language, it is often not enough for correct articulation and orientation of the hand. Non-signers are not used to looking for the visual cues in hand articulation, and may confuse the hand configurations necessary for particular signs. Spelling them out, however, can help them realize the correct finger flexing and wrist rotation, in an environment where an interpreter or teacher may not be readily available.

This also leads to a different application of this tool: education. Both Deaf and non-Deaf people can find it challenging to self-study sign language, due to the scarcity of resources and the challenge of a lack of a linear transcription system. Translating SignWriting into text can improve understanding of this system, helping both signers and students learn the representation of sign language in SignWriting symbols in a dynamic environment with immediate feedback.

5.3.4. Animated Execution by an Avatar

Another result of the project will be a three-dimensional animated avatar, capable of executing the signs contained in the parametric representation. SignWriting is an (almost) complete transcription of sign language, including spatial and movement-related information. This information, after its computational transformation into parameters, can be directly fed into a virtual avatar to realize the signs in three dimensions.

The advantages of the use of avatars are known in the community, and have been studied before. Kipp, Heloir, and Nguyen (2011) give an informative account of different avatar technologies and the challenges in their use,

and propose methodologies for testing and evaluation of the results. Bouzid and Jemni (2014) describe an approach very similar to ours in its goal, using SignWriting as the basis for generating the sign language animations. However, this process is not done automatically, but manually as in other avatar technologies.

Manual preparation of the execution of signs is a costly procedure, even if not as much as video-taping interpreters. An expert in the system as well as one in sign language are needed, and it is difficult to find both in one person. With our approach, instead of knowing the intricacies of the avatar technology, just an expert in SignWriting would be needed. It is far easier that this expert is the same as the sign language translator or author, or minimal training can be provided. SignWriting is also easier to carry around and edit, compared to systems like SIGML (Kennaway 2004), which may be intuitive and easy for computer engineers but no so much to non-computer-savvy users.

Our system will also strive to be dynamic, not presenting a static sequence of images but rather an actual animation of the sign. While sign language generation is very complicated, it is important to note that this is not what our system needs to do. Placement and movement are already encoded in SignWriting, and our system only needs to convert the parameters into actual coordinates in three-dimensional space.

The technology for this tool will be Javascript and WebGL, which are increasingly mature and seeing wide-spread adoption in the industry. Web technology is ubiquitous nowadays, and browsers present an ideal execution environment where users need not install specific libraries or software but rather use the same program they use in their everyday digital lives.

5.4. Conclusions

As we have seen, the goal of the VisSE project is to develop a number of tools for the use of SignWriting effectively in computers. First, the recognizer will allow SignWriting to be used as input in a comfortable way. Users will not need to search for symbols, and then drag and drop them to the canvas, nor will they have to memorize an arbitrary mapping from ASCII characters to SignWriting

symbols. Hand written transcriptions or existing images will be able to be processed, making the use of SignWriting practical for continuous use. Then, the description generator and the avatar will use this input to transform SignWriting transcriptions of signs into alternative representations, which will help users understand both the meaning and the use of SignWriting.

All the developed tools will be publicly released, and the full pipeline might include software that allows a user to dynamically input SignWriting into an interface and immediately watch its realization by the avatar. The data generated in the form of the corpus can also be transformed into a dictionary, one where words are stored and indexed directly in sign language. Often, sign language resources are only accessible via oral language glosses, but the use of SignWriting allows sign language to be the primary language in its own dictionary.

These are all future works worthy of research and development, which will benefit the Deaf community in Spain. But the methodology and principles used are not specific to Spanish Sign Language, so we expect they will be able to be adapted to other sign languages.

Apart from the results benefiting the Deaf community, there will also be results for the language resource community. The data collected, in the form of the corpus, and the recognizer algorithm, will be released for the use of other researchers. Additionally, if this project helps SignWriting to become even more widespread and easier to use in computational contexts, this might become another powerful tool for the sign language linguistics community.

Therefore, we present this article to the community, with the goal of receiving feedback and comments during the early stages of the project so that it can inform and improve its development and its usefulness for the Computational Linguistics field.

5.5. Acknowledgements

The research leading to and contained within this project is partially funded by the project IDiLyCo: Digital Inclusion, Language and Communication, Grant. No. TIN2015-66655-R (MINECO/FEDER) and the FEI-EU-17-23 project InViTAR-IA: Infraestructuras para la Visibilización, Integración y Transfer-

encia de Aplicaciones y Resultados de Inteligencia Artificial (Universidad Complutense de Madrid).

Funding for the development of the project “Visualizando la SignoEscritura” has been awarded by Indra and Fundación Universia as part of the program for funding of research projects on Accessible Technologies.

6. Building the VisSE Corpus of Spanish SignWriting

Antonio F. G. Sevilla, Alberto Díaz Esteban, José María Lahoz-Bengoechea

This article was published in the journal "Language Resources and Evaluation" in October 2023.

The accepted article text is reproduced here, as part of the contributions to this thesis, but the version of record can be found online at `https://link.springer.com/article/10.1007/s10579-023-09694-9`.

Abstract

SignWriting is a system for transcribing sign languages, using iconic depictions of the hands and other body parts, as well as exploiting the possibilities of the page as a two dimensional medium to capture the three-dimensional nature of signs. This goes beyond the usual line-oriented nature of oral writing systems, and thus requires a different approach to its processing. In this article we present a corpus of handwritten SignWriting, a collection of images which transcribe signs from Spanish Sign Language. We explain the annotation schema we have devised, and the decisions which have been necessary to deal with the challenges that both sign language and SignWriting present. These challenges include the transformational nature of symbols in SignWriting, which can rotate and otherwise transform to convey meaning, as well as how to properly codify location, a fundamental part of SignWriting which is completely different to oral writing systems. The data in the corpus is fully annotated, and can serve as a tool for computational training and evaluation of algorithms, as well as provide a window into the nature of SignWriting and

the distribution of its features across a real vocabulary. The corpus is freely available online at https://zenodo.org/record/6337885.

Acknowledgments

The VisSE corpus was created thanks to funding from the project "Visualizando la SignoEscritura" (VisSE, Visualizing SignWriting[1]), reference number PR2014_19/01, funded by Indra and Fundación Universia, and development continues thanks to the project "Signario de LSE: Diccionario paramétrico de la Lengua de Signos Española" (SSL Signary: A parametric dictionary of Spanish Sign Language[2]), reference number IN[21]_HMS_LIN_0070, supported by a 2021 Leonardo Grant for Researchers and Cultural Creators from the BBVA Foundation. The BBVA Foundation accepts no responsibility for the opinions, statements and contents included in the project and/or the results thereof, which are entirely the responsibility of the authors.

6.1. Introduction

Modern linguistics rely ever increasingly on digital data, source instances of language along with annotations of their origin, meaning, or features. These assets are often organized into datasets or corpora, collections of annotated linguistic data sharing a theme or object of study. The creation and sharing of datasets can help immensely in the research of a certain subject, allowing empirical investigation as well as providing a shared substrate on which to discuss and compare theories.

As an object of increasing linguistic scrutiny, sign languages have also seen the construction of diverse corpora in recent times, covering some of the more than a hundred different sign languages in use in the world. Sign languages, however, present unique challenges due to their viso-gestual nature and, especially for linguists, their lack of a standard and widespread form of writing.

Often, sign languages are recorded using video, and the meaning is annotated using glosses from the oral language in the same geographic region.

[1] https://www.ucm.es/visse
[2] https://www.ucm.es/signariolse

Fig. 6.1. – SignWriting transcription of the Spanish Sign Language sign for "lie". A video can be seen online at SpreadTheSign: `https://www.spreadthesign.com/es.es/word/22502/mentira/0/?q=mentira`

This is, however, not a proper transcription, since sign languages are natural languages with a grammar and lexicon of their own, and, in order to properly capture them, a native system is needed.

There are a few existing proposals for transcribing sign languages into a written form. Unique among them, SignWriting (Sutton and Frost 2008) transposes the three-dimensional nature of signs into a bi-dimensional arrangement of symbols, as can be seen in Figure 6.1. Different iconic symbols are used to represent the head, hands and other body parts, and their location and movement is recorded in an abstract and systematic manner.

However, the graphical nature of SignWriting means it is very different from the usual writing systems for oral languages, making it harder to process with existing tools and standards. Moreover, while there exist some computational representations, very often SignWriting is shared in the form of images, which do not require special fonts and software installed to be viewed, but are impossible to process as text.

Therefore, to be able to linguistically process SignWriting in its image form, tools and standards are required. If these are to be developed empirically, source data are also needed. A few collections of sign language transcribed using SignWriting exist, but are not research oriented and deal only with the digital representation. Additionally, SignWriting can also be handwritten, and there are no research corpora of handwritten SignWriting that we are aware of.

In this article, we present the VisSE corpus of Spanish SignWriting, a collection of handwritten SignWriting instances representing signs of Spanish Sign language. The instances have been graphically annotated, for which an extensive schema has been developed, recording both the lexical meaning of the different symbols involved as well as their spatial information, a fundamental part of SignWriting.

This corpus can be used to extract information on SignWriting, for research on the processing of graphical languages, for empirical study of the features of sign languages, for the training and evaluation of machine learning methods, or for other research purposes which have not occurred to us. We have used it in our previous and ongoing research, and therefore, believing it may be of use to the research community at large, we have freely released it online (Sevilla, Lahoz-Bengoechea, and Díaz 2022). We will continue to expand and improve it, and this article relates its current state, how it has been built and the annotation schema used.

In Section 6.2, related corpora and tools are discussed, as well as systems for computationally storing SignWriting. Section 6.3 explains the different objects in the corpus and how have they been annotated, while Section 6.4 is centered on the concrete details and computational aspects of its construction. Section 6.5 gives an overview of the data and some statistics, and in Section 6.6 a few conclusions are drawn and future work described.

Due to its complexity, explaining SignWriting is out of the scope of this article, but enough detail will be given to allow the reader to follow the discussion. For more information, interested readers can see the extensive documentation available online at `https://signwriting.org`.

6.2. Related Work

Existing sign language corpora or datasets are usually comprised of videos of utterances, whether isolated signs or phrases. Many are not intended for research, but rather for regular use, and are structured as dictionaries.

Spread the Sign[3] (Hilzensauer and Krammer 2015) and DILSE[4] (Moreno 2012) are two of these dictionaries containing the Sign Language transla-

[3] `https://www.spreadthesign.com`
[4] `https://fundacioncnse-dilse.org`

tions (as video) of words in one or many oral languages. These videos are provided without any phonetic annotation, though DILSE is interesting in that, additionally to the video, it includes static photographs of signers where movement is annotated using superposed arrows, and when needed, different instants of the sign are recorded as consecutive photographs. We see this as an approach halfway to SignWriting, which follows similar principles but in a more abstract and standardized manner.

Other datasets, even if often also intended to be usable as dictionaries, are structured to allow research by examining the data or searching by features instead of just by meaning. LSE-Sign (Gutierrez-Sigut et al. 2016) is a web tool that contains 2400 signs from Spanish Sign Language, annotated with linguistic features to enable searching for concrete characteristics of signs. The signs are stored as videos and glosses, but the annotation is rich, with entries for hand shape and orientation, movement shape, and other features. Other such corpora of sign language videos exist, such as those for Australian Sign Language, British Sign Language, and others, based on the Signbank software (Cassidy et al. 2018).

A common necessity of sign language corpora is a relevant and meaningful annotation of the signs depicted, since video by itself is not computationally processable. To this end, phonological or phonetic transcriptions of signs can be used, but there is not a universally accepted way to represent the movements and gestures of sign language neither formally nor computationally. The forefather of sign language linguistics, William Stokoe, proposed a linear writing system consisting of abstract symbols to encode the different parameters of the language (Stokoe 1960). The Hamburg Notation System (Hanke 2004) uses a similar approach but with a different set of symbols, while Ángel Herrero Blanco, in his study of Spanish Sign Language, developed another featured writing system but using characters from the roman alphabet (Herrero Blanco 2003).

A different approach to sign language transcription can be found in SignWriting (Sutton and Frost 2008), a featural writing system (Galea 2014, pp. 76-77) where abstract symbols are used for representing linguistic features. Their shape is chosen as iconic as possible, helping the reader and writer remember the actual physical articulator the symbol represents. The main difference that SignWriting introduces is that symbols are also arranged iconically, in-

stead of in a linear fashion. The bi-dimensional page is used to represent three-dimensional sign space, and symbols are set on it according to their actual location in the realization of the sign. This enables the writer to capture the spatial richness of sign language almost directly, but is a radical departure from the main paradigm of oral writing systems.

One of the problems this presents is that the common computational representation of oral writing systems, as sequences of individual and mostly independent symbols, is insufficient for representing SignWriting. Nonetheless, there is ongoing effort to solve this problem, and some of SignWriting can be represented using Unicode.

Unicode is a "universal character encoding standard for written characters and text" (The Unicode Consortium 2021) which assigns a number to each possible character in use in a documented human language, so that text can be computationally stored as a sequence of bytes. It includes character points and combinations for many of the symbols in SignWriting, and The International SignWriting Alphabet (Sutton and Slevinski 2010) provides fonts which, when installed in the user's computer, allow for the proper display of the symbols.

However, "the spatial arrangement of the symbols (...) constitutes a higher-level protocol beyond the scope of the Unicode Standard" (The Unicode Consortium 2021, p. 831), meaning Unicode is not enough to fully codify SignWriting. To solve this problem, computational solutions such as Formal SignWriting or SignWriting Markup Language (Rocha Costa and Dimuro 2002; Verdu Perez et al. 2017; Slevinski 2016) often store positional information as numerical coordinates alongside the Unicode bytes, indicating where to place them in bi-dimensional symbol space. Nonetheless, these systems are intended mainly for creation and display of SignWriting, not for its linguistic processing, and thus lack many annotations needed for the fully automatic understanding of the transcriptions.

Compared to video-based corpora of sign language, there are not many databases of sign language which use SignWriting as their representation form. SignPuddle[5] is a dictionary and database of sign language which stores SignWriting using Unicode and storing symbol positions as coordinate pairs.

[5] https://www.signbank.org/signpuddle2.0/

It is a multilingual dictionary, containing entries for many different sign languages across the world, including Spanish Sign Language. The web interface allows searching by word, symbol or searching full signs, and data can be exported for offline processing. However, since it uses the computational systems mentioned before, it does not contain the "higher-level protocol" information identified by the Unicode Standard and needed for decoding the complete meaning of transcriptions.

6.3. Annotation Schema

The VisSE corpus is a collection of handwritten SignWriting instances (images) representing signs or parts of signs from Spanish Sign Language. These instances are annotated both graphically, by demarcating the relevant regions of the image where meaning is codified, and more conventionally using textual tags to codify the meaning and attributes of the different symbols.

6.3.1. Logograms

Each SignWriting instance is called a "logogram" (Slevinski 2016), since it represents units of meaning which are either words or word-like (not necessarily full signs[6]). Although the term logogram is commonly used to emphasize the non-phonetic nature of characters, it is worth noting that there are instances where sub-units may contain phonetic information (Liu et al. 2020). In the case of SignWriting, all of the sub-units are phonetic, conveying the gestures (understood in the broad articulatory sense) necessary to articulate each sign. We call each of these sub-units of writing "graphemes".

Graphemes codify most of the phonetic content of SignWriting, and so they require the most complex annotation, consisting of not a single label but a set of features for each grapheme. The list of graphemes present, along with their feature set, is the core of each logogram's annotation.

However, the meaning of each grapheme is not only determined by its graphical properties, but also by its position relative to the other graphemes in the logogram. It is only after contemplating both each grapheme's features

[6] Whether these are syllables, morphemes, or maybe something else is a question of linguistic research.

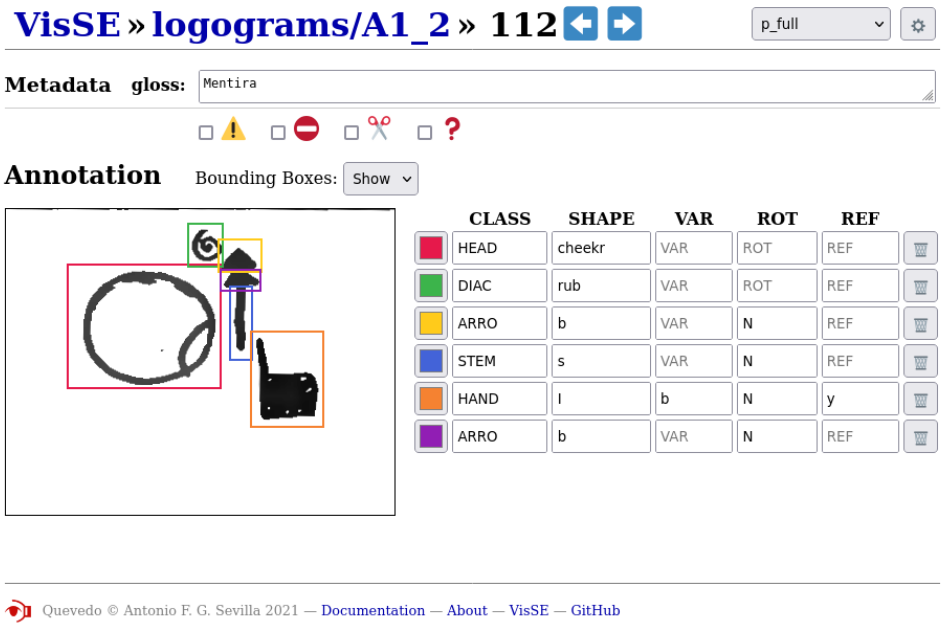


Fig. 6.2. – Visual annotation of the sign "lie/to lie". Superposed over the logogram image, the bounding boxes of the different graphemes are drawn. The color codes serve to match each region to their grapheme, represented to the right as a table of feature names and values.

and their holistic arrangement in the page that can the sign transcribed by a logogram be understood.

Therefore, logogram annotation consists of a list of graphemes with their relative locations, each annotated with their own independent feature set. The locations are annotated as 'bounding boxes', the geometrical regions within the logogram where each of the graphemes can be found. An example of this logogram annotation can be seen in Figure 6.2.

6.3.2. Graphemes

Unlike other writing systems, graphemes in SignWriting are not a one-to-one mapping from a shape or picture to a phoneme or phonemic feature, but rather encode complex meaning in their graphic form and visual properties. They have internal structure, both in their graphical properties (strokes, fill) and in their presentation (rotation, reflection). Each of these properties are

encoded in a set of tags, a mapping from feature names to feature values that stores their independent meaning.

Some of this meaning is lexical, in that the shape of the grapheme must be looked up in a dictionary to understand what it represents. This is mostly the shape and outline of the grapheme's form, which while often iconic and thus intuitive for humans to remember, is in the end conventional and abstract. The rest of the grapheme's meaning is morphological, in that it is derived from the graphical transformation of the grapheme's form. For example, the grapheme may be filled with different patterns of black and white, or drawn rotated around its center.

To properly annotate this meaning, a schema of five different features or tags is used. The first two, the `CLASS` and `SHAPE`, codify the lexical part of the grapheme's meaning, while the rest, namely `VAR`, `ROT` and `REF`, encode the graphical transformations. As we will see, not all graphemes can be transformed in the same way or at all, so not all graphemes use the same set of features. Which features need to be used is determined by the first tag, the `CLASS`, which separates graphemes into different groups according to their visual characteristics (mostly size and variability) as well as their transformation possibilities. We have settled in six classes for our corpus: `HEAD`, `DIAC`, `HAND`, `ARRO`, `STEM` and `ARC`.

Once the `CLASS` is determined, the `SHAPE` completes the lexical meaning by refining the classification down to what a user of SignWriting might actually identify as a 'character'. For example, a concrete hand shape, a symbol for a head or a contact mark. The rationale for the grouping and annotation of the different classes, as well as any further tags needed for any of them, are covered in the following.

`HEAD` and `DIAC` have the simplest annotation, so are explained together in 6.3.2, while `HAND` graphemes are the most complex and Section 6.3.2 is fully dedicated to them. `ARRO`, `STEM` and `ARC` are grapheme classes used to annotate the significant components of movement markers, and so are described together in Section 6.3.2.

Fig. 6.3. – Three HEAD graphemes, representing different parts of the head as place of articulation. The first is SHAPE=chin; the second SHAPE=mouth, and the third SHAPE=smile. This third grapheme represents not only the head as a body part, but also the "smiling" facial expression, which can be semantically relevant in sign languages.

Invariant graphemes

Two first classes of graphemes are HEAD and DIAC. These groups of graphemes do not transform, so are always presented with the same picture, and the SHAPE feature is enough to discern their independent meaning. They are separated into two classes mostly due to their graphical characteristics. HEAD graphemes are big and sparse, while DIAC graphemes are small and compact. Nonetheless, they also have different characteristics in how they contribute to the meaning of a logogram.

HEAD graphemes, by depicting the head or some of its parts (eyes, nose, etc), establish a place of articulation, and the location of other graphemes is decided relative to them. Additionally, iconic representations of the eyes, mouth, and other elements can be used to transcribe facial expressions, an important non-manual parameter of sign languages. Some examples of HEAD graphemes can be seen in Figure 6.3.

DIAC graphemes act more like diacritics, modifying the meaning of nearby graphemes in some predictable way—hence the name, though no profound thinking has been given to whether they actually count as traditional diacritics. Some examples of DIAC graphemes are dynamic marks, which establish the coordination of the hands, or the velocity of the signing, thus affecting the whole logogram and having a mostly arbitrary place within it. Other marks, such as internal movements of the hand, or contact markers, must be placed nearby the graphemes which they modify, though there are no hard rules as to where exactly.

In Figure 6.4 some example DIAC samples can be seen, and their use in combination with other graphemes in the logogram.

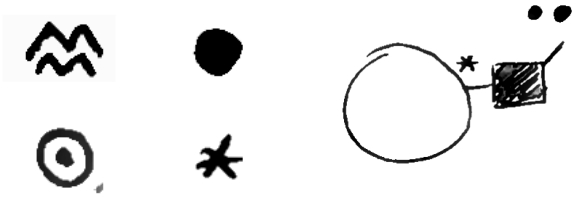


Fig. 6.4. – On the left some DIAC graphemes from the corpus are shown. Clockwise from top left, the SHAPEs are: wiggle, flex hook, touch and brush. On the right, the logogram for the sign 'left-handed' is shown. The touch grapheme marks that the hand should be in contact with the head, while the top two flex hook graphemes specify that the little finger must bend twice in the "hooking" manner (they are above a HAND grapheme, explained in the following).

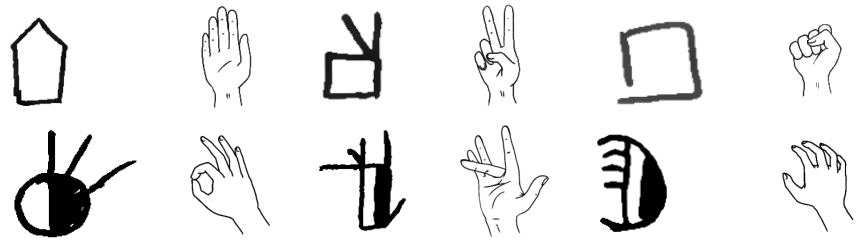
Hand graphemes

Hands are the most prominent articulators of Sign Language, and have many degrees of freedom and articulatory possibilities. Different authors assign different features to signs, but some commonalities can be found. The "hand shape" or configuration is a feature that accounts for the articulatory possibilities of the fingers, i.e. how are they bent to produce a unitary and meaningful shape; orientation is a feature specifying the rotation of the hands as 3D objects in sign space.

In SignWriting, each different hand shape is assigned a picture, a combination of strokes that iconically represents the hand and the fingers. This basic picture can have different filling patterns of black and white, and can also be rotated or reflected. These different attributes are annotated separately in our corpus, to represent their combinatory possibilities. Hands are grouped under the CLASS=HAND, and present the most complex annotation, requiring all the features available in the corpus for their annotation.

The outline of the character, which SignWriting uses to represent the hand shape, is annotated in the tag SHAPE. This roughly corresponds to the sign language parameter of hand configuration, and therefore a suitable linguistic notation system can be used to transcribe it. Some different notation systems exist for hand shapes, varying in their applicability to different sign languages and their ease of use. We use our own notation system, somewhat similar to

Tab. 6.1. – Some SignWriting hand grapheme samples from the corpus along with the hand shape they represent.



that of (Eccarius and Brentari 2008), but specific for Spanish Sign Language. Some examples of hand shapes can be seen in Table 6.1.

These basic "forms" for hand graphemes can suffer a number of graphical alterations in SignWriting, used to transcribe the hand as a three dimensional object in the flat page. The hand grapheme can be filled with three different patterns: full white, full black or half and half. Then, the grapheme can be rotated using a set of eight possible different angles. These two transformations encode the orientation of the hand, and are annotated in our corpus in the VAR and ROT features.

A final graphical transformation allows the hand graphemes to be reflected across their longitudinal axis, turning them into their mirror image. This transformation is used by SignWriting to better iconically depict the hand as would be seen by the signer, representing the fingers in their correct position across the hand and also being useful to represent left hands (which are mirror images of the right hand). This reflection is coded in our corpus in the REF feature, using a yes/no value.

To decide whether a hand grapheme is reflected or not, however, is not as straightforward as it may seem. Without the wider context, it is impossible to predict whether a grapheme is an unreflected left hand or a reflected right one. For example, in Table 6.2, the fourth grapheme could be either a right or a left hand, we only know that the palm is looking left. Conversely, the fifth grapheme does not tell us, without wider context, whether it is a left or right hand, but only that the palm is oriented to the right. This wider context might be two hands, side by side, or a nearby body part, allowing humans to

Tab. 6.2. – Hand graphemes and their transformational annotation. The label values have been expanded from the actual abbreviatures used in the corpus to be more informative.

						
VAR	white	white	white	half	half	black
ROT	North	South	NorthEast	North	North	North
REF	no	no	yes	no	yes	yes

deduce which hand is depicted. This is inherently ambiguous, however, and requires understanding of the human body and sign language phonotactics.

We choose to optimize graphical stability, meaning graphemes which are similar in features should also be similar graphically. To this effect we choose the `half` variant as the guide for whether graphemes are reflected or not (choosing as non-reflected the one with the palm to the left), and base the `REF` feature for white or black variants on their graphical similarity to the half one.

More examples of hand grapheme alterations can also be seen in Table 6.2. For the complete enumeration and explanation of tag values, please refer to the annotation guide that can be found in the corpus (in English and Spanish, available inside the corpus distribution file or directly at `https://zenodo.org/record/6337885`).

Movement marks

Hand movements are an integral part of sign language, and therefore a substantial part of SignWriting. They are codified with paths and arrows depicting the 3D movements of the hands in the page, describing the shape of the movement by drawing it in an intuitive way. To properly encode 3D space in 2D writing, they use graphical attributes to distinguish between planes of movement, similar to how `HAND` graphemes have variations to represent different palm orientations. Some examples of movements in the corpus can be seen in Figure 6.5.

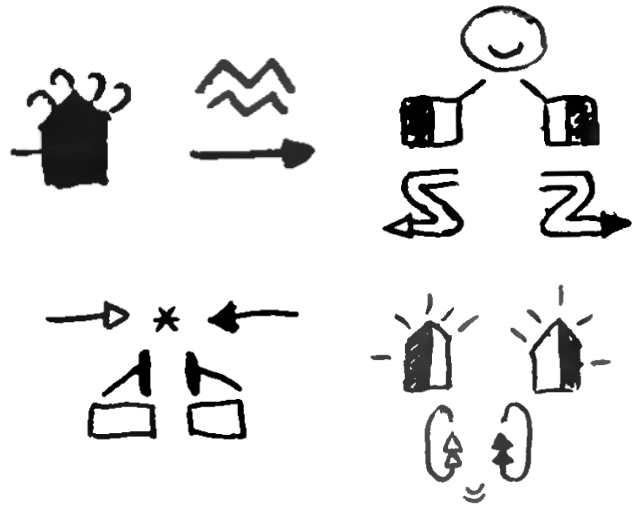


Fig. 6.5. – Some logograms in the corpus which include movements of the hand. On the top left ("fingerspelling"), the hand moves to the right while the fingers "wiggle" (a `DIAC`). Next, top right ("happy"), the hands simultaneously do parallel zigzagging downward movements. On the bottom left ("together"), the hands move from the sides to the center until they touch. Finally, to the right ("Sign Language"), the hands make repeated circular movements in the vertical plane.

What constitutes a grapheme in movement markers, however, is not an easy decision. As with other elements of SignWriting, movement symbols use both symbolic graphical properties (strokes, filling, shapes) as well as location in the page to record information. The shape of sign language movement is directly and iconically converted into bi-dimensional trajectories, with a few graphical attributes to bridge the gap to the extra dimension.

One approach would be to understand the full trajectory and associated symbols of the movement as a unit. This is the approach used by the Unicode standard and the International SignWriting Alphabet, even if some of the features, such as rotation, are encoded as different codepoints forming combining characters. Indeed, there are tens of thousands of glyphs in the International SignWriting Alphabet fonts to try and account for as many possible movements as possible, a small sample of which can be seen in Figure 6.6. Encoding movements holistically is, however, problematic for annotation due to the sheer number of them, and there is arguably a loss of information incurred when transforming a visual, meaningful, spatial representation

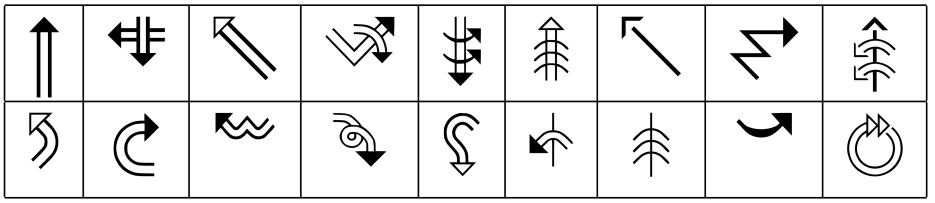


Fig. 6.6. – Small sample of characters that can be found in the Sutton SignWriting fonts to represent different movements of the hand.

into an index in a table (the lost information must be looked up in the table, instead of being directly available). Additionally, dealing with handwritten SignWriting renders this approach even more difficult, since it is not guaranteed that transcriptions will use only the abstractions that are collected in the Unicode representation.

Instead, we have opted to characterize the different elements of movement markers as individual graphemes. From the lexical annotation point of view, this makes sense because there are repeated elements, with identifiable semantics of their own, which can be used in different context. While these elements could be transcribed as different tags for the same grapheme, as is done for hands, their independent spatial characteristics make a subdividing approach more useful.

In Figure 6.6 we can see that arrow heads are repeating elements, as well as straight path segments, arcs and circles. Since each of them has a distinctive meaning (black arrow heads represent right hand movements, white ones left hand movements, double stemmed movements occur in the vertical plane, etc.) and there are movement marks which are distinguished only by the presence or absence of one of them, it is reasonable to think of each independent segment as individual graphemes.

Therefore, we have settled on three different classes of movement graphemes: arrow heads `ARRO`, straight segments `STEM` and curved segments `ARC`. The `SHAPE` tag for each of them distinguishes between the few different variants in the `CLASS`: in the case of `ARRO`, the color of the arrow head marks which hands move, and is encoded in the `SHAPE`. `STEM` and `ARC` can occur in different planes of movement, distinguished by their stems, so this is annotated in the `SHAPE`. For `ARC`s, the `SHAPE` additionally stores the amplitude

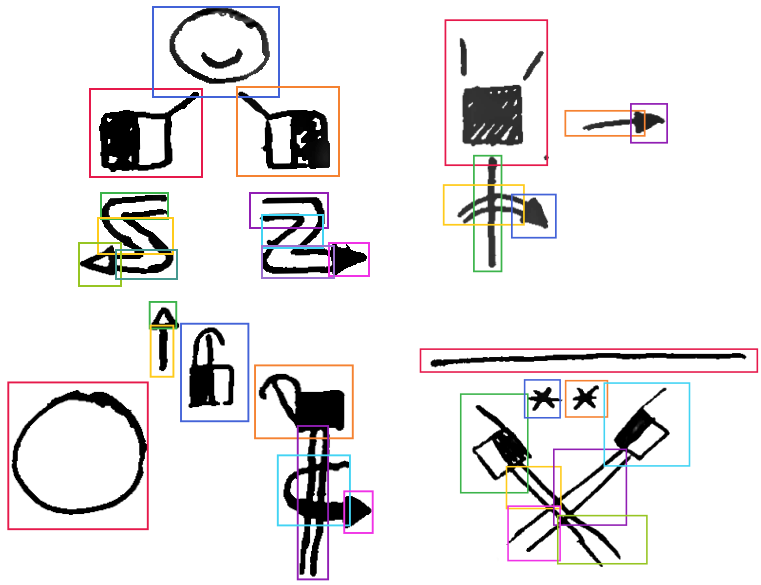


Fig. 6.7. – Annotation of sub-segment bounding boxes for movement and forearm markers in some logograms from the corpus. The signs, from the top left and clockwise are “happy”, “bridge”, “November”, and “Zaragoza”.

of the movement, since curved movements can be shorter or longer arcs or even full circumferences.

Additionally to the SHAPE, all movement graphemes can be rotated to convey orientation in 3D space. This is annotated in the ROT tag, as is done for hands, and following the same notation using cardinal directions[7]. The REF tag is not needed thanks to subdividing movements into segments, since each individual segment is symmetric. If movement markers were annotated as a whole, annotating reflection would be necessary for ARCs to distinguish clockwise and anti-clockwise orientations, but with our approach it is not necessary (it is marked by the orientation of the arrow head).

Some examples of this path subdivision can be seen in Figure 6.7.

The subdivision approach also helps with marking the bounding boxes for movement graphemes. Movement paths are very graphically sparse, the actual strokes often occupying a small part of the whole rectangular region they occupy. The bounding box of the full path is not very informative either,

[7]For the specific details please see the annotation guide of the corpus.

not being able to distinguish the actual shape of the paths or their direction. This could be annotated a posteriori, but it seems a waste of effort to try to reconstruct a geometric meaning by abstract terms when the description is there, on the page, using geometric features which we can spatially annotate.

Another instance where the segmenting approach helps is with crossings and overlapping graphemes. It is very common for movement markers to overlap each other. Sometimes this is not a problem, for example a small ARC crossing over a long STEM, as is done to indicate forearm rotation. Their geometric properties are distinct enough that spatial annotation can be reasonably performed, their bounding boxes in a distinguishable and characteristic arrangement even if there is much overlap (see Figure 6.7, top right). However, when there is a diagonal crossing of STEMs, as in Figure 6.7 (bottom right), the bounding boxes overlap so much that they become meaningless, impeding grapheme discrimination. In this case, the paths are subdivided into consecutive sub-segments, forming a meaningful arrangement of movement graphemes which can be annotated and processed.

Despite the advantages we have explained, subdividing the movement markers present one small problem. With this schema, some markers used in SignWriting to represent body parts like shoulders, the waistline, or forearms become indistinguishable from STEMs. Indeed, in the case of forearms, this similarity is not a coincidence, since they have the same features and behaviours of movement paths. They can be double or single, depending on whether the forearm is vertical or horizontal, and they can have overlapping ARCs to represent forearm rotation. The only difference is that these "STEM"s are not topped by an arrow head. Fortunately, this is also the solution to our problem. Shoulders, forearms etc. are annotated as STEMs, and their actual meaning is contextual, depending on the presence or absence of actual ARRO graphemes in their extremes. While this pushes interpretation of graphemes' meaning to a further layer of processing which can take context into account, this is already a feature of SignWriting and our annotation, so this decision maintains coherence and makes the corpus consistent.

6.4. Corpus Construction

The VisSE corpus was built as part of the equally named VisSE project[8] ("Visualizando la SignoEscritura", "Visualizing SignWriting" in Spanish) which had the goal of improving the use and access of SignWriting in digital contexts. To form the base for further study as well as training samples for machine learning development, around 1950 instances of raw SignWriting were collected. These samples had been produced by Dr. José María Lahoz-Bengoechea during a span of years while learning Spanish Sign Language at Universidad Complutense de Madrid, and were originally a tool for his private study.

The samples were handwritten with pen or pencil, and collected in vocabulary sheets. As part of the project, they were digitized, separated into different files for each entry, and graphically enhanced to reduce scanning artifacts and other noise. During this process, a reference to the original vocabulary entry was kept, and remains as a tentative gloss (in Spanish) for logograms in the corpus.

The logograms are collected in subsets according to the academic level at which they were collected. This has no further intended meaning than being a way to organize the corpus. Nonetheless, due to the temporal separation of the records, this organisation results in some greater graphical and usage consistency within each of the subsets. This has let us do annotation incrementally, learning from each phase and improving the annotation schema each time a new set was added. In the current release of the corpus, not all of the original logograms are present, but only those which have been annotated and revised. These are sets `A1_1`, `A1_2`, `A1_3`, `A2` and `B2_2`, which include 1146 annotations.

Apart from learning from the annotation process and improving it for the following subset, this incremental approach allowed us to use a bootstrapping approach to annotation. Once the first set had been fully manually annotated, machine learning algorithms were trained on it and used to perform a preliminary annotation of the next subset. The resulting annotations had to be checked and corrected manually, but the process was somewhat faster. Some graphemes are easy to detect for the machine learning algorithms,

[8] `https://www.ucm.es/visse`

meaning the human annotators could focus on the more difficult parts. As the algorithms improved, the speedup was evident, and some of the tasks could be somewhat automatized, like the drawing of bounding boxes for each grapheme. Readers interested in the machine learning aspect of our research can find more information in Chapter 8 (Sevilla, Díaz, and Lahoz-Bengoechea 2023).

6.4.1. Quevedo

The process of collecting, organizing, annotating, and performing machine learning on the data was a complex one, compounded by the fact that we were developing the annotation schema in parallel to the actual annotation of the data. Moreover, our annotations are complex and very specific, including both visual annotation of logograms and a multi-feature annotation of graphemes. To deal with this complexity and the specific requirements of our task, a specialized tool was developed as part of the VisSE project, named Quevedo[9]. Therefore, computational access to the corpus and its features is easiest when using Quevedo, and the on-disk format and organization of the corpus is as a Quevedo dataset.

Quevedo is available on the Python Package Index, so installing it can be done with the command `python3 -m pip install quevedo[web]` if Python and Pip are available. This will also install the web interface, which can be launched with `quevedo web` at the corpus root directory, allowing visual inspection of the logograms and their annotation.

The features of Quevedo and the format on disk are all explained in the online documentation, but are also briefly detailed in the following for parties who want to use the corpus with other tools or need access to the low-level details. All formats are open and standard, so all the data and features in the corpus can be thus accessed.

6.4.2. Computational representation and access

Logograms in the corpus are stored in the `logograms` directory, in subdirectories representing each of the subsets. Files are sequentially numbered,

[9] Available at `https://github.com/agarsev/quevedo`. An article detailing its features and internal working can be found in (Sevilla, Díaz, and Lahoz-Bengoechea 2022).

starting from 1, and each instance consists of two files. The source image is named with the index of the annotation plus file extension `.png`, and the annotation data uses the same filename (the index) but with `.json` extension. For example, the annotation data corresponding to image `logograms/A1_1/1.png` can be found in the file `logograms/A1_1/1.json`.

The json annotation file is a dictionary of attributes, among which there is a `graphemes` key containing an array of the different graphemes found in the logogram. Each of them is a dictionary as well, having a `box` key with the coordinates of the bounding box, and a `tags` key which is another dictionary representing the mapping from feature names to feature values.

The coordinates of the bounding boxes are 4-tuples of floating point numbers, in the format (cx, cy, w, h). (cx, cy) are the coordinates of the center of the box relative to the logogram, which range from 0 to 1, $(0, 0)$ being the top left corner, and $(1, 1)$ the bottom right one. (w, h) are the width and height of the grapheme region, again relative to the width and height of the logogram, so ranging from 0 to 1. The grapheme tags are stored as strings of characters both for feature names and value.

Aside from the `graphemes` key, logogram annotation files include some other information used by Quevedo. The `meta` key stores a dictionary of additional metadata keys for the logogram, where the original gloss for the logogram can be found, as well as some boolean `flags` which we have used to mark and exclude a few problematic graphemes.

There is also a `fold` key which stores a number, the index of the fold to which the annotation belongs. Folds can be used to split the data in the corpus along a different dimension from that of the subsets, which can be useful, for example, for logically partitioning the data into training and evaluation sets. Storing this split into the annotations as a fold number helps make experiments reproducible and sound and results comparable. Each logogram is assigned a number ranging from 0 to 9, and this numbers split the corpus data into 10 approximately equally-sized folds. In our experiments we use folds 0-7 for training, and 8-9 for testing.

An example annotation file can be seen in Listing 6.1 at the end of the article.

6.4.3. Other files

Inside the corpus root directory there are a number of other files and directories not mentioned above. Especially relevant is the `networks` directory, where the weights of neural networks trained with the corpus data are stored. These are included with the corpus so that interested parties can reproduce some of our experiments and pipelines without having to train the algorithms themselves. Visual testing of the networks and pipelines is also possible using the Quevedo web interface.

In the `scripts` directory, some utility Python scripts are also included, and some researchers might find them useful. A `dvc.yaml` file can also be found in the root directory, for use with DVC (Kuprieiev et al. 2021). This file can be used to run some common tasks with the data from the corpus. Relevant here may be the `extract` step, which will extract all graphemes from the logograms and turn them into their own annotation files in a `graphemes` directory, allowing them to be processed independently of the rest of the logogram.

For more information on the dataset structure or other files, please refer to Quevedo's documentation at `https://agarsev.github.io/quevedo`, or to our other articles.

6.5. Data Description

There are 1146 annotations in the corpus, 982 of which are fully annotated logograms. The rest of the annotations are marked using the `exclude` flag, most of them being long transcriptions of polysyllabic signs rather than single logograms. These have been split into two (or more) independent logograms, but the original annotations are also included for reference. Some other transcriptions, marked with the flag `problem`, present some kind of graphical or representational problem, which has led us to exclude them for now from annotation, but are kept in the corpus.

Within the 982 fully annotated logograms, 6060 different graphemes can be found. Of these, 330 belong to the `HEAD` class, 1047 to `DIAC`, 1649 to `HAND`, 1369 to `ARRO`, 1292 to `STEM` and 373 to `ARC`. In Table 6.3, these numbers can be compared to the number of different `SHAPE`s that can be found for each

Tab. 6.3. – Counts of observations in the corpus by CLASS. Unique observations refer to those which share the same set of features. The rate of occurrence is the number of observations divided by the total number of logograms, measuring how likely a grapheme class is to appear in a logogram.

CLASS	Graphemes	Unique Observations	SHAPEs	Appearance Rate
HAND	1649	560	72	1.68
ARRO	1369	23	3	1.39
STEM	1292	15	2	1.32
DIAC	1047	19	19	1.07
ARC	373	37	6	0.38
HEAD	330	20	20	0.34
total	6060	674	122	6.17

CLASS. As can be seen, the proportions are very different, meaning that some classes, like ARRO or DIAC have only a few different possible grapheme SHAPEs but are abundantly represented in the data, while other classes like HAND or ARC are less abundant compared to the variability within the class.

Examining the rate of appearance of grapheme classes per logogram, we can also make some interesting observations. The average amount of graphemes per logogram is about 6, 1.68 of which are hands. Unfortunately, we cannot distinguish between bimanual signs and transcriptions where a transformation in handshape is encoded, but further, semantic annotation would make this clear. Easier to compute is the complexity of movement paths. Since most movements are marked with an arrow head, the ratio of segments (STEM and ARC) to arrow heads (ARRO) can give us an approximate measure of the mean number of segments for paths: $\frac{1292+373}{1369} \approx 1.22$. This means that most movement markers are simple, with just one stem segment, but a non-trivial amount (approximately one every five) is more complex, having two or more segments.

If we examine the distribution of tag combinations, however, we can see a very skewed distribution, as is depicted in Figure 6.8. Some graphemes are very common, while many combinations are rare, forming a very long tail of infrequent graphemes. This also happens if we just look at the SHAPE feature, and across classes, as can be seen in Figure 6.9.

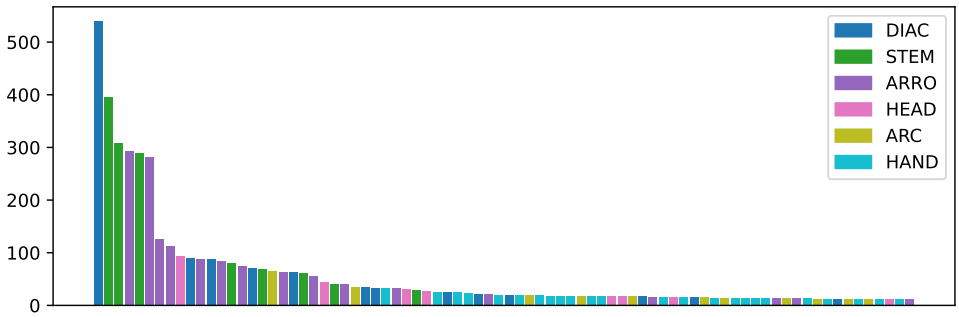


Fig. 6.8. – Distribution of unique tag combinations in the corpus. The most common 80 are arranged in the horizontal axis (not labeled for clarity), while the vertical axis represents the number of times that unique combination appears in the corpus. The bars are also color-coded by CLASS. The plot would extend to the right more than 6 times, making the long tail even longer and thinner.

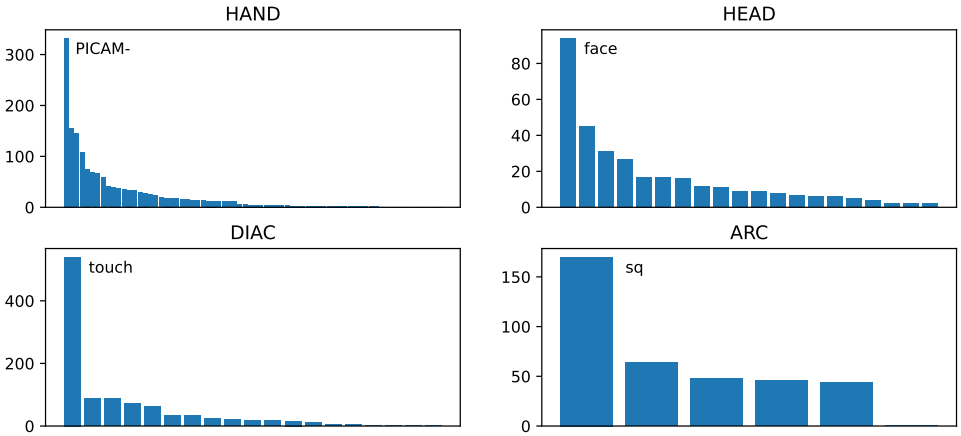


Fig. 6.9. – Distribution of shapes for some classes. The horizontal axis represents the shapes (not labeled for clarity), the vertical axis is the number of times that shape appears in the corpus. The most common CLASS is labeled near its bar for reference.

Since this is not a corpus of real utterances or texts, but rather a vocabulary, conclusions cannot be directly inferred about the frequency of elements in SignWriting in use. However, we can make observations across the vocabulary of the transcribed sign language, seeing that some particular gestures (understood in the broad linguistic sense) are much more common than others. For example, the "touch" grapheme is extremely common, as well as the `PICAM-` hand shape —fingers extended but together, which acts as the "flat" object descriptor in LSE (top left in Table 6.1).

6.6. Conclusion

SignWriting is a featureful writing system used to transcribe sign languages. Its power as a faithful phonetic representation of signs, both in the lexical and the spatial domains, means it can be a useful tool for empirical linguistic research. For this to work, examples and empirical processing methodologies need to be developed and tested.

To this end, we have built the VisSE Corpus of Spanish SignWriting, a collection of handwritten SignWriting logograms which can be used for automatic processing or linguistic analysis of the data. The corpus is freely available online, and in this article we have presented its annotation schema and construction, and we have performed a brief analysis of the data it includes. We have seen that processing SignWriting requires careful analysis, and believe our work may be useful for the research community both in how we have proceeded as well as in the end result, the published corpus.

As an example of the insight that can be gained from having data and annotating it, we have observed in our research some similarities between SignWriting and oral language writing systems, and some meaningful differences. This similarity and differences parallel the relation of sign languages with oral languages: where sign languages are similar to oral ones, SignWriting resembles oral writing. Both differ to their oral counterparts in the same way: use of space to convey meaning and syntactic relations.

Another insight can be found in how to think of SignWriting. Due to the complexity of sign languages at the phonetic level, SignWriting can be a powerful and flexible tool for dealing with them, thanks to its very detailed and

phonetically precise nature. However, it requires a mental and computational model more complex than that of other writing systems. To deal with this complexity, we think it is useful to think of SignWriting as a language in itself, even if a graphical one.

This mental model can be seen in our classification of graphemes in hierarchies, in which if we think of SignWriting as a language, the `CLASS` tags might be parallel to parts of speech, and `SHAPE` to particular words. The rest of the annotation, using sets of features, might somewhat resemble the morphological features of words. In the corpus and in this article not much attention has been paid to these semantic properties of graphemes, since we have focused on their descriptive annotation, but we want to note that this similarity of SignWriting elements to words in an oral language, with their syntactic and morphological properties, is not only found in their appearance, but also in their interpretation. Properly extracting this interpretation is a task which remains as future work.

On the other hand, just as space and movement are still unresolved issues for sign language theoretical description, annotation of spatial properties of SignWriting graphemes has been a challenge in itself. We could not base ourselves in any widespread linguistic standard or know-how, since these characteristics are not found in oral languages. We have seen that some of the graphical attributes of graphemes are similar to morphological derivational processes, manifesting as rotations and reflections of characters rather than their insertion or deletion. But others, intrinsically locative, require numerical annotation of positions and regions, and subdivision of paths into smaller elements.

We believe this insight might not be only useful for the computational treatment of SignWriting, but may also mirror some of the problems of computational treatment of sign language per se, and may inspire solutions or ideas in that space.

Moreover, as SignWriting elements map very well to the phonetic features of sign languages, especially when annotated in detail like is done in the VisSE corpus, we think that the study of a corpus of SignWriting can very well be useful to draw conclusions about the sign language it transcribes. This can help make data-based linguistic research on sign languages less costly, as collection of video corpora requires much time, face-to-face collaboration of

native informants, and attention to issues such as privacy and distribution of the videos. Linguistic annotation of video is difficult and essentially a manual process, while annotating SignWriting can be faster with the right tools, or can be even done with hand written samples as we show in this corpus, and using the machine learning algorithms trained on it.

6.7. Future Work

While the presented corpus is currently usable (as demonstrated in our real use-case application described in Sevilla, Díaz, and Lahoz-Bengoechea 2023), it represents only an initial step in its development.

There remains a significant portion of the corpus to be annotated, and this further annotation might uncover issues that require some revision of the annotation schema.

Additionally to expanding the coverage of the annotation, the data collected can be augmented along two axes. Collecting data from more informants will make the corpus more robust and representative of actual SignWriting. Collecting data from continuous, real-world usage, instead of isolated examples, may also further this goal.

On a different note, these last years have witnessed significant advancements in deep learning approaches. New, state-of-the-art algorithms may be able to tell us more about our collected data, expedite the annotation process, or maybe even open up avenues for exploitation which we had not thought about before.

Finally, it is our hope that the public availability of the corpus will invite other researchers to contribute to this future work or add their own insights. The involvement of a broader research community will surely lead to a richer understanding of SignWriting or even sign languages themselves.

List. 6.1. – Simplified JSON annotation file for the transcription in Figure 6.2.

```
{
  "meta": {
    "glosa": "Mentira"
  },
```

```
  "fold": 8,
  "graphemes": [
    {
      "tags": {
        "CLASS": "HEAD", "SHAPE": "cheekr"
      },
      "box": [ 0.3561, 0.3867, 0.3992, 0.4156 ]
    },
    {
      "tags": {
        "CLASS": "DIAC", "SHAPE": "rub"
      },
      "box": [ 0.5134, 0.1207, 0.0944, 0.1507 ]
    },
    {
      "tags": {
        "CLASS": "ARRO", "SHAPE": "b",
        "ROT": "N"
      },
      "box": [ 0.6032, 0.1532, 0.1159, 0.1145 ]
    },
    {
      "tags": {
        "CLASS": "STEM", "SHAPE": "s",
        "ROT": "N"
      },
      "box": [ 0.6038, 0.3787, 0.0619, 0.25 ]
    },
    {
      "tags": {
        "CLASS": "HAND", "SHAPE": "I", "VAR": "b",
        "REF": "y", "ROT": "N"
      },
      "box": [ 0.7224, 0.5644, 0.1913, 0.3229 ]
    },
    {
      "tags": {
        "CLASS": "ARRO", "SHAPE": "b",
        "ROT": "N"
      },
      "box": [ 0.6025, 0.2363, 0.1078, 0.0763 ]
    }
  ]
}
```

7. VisSE Corpus Annotation Guide

The VisSE corpus is published at https://doi.org/10.5281/zenodo.6337885, and a guide to the data annotation is also included. This guide serves as a normative document for annotators and contributors, but also as an explanation and listing of the different tags and values used. A reproduction of the guide, as part of the contributions of this thesis, is included here.

7.1. Introduction

SignWriting is a writing system for sign languages (Sutton and Frost 2008). It uses the graphical possibilities of the bidimensional page to encode the visual characteristics of movement and space used in sign languages, so it is very different to other writing systems, especially in its non-linear nature.

The VisSE corpus is a collection of SignWriting instances, annotated graphically and semantically to capture all the meaning, both conventional and visual, of SignWriting. The samples are all handwritten, and codify signs or parts of signs from Spanish Sign Language. The samples were collected by Dr. José María Lahoz-Bengoechea during a span of years while learning Spanish Sign Language at Universidad Complutense de Madrid, and cover a wide range of vocabulary. However, since they were originally a tool for his private study and not for research, there may be minor errors and inconsistencies in the transcription. They should not be viewed as a collection of Spanish Sign Language Signs, but rather as a collection of SignWriting examples. Nonetheless, a tentative "gloss" (in Spanish) is included for most signs.

Due to the special nature of SignWriting, a tailored annotation schema is needed for its proper codification, and this is reflected both in the digital format of the corpus and in the logical structure of the annotations. This

document is a guide for the later as well as a full and normative specification of it. The logical structure and possible values for the annotations are given, as well as some motivation or explanations when needed. For more detail on this, please see our forthcoming article "Building the VisSE corpus of Spanish SignWriting".

It is not within the scope of this document to explain SignWriting. Interested readers can see the excellent and extensive documentation available online at `https://signwriting.org/`.

7.1.1. About examples

In this document, glyphs from the official Sutton SignWriting digital fonts are used to present examples for the different graphemes. This is done for two main reasons. On one hand, it is useful to present a "standard" and abstract example in each case and not choose any actual handwritten grapheme from the corpus as a "better" example. On the other hand, it may help users knowledgeable of SignWriting understand the schema even if some of the handwriting in the corpus is inconsistent or non standard.

It does not mean that only graphemes identical to the digital glyph will be tagged as such. SignWriting is very complicated, and there are variations available for many of its symbols. Additionally, handwriting tends to be less rigid than digital fonts, so variation is to be expected. Finally, some graphemes which are different symbols in the SignWriting specification have been tagged with the same label in this corpus. This is a conscious decision based on the phonology and the meaning of the graphemes in use in Spanish Sign Language, rather than the accurate and detailed phonetic description that SignWriting provides.

7.1.2. Annotation schema

SignWriting transcriptions are arrangements of symbols in a 2D space that represent the configuration of the hands, their orientation and movement, and other relevant features of signs in Sign Language. Each transcription can represent a single sign, or part of it. We call each independent arrangement of symbols a "logogram", and we call each of the different symbols or graphical components a "grapheme".

Graphemes have internal structure, both in their graphical properties (strokes, fill) and in their presentation (rotation, reflection). We encode these properties in a set of tags for each grapheme, a mapping of feature names to feature values that stores their meaning 'in isolation'.

However, the meaning of each grapheme is not only determined by its graphical properties, but also by its position relative to the other graphemes in the logogram. This is codified for each grapheme with a 'bounding box', a square region within the logogram within which the grapheme can be found.

7.1.3. Grapheme tags

Not all classes of grapheme require the same number of features to annotate them. For this reason, graphemes are first roughly classified into six groups: `HEAD`, `DIAC`, `HAND`, `ARRO`, `STEM` and `ARC`. This is stored in the `CLASS` feature, common to all graphemes in the corpus. A more detailed classification of graphemes is then stored in the `SHAPE` feature, which is also common to all graphemes. The `CLASS+SHAPE` combination establishes the full 'lexical' meaning of the grapheme, but some grapheme `CLASS`es can have additional `VAR`, `ROT` and `REF` features which encode the rest of its 'spatial' meaning.

Each different `CLASS` in the corpus has a section in this guide, explaining the possible values of `SHAPE`, as well as any further tags needed to annotate them. In the case of `HAND`s, due to their complexity, a full chapter is dedicated to their annotation.

7.1.4. Bounding boxes

The location of each grapheme within the logogram is stored alongside its tags in the 'bounding box' attribute. This is a 4-tuple of floating point numbers, in the format (cx, cy, w, h). (cx, cy) are the coordinates of the center of the box relative to the logogram. The coordinates of the logogram go from 0 to 1, $(0, 0)$ being the top left corner, and $(1, 1)$ the bottom right one. (w, h) are the width and height of the grapheme region, again relative to the width and height of the logogram (so ranging from 0 to 1).

To visualize and create bounding boxes, a graphical tool is needed. Quevedo web interface provides such a tool, and shows the boxes as in Figure 7.1. Boxes should cover the full graphical extent of the grapheme, preferably with some

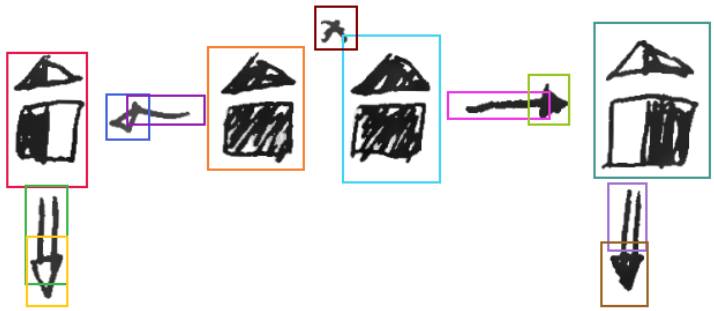


Fig. 7.1. – Graphical example of general bounding box annotation.

padding around it. The exact location of the boxes is not important, but rather the general relative colocation of graphemes, as well as the full area covered by the grapheme's symbol.

7.1.5. Programmatic access

This corpus is formatted as a Quevedo dataset, so the easier way to access the annotations is using the Quevedo library or command line tool. Since annotations have an important graphical component, Quevedo's web interface can be especially useful in their visualization. To install Quevedo with the web interface, the command `pip install quevedo[web]` can be used in any system with python and pip installed.

Nonetheless, the annotations are stored in an open and standard format, so they can also be manually inspected or consumed by other tools.

7.1.6. Format on disk

Logograms in the corpus are stored in the `logograms` directory. They are split into subsets according to when they were collected, but subset structure is more organizational than semantically relevant. In each subset directory, `logograms/A1_1` for example, logograms are sequentially numbered starting from the number 1.

Each logogram instance in the corpus consists of two files. First, the source image, with file extension `.png`, and then the annotation itself, with the same

name but extension `.json`. For example, the annotation data corresponding to image `logograms/A1_1/1.png` is in file `logograms/A1_1/1.json`.

The `json` annotation file is a dictionary of attributes. It has a `graphemes` key, an array of the different graphemes found in the logogram, each of them having a `box` key with the bounding box coordinates (an array) and a `tags` key (a dictionary).

7.1.7. Other corpus objects

Inside the corpus root directory there are a number of other directories not mentioned above. The `graphemes` directory stores isolated graphemes samples. There are currently no such samples in the corpus, but they can be automatically extracted from the logograms. The `networks` directory stores the weights for neural networks trained with the corpus data, and useful code for processing the dataset can be found in the `scripts` directory. For more information on the dataset structure, please refer to Quevedo's documentation at `https://agarsev.github.io/quevedo`. For information on the machine learning artifacts, please refer to Chapter 8 (Sevilla, Díaz, and Lahoz-Bengoechea 2023).

The rest of this document covers the annotation schema for the different grapheme classes.

List. 7.1. – Example json annotation file.

```
{
  "graphemes": [
    {
      "tags": {
        "CLASS": "HAND", "SHAPE": "PICAM-",
        "VAR": "hb", "ROT": "N", "REF": "n"
      },
      "box": [ 0.3160, 0.4671, 0.3218, 0.6927 ],
      // ...
    },
    {
      "tags": {
        "CLASS": "ARRO", "SHAPE": "b", "ROT": "N"
      },
```

```
      "box": [ 0.5404, 0.1947, 0.1441, 0.1434 ],
      // ...
    },
    // ...
  ],
  "meta": {
    "gloss": "Carrera universitaria",
    // ...
  },
  // ...
}
```

7.2. Invariant Graphemes

Some graphemes of SignWriting are always represented upright and in the same orientation, namely HEAD and DIAC graphemes. In the case of HEAD, the body is normally assumed to be upright while signing, so it is drawn as such. When there are alterations to this principle, such as head movement, they are indicated with additional symbols rather than by transforming the grapheme. In the case of DIAC, these are abstract symbols with no internal spatial information, so are always represented with a constant, invariant shape (see their section for a caveat).

Therefore, HEAD and DIAC graphemes only use the CLASS and SHAPE tags in the corpus tag schema. The possible SHAPEs are enumerated in the following.

7.2.1. HEAD

HEAD graphemes represent the location of the sign relative to a number of different bodily locations in the head, and at the same time depict some of the non-manual parameters of the sign. In this corpus, this articulation is only annotated for the mouth. There rarely appear logograms with other marks, such as head or eye inclination, but it has not been annotated (for now).

In HEAD graphemes, the SHAPE tag specifies the holistic meaning of the full grapheme, including place of articulation and mouth gesture. Since HEAD graphemes are invariant, no other tags are used.

Tab. 7.1. – Values of SHAPE for HEAD graphemes.

face		fore		forer		chin	
cheeks		cheekr		cheekl		mouth	
moutho		smile		teeth		tongue	
nose		ears		earr		eyes	
eyer		hair		back		neck	

7.2.2. DIAC

DIAC graphemes are small, invariant marks, such as contact or dynamics marks or internal movements of the hand. They are classified as such due to their graphical characteristics, rather than after a thorough examination of whether they count as diacritics or not, or due to some internal semantic coherence of the class.

Their concrete meaning is codified in their SHAPE tag. While some are graphical transformations of each other, their rotation and reflection is not productive so ROT or REF tags are not used.

However, sometimes DIACs can appear rotated or reflected for stylistic reasons. This does not alter meaning, but it is important to distinguish between DIACs that are mirror one of the other, with different meanings, and DIACs that accept some stylistic variation to better convey location or what other graphemes they modify.

Tab. 7.2. – Values of SHAPE for DIAC graphemes.

touch		inter		brush		grasp	
between		rub		flex_hook		flex_base	
flex_alt		ext_hook		ext_base		ext_alt	
strike		tense		wiggle		sym	
anti		altern		fast			

7.3. Hands

Hands are the most prominent articulators of Sign Language, and have many degrees of freedom and articulatory possibilities. They are represented in SignWriting with complex graphemes which encode in their graphical attributes the different features of the hand. In the corpus they are assigned the CLASS=HAND, and all four other tags are used: SHAPE, VAR, ROT and REF.

7.3.1. SHAPE

The first feature of the hand is its "shape" or configuration: how the fingers are bent and placed to form a unique shape that acts as a unit. Graphically, the SHAPE tag is roughly the outline of the grapheme, mainly the strokes representing the fingers. Fingers are very flexible, so there are a great many possible configurations that the hand can adopt, and SignWriting strives to provide symbols for every one of them. However, not all of them are in use in every sign language. There is also allophonic variation, meaning some different finger configurations are perceived to be the same hand shape for native signers, and so there can be vacillation and inconsistencies in their transcription.

In this corpus, hand shapes are labelled according to the phonology of Spanish Sign Language, not phonetically, so some different symbols are tagged with the same SHAPE. The phonological base for our labeling is to be published in a forthcoming article. Since there is not a standard notation for hand shapes across languages, we use our own ASCII-based notation which is derived from the previously mentioned phonology. For users not interested in the underlying linguistic theory, these labels can be assumed to be arbitrary strings uniquely identifying the different configurations. Currently, 72 different hand SHAPEs can be found in the corpus.

7.3.2. VAR

Apart from finger configuration, hands can rotate in the three dimensions of space, which complicates their transcription in the flat page. SignWriting uses a combination of graphical features to represent hand orientation, encoded in the remaining tags for hands in this corpus.

The first of them is the VARiation. Graphically, it encodes the "alteration" of the basic shapes encoded in the previous label. This variation can happen in two ways. First, the body of the hand can be filled with different patterns of black and white. White represents the palm, and black represents the back of the hand, as viewed from the point of view of the signer. The fingers can also be detached, meaning the orientation is horizontal.

Fill variation is encoded with the letters w (white), b (black), and h (half). Finger detachment is encoded by prepending the letter h (horizontal) to the tag. This gives six possible VAR tags:

Tab. 7.3. – Values for the VAR tag.

w		b		h	
hw		hb		hh	

There is also the possibility that a hand grapheme has a "black left" and "white right" fill pattern. This is encoded as h or hh, and treated as a graphical reflection (see below for the REF tag).

7.3.3. ROT

To complete the graphical representation of hand orientation, HAND graphemes can also be rotated around their center. This rotation is not continuous but rather has 8 possible values, encoded in this corpus using the notation for the cardinal directions. The hand is considered to be pointing along is distal axis, that is, the straight line from the forearm to the fingertips when they are fully extended.

Tab. 7.4. – Values for the ROT tag in HANDs.

N		NE		E		SE	
NW		W		SW		S	
N		NE		E		SE	
NW		W		SW		S	

7.3.4. REF

As a last transformation, HAND graphemes can appear "mirrored" in SignWriting. Mirroring of a grapheme is not reflective of any one phonological feature, but rather a graphical attribute that can be used to convey different meanings. For example, right and left hands are mirror images, so the corresponding graphemes can be mirrored to better identify each of them. "Black" VARiants are also often mirrored, to better iconically depict the hand as it would be seen by the signer.

Therefore, the meaning of reflecting a grapheme has to be extracted from the context, and can not be deduced from the isolated grapheme at all. This also means that there is not a phonological criterium to decide on a "normal" form of a grapheme, so the criteria chosen in this corpus may seem arbitrary. However, they are chosen to maximize graphical homogeneity and predictability, which can help in the computational treatment of SignWriting.

Reflection is codified in the REF tag, which can take the values n (not reflected) or y ("yes", reflected). To decide whether an instance is reflected, the following algorithm is used:

1. Always, reflection must be decided from "North' rotation. If a grapheme is rotated, it must first be (mentally) set upright.
2. If the VAR is h or hh, the variant with the black on the right is n, and the one with black on the left is y.
3. If the VAR is w or b, attention must be paid to the fingers. If they are in the same position as the unreflected h VARiant, then they are themselves not reflected. In other words, the h VARiant decides, and the b and w ones copy it.
4. If the w o b VARiant is not identical to the h one, attention is paid to the flexion of the fingers. If they bend to the left, the REF is n, otherwise it is y. In the case of the single little finger SHAPE, where the finger bends to one side but curls to the other, the not reflected grapheme is that where the finger is to the far left (white right hand). This step can also help decide the REF for other w or b graphemes without having to look up the h version.
5. Horizontal VARiants follow the same pattern as vertical ones.

The algorithm above is also important because when dealing with hand-written SignWriting, such as this corpus does, there can often appear "non-normative" uses of graphemes which are however understandable and need to be annotated. In any case, since pictures convey graphical information better than words, the following tables present some examples of REF tags.

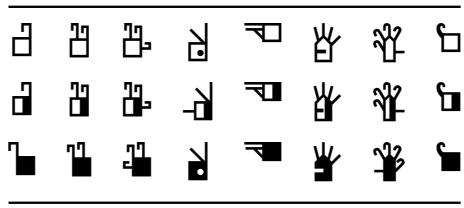
Tab. 7.5. – REF in relation to VAR

n		n		n	
y		y		y	

Tab. 7.6. – REF in relation to ROT

n		n		n		n	
y		y		y		y	

Tab. 7.7. – REF=n for some flexed SHAPEs

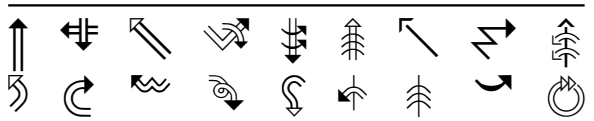


7.3.5. Ambiguous graphemes

Sometimes grapheme SHAPEs are symmetric, meaning that the REFlected versions end up being graphically identical. In this case, REF is always taken to be n. In a few cases, rotation can also be ambiguous (for example the closed fist, which is a square). In this case, the first possible ROT in this sequence is chosen:

N→NE→E→SE...

Tab. 7.8. – Small sample of possible movements as encoded in the Sutton SignWriting fonts.



7.4. Movements

Hand movements are an integral part of sign language, and therefore a substantial part of SignWriting. They are codified with paths and arrows that iconically depict the 3-D movements of the hands in the page. To properly encode 3-D space in 2-D writing, they use graphical attributes to distinguish between planes of movement.

In the digital typographies of SignWriting, there are tens of thousands of characters to account for a wide variety of possible trajectories and types of movement. When devising an annotation schema for movement, and especially when dealing with handwritten SignWriting, using a flat classification is unpractical.

Therefore, in this corpus, movements are annotated by decomposing them into segments, and annotating the segments. These segments convey either straight (`STEM`) or curved (`ARC`) paths, and end of movement markers or "arrow heads" (`ARRO`). Additionally, since the shoulders and waistline are depicted in SignWriting with straight lines, they are provisionally annotated as `STEM`. The forearm shares many characteristics of `STEM`s, so it too is annotated as such.

Movement segments have directional information, annotated in their `ROT` tag, and plane distinctions, annotated in the `SHAPE` tag. Arrow heads have directional information as well, annotated in the `ROT` tag, but the `SHAPE` tag is used to annotate the type of arrow head, which is used in SignWriting to denote what hands move along each path.

Segments often overlap, depicting for example crossing movements of the hands, or a curved segment can be superposed over a straight segment to convey rotation simultaneous to displacement. If these segments have different `CLASS`es, their bounding boxes are annotated as usual. If the `CLASS`

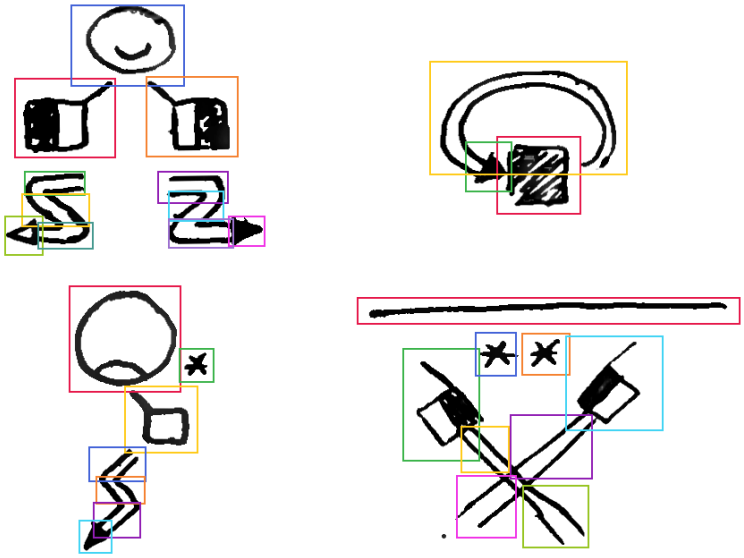


Fig. 7.2. – Graphical example of bounding box annotation for some complex trajectories.

is the same, the overlap in bounding boxes can make them meaningless. This is very common with crossed forearms configurations, or with X-shaped movements. In these cases, straight segments are subdivided, and each division annotated independently, as in Figure 7.2.

7.4.1. ARRO

Arrow heads mark which hand moves, encoded in their SHAPE tag. They can be black (right hand), white (left hand) or j, for both hands joining in the movement. They also point in a cardinal direction, annotated in their ROT tag.

Tab. 7.9. – Values for the SHAPE tag for ARRO.

b		w		j	

Tab. 7.10. – Values for the ROT tag in ARROs.

N		NE		E		SE	
NW		W		SW		S	

7.4.2. STEM

Straight lines can represent the shoulders or waistline, straight movements, the forearm, or both the forearm and a movement at the same time. Their direction is marked in the ROT tag using cardinal directions. Since they are symmetric, only half of the possible ROT values are used.

To distinguish between vertical and horizontal movements or forearms, STEMs can be single or double, which is annotated in the SHAPE tag. Shoulders and waists are always single.

Tab. 7.11. – Values for the SHAPE tag for STEM.

s		d	

Tab. 7.12. – Values for the ROT tag in STEMs.

N		NE		E		SE	

7.4.3. ARC

Curved paths represent arcing or circular movements.

As in STEMs, single and double paths represent horizontal and vertical planes of movement. This is encoded with the first letter of the SHAPE tag. The second letter is used to determine the amplitude of the movement, and can take three values: q for 'quarter', a small arc; h for 'half', a bigger arc which covers around half a circle, and f for 'full' for fully circular paths. Distinction between q and h can be difficult without underlying understanding of the sign language depicted, but with some practice it becomes more intuitive.

To determine the ROTation of ARCs, two points need to be mentally found. The first one is the center of the circle on which the ARC lies. The second one is the middle point of the ARC segment. 'Fully' circular ARCs still have a middle points, since they start and end at the arrow head. Once these two points are determined, the cardinal direction to annotate for the ROT is the direction in which points the segment from the center to the middle point. Table 7.14 will probably make this clearer.

Tab. 7.13. – Values for the SHAPE tag for ARC.

sq		sh		sf	
dq		dh		df	

Tab. 7.14. – Values for the ROT tag in ARCs.

N		NE		E		SE	
NW		W		SW		S	

8. Automatic SignWriting Recognition: Combining Machine Learning and Expert Knowledge to Solve a Novel Problem

Antonio F. G. Sevilla, Alberto Díaz Esteban, José María Lahoz-Bengoechea

This article was published as an open access article in the journal "IEEE Access" in February 2023.

It was originally submitted on December 2021 to the journal "Expert Systems and Applications", where it went through a first round of reviews. After six months, minor revisions were asked for the article text. They were performed, but for more than six months, the editorial process did not advance. Despite repeated inquiries, the journal and its editors did not respond, so we were forced to withdraw the article and submit it to "IEEE Acess" on November 2022. The pre-print version as submitted on 2021 to "Expert Systems and Applications" can be found online.

The final article text is reproduced here, as part of the contributions to this thesis, but the primary source can be found at `https://ieeexplore.ieee.org/document/10035941`.

Abstract

Sign languages are viso-gestual languages, using space and movement to convey meaning. To be able to transcribe them, SignWriting uses an iconic system of symbols meaningfully arranged in the page. This two-dimensional system, however, is very different to traditional writing systems, so its auto-

matic processing poses a novel challenge for computational linguistics. In this article, we present a novel problem for the state of the art in artificial intelligence: automatic SignWriting recognition. We examine the problem, model the underlying data domain, and present a first solution in the form of an expert system that exploits the domain knowledge encoded in the data modelization. This system uses an adaptable pipeline of neural networks and deterministic processing, overcoming the challenges posed by the novelty and originality of the problem. Thanks to our data modelization, it improves the accuracy compared to a straight-forward deep learning approach by 17%. All of our data and code are publicly available, and our approach may be useful not only for SignWriting processing but also for other similar graphical data.

8.1. Introduction

Sign Languages are a family of viso-gestual languages in use by the Deaf and Hard-of-hearing communities. Instead of sound, they rely on hand and body gestures to communicate meaning, making use of the rich possibilities of three-dimensional space and movement to build words and sentences (Barberà Altimira 2015; Brentari 2019). They are not, as it was sometimes thought by the general population, mere codings of oral language in gestures, but full natural languages with their grammar, vocabulary and evolution (Parkhurst and Parkhurst 2001; Senghas, Senghas, and Pyers 2005; Sandler et al. 2014). This recognition as actual human languages has turned them into an object of increasing interest in the research community.

Nonetheless, there is not a standard writing system in use by the signing population. The complex nature of viso-gestual communication, very different to speech, which is based on sound, makes creating or adapting a writing system not a trivial task. A number of proposals exist, most coming from the research community (Hanke 2004; Herrero Blanco 2003; Stokoe 1960). A different proposal, purported to be more user friendly, is SignWriting, which utilizes more of the graphical potential of the blank page to capture signing and its use of space in an expressive and iconic manner (Sutton and Frost 2008).

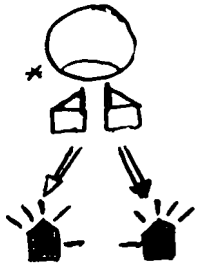


Fig. 8.1. – SignWriting transcription for the sign "History" in Spanish Sign Language, showing the hands near the chin in an initial configuration and moving down and sideways to a finishing configuration.

SignWriting uses abstract but recognizable symbols for the hands and other parts of the body, and then places them in 2D space to represent their relative locations in 3D signing space (Thiessen 2011). Arrows and other graphical tricks serve to fully capture the missing third and fourth dimensions (depth and time) in a system that is arguably intuitive for both signers and non-signers, making it not only a valuable recording and communication tool but also very useful for education.

In Figure 8.1 an example SignWriting transcription is shown. This sign starts with the hands touching the chin, with the fingers flexed in a "pin" configuration. This configuration, as well as the hand orientation, are represented by the two top hand symbols. The location is signified by the circle, an iconic representation of the head, with a line marking that the chin is the actual point of contact. That there is contact at all is represented by the small asterisk to the left of the head. The hands then move downwards and to the sides, represented by the arrows, and end up fully extended and in a different orientation. In this final position, the hands are filled in black and with the fingers separated from the body of the hand to represent that the palm is facing downwards. This is just a small sample of the richness and complexity of SignWriting, but more can be read in Section 8.2 or online at `https://www.signwriting.org/`.

The use of graphic properties and spatial relationships makes SignWriting very different from traditional writing systems. Most writing systems in use for oral languages, if not all, are based on the sequential concatenation of characters, with only slight deviations in the co-location of diacritics and

sometimes punctuation. In SignWriting, however, "characters" do not occur in a particular one-dimensional order. Instead, they appear in a complex bi-dimensional relationship, where their relative position is not arbitrary but actually represents some spatial meaning. Symbols can be rotated and reflected, and as seen before, filled in with different patterns to represent different orientations. If we count every possible graphical transformation or variation of the characters in SignWriting, there are beyond thirty thousand unique symbols to remember, understand, and be able to produce[1], which again makes it very different from the "usual" writing systems.

These challenges mean that most SignWriting is produced and consumed in graphical format (i.e. images), making it difficult to process with existing language technologies or to combine with oral language resources in equal footing. Existing approaches avoid this problem by using code representations of SignWriting, like SignWriting Markup Language (Rocha Costa and Dimuro 2002; Verdu Perez et al. 2017) or Formal Signwriting (Koller, Ney, and Bowden 2013), but to be able to computationally process all the SignWriting data that exists in image form, techniques from artificial intelligence and computer vision are required.

We present in this article such a system, capable of understanding instances of SignWriting in image form by extracting a meaningful representation from the raw pixel data. This meaningful representation encodes domain knowledge of SignWriting, and we have developed an intelligent system able to automatically extract it. As far as we can ascertain from the literature, we are the first to process SignWriting images in this way.

For the processing of the graphic data in the SignWriting images, inevitably noisy and very variable, we use deep neural networks, which have very good results in pattern matching and recognition. First, we present a single state-of-the-art neural network to solve the full task, as an example one-shot approach and as baseline for comparison. Improving on the issues that this first solution presents, our proposed system further utilizes the domain knowledge encoded in our data annotation. Neural networks are arranged in a branching pipeline, with rules coming from domain knowledge used in between to reduce the complexity of the problem and make it more tractable. Later

[1] Counted in the SignWriting fonts, downloaded from `https://www.signwriting.org/catalog/sw214.html`

steps utilize knowledge extracted from previous steps to facilitate further processing, significantly improving accuracy compared to the naive, direct approach. This divide-and-conquer approach is possible thanks to our annotation of data into a hierarchical scheme, which lets us partition the problem into a series of sub-tasks, each easier than the full task.

One difficulty of our research lies in the sparseness and complexity of the data. There are many different symbols to learn to predict, some with different graphical variations, and some of which can be rotated or mirrored to convey different sign language meanings. These symbols are arranged in a single SignWriting transcription in a meaningful way, so it is necessary to find their relative positions at the same time as the symbol meaning is found. Combined with the small size of the available dataset, this complexity makes the search space of our problem too sparse to solve with direct application of existing neural networks. By reducing the features to extract at each step, each search space is made more dense, and entropy of the problem reduced. Not only are each of the sub-tasks then easier to solve, but the information extracted can be further processed, informing the next step and improving its results.

To recapitulate, in this article we present a novel problem for the artificial intelligence community: automatic recognition of SignWriting instances. We propose a modelization of the problem, by giving a computationally tractable description of the underlying data, and we offer a solution using some common machine learning algorithms. Due to the originality and inherent difficulty of the problem, our solution combines the machine learning algorithms with rules coming from domain knowledge. We validate our approach by evaluating its performance in solving the task, and also compare it to the performance of a direct, single algorithm, one-shot approach. These first solutions to the novel problem of SignWriting recognition validate our computational modelization of the data domain, and present an opportunity for future research to improve on the task.

The rest of the paper is structured as follows: in Section 8.2, we present an overview of Sign Language and SignWriting, to show the complexity of the problem and introduce the key points necessary to understand our system. Section 8.3 is devoted to other solutions that try to solve similar problems, and briefly presents useful deep learning approaches. Our contribution is divided

into two sections: Section 8.4 discusses the underlying data engineering, and Section 8.5 presents our solution as well as the baseline approach. These are evaluated and compared in Section 8.6, and finally, Section 8.7 gives our conclusions and outlines future lines of research and development that can be followed.

8.2. Sign Language and SignWriting

Sign languages, as the natural languages of the Deaf and Hard-of-Hearing communities, utilize not sound, but vision. Sound, as used in oral speech, is a wave of air pressure modulated in time to create different qualities to which we then assign meaning. These sounds are easily quantized into individual units, the phonemes, so the fundamental model of speech in linguistics and language technology is as a sequence of individual phonemes.

This simple model is then easily transferable to writing, where phonemes are represented by characters. To each individual sound, we assign a character (sometimes more), and then write these characters in order. Instead of phonemes, some languages assign characters to syllables, morphemes, or whole words, but in their fundamental, written texts are strings of characters each representing discrete sounds or sound sequences.[2]

In sign languages, it is not clear what the fundamental unit is, and this is quite an actual issue of linguistic research. What constitutes phonemes, syllables or even words in sign language? There is a tension in linguistics between trying to apply existing categories, developed from oral language research, and recognizing the uniqueness of sign language characteristics, but there are many examples of in-depth descriptions of different sign languages' grammar and phonology in the literature (Branchini and Mantovan 2020; Herrero Blanco 2009; Langer et al. 2020; Liddell and Johnson 1989);

Some systems developed to write sign languages, such as that devised by (Stokoe 1960), or the Hamburg Notation System (Hanke 2004), are in their origin linguistic notation systems, but SignWriting (Sutton and Frost 2008) is a naturalistic system based on iconicity rather than linguistic categories.

[2] "Ideographic" writing systems, such as the Han characters in use in Chinese and Japanese, seem different on the surface, but in reality each character is assigned a set of possible sounds, so the sequential, one dimensional model still applies.

As such, the fundamental "unit" in SignWriting is the hand, since it is the most prominent articulator in the phonology. We will call the symbols used in SignWriting "graphemes" from now on, to highlight their nature as units of a writing system, but different from the linear "characters" of oral writing systems. We will introduce the different SignWriting graphemes in the following, to give a sense of the problem we are trying to solve, and use them to introduce the relevant parts of sign language phonology which give rise to the complexity of its writing systems.

Since hands are the most prominent and main articulator of sign languages, so hand graphemes are the most visible graphemes in SignWriting. They are iconic depictions of hands, using polygons to represent their configuration, namely the shape the palm and fingers make. The fingers can flex against the palm, extend, join laterally, curl, or even cross. The possibilities are somewhat different in each sign language, but see (Eccarius and Brentari 2008) for an analytical account and coding system for those in American Sign Language. Hand graphemes can be used to represent right or left hands, which are mirror images of each other, so the shape is horizontally flipped (reflected) to represent them when the hand grapheme is not symmetric.

As objects in three dimensional space, hands have a location and rotation which, when relevant, need to be specified.

First, orientation captures the rotation of the hand in three dimensional space. The same sign, with hand rotated in different angles, can mean different things, so orientation is an essential parameter to notate. However, it is an intrinsically three-dimensional feature (see our somewhat mathematical account in Sevilla and Lahoz-Bengoechea 2019), so a number of graphical techniques are required to depict it in the flat surface that SignWriting uses, enumerated in the following and depicted in Figure 8.2.

1. To project the 3D hand into a 2D plane, a plane of observation has to be determined. This can be the vertical plane (parallel to a wall in front) or the horizontal plane (parallel to the floor).
2. If the chosen projective plane is the floor plane, the fingers are drawn separate to the hand, like in the last example of Figure 8.2.
3. The hand grapheme can then be rotated, to iconically represent the rotation as observed from the chosen point of view.

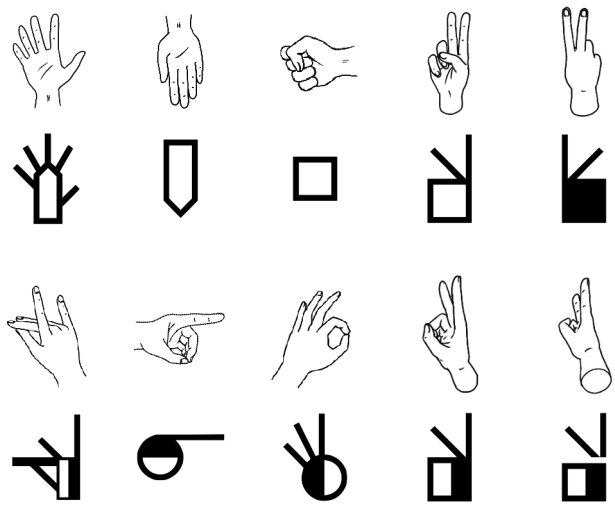


Fig. 8.2. – Some hand graphemes, comprising different hand shapes and orientations. Based on (Sevilla and Lahoz-Bengoechea 2019).

4. To indicate wrist rotation, the palm of the grapheme is filled in with different patterns according to how it would be seen from the chosen plane. If the palm is facing toward the signer, the grapheme is white, while if the back is seen, it is filled with black. If the hand is partially rotated, it is filled half white half black, with the white side indicating where the palm is.
5. To improve iconicity, when the different fills are used, the positioning of the fingers in the grapheme can change, to more iconically capture the observed shape of the hand. This can mean that there is sometimes ambiguity between left and right hands, when their wrists are rotated such as the filling ends being the same.

Next, the hand has to be located in space, since a sign performed at the height of the head is not the same as the same movements performed at the chest, for example. In SignWriting, a number of graphemes exist to denote different parts of the body. These graphemes can then be placed in the page, and hand graphemes located relative to them to iconically represent their three-dimensional location. If the sign is realized in the "neutral" space (in front of the body, but not relative to any particular body part) no body

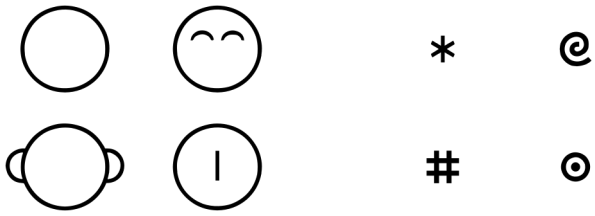


Fig. 8.3. – Examples of non-rotating graphemes. On the left, some body parts (on the head) are shown. Clockwise from top left: generic face, eyes, nose and ears. On the right, diacritics marking different types of contact are show. Clockwise from top left: touch, rub, brush and strike.

graphemes need to be used, or if the head is required[3], the hands are placed sufficiently distant to it to not give rise to confusion. Of course, the exact placement of hand and body graphemes is a subjective and stylistic decision, which may not often be a problem for humans but can be an obstacle for computational processing.

Related to location, an important feature in sign language is contact. Hands can touch, rub, brush and strike each other or different parts of the body. This is represented using small graphemes, which we call diacritics, placed in suggestive positions near the point of contact. These are the main diacritics, but we also group under this umbrella other small, independent graphemes which capture things like movement dynamics, pauses, or internal hand movements. Internal hand movements capture an evolution of the hand shape in the sign, such as fingers being bent, extended, or wiggling. Some examples of heads and diacritics can be seen in Figure 8.3.

Lastly, hands are often not static, but rather move in space, and these movements can have meaning in themselves. In SignWriting, movements are represented with arrows, which can have a start and an end, and depict a straight, curved, or more complicated trajectory between those points. Movements can also be circular, naturally represented with a closed circular trajectory. To represent three dimensional movements in the flat page, the same observation planes used for hand orientation are required. When the movement occurs in the plane parallel to the floor, single-stemmed arrows are used, while double-stemmed arrows represent movements in the vertical

[3]The head may be required to show the eyes or mouth, which can alter a sign's meaning, for example with a frown or a smile.

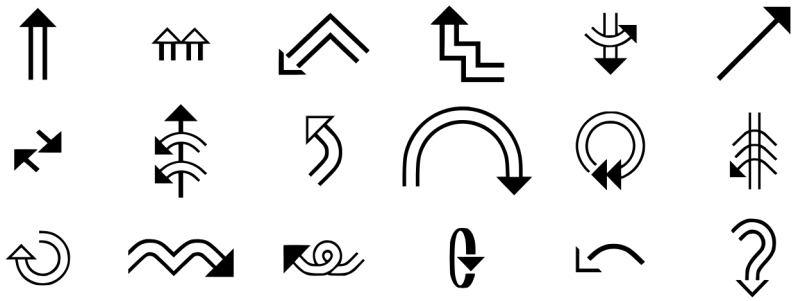


Fig. 8.4. – A selection of movement markers in SignWriting. Each of these is a different character in SignWriting fonts, but we can think of it as composed of smaller units: straight or curved segments, single or double stems, and black, white or empty arrow heads.

plane. Additionally, the arrow heads can be black, if the right hand is moving, filled with white, if it is the left hand moving, or left open, when both hands move together as a unit. Movements can get very complicated, and some examples are presented in Figure 8.4.

These are not all of the possibilities of SignWriting, and even within the described graphemes there is complexity that is out of scope for this article. This introduction should be enough to understand the complexity of the problem in hand, and the reasons for our proposed solution. Nonetheless, a complete textbook by the inventor can be found in Sutton (1995), or online at `https://www.signwriting.org/`.

8.3. Computational Approaches

From a language technology point of view, one might think of our problem as a task of Optical Character Recognition (OCR). OCR is the task of recognizing the written form of language found in images, often scanned, and either coming from a printer, other type of digital production or handwritten. Indeed, that is exactly our problem, recognizing text from rasterized images where the language information is lost but only pixels remain. However, existing OCR solutions do not work for our problem. Smith (2013) gives a fascinating account of the history and architecture of Tesseract, an open source OCR engine, but the reader will note that most of the issues, tricks and clever solutions are based on an understanding coming from oral language.

Like Tesseract, existing OCR systems are developed for the writing systems of oral languages, and thus often rely on two assumptions that don't hold for SignWriting, both of them related to the spatial uniqueness of SignWriting–which is in turn an effect of the spatial use of the viso-gestual modality of sign languages.

First, graphemes are assumed to form one-dimensional sequences. Text flows in a main axis, sometimes wrapping along a second axis (e.g, in western languages, characters flow from left to right and then lines can wrap from top to bottom). Even with the more advanced OCR systems that can detect text in different, random positions in the image, when characters form a word they are expected to form a line. In SignWriting, graphemes are distributed spatially in a significant but non-linear way.

Related, graphemes in oral language writing systems are expected to be mostly upright. Even some typographical variations like cursive don't rotate the character more than a few degrees, and in any case this rotation is not significant but rather stylistic variation or noise. In SignWriting, hand graphemes and movement markers rotate and reflect to represent different spatial meanings. While this could be solved by treating each of the possible transformations as a different grapheme, this multiplies the number of graphemes to recognize in more than an order of magnitude. This makes it unpractical both for the manual processing and tagging required to get the expert system running, and rarefies the data, making it sparser and therefore the use of deep learning less robust.

A similar problem to ours is faced by Amraee et al. (2022), who need to recognize a different kind of language: handwritten logical circuit diagrams. In those diagrams, the 2D position of circuit elements is meaningful, and so, as for us, OCR algorithms are not applicable. However, they have a much smaller vocabulary than ours. SignWriting graphemes number in the hundreds, while logic circuit elements in their dataset seem to be less than ten. Additionally, they only deal with their elements in a standard orientation, the circuit flowing from left to right, so they do not have the problem of rotations and reflections of the graphemes.

Indeed, with the recent advances in artificial intelligence and deep learning, more such graphical writing representations may become apparent[4]. Since ready-made OCR systems cannot be used, nor OCR technologies be adapted to the problem, it is necessary to directly use the underlying technology: computer vision.

8.3.1. Computer vision

Computationally understanding images is a hard problem due to a number of factors. Of course, what humans see and interpret in an image is a rich and subjective composition of different meanings, but we often just need a simpler understanding, such as labeling the object depicted (classification), finding different objects in a scene (object detection, scene understanding) or separating image regions according to the real life object to which they pertain (image segmentation).

The main issue is that the way images are usually stored and manipulated is completely unrelated to the way humans understand them. Instead, they are optimized for display on square arrays of color elements (monitors and screens) and are therefore stored as arrays of pixel values. If an image is stored in row-order, neighbour pixels in the vertical direction will be far apart in memory. If an image stores the different color information (channels) separately, even the values needed to reconstruct a pixel will be far apart in the computational representation.

This pixel array-based representation thus presents a problem for algorithms that try to extract meaning from images, since it is unfeasible to write deterministic and exhaustive rules that relate when pixels form lines, shapes, or more complex objects.

To deal with this, techniques which compute aggregated features from the pixel information have been used to great success. Some of these act globally, computing mathematical properties of full images, while kernel methods use convolutions to compute local properties of images, by integrating with a kernel function suited to the particular task which takes into account the values of nearby pixels. Many different mathematical algorithms and statistical techniques have been developed, and a comprehensive overview of the

[4]We discuss some more of this in Chapter 9 (Sevilla, Díaz, and Lahoz-Bengoechea 2022).

state of the art before the uprise of deep learning can be read in Granlund and Knutsson (1995).

But in recent years, there has been an exceptional expansion of machine learning techniques, especially around the use of deep learning, which has notably improved the state of the art both in accuracy and range of problems which can be tackled. While neural networks haven't necessarily replaced traditional methods (see O'Mahony et al. 2020, for a discussion on this), they have become very popular, not only for their success rate but also because of their relative ease of use.

A rough description of machine learning techniques is to use training data, a large number of input examples where we know the desired output result, to decide the parameters in a regressive (predictive) algorithm by minimizing the error made. For example, in the most common neural networks, an iterative process of prediction and error computation (forward and backward propagation) is performed and the algorithm parameters are iteratively improved.

The strong suit of neural networks is their ability to extract and remember patterns in source data, without the need for the researcher to accurately formalize or describe these patterns. Deep neural architectures can build these patterns from other patterns, in a cascade of increasing complexity, which makes them particularly suited to computer vision. One can imagine pixel arrays turning into lines and shapes, lines and shapes into body parts, and these body parts being then combined to discover that an image is that of a dog.

8.3.2. Neural architectures

As we said at the start of the section, there are different tasks in computer vision, and neural networks are not only applied to computer vision, but rather a wide variety of problems. All these networks are not the same, but each problem requires a specific architecture (combination of layers, activation functions, and other parameters) suited to learning the particular patterns that solve it.

For image classification, that is, finding a suitable label to describe the content of an image, a usual neural architecture used is that of convolutional

networks, for example AlexNet (Krizhevsky, Sutskever, and Hinton 2017). This architecture is composed of convolutional layers, which combine local features of an image into aggregated characteristics. These patterns are built upon, layer after layer, and in the end, sufficiently sophisticated graphical characteristics are found, so a probability of the original image belonging to a particular class can be given. As in all neural networks, the patterns and convolutions found by the network are not pre-determined, but rather optimized in a training step using already annotated images.

"You Only Look Once" networks (YOLO) are a type of neural network used for object detection and classification. Given an input image, the different objects contained are found, and a label assigned to each of them. This architecture can be trained for different corpora, and is described in Redmon, Divvala, et al. (2016). Roughly, YOLO networks learn, for the different regions in the image, the probabilities that a particular object or its boundary can be found there, and then reconstruct the objects' positions and sizes from this information.

8.4. Data design

In the previous section we have seen machine learning approaches to the task of computer vision. But the quality of any machine learning approach rests on a sometimes overlooked cornerstone: data. Training data must be sourced, prepared, preprocessed, and often annotated. This is a costly process in terms of time and effort, but on its correctness lies the maximum real performance of the trained algorithms in the real world. As Sambasivan et al. (2021) say, an error in any step of the data pipeline propagates to further steps, compromising the accuracy and reliability of the algorithm. When approaching a novel problem, often new data must be sourced. Furthermore, the problem must be modeled, and the data annotation schema designed. Neural networks learn patterns very accurately, but we have to decide what patterns to learn, what features to discriminate and which to abstract.

This process of data engineering is a fundamental step of the expert resolution of a problem, a step where much of the domain knowledge to be used

is embedded in the final system, and thus it is as much part of the solution as the system's code and implementation.

Unfortunately, there is not a lot of SignWriting data available to us, and what can be found is not in an easily processable format. Sometimes there is an association of transcription to meaning in oral language (which doesn't really help, since the oral word is almost never related to the sign parameters) and most of the time there is no indication whatsoever of the symbols contained in the SignWriting transcription.

This has meant that we have had to prepare our own corpus of SignWriting that can be used for machine learning or linguistic research, using data collected in a collaboration with linguists as part of a project to develop tools for the ease of SignWriting visualization: `https://www.ucm.es/visse`.

In the corpus, there are two types of data: logograms and graphemes. Logograms are full transcriptions, images of SignWriting which correspond to a sign. Graphemes is the name we have chosen for individual graphic components, units of linguistic information which by themselves convey some of the meaning necessary to reconstruct the sign. The full information is obtained by combining the meanings of the different graphemes while taking into account their relative position and rotation within the logogram.

The information represented by each grapheme, however, is itself complex, and tagging each of them with a single label is not enough. We want to be able to discern the different features described in Section 8.2, and so we have developed an annotation schema for our corpus, where we assign a variable number of labels to each grapheme, depending on how much information is needed.

A first label, `CLASS`, divides graphemes into 6 classes of symbols, with related graphical properties and similar semantic information: `HEAD` for head symbols, `DIAC` for small diacritics, often used for contact information; `HAND` for graphemes which represent hands, and `ARRO`, `STEM` and `ARC` for arrows.

Graphemes are then annotated with an additional tag, `SHAPE`, which identifies the actual SignWriting symbol. For `HEAD` and `DIAC` graphemes, this is enough information, but other graphemes require more annotation. `HAND` graphemes, such as the one tagged in Figure 8.5, have an additional label, `VAR`, which represents the wrist rotation (whether the hand is white, black,

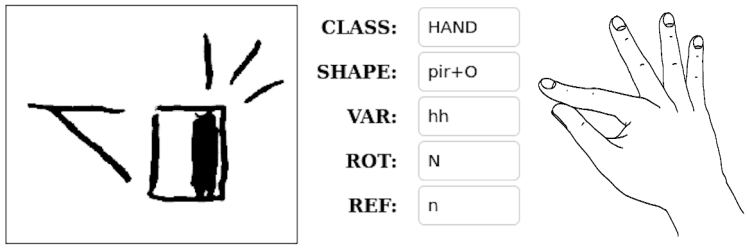
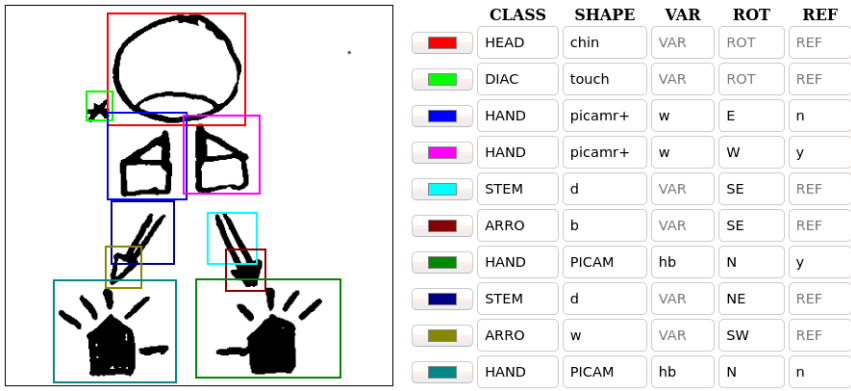


Fig. 8.5. – Annotation for a hand grapheme, showing the different labels and values that need to be assigned. To its right, an illustration of the depicted hand shape is shown.

or half and half, see Section 8.2 about orientation). With this, the "lexical" information of a hand grapheme is fully specified, but we still need to record the rest of its orientation, as graphically represented by the symbol rotation and mirroring. Rotation (ROT) can take one of 8 different values, described with the cardinal points (N for north, NE for northeast, etc.). If the symbol is mirrored (horizontally flipped) a value of 'y' is added for the reflection (REF).

The remaining classes of graphemes, ARRO, STEM and ARC, are those needed to represent movements. As we saw in Figure 8.4, movement markers in SignWriting can be very complex, as well as mixed and superposed to create compositional meaning. Nonetheless, they can be seen as composed of different segments: the straight or curved trajectories, or the end of movement arrow heads. Even if, from the point of view of SignWriting, movement markers are single holistic "symbols", each of the segments has its own visual identity and meaning, so in our corpus we have chosen to identify them as separate graphemes.

Arrow heads (class ARRO) can be black, white, or not filled, to represent which hand is moving. The body of the arrow can then be straight (STEM) or curved (ARC). STEMs and ARCs can be either single or double, to represent whether the movement is parallel to the floor or to the wall, and finally ARCs can be a full circle, half of one, or a quarter. This is all encoded into the SHAPE, and it is our hypothesis that with this graphemes most if not all movement markers can be recorded. Finally, graphemes of ARRO, STEM and ARC class can rotate, so need an additional ROT tag.



	CLASS	SHAPE	VAR	ROT	REF
	HEAD	chin	VAR	ROT	REF
	DIAC	touch	VAR	ROT	REF
	HAND	picamr+	w	E	n
	HAND	picamr+	w	W	y
	STEM	d	VAR	SE	REF
	ARRO	b	VAR	SE	REF
	HAND	PICAM	hb	N	y
	STEM	d	VAR	NE	REF
	ARRO	w	VAR	SW	REF
	HAND	PICAM	hb	N	n

Fig. 8.6. – Annotation of the logogram for the sign "History".

We have collected and annotated 982 handwritten logograms, within which 6071 graphemes can be found. The logogram annotation consists of the bounding boxes (location and size) of the graphemes, and then tags for each grapheme with the appropriate combination of features. An example annotation can be seen in Figure 8.6. The dataset we have created is publicly available at `https://zenodo.org/record/6337885`, including an annotation guide with detailed information about the different grapheme classes and their tags.

8.5. Proposed solution

In the previous section, we have described the data that we have available, and an annotation schema with which to organize and store the information contained within each image in our dataset. This schema is a modelization of the problem we want to solve: extracting a computational representation of the semantics represented by SignWriting images. As we saw in Section 8.3, we can train machine learning algorithms to learn the underlying patterns in annotated data, so that they are able to reproduce them in other, previously unseen, data samples. Therefore, we can use these algorithms, in particular neural networks, to learn and reproduce the modelization of our problem implicit in our corpus annotations, and solve our problem this way.

As a first approach and baseline for comparison, we use a state-of-the-art computer vision algorithm: YOLO. This is a deep learning detector and

classifier neural network architecture, which can solve the full task with one single network, but has many limitations. Our proposed solution is a second approach which addresses some limitations of the first, combining additional networks into a branching pipeline which intelligently decides the paths to follow. This allows us to exploit the expert knowledge encoded into the corpus annotations, and solve a data problem in which a Big Data approach is not enough.

8.5.1. One-shot Approach

YOLO networks, as introduced in Section 8.3.2, seem like a perfect fit for our problem. They solve detection and classification in one step, so given a logogram, they should be able to locate the different graphemes depicted and assign a label to each of them. A problem might be that our graphemes are tagged with many different features, but we can create a different label for each feature combination (concatenating tag values) and give this resulting label to the network to learn. From this single label, the different features can then be extracted, and so the full task of SignWriting resolved. We use YOLO version 3, from Redmon and Farhadi (2018), as implemented in `https://github.com/AlexeyAB/darknet`.

We will examine the results later, in Section 8.6, but this one-shot solution will prove to be not enough for our problem. Essentially, there is too much complexity in our data for the single neural network to solve it successfully.

8.5.2. Expert Knowledge-based Pipeline Approach

To make the detection problem more approachable, we have to reduce the information that is passed to the neural network. Instead of telling it that each possible different grapheme is an object to locate, we can omit some of this information, and just have the network learn to find a few rough groups of graphemes (the `CLASS` tag from Section 8.4). This way, the detector network learns to find objects abstracting away some of their details, which makes it more able to generalize, and thus increasing its ability to find graphemes which are not seen exactly so in the training data.

Then, we can use a different procedure to complete the missing features. Since detection has already been performed, we can extract the region of the

image where the detected grapheme is found, and utilize only this sub-image. This lets us ignore all information unrelated to the particular grapheme in question, and what we have left is a new task of, given an image depicting just a grapheme, find out the features it represents. Moreover, since we already have the rough grouping given by the detection step (the CLASS tag), we can use different processes for each of the grapheme classes.

For most of them, it is enough to use a single neural network to learn all of the remaining features. We use AlexNet, as introduced in Section 8.3.2, and treat this problem as one of image classification. For each grapheme, a label is computed by concatenating its features, just like was done for the one-shot YOLO. The difference is that now we train different networks for each of the grapheme classes, reducing the number of labels they have to learn, and letting them concentrate in the specifics of each different grapheme instead of the commonalities to the whole group.

The case of hand graphemes is a little different. As we saw in Section 8.4, hand graphemes have each four distinct features in addition to the CLASS, that is the SHAPE (finger configuration), the VAR (palm orientation) as well as the graphical transformation (ROTation and REFlection). There are more than 50 hand shapes in our corpus, which multiplied by a possible 6 different variations, 8 rotations and 2 reflections (normal and mirrored) give a total of more than 4000 possible combinations. This is too difficult for the network to learn, more so given the scarcity of our data, so we further subdivide the problem.

First, a classifier (again AlexNet) is used to detect the transformation of the grapheme (rotation and reflection). Once the transformation is detected, the grapheme is normalized to a standard form (North rotation, no mirroring). A second AlexNet classifies this normalized form, finding its SHAPE, and a third and last network finds the VAR. This way, we encode our knowledge of SignWriting into deterministic rules and steps of processing. The neural networks are still in charge of detecting the visual patterns that constitute the graphemes, a task better suited to deep learning, but reconstructing the grapheme tags from the different visual patterns is done in the glue between the networks. This means each network only has a smaller task to learn, which it can tackle even with the limited data available. The normalization of rotation and reflection is again a deterministic rule, which transforms the

data in ways that we know make sense, freeing the networks from having to do the work of generalizing such transformations.

The full pipeline for SignWriting can be seen in Figure 8.7, with an example logogram for reference.

8.6. Evaluation

There are many different metrics which can be used in machine learning and computer vision, including precision, recall, mean average precision, etc. These metrics are each useful for different things, and for a complete understanding of an algorithm it is often necessary to measure all of them, and obtain a rounded view of the problem. Indeed, we have used them while building our system. However, we are not so much interested in the performance of the networks themselves, since there is extensive research on this, but rather in the performance of our system for our full problem, that is, transformation of SignWriting images into useful computational representations.

To measure this, we use accuracy, which is a balanced scoring metric and has the added benefit that it can be directly compared between our solution and the baseline direct approach. Accuracy is a measure of how good predictions by an inference system are, by dividing correct predictions by the total number of instances. We adapt the concept (proportion of correct over total) to three different measures which are useful to us. The key aspect to evaluate is the predicted graphemes: how many of them can be found, and how accurate their predicted features are. The third measurement combines the first two into an overall performance of the full task.

First, we need to measure detection accuracy, that is, whether predicted graphemes are actually there, and whether the graphemes which are there are found at all (equation 8.1).

$$\text{Detection Accuracy} = \frac{\text{correct detections}}{\text{total detections} + \text{missed dets.}} \tag{8.1}$$

A grapheme is considered correctly predicted if the area of the bounding box sufficiently overlaps the true bounding box. If there is no true grapheme where a predicted grapheme is found, it is not counted as a correct detection,

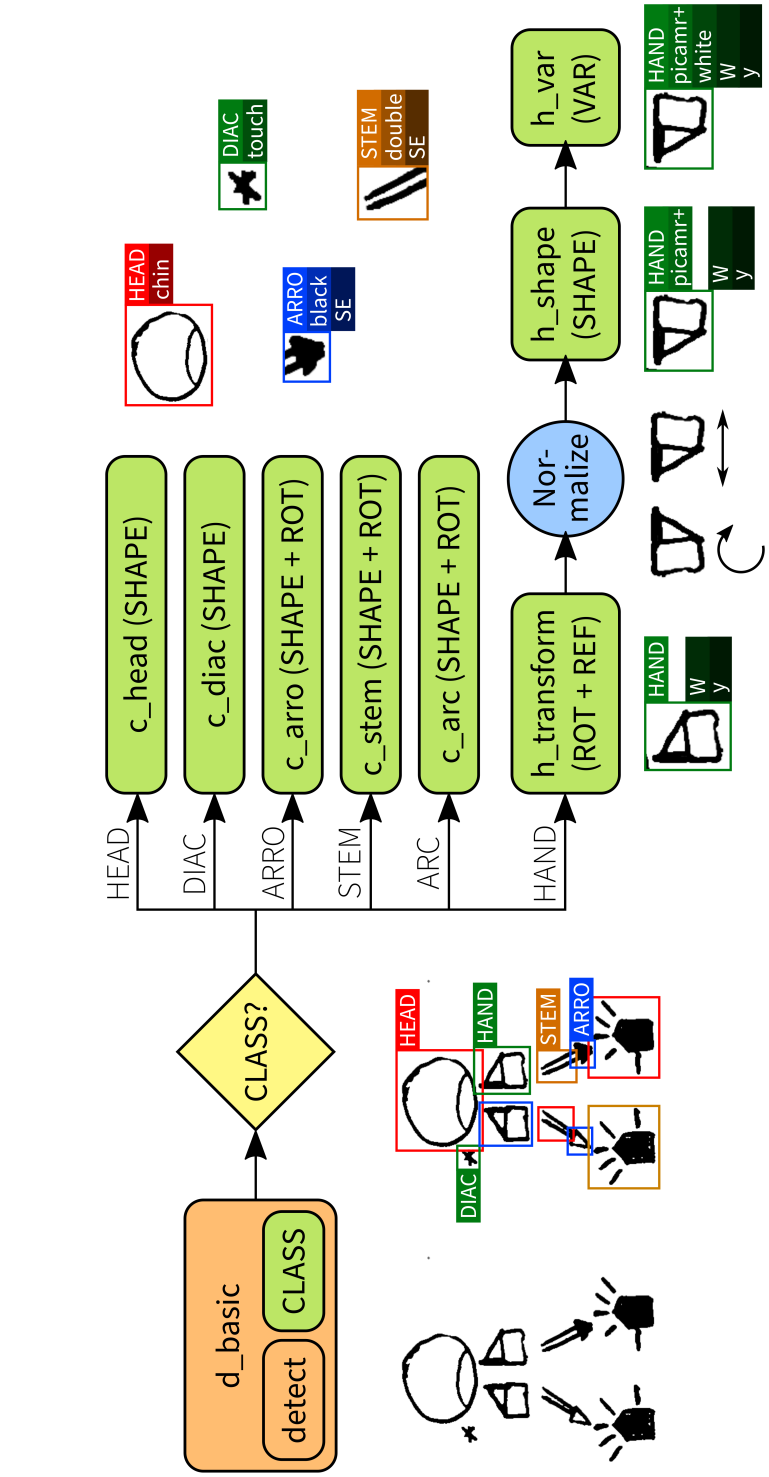


Fig. 8.7. – Full pipeline architecture, including the different networks, decisions and processes. An example logogram and some of the detected graphemes are shown as they progress through the pipeline.

and if no grapheme is predicted where the true annotation has one, it is counted as a missed detection.

Then we need to measure whether the labels assigned to each grapheme are correct, that is, classification accuracy. This is only measured for graphemes which are correctly detected, and it is the proportion of the correctly classified graphemes over the total number of correct detections (equation 8.2).

$$\text{Classification Accuracy} = \frac{\text{correctly classified}}{\text{correct detections}} \tag{8.2}$$

Finally, a combined measure is computed, overall accuracy (equation 8.3), which scores each solution for the global task of recognizing SignWriting. It counts the graphemes correctly detected and classified, dividing it among the total number of predictions and missed predictions.

$$\text{Overall Accuracy} = \frac{\text{correctly classified}}{\text{total detections} + \text{missed dets.}} \tag{8.3}$$

Accuracy measures are proportions, so the three given scores range from 0 (worse) to 1 (perfect) values.

To evaluate the performance of the algorithms closer to how they would be used in the real world, and as is standard practice, we split our logogram data into two sets: a training set, used to automatically learn the patterns, and a testing set which is not used in training. This set is never seen by the algorithm, and thus simulates real world, previously unseen, data. The different accuracy measurements are evaluated on this test set. There are 791 logograms in the training set, and 191 in the test set, resulting in a total of 4840 graphemes in the training set and 1231 in the test one.

The results of computing these metrics for our solution and the baseline single YOLO network are shown in Table 8.1. The overall accuracy of the direct approach is of 0.58, which may seem low but is impressive for the complexity of the problem. Our pipeline manages to increase performance by 17%, to an overall accuracy of 0.68. This improvement alone shows the validity of our approach, and a human analysis of the pipeline results gives an even more optimistic view. Often, the pipeline makes mistakes that are less severe than a complete failure. For example, similar hand shapes are often confused, or diacritics mixed up. While these are indeed wrong predictions,

Tab. 8.1. – Performance of our solution and a baseline one-shot algorithm for the task of SignWriting recognition: grapheme detection and classification within logograms.

Solution	Detection	Classification	Overall
Single YOLO	0.67	0.88	0.58
Pipeline	0.86	0.78	0.68

and are counted as such in the accuracy computations, the partial truth that they are able to predict can still be useful for downstream applications to process, and this is the great advantage of the pipeline.

Our full experimental setup can be reproduced using our published dataset (`https://zenodo.org/record/6337885`, Sevilla, Lahoz-Bengoechea, and Díaz 2022), which includes the Data Version Control ("DVC", Kuprieiev et al. 2021) configuration files that define the experiments, and scripts for performing every step. These scripts use our software Quevedo (Sevilla, Díaz, and Lahoz-Bengoechea 2022), a tool for annotating and processing graphical language datasets. To examine the repository, for example one could use the following commands:

List. 8.1. – How to examine the visse-corpus dataset with Quevedo.

```
$ wget https://zenodo.org/record/6337885/
  files/visse-corpus-2.0.0.tgz?download=1
$ tar xzf visse-corpus-2.0.0.tgz
$ cd visse-corpus
$ pip install quevedo[web]
$ quevedo web
```

To reproduce the machine learning evaluation, one also needs to have DVC and Darknet installed. Then, it is as simple as issuing the DVC "reproduce" command:

```
$ dvc repro
```

This includes all information needed to reproduce our experiments, since all the steps and data of our experimental setup are contained in the different configuration files and software. However, since Quevedo is generic, not

only can our SignWriting experimentation be reproduced, but also its ideas applied to other datasets and domains.

8.6.1. Error Analysis

In a more in-depth analysis, the first immediate observation is that the single YOLO detection performance is too low to be useful. While its classification score is good, this is an effect of only classifying a handful of graphemes, the ones that have been detected. Detection, however, misses too many graphemes, essentially ignoring those which are not common enough. While focusing on the most common data is not a bad strategy in many situations, in this case detection performance is too low to justify it. Furthermore, for our purposes, incomplete predictions can be useful, since a lot of the meaning in the transcription can be later reconstructed, which can not be done for a missed detection.

As was advanced before, the probable reason for this lower detection accuracy is that, by giving all the features to the YOLO algorithm, it can not see the common properties of the different grapheme classes. We tell it that a touch diacritic is a different thing than a rub diacritic, so it needs to differentiate them and can not exploit their graphical similarities. This impedes proper generalization, and thus the network only learns to detect and classify graphemes which it has seen enough, ignoring the rest.

In our pipeline solution, only the `CLASS` feature is given to the detector network to predict. The different grapheme classes have been chosen due to their graphical properties, so the network can exploit this to learn to discriminate them while at the same time being able to generalize to instances not seen before. In fact, the YOLO network is better at this rough classification than a grapheme classifier network trained specifically for this task. It is likely that in this case, having the full logogram context can help, rather than hinder, prediction. Diacritics are smaller than hands, which are smaller than head graphemes. Arrow components (heads, stems and arcs) are usually found together. This context is lost when isolating the grapheme, but present in the full logogram, so the detector can use it to give us a first rough classification which we then apply to split further processing into branches.

Regarding topographic accuracy of detection, that is, how close the predicted bounding boxes are to the annotated ones, it is generally good across different configurations that we have tried. Detecting and separating black-on-white objects is generally easy for the network, with two very relevant exceptions. The first problem are diagonal elongated objects, that is, arrow stems. These graphemes are sometimes very long, and when rotated, they may actually occupy very little of their bounding box—the bounding box is square, but the stems only fill the diagonal. This can be a problem when other objects are present in the bounding box, even if not overlapping the actual arrow.

The second problem is, precisely, overlaps. While YOLO networks seem to do a good job with partially obstructed objects, sometimes graphemes are placed in a "cross" configuration, where they overlap diagonally, and the ends of the lower grapheme spread to both sides of the overlapping one. To further complicate this issue, graphemes so placed often have the same CLASS. Hands can be placed on top of each other, movements can have cross-like trajectories, etc. It is likely that YOLO networks have trouble with these combinations due to the way the edge and interior probabilities are merged by the network to find the predicted bounding boxes.

Fortunately, while not uncommon, these two detection problems are the only issues in this step, and do not hinder further classification of graphemes.

By splitting the accuracy measurement for each grapheme class as in Table 8.2, more detail can be seen. It is clear that two grapheme classes are especially problematic: HAND and ARC. Their classification is a tough problem, which can be seen in the low detection of the single shot YOLO network (remember that it performs detection and classification at the same step, so difficult to remember graphemes will not be detected at all) and in the classification accuracy of the pipeline.

ARC graphemes, which represent curved movement trajectories, are numerous and very varied, while at the same time being the class with lowest number of instances found in the corpus. It is also the case that hand-drawn arcs tend to present the highest graphical variability, since many different angles and sizes can be used by the transcriber to represent the same meaning. Both issues lead us to think that increasing the number of ARC instances in

Tab. 8.2. – Detection, classification and overall accuracy of our pipeline solution and the baseline one-shot algorithm, computed for each grapheme CLASS.

	Oneshot			Pipeline		
CLASS	Det.	Class.	Overall	Det.	Class.	Overall
ARRO	0.79	0.93	0.73	0.90	0.85	0.76
HEAD	0.80	0.81	0.65	1.00	0.65	0.65
DIAC	0.82	0.89	0.73	0.90	0.83	0.75
STEM	0.69	0.91	0.62	0.83	0.89	0.73
HAND	0.48	0.74	0.35	0.88	0.61	0.54
ARC	0.51	0.56	0.28	0.44	0.55	0.24

the training data is the most important step to be taken, but more detailed processing like the one done for HANDs may also help.

HAND graphemes are also very difficult, probably more so than ARCs, due to the multitude of features to be predicted and how they interact. However, hands are probably the most prominent articulator of sign languages, and as such appear in good numbers in the corpus. This can be seen in the huge leap in detection accuracy, from 0.48 by the single YOLO network to 0.88 by the pipeline, and the overall accuracy improvement from 0.35 to 0.54.

In both cases, classification accuracy affects overall performance, but detection accuracy is much better with the pipeline than in the YOLO single shot solution. As said before, being able to detect that a grapheme is present is fundamental for correct computational representation of SignWriting. It is of utmost importance to detect hands, which the single YOLO often fails to do, and in the case of ARCs knowing that a sign has some curved or circular movement, even if the concrete details are not known, is already very useful information.

8.6.2. Limitations of the system

There are a number of limitations to our work, the most important of which is related to the data used for training the deep learning algorithms. The corpus we have collected is very representative of the different graphemes available for composing SignWriting, but there are other dimensions along

which there is not so much variation. For example, most of our transcriptions come from a single informant, so there can be stylistic choices made by that informant and peculiar to their use of SignWriting. Additionally, our logograms represent signs from Spanish Sign Language, so graphemes that codify features not common in Spanish Sign Language may not be properly represented. To overcome this limitation, more data need to be available, and we will work in the future to acquire them.

Thinking about the graphical context of the logograms, all of our samples are in black font on white background, and have been preprocessed to maximize contrast. This can be solved with more data, from more contexts (e.g. ruled or squared paper, using colored ink, from a camera photo with bad lightning) or by adding a preprocessing step to our system which corrects for this kind of variations.

As to the machine learning algorithms employed, it is likely that there are better algorithms available. The state of the art in deep learning is rapidly improving, and it seems every other year there is a major breakthrough. This algorithms could be swapped in, replacing the neural networks we use and improving accuracy. However, since the difficulty of our problem lies in the complexity of the data and the small amount available, our system can continue to work as is, augmenting the machine learning techniques with the expert knowledge and therefore improving overall performance.

8.6.3. Practical application

The system described in this article, including the featureful description of SignWriting instances and the automatic pipeline capable of extracting it, underlies the "Visualizing SignWriting" application (`https://github.com/agarsev/visse-app`). This application was one of the results of project "Visualizando la SignoEscritura" (VisSE, "Visualizing SignWriting" in Spanish) which seeked to create tools for the better use of SignWriting in the digital world. The user-facing result is a web application able to recognize instances of SignWriting, be them scanned from the user device or created from an image processing software, and explain them to the user via textual descriptions of the graphemes and 3D models of the hands. This application validates our approach by showing, on the one hand, the use of image recognition to

process SignWriting, and on the other hand, the usefulness of our annotation schema which can be leveraged to generate explanations for the components of a newly seen logogram.

8.7. Conclusions and Future Work

As the importance of sign languages is recognized throughout the world, the inclusion of the signing community in the digital economy is fundamental. To this end, computational representations of sign languages are necessary, both for end users and researchers alike. In the case of SignWriting, much of the available data exist in image formats, understandable by humans but not machines. Converting SignWriting images into a computational representation is a necessary first step to automatically process them, but requires state-of-the-art applications of artificial intelligence, since images are not easy to process using hand-crafted rules or ad-hoc procedures.

We have presented a careful analysis of the data underlying the problem, establishing a categorization of the different meanings of SignWriting symbols into hierarchical features. This formalization is itself one of our contributions, a computationally valid representation which captures the intended meaning of SignWriting transcriptions into a numerical representation of the positions of graphemes within a logogram and additional key-value pairs of features for each of them.

To automatically reproduce this representation for new instances of SignWriting, the best approach is to use deep learning, able to capture complex relationships in the data and generalize from the patterns present in a training corpus to the general case. However, large amounts of data are necessary to make deep learning approaches work reliably. For our problem, there does not exist a reference corpus of data, or a similar problem from which to transfer learned neural network weights.

We have collected the necessary samples, and created a corpus (Chapter 6, or Sevilla, Lahoz-Bengoechea, and Díaz (2022)) with which to train the algorithms. Still, the amount of data available is small, and costly to annotate. However, we have shown that the annotation pays off if done carefully. Our use of many features, decided both from a semantic point of view and the

necessities of the visual processing required for the problem, has allowed us to build an expert solution able to automatically recognize SignWriting. Compared to the simple, direct approach using a single YOLO network, our proposed improved system uses many deep learning networks, combined in an intelligent pipeline which can extract additional information and make decisions based on previous steps of processing, achieving a 17% improvement in recognition accuracy and additionally being able to extract partial information even when recognition fails.

On the whole, domain knowledge about the problem has let us create a system which utilizes deep learning approaches even in a situation where no existing data can be found, by collecting the corpus ourselves, defining a formal schema for its annotation, and exploiting it to get the best performance from the neural networks employed. Our ideas and approach may be useful not only to process SignWriting instances, but may be applicable to other problems where the data available are less numerous than the expert knowledge that can be collected.

8.7.1. Future work

There are three straight-forward directions in which this research can be improved. On one hand, collecting more and more varied data will likely improve performance of the system, and solve some of the limitations outlined above. A second direction to go is downstream, putting the recognized SignWriting representation to use in more consumer applications. The needs of these applications will tell us what the strong points of our approach are, and where its need to improve to support their use case. Finally, the components themselves used in the system may be improved. We have used readily available neural network architectures, as can be found in the literature, and with implementations that we can directly use. Fine-tuning the network parameters, or swapping some of them for networks better suited to each particular sub-task, will surely improve overall performance.

There is also room for alternative approaches to be tried. An ensemble of neural networks, where their results are weighed and combined, can help improve detection and classification of rarer graphemes, or correct frequent errors for certain common or uncommon situations. A custom

neural architecture that embeds all the steps in our pipeline may be possible, which would facilitate feedback between steps, or some other technique for improving the results of earlier steps in the pipeline by taking into account the confidence of later steps.

Acknowledgments

The development of the expert system described in this paper is part of the project "Visualizando la SignoEscritura" (Visualizing SignWriting, `https://www.ucm.es/visse`), reference number PR2014_19/01, developed in the Faculty of Computer Science of Universidad Complutense de Madrid, and funded by Indra and Fundación Universia in the IV call for funding aid for research projects with application to the development of accessible technologies. We want to acknowledge the collaboration of the signing community, especially the Spanish Sign Language teachers at Idiomas Complutense and Fundación CNSE.

9. Quevedo: Annotation and Processing of Graphical Languages

Antonio F. G. Sevilla, Alberto Díaz Esteban, José María Lahoz-Bengoechea

This article was presented at LREC 2022, Marseille, and the text is reproduced here as one of the contributions to this thesis. The proceedings, article and poster are linked to from my personal page: `https://garciasevilla.com/2022/06/30/Quevedo-Annotation-and-Processing-of-Graphical-Languages/`. Quevedo is available online at `https://github.com/agarsev/quevedo`.

Abstract

In this article, we present Quevedo, a software tool we have developed for the task of automatic processing of graphical languages. These are languages which use images to convey meaning, relying not only on the shape of symbols but also on their spatial arrangement in the page, and relative to each other. When presented in image form, these languages require specialized computational processing which is not the same as usually done either for natural language processing or for artificial vision. Quevedo enables this specialized processing, focusing on a data-based approach. As a command line application and library, it provides features for the collection and management of image datasets, and their machine learning recognition using neural networks and recognizer pipelines. This processing requires careful annotation of the source data, for which Quevedo offers an extensive and visual web-based annotation interface. In this article, we also briefly present a case study centered on the task of SignWriting recognition, the original motivation for writing the software. Quevedo is written in Python, and distributed freely under the Open Software License version 3.0.

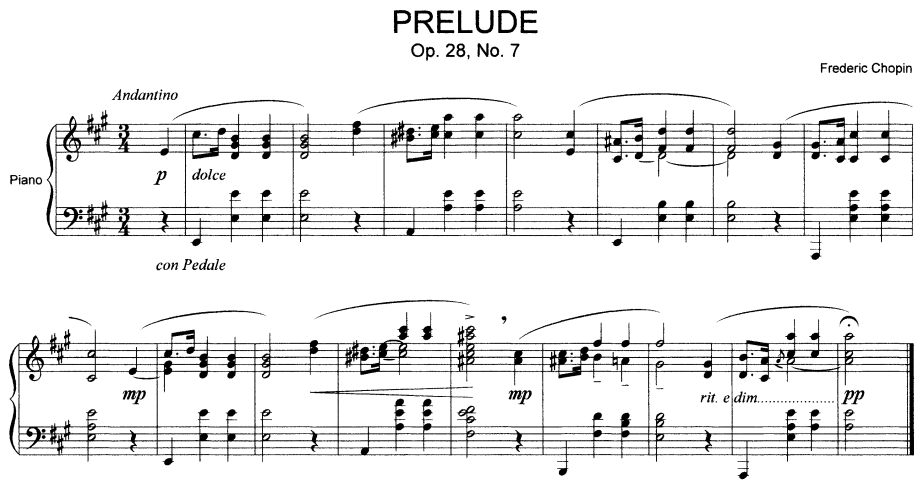


Fig. 9.1. – Modern musical notation as an example of a graphical language. Image by Prof.rick[1] (public domain).

9.1. Introduction

The human language capacity is flexible and very powerful. It gives us not only the usual languages that are the focus of linguistics, i.e. natural, oral languages, but also artificial systems for communication that have some or most of the features of languages and can therefore be studied and processed with linguistic tools.

An example of these systems are graphical languages, which rely on images for communication. Musical notation (Figure 9.1), Feynman diagrams, elementary arithmetic notation, or the Unified Modeling Language (UML, ISO (2012), Figure 9.2) are systems which share some of the important characteristics of languages: signs with a signifier-signified nature, syntactic rules for combining them, and meaning which is compositional, a whole resulting from the consideration of the symbols but also their context and relative arrangement.

However, graphical languages have an important characteristic which is not common in natural languages: they are visual, and exploit the two dimensions of the page as a fundamental feature for codifying meaning. Location of

[1] https://commons.wikimedia.org/wiki/File:Chopin_Prelude_7.png

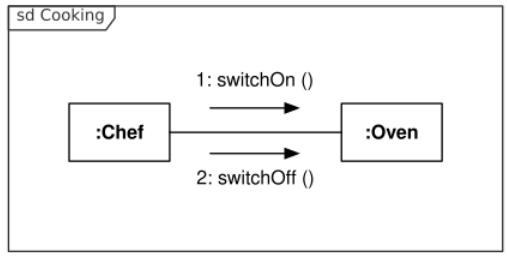


Fig. 9.2. – An UML communication diagram. Image by Oemmler[2], distributed under a CC BY-SA 3.0 license.

symbols in the two dimensions and their relative arrangement is key to the correct understanding of graphical languages, but it is a problem beyond the tools habitual to natural language processing.

If a graphical language is not stored in an abstract, semantic representation, but rather in its graphic realization, it is a challenge to process it automatically. It is necessary to locate symbols within the page, storing their relative locations since these are meaningful data, as well as finding the meaning intended for each symbol, meaning which can be encoded with overlapping graphical features such as size, rotation, color, etc.

In this article we present a software tool we have developed to deal with the annotation and automatic processing of images of graphical languages: Quevedo.

Quevedo is an open source python library and application with both command line and web interfaces. It can create, organize and manage sets of images containing samples of a graphical language. The web application provides a way of visually inspecting and exploring the datasets, as well as featuring an extensive annotation interface. It can be run locally, or deployed as a server for access or annotation by a team.

Processing of the images and their annotations can then be performed, or they can be used in other programs by importing Quevedo as a library. The processing allowed by Quevedo includes training and testing computer vision neural networks for the recognition of the target graphical language. These networks are configured using a declarative format, and can then be

[2] https://commons.wikimedia.org/wiki/File:UML_Communication_diagram.svg

Fig. 9.3. – SignWriting transcription of the Spanish Sign Language sign for "coffee", performed iconically as stirring the beverage in a cup. A video can be seen online at SpreadTheSign[3].

organized into a pipeline, allowing more complex processing and automatic recognition to be performed.

We strive for Quevedo to be a complete solution for the computational processing of graphical languages, guided by our own efforts in that domain. However, we are aware that every task has its own quirks, so Quevedo is very configurable, and extensible with user scripts. The organization of the datasets uses standard file formats and is straightforward and accessible, so the source data, annotations, and neural networks can also be consumed externally or enhanced with other tools.

In our research, we use Quevedo for managing SignWriting data. SignWriting (Sutton and Frost 2008) is a graphical system for transcribing sign languages, using the two dimensional possibilities of the page to encode the movement and use of space of sign languages. In Figure 9.3, an example SignWriting transcription can be seen, and in Section 9.5 we give more detail into this use case.

Before that, in Section 9.2 we examine some related work that is relevant, Section 9.3 gives an overview of the different features in Quevedo, and Section 9.4 gives some notes into how to use Quevedo as a library and application. After we have presented our SignWriting case study in Section 9.5, Section 9.6 summarizes our conclusions with Quevedo and the research it has enabled, and Section 9.7 explains our future development and ideas for how we want to improve it and expand the range of problems it can solve.

[3] `https://www.spreadthesign.com/es.es/word/6209/cafe/0/?q=caf%C3%A9`

9.2. Related Work

Recognizing graphical language images might look, at first glance, somewhat similar to the task of Optical Character Recognition (OCR). Images of writing, be they handwritten, or maybe printed, also consist of shapes in a particular arrangement which need to be recognized by taking into account the natural variability of the graphical realization. Open source tools such as Tesseract (Smith 2013) exist to solve this problem, and indeed often use machine learning and similar algorithms to what we will later propose. The problem with OCR solutions is that they assume an underlying linear ordering to characters, that is, words are formed by characters in a sequence, and sentences by words linearly arranged. Graphical languages such as the ones Quevedo is suited to study are intrinsically two-dimensional, so OCR tools can not be practically used to process them.

In the realm of artificial vision, however, finding and recognizing objects in 2D (or even 3D) is a well studied task. Deep learning neural networks are a common and successful approach to this problem, and there is a wealth of software dedicated to this task, such as PyTorch (Paszke et al. 2019) or TensorFlow (Martín Abadi et al. 2015). Indeed, Quevedo uses one such software, Darknet (Redmon 2013), to train neural networks on the annotated graphical language data, and uses it for inference on new data. These tools often give access to GPUs and other optimizations to make deep learning algorithms usable in reasonable time frames, and provide high-level abstractions to the networks' internal details. However, they still require the data to be prepared for the task, and accessory processing beyond the algorithm itself to be coded by the user.

The labeling task is an important part of this, and there is existing software to make this highly visual task efficient and correct. YOLO Mark[4] is a tool by the authors of the Darknet software that allows visually tagging image files with the objects within, marking their bounding boxes and their assigned class. However, this tool presupposes that the task consists of single class labeling: each object has a tag and that is what is to be stored. The rich annotation often necessary when dealing with linguistic data, and which

[4] https://github.com/AlexeyAB/Yolo_mark

Quevedo makes possible, is not directly possible with YOLO Mark or other similar tools.

This problem is also visible regarding dataset organization and formatting. Datasets for image recognition and understanding, such as ImageNet (Deng et al. 2009) and COCO (Lin et al. 2015), do not need to deal with such a rich annotation as linguistic data often presents. It is often sufficient to present raw images on disk, with a properly formatted name that contains the single label, or maybe a text file beside the image containing the annotation. This straight-forward approach facilitates sharing and reproduction of results, and using it requires a small amount of code that is reasonable to expect every researcher to write themselves. But when data begin to be more complex, and annotations richer, it is also reasonable to have a tool to load the data from disk and access the annotation, a tool tailored for the problem at hand, such as Quevedo and its custom dataset organization architecture.

Data Version Control[5] (DVC, Kuprieiev et al. 2021) is another tool that provides some dataset organization and experiment preparation tools, but again is a generalist tool, requiring the researcher to write the specific code for their domain. Its concerns are, however, orthogonal to Quevedo's, so Quevedo datasets are very compatible with DVC due to their architecture and the use of regular files. Quevedo commands can be tracked in DVC pipeline files, and DVC can understand the parameters in Quevedo configuration files thanks to using TOML[6] as configuration language.

As we have seen, and as often happens in ML, there exist many independent but related tools, some more general and some more specialized. Many of them can be applied to our problem, but require a non-trivial amount of code and design to make them work for our purposes. To alleviate this problem, Quevedo is a tool that understands a more specialized domain, providing features for graphical language processing, while delegating to other tools when necessary.

[5] https://dvc.org/
[6] https://toml.io/en/

9.3. Features

Quevedo can help organize the dataset, storing the source data, metadata and annotations. Quevedo datasets are file system directories, with a `config.toml` configuration file in the top level. This configuration file keeps common information about the dataset, such as annotation schema (an array of possible tags to give to the symbols of the graphical language), number of splits and their use for training or testing, or web interface configuration parameters. A title and description of the dataset can also be given.

The top level directory of the dataset is also a perfect place to have non-Quevedo files such as a "readme", license, or other information. It can also function as a Git[7] and/or DVC repository for better distribution.

Source images are organized into directories, keeping them as raw images on disk. Beside the image files, their annotations are stored as JSON files, allowing easy interoperability. Additional directories are used to store neural network configuration and trained weights, as well as inference results, or user-defined scripts and programs. This straightforward organization into directories and files is easy for other tools to consume if necessary, but Quevedo creates and manages it automatically for the user's convenience.

9.3.1. Annotation Features

Since Quevedo deals with visual data, source files in Quevedo datasets are images in bitmap format. These images are divided into two types: logograms and graphemes.

Graphemes are atomic, individual symbols that represent some particular meaning in the target graphical language, while logograms are images made of graphemes in complex and meaningful spatial arrangements. In the UML example in Figure 9.2, the different boxes, arrows and characters are graphemes. In the SignWriting example (Figure 9.3), the hand symbols along with the arrows indicating movement are the graphemes. In the sheet music excerpt in Figure 9.1, one can identify the notes, accidentals and other symbols as graphemes.

[7] `https://git-scm.com/`

Both logograms and graphemes have dictionaries of tags, following a global schema defined for each dataset. Using a dictionary permits having more complex annotations than just a single label per object, for example having tags for different independent features, or a hierarchy of tags where some values depend on the values of other tags. Graphemes can be independent, or part of a logogram.

Logograms are comprised of graphemes, but the meaning of the logogram is not just the concatenation of the individual graphemes' meanings, but rather is derived from their geometric arrangement in the page. Logograms therefore have a list of contained graphemes, each with their own annotation, but these "bound" graphemes also have box data, representing the coordinates in the image where they can be found. Location data is fundamental for graphical languages, since the relative positions and sizes of the graphemes can have important repercussions on meaning.

Since annotation is a highly visual process, especially for the kind of data in Quevedo datasets, Quevedo can launch a web interface as shown in Figure 9.5. This web interface allows editing tags and metadata for all annotations, and drawing bounding boxes in logograms. Custom functions can also be run from the web interface, either aiding with the annotation process, or letting users visualize the results of these functions without having to run any code.

Annotation files (logograms or independent graphemes) can also be assigned metadata, such as source of the data, annotators, or other custom values to aid in the use of the dataset. Additionally, they can be automatically assigned to different "splits". These splits can then be used to train and test on different subsets of the data, or even perform cross-validation.

9.3.2. Processing features

Quevedo can be used as a library, giving access to the annotations in an easy and organized way, so user code can perform custom processing without having to worry about files and directories or storage formats. However, there is also some higher-level functionality implemented, providing an abstraction over complex tasks that the owner of the dataset may want to carry on.

In the field of Computer Vision, a number of algorithms have been developed to deal with the task of automatically recognizing images or finding

relevant objects in them. Quevedo can train neural networks for this task using the data annotated in the dataset. High-level configuration for the neural network is specified in the dataset configuration file, mainly the task to solve, the annotations to use for training, and which tags to learn to recognize. Based on this high-level description, the necessary Darknet configuration files are generated, and the data prepared so Darknet can process it. The neural network can then be trained with a single command, and a simple evaluation can also be performed. The resulting weights can be used by other applications, or directly from Quevedo.

However, linguistic processing is often not as simple as a single labelling task. There may be different preprocessing steps to run, or some analysis required beyond a machine learning algorithm. This is especially true when the available data is scarce, so rule-based processing is necessary alongside whatever data-based processing is possible. Moreover, language is often seen as organized in layers, and processing mimics these layers by building aggregated representations one step at a time. For this purpose, Quevedo can run pipelines, configured again in a declarative and high-level format in the dataset. Pipelines consist of series of steps, including neural network inference, custom processing scripts, or branching pipelines depending on tag values.

9.4. Usage

Quevedo is freely available on the Python Package Index (PyPI[8]) so the latest version can be installed with the command `python3 -m pip install quevedo`. Source code is available on GitHub[9] under the Open Software License 3.0[10] and documentation is also maintained using GitHub Pages[11].

To build a dataset, we need a collection of images to annotate. We can then use the command line to create the dataset, configure it, and add the images to the relevant subset:

[8] https://pypi.org/project/quevedo/
[9] https://github.com/agarsev/quevedo
[10] https://opensource.org/licenses/OSL-3.0
[11] https://agarsev.github.io/quevedo/latest/

List. 9.1. – How to create a Quevedo dataset.

```
[path/to]$ python3 -m pip install quevedo
[path/to]$ quevedo -D dataset create
[path/to]$ cd dataset
[path/to/dataset]$ quevedo add_images -i
  ↪  source_image_directory -g triangles
```

At this point, we will want to annotate the images. The first step is to decide on an annotation schema, an array of tags to give to logograms and graphemes, and the metadata schema, additional data which we will want to store about each file. This is configured in the dataset configuration file, which is in TOML format so easily editable with a text editor. An example configuration could be the following:

List. 9.2. – Example Quevedo dataset configuration.

```
title = "Example dataset"

description = """
    This dataset is an imaginary example
    for how to use Quevedo. Annotations
    would be similar to those in vector
    graphics format such as SVG.
"""

tag_schema = [ "shape", "fill", "stroke" ]
meta_tags = [ "filename", "meaning" ]
...
```

With this, the web interface can be launched, useful for both visualization of the source images and annotation of their meaning:

List. 9.3. – How to launch Quevedo's annotation web interface.

```
[path/to/dataset]$ quevedo web --host 'localhost' --port 8080
```

Once the data have been annotated, the dataset can be accessed from user code to compute corpus statistics, perform user processing, or train machine

learning algorithms. In the following example, we find the most common colors used in our imaginary dataset:

List. 9.4. – Example use of Quevedo as a library to access a graphical language dataset programatically.

```
from collections import Counter
from quevedo import Dataset

colors = Counter()
ds = Dataset('path/to/dataset')

for a in ds.get_annotations():
    fill = a.tags['fill']
    stroke = a.tags['stroke']
    colors[fill] += 1
    colors[stroke] += 1

print(colors.most_common(5))
```

To use the machine learning functionality provided with Quevedo, first we have to configure the networks and pipelines in the dataset configuration file:

List. 9.5. – Neural network configuration with Quevedo.

```
[network.monochrome]
subject = "Classify black and white shapes"
task = "classify"
tag = [ "shape" ]
subsets = [ "squares", "triangles",
            "circles", "other" ]

[network.monochrome.filter]
criterion = "fill"
include = [ "black", "white" ]
# With this filter, only graphemes with a
# 'fill' tag of black or white will be
# used for training. This lets us have
# different networks for different tasks.
```

With the network configured, we can then use the command line to train it. This will take a bit of time, and at the end the network weights will be

stored in the `network/monochrome` directory. These weights can be used to predict the "shape" tag of new data, and we can do a basic test of its accuracy on our own data:

List. 9.6. – Usage of Quevedo's CLI to train and test neural networks.

```
[p/t/dataset]$ quevedo -N monochrome train
Neural network 'monochrome' trained

[p/t/dataset]$ quevedo -N monochrome test
Annotations tested: 136
{
    "overall": 0.9632352941176471,
    "det_acc": 1.0,
    "cls_acc": 0.9632352941176471
}
```

This is a basic introduction to Quevedo usage, and more detailed documentation can be found online at `https://agarsev.github.io/quevedo/latest/`. The command line interface can also list the available commands and parameters with the command `quevedo --help`[12].

In the next section we will give a brief overview of our own dataset and research using Quevedo, including an example of the web annotation. This dataset can be used to follow along with the explanations in this section or on the online documentation. A simpler example dataset is also provided with the Quevedo source code, and can be found in the `examples/toy_arithmetic` directory. This dataset also serves as an example of the how Quevedo can be used for the annotation of different graphical languages, as it contains examples of elementary arithmetic operations —additions, substractions, etc., performed visually, as would be performed by students. An example can be seen in Figure 9.4.

If the source code of Quevedo is downloaded, the latest development features can be tested. For this, we recommend using `Poetry`[13], a python environment and dependency manager. For example, we could clone the source code repository and use `Poetry` to install dependencies, allowing us to exam-

[12] Quevedo's command line functionality is implemented with the excellent "Click" library: `https://github.com/pallets/click`.

[13] `https://python-poetry.org/`

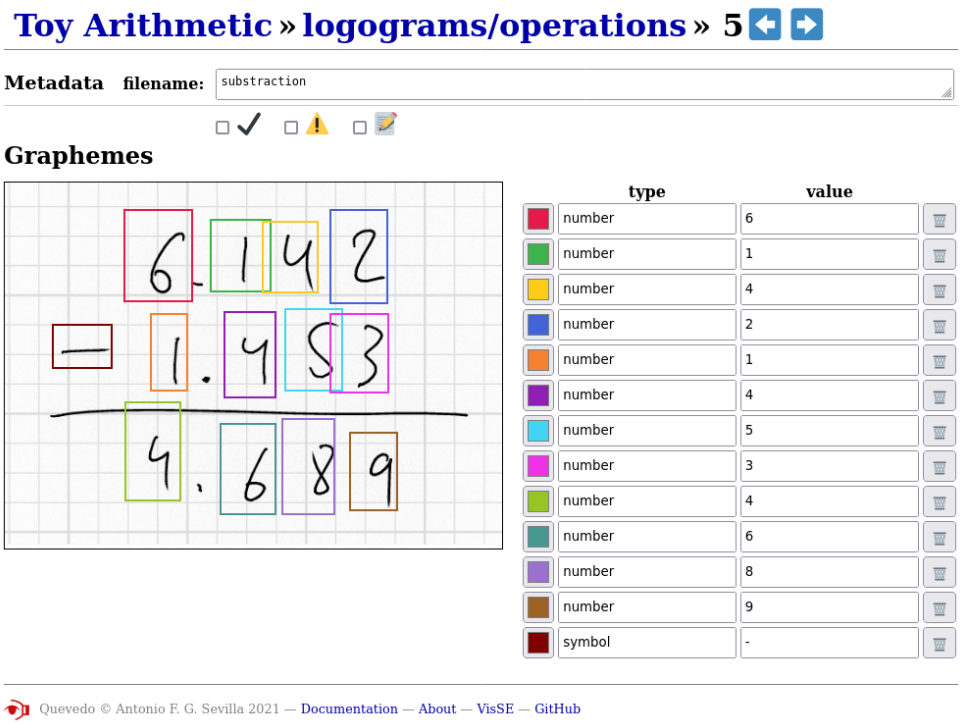


Fig. 9.4. – Example of the use of Quevedo to annotate the graphical language of elementary arithmetic, where the bidimensional position of elements is semantically relevant. Section 9.5 describes a real example of annotation and processing of a different graphical language, but this example shows that the same techniques can be used for different languages and systems.

ine the example "toy arithmetic" dataset using the web annotation interface. This would give a result similar to Figure 9.4, accessible using our own local browser. The following sequence of commands, adapted to our own environment, could be used to this end:

List. 9.7. – How to install and run the web interface to see the example dataset.

```
[~]$ git clone https://github.com/agarsev/quevedo
[~]$ cd quevedo
[quevedo]$ poetry install --extras "web"
[quevedo]$ cd examples/toy_arithmetic
[toy_arithmetic]$ poetry run quevedo info
[toy_arithmetic]$ poetry run quevedo web
```

9.5. Use Case

Quevedo is one result of our project "Visualizando la SignoEscritura"[14] (VisSE, "Visualizing SignWriting" in Spanish). One goal of the project was to create tools capable of automatically processing SignWriting, a graphical system for transcribing sign languages. SignWriting uses iconic symbols to represent the hands, body parts, and their movements and configurations, as the example in Figure 9.3 shows. Quevedo and its features have enabled us to create a system capable of automatically extracting the graphemes within a SignWriting transcription, assigning each of them a label representing their meaning. This required us to collect a corpus, publicly available at `https://zenodo.org/record/6337885` and formatted as a Quevedo dataset. The corpus was manually and visually annotated, and a complex hierarchy of neural networks was trained on these annotations to be able to find the correct tag set for every grapheme in new instances.

For every step of this process we used Quevedo, each step guided by a few configuration lines or command line options, and the ground truth kept in the annotation files. The different steps in the process and their results can be recorded with DVC, easing experimentation and storing of measurements, as well as increasing reproducibility.

Our SignWriting data are handwritten transcriptions of Spanish Sign Language signs, organized into sets to be able to do incremental annotation. Metadata for each file store the annotation progress, doubts and errors, and possible problems in the source images to take into account. Within each logogram, the different graphemes are marked and tagged with a set of features that captures their meaning. This set of features, the annotation schema of the dataset, includes a coarse-grain class for graphemes (`CLASS`), a fine-grain label (`SHAPE`) and a possible variation (`VAR`). Rotation (`ROT`) and reflection (`REF`) are also specified, since they can alter the meaning of the sign. The relative location of the graphemes is marked with the bounding boxes, which are shown visually in the web interface. An example of this annotation process can be seen in Figure 9.5.

Splitting the grapheme labels into a coarse classification (hands, head symbols, arrows, and some others) and then a finer one has allowed us to

[14] `https://www.ucm.es/visse`

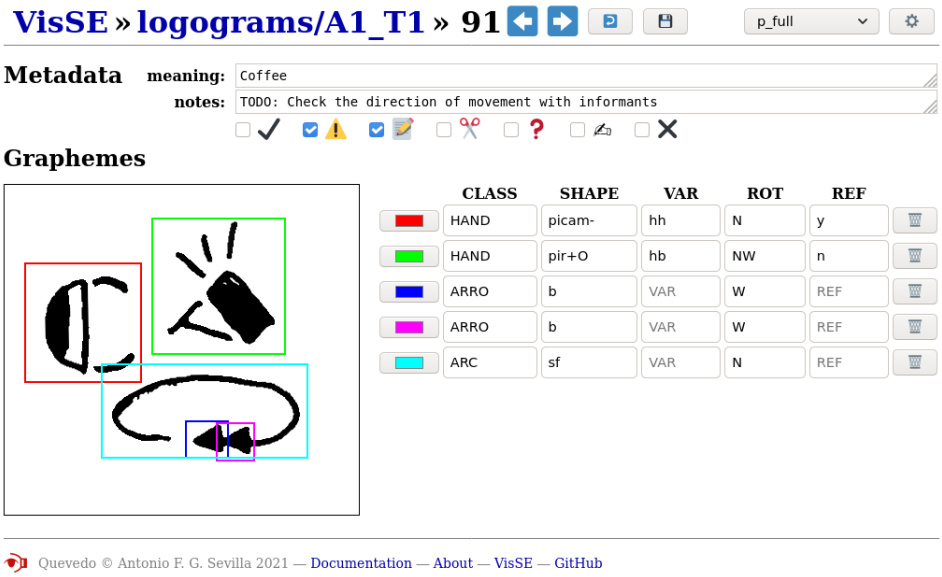


Fig. 9.5. – Annotation of a logogram using the Quevedo web interface. The sign, the same as in Figure 9.3, means coffee in Spanish Sign Language. The graphemes marked and annotated within the image represent the two hands and their relative location and orientation, along with their finger configuration, and the circling movement they perform. Metadata can help with the annotation process, and user-defined flags can be configured to represent task-specific messages for other annotator or reviewers. More details on the labels and their values can be found in the corpus annotation guide at `https://zenodo.org/record/6337885`.

perform automatic recognition of the graphemes in steps, instead of with one single recognizer. Storing rotation and reflection of graphemes in the tags, combined with a script to "straighten" them before recognition, reduced the number of classes of graphemes to be recognized 16-fold, de-sparsifying our data and again dividing the problem of recognition into smaller steps. The trained neural networks, each specialized in a different task, were then collected into a Quevedo pipeline, making the full automatic recognition process available as a single command line invocation or a few lines of glue code.

The use of domain knowledge and deterministic rules to do part of the processing was a necessary step in our research, turning a very difficult problem into a tractable one. "Dividing and conquering" is a common strategy in

computer science, and is especially necessary if the amount of data available is not as extensive as the complexity of the problem requires for a purely data-based, blind approach. Not having "Big Data" at our disposal means we can not rely on ready made, single-shot solutions to our problem, but we can still use some of the algorithms coming from the state of the art. This requires thoughtful preparation and organization of data, and building custom processing sequences. Quevedo was born to help us in doing this for our SignWriting research, but is general enough that it can be useful for other similar tasks of graphical language recognition, where data is not plentiful and requires careful annotation and processing.

For the VisSE project, the results of the data collection, annotation and machine learning were integrated into a user-facing application that explains SignWriting instances[15]. The application works by using Quevedo's recognition pipeline to find the graphemes, and creating textual descriptions for each of them from the predicted tag set. While we have given a shallow overview of our SignWriting pipeline here, researchers interested in reproducing our research or extending it to new problems can find a more in-depth description in our forthcoming work "Automatic SignWriting Recognition".

The success of the VisSE project in achieving its goals serves as an initial evaluation of Quevedo as useful software for the processing of graphical languages, as well as a demonstration of its potential for solving similar problems in the future.

9.6. Conclusion

Graphical languages use symbols arranged in the page to convey meaning, but in contrast to the mostly linear writing systems of oral languages, the two-dimensional placing of symbols is fundamental to their decoding. To properly process and recognize images of graphical languages, we can use techniques from artificial vision, but also need the rich annotation of meaning found in natural language processing tools.

Existing software can help in many parts of the process, but none is focused on the concrete task of graphical language processing, requiring researchers

[15] `https://garciasevilla.com/visse/` (in Spanish).

to write much code to account for its unique features. Quevedo is our answer to this problem, a high-level python library and application that can help in building datasets of graphical language images, and annotating them with the necessary information for their automatic recognition and processing.

Quevedo has enabled our research into SignWriting, a complex writing system for transcribing sign languages which is a graphical language in itself. However, Quevedo is general and domain-agnostic, so it can be used for other tasks and with other datasets. It offers the researcher tools for performing the chores of dataset collection and organization, a visual and fully featured annotation interface, and functions for performing common tasks such as machine learning algorithms training. These tasks are all part of modern data science, but take time and expertise, time which is also needed for the domain-specific tasks of deciding on an annotation schema, relevant processing pipelines, and actual annotation of the data.

We have briefly shown how we use Quevedo, and given a quick primer on its usage for other researchers. There is more documentation available online, and our code is freely distributed at GitHub. We believe that there are many other graphical languages which could be processed with similar techniques to ours, and sharing our software may be a way to help other researchers do so.

9.7. Future Work

In the future, there are many improvements we want to make, and we are conducting lines of research that will contribute more functionality to Quevedo. We want to integrate with alternative deep learning platforms, such as TensorFlow, to be able to utilize the wide array of features they provide, as well as making the interaction of Quevedo with other software easier.

While our focus is on SignWriting, where there is still much work to be done for its fully automatic processing, we already have many ideas of languages where it might be interesting to try to apply our techniques. The examples given in the introduction, such as musical notation and UML diagrams, are only some of them.

Finally, there is current work on developing the next step in Quevedo processing. When representing graphical languages "semantically", often the chosen representation is a list of symbols with their attributes and positions. This is indeed the representation that Quevedo is currently geared to extract, and the one majorly used when dealing with SignWriting computationally. However, this representation leaves interpretation of the image meaning to the human reader. To computationally extract the semantics of graphical language images, further processing is needed, taking into account the whole context of the logograms, as well as the functional relations between the graphemes. This is our current research, and the related code is already under development in the Quevedo repository.

9.8. Acknowledgements

Initial development of Quevedo was part of the project "Visualizando la SignoEscritura", reference number PR2014_19/01, funded by Indra and Fundación Universia in the IV call for funding aid for research projects with application to the development of accessible technologies.

Current development and improvements are part of the project "SSL Signary: A parametric dictionary of Spanish Sign Language"[16], reference number IN[21]_HMS _LIN_0070, supported by a 2021 Leonardo Grant for Researchers and Cultural Creators from the BBVA Foundation. The BBVA Foundation accepts no responsibility for the opinions, statements and contents included in the project and/or the results thereof, which are entirely the responsibility of the authors.

[16] https://www.ucm.es/signariolse

10. Quevedo Documentation

Quevedo is a python tool for creating, annotating and managing datasets of graphical languages, with a focus on the training and evaluation of machine learning algorithms for their recognition.

Quevedo is published under the Open Software License version 3.0, and the source code can be found online at:

- `https://github.com/agarsev/quevedo`

This is a reproduction of the documentation for Quevedo version 1.3 for the purposes of this thesis. The live version can be found here:

- `https://agarsev.github.io/quevedo/latest/`

Due to space concerns, the API for Quevedo as a library and the command line interface are not included here. They can be found online at `https://agarsev.github.io/quevedo/latest/api/` and `https://agarsev.github.io/quevedo/latest/cli/`.

Installation can be performed using pip: `pip install quevedo[web]`. The `web` optional dependency is recommended, so as to be able to follow the discussion around visual annotation and the web interface.

Being a data-centric tool, having data on hand can make following this documentation easier. The VisSE corpus presented in this thesis can be used, and an example dataset can also be found along the source code, in the directory `examples/toy_arithmetic` of the repository.

10.1. Quevedo Datasets

Quevedo datasets consist of source images, annotations on those images, and other metadata that can help with their interpretation. While it can be used for less complex images, Quevedo's focus is on **images with compositional meaning**, such as constitute visual languages, like in Figure 9.2, or complex writing systems, like SignWriting (Figure 9.3) or musical notation (Figure 9.1).

10.1.1. Logograms and Graphemes

Quevedo recognises two types of source images: **logograms** and **graphemes**. Graphemes are atomic, individual symbols that represent some particular meaning in the target visual language, while logograms are images made up of graphemes in complex and meaningful spatial arrangements. In the UML example above, the different boxes, arrows and characters are graphemes. In the SignWriting example, the hand symbols along with the arrows indicating movement are the graphemes. In the sheet music excerpt, one can identify the notes, accidentals and other symbols as graphemes. As for logograms, what constitutes a logogram depends on the target language and the goal of the researcher, but to Quevedo, any logogram is an image file where graphemes are arranged according to some underlying meaning.

The names logogram and grapheme come from the original problem for which Quevedo has been designed, which is automatic recognition of visual languages, but the software imposes little meaning to the terms beyond the fact that graphemes are independent and atomic, and logograms are composed of spatially arranged graphemes. Therefore, Quevedo can be used to manage datasets for problems of varying complexity, as long as the source data are images with some compositional structure.

10.1.2. Annotation of logograms and graphemes

One of the characteristics of visual writing systems is that they can encode multiple meanings within a single symbol, taking advantage of the possibilities offered by the visual medium. In Quevedo, annotation consist not of a single tag, but rather of a dictionary of tag names and values. This al-

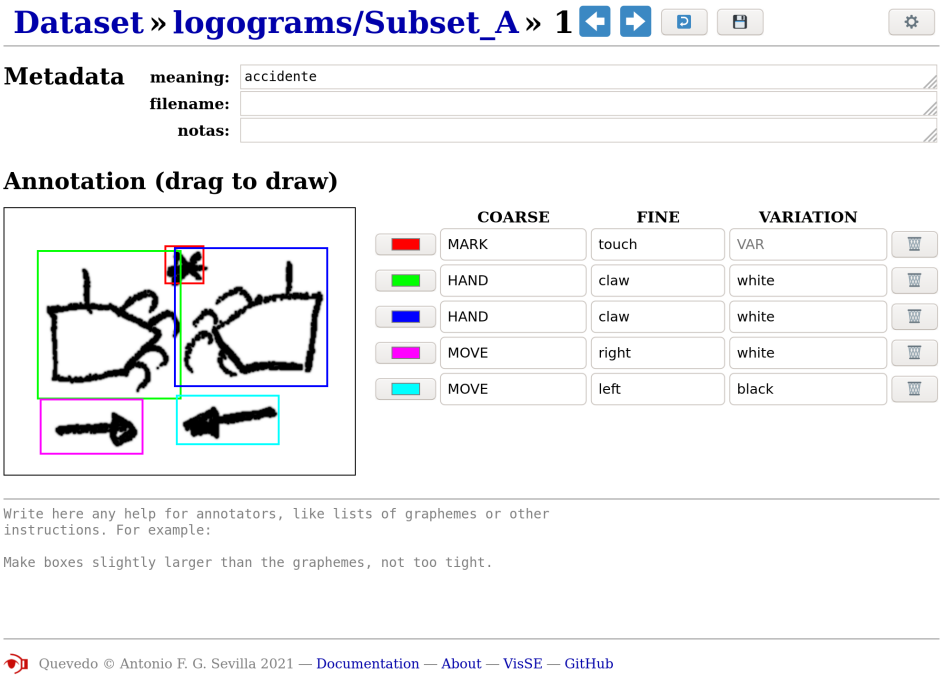


Fig. 10.1. – Example of the annotation of a logogram in the web interface.

lows different systems to use different aspects of the symbols' meaning, and also lets researchers experiment with different, simultaneous and possibly overlapping annotation schemas for the dataset.

Each annotation, both logograms and graphemes, has one such dictionary of tags associated, manually entered by an annotator or automatically filled by some process. Logograms, additionally, have a list of graphemes found within them, and each of these also has a dictionary of tags. Independent annotations (so, not the graphemes within logograms) can also have "meta" tags which can be used to represent other information, such as source of the annotation, status, notes, etc.

To represent the spatial relations between graphemes, logograms also contain a graph of edges between the graphemes. These edges again have their own annotation schema and dictionary of tags.

10.1.3. Dataset structure

Each Quevedo dataset is a directory on disk, containing a configuration file (Section 10.4) named `config.toml`, and a number of directories. Files which Quevedo doesn't recognize will be ignored, so it is safe to add these files that other programs (such as git[1] or DVC[2]) may need.

Annotations are stored in subdirectories of the `logograms` and `graphemes` directories (depending on their type). Each subdirectory represents a data subset, which can be used to perform different experiments on different sets, or just to organize data in some meaningful way.

Annotations in each subset consist of two files: `<number>.png` and `<number>.json`. The `.png` file is the source image, in PNG format, and the `.json` file contains the annotations in [3] format. These are standard formats, so annotations in a Quevedo dataset can be read and modified by external tools and inspected by humans. The annotations are sequentially numbered, so corresponding images and json files are easily found.

There are two additional directories which Quevedo uses: `networks` and `scripts`. In the `networks` directory, the training configuration and weight files for each different neural network (Section 10.2.2) are stored. Each network has a name, and its files are all organized in the subdirectory of `networks` with the network name.

The `scripts` directory can contain useful scripts for additional management of the dataset. For example, a researcher can store the `.r` scripts used to evaluate different metrics on the dataset, or shell scripts to process images or extract annotation information. A special case are python files (ending in `.py`) which Quevedo can understand (Section 10.8.2).

List. 10.1. – Example of a Quevedo dataset directory structure

```
1 dataset_root
2 ├─ config.toml
3 ├─ logograms
4 │  ├─ subset_1
```

[1] https://git-scm.com/
[2] https://dvc.org/
[3] https://www.json.org/json-en.html

```
5  │   │   ├── 1.png
6  │   │   ├── 1.json
7  │   │   ├── 2.png
8  │   │   ├── 2.json
9  │   │   └── ...
10 │   └── other_subset
11 ├── graphemes
12 │   ├── subset_1
13 │   │   ├── 1.png
14 │   │   ├── 1.json
15 │   │   └── ...
16 │   └── other_subset
17 ├── networks
18 │   ├── network_1
19 │   │   ├── train
20 │   │   ├── darknet.cfg
21 │   │   ├── darknet_final.weights
22 │   │   ├── results.json
23 │   │   └── ...
24 │   └── network_2
25 └── scripts
```

10.1.4. Interaction with git and DVC

Since Quevedo datasets are directories on disk, and the different files use standard formats, Quevedo datasets can interact nicely with other tools, such as git and DVC. In particular, a Quevedo dataset can also be a git repository, and therefore a DVC repository too. This can help with dataset sharing and experiment reproducibility. We recommend using git to track configuration files and scripts, and DVC to track source data, annotations, and experiments.

10.2. Neural networks

One of the difficulties of processing visual languages automatically is input, when it is presented in the form of images. Images are represented digitally as collections of pixels, arrayed in memory in a way that makes sense for display and storage, but which is completely disconnected to the meaning these

images have to humans. Additionally, if input is hand written, graphemes can present variations which don't affect human understanding but which mean completely different pixel patterns are present. And positioning of objects is again not based on hard rules, but rather on visual interpretation.

For these reasons, machine learning techniques developed in the field of computer vision are necessary to adequately process logograms and graphemes. While the researcher can use any toolkit and algorithm they prefer, Quevedo includes a module to facilitate using neural networks with Quevedo datasets.

10.2.1. Darknet

Darknet[4] is "an open source neural network framework written in C and CUDA", developed by the inventor of the YOLO algorithm, Joseph Redmon. This framework includes a binary and linked library which make configuring, training, and using neural networks for computer vision straightforward and efficient.

The neural network module included with Quevedo needs darknet to be available. This module automatically prepares network configuration and training files from the metadata in the dataset, and can manage the training and prediction process.

Installation

We recommend using this fork by Alexey Bochkovskiy: `https://github.com/AlexeyAB/darknet`. Installation can vary depending on your environment, including the CUDA and OpenCV (optional) libraries installed, but with luck, the following will work:

List. 10.2. – How to install darknet.

```
$ git clone https://github.com/AlexeyAB/darknet
$ cd darknet
<edit the Makefile>
$ make
```

[4] `http://pjreddie.com/darknet/`

In the Makefile, you probably want to enable `GPU=1` and `CUDNN=1`, otherwise training will be too slow. Depending on the GPU available and CUDA installation, you might need to change the `ARCH` and `NVCC` variables. For Quevedo to use Darknet, it is also necessary to set `LIBSO=1` so the linked library is built. Finally, if you want to use Darknet's data augmentation, you probably want to set `OPENCV=1` to make it faster.

After darknet is compiled, a binary (named `darknet`) and library (`libdarknet.so` in linux) will be built. Quevedo needs to know where these files are, so in the `[darknet]` section of the configuration, the path to the binary and library must be set. By default, these point to a darknet directory in the current directory. Some additional arguments to the darknet binary for training can be set in the `options` key.

10.2.2. Network configuration

Neural networks are ideal to deal with image data, due to their ability to find patterns and their combinations. Quevedo can help with preparing the configuration and training files to train darknet neural networks, can launch the actual training, and can compute evaluation metrics on the resulting network weights. It can also be used as a library to peruse the trained network in an application, not only for research.

But no net is a silver bullet for every kind problem, and Quevedo datasets deal with different types of data with complex annotations. Therefore, Quevedo allows different network configurations to be kept in the configuration file, aiding both ensemble applications and exploration of the problem space.

To add a neural network configuration to Quevedo, add a section to the `config.toml` file with the heading `[network.<network_name>]`. The initial configuration file that Quevedo creates for every dataset contains some examples that can be commented out and modified.

Under this heading, different options can be set, like a `subject` key that gives a brief description of the purpose of the network. The most important configuration option is `task`, which can take the values `classify` or `detect`.

In Quevedo v1.2, a key "extend" has been added that can be used to share network configuration. If a network `net_a` has a key `extend = "net_b"`, parameters from `net_b` will be used when no other value has been set in `net_a`. This can be useful to share common options when testing different networks, or to set a single source of truth for options that must be common. Since v1.3, "extend" is recursive, so a chain of configuration inheritance can be used.

Classifier

Classifier networks can be used with individual graphemes, and therefore use the data in the grapheme subsets of the dataset. Classify networks see the image as a whole, and try to find the best matching "class" from the classes they have been trained in. In Quevedo, classify networks are built with the AlexNet (Krizhevsky, Sutskever, and Hinton 2017) architecture, a CNN well suited to the task.

Detector

Detector networks try to find objects in an image, and therefore are well suited for finding the different graphemes that make up a logogram. Apart from detecting the boundary boxes of the different objects, they can also do classification of the objects themselves. Depending on the nature and complexity of the data, classification of graphemes can be performed by the same network that detects them within a logogram, or can be better split into a different (or many) classifier networks. The detector network architecture used by Quevedo is YOLOv3 (Redmon and Farhadi 2018).

Tag selection

Since Quevedo datasets support a multi-tag annotation schema, a single "class"/"label" has to be selected for the networks in order to perform classification (including detector networks, since they have a classification step). By default the first tag of the tag schema will be used, but other tags can be selected by writing `tag = "some_tag_in_the_schema"`. A combination of the tags can be used by listing them, for example `tag = [ "some_tag",`

`"some_other_tag" ]`. This will produce a single label for each grapheme by combining the values of the tags with an underscore in between, and train and evaluate the network with that single label.

Annotation selection

To specify what subsets of data to use for training and testing of a neural network, we can list the names in the `subsets` option. Additionally, we might want to select some logograms or graphemes to use for a particular network based on the tag values. We can do this by leaving the relevant tags for that network empty, in which case Quevedo will skip the annotation.

In classify networks, finer control can also be achieved using a "filter" section for the network configuration. This filter accepts a key `criterion` which determines what tag from the annotation schema to use to select annotations. Then, an `include` or `exclude` key can be set to the list of values to filter. When `include` is used, if a grapheme is tagged with any of the values in the list, it is included for training and test, otherwise it is ignored. With `exclude`, the reverse happens.

Data augmentation

Recent versions of darknet include automatic data augmentation that happens "on the fly", while the network is being trained. This data augmentation is not based on semantics of the images, but on image properties like contrast or rotation. By slightly and randomly modifying the images that the network is trained on, overfitting can be avoided and better generalization achieved. Some relevant options for grapheme and logogram recognition are supported by Quevedo, and if set in the network configuration will be written into the Darknet configuration file.

The header to use is `[network.<network_name>.augment]`, and the options supported are `angle` (randomly rotate images up to this amount of degrees), `exposure` (change brightness of the image), `flip` (if set to `1`, images are sometimes flipped), and, only for classify networks `aspect`, which modifies the grapheme width/height relation.

In visual writing systems, not all of this transformations are without meaning, so by default they are disabled so that the user can choose which options make sense for their particular use case and data.

10.2.3. Usage

At the command line

Once the network has been configured, the files necessary for training it can be created by running `prepare`. This will create a directory in the dataset, under `networks`, with the name of the neural network. By default, Quevedo will use the neural network marked with `default = True`, so to change to a different one use the option `-N <network>` (since this is an option common too many commands, it must be used after the `quevedo` binary name but *before* the command).

Once the directory with all the files needed for training has been created, a simple invocation of `train` will launch the darknet executable to train the neural network. This command can be interrupted, and if enough time has passed that some partial training weights have been found, it can be later resumed by calling `train` again (to train from zero, use `--no-resume`).

The weights obtained by the training process will be stored in the network directory with the name `darknet_final.weights`. This is a darknet file that can be used independently of Quevedo.

To evaluate the results, the `test` command can be used, which will get the predictions from the net for the annotations marked as "test" (see `split`) and output some metrics, and optionally the full predictions as a **csv** file so that fine metrics or visualizations can be computed with something else. The `predict` command can be used to directly get the predictions from the neural network for some image, not necessarily one in the dataset.

Since commands can be chained, a full pipeline of training and testing the net can be written as:

List. 10.3. – Prepare, train and test a neural network.

```
$ quevedo -D path/to/dataset -N network_name prepare train
  ↪  test
```

At the web interface

Trained neural networks can also be used on the web interface (Section 10.7.3). Networks for detection will be available for logograms, and classifier ones will be available for graphemes. They will be listed at the top right of the interface. When running them, the current annotation image will be fed to the neural network, and the predictions applied (but not saved until the user presses the **save** button). This can be used to visualize the neural network results, or to bootstrap manual annotation of logograms and graphemes.

10.2.4. Example Configuration

List. 10.4. – Example network training and testing configuration for a Quevedo dataset.

```
# Annotations for each grapheme
tag_schema = [ "COARSE", "FINE", "ALTERATION" ]

# Configuration for the darknet binary and library
[darknet]
path = "darknet/darknet"
library = "darknet/libdarknet.so"
# By passing the -mjpeg_port argument to darknet, a live
  ↪  image of training
# progress can be seen at that port (in localhost)
options = [ "-mjpeg_port", "8090" ]

# Detect graphemes in logograms, and also assign a
  ↪  coarse-grained tag
[network.logograms]
subject = "Detect and classify coarse grain graphemes in a
  ↪  logogram"
default = true
task = "detect"
tag = "COARSE"
```

```
subsets = [ "italian", "spanish" ]

[network.shapes]
subject = "Classify grapheme shapes"
task = "classify"
tag = [ "FINE" ]
subsets = [ "simple", "complicated" ]

# When training grapheme classification, augment the data
[network.shapes.augment]
angle = 10
exposure = 0.5

# Some graphemes present alterations, annotated in the
  ↪ "ALTERATION" tag. We want
# to train a specific classifier for these graphemes
[network.altered]
subject = "Classify the alterations of 'complicated'
  ↪ graphemes"
task = "classify"
# The label to train will be a concatenation of the "fine"
  ↪ tag and the
# "alteration"
tag = [ "FINE", "ALTERATION" ]
# We have stored the graphemes with these alterations in the
  ↪ "complicated"
# subset
subsets = [ "complicated" ]

# Only graphemes with the values "multifaceted" or "
  ↪ accentuated" for the
# "FINE" tag will be used
[network.altered.filter]
criterion = "FINE"
include = [ "multifaceted", "accentuated" ]
```

10.3. Pipelines

Quevedo allows you to train different neural networks (Section 10.2) to recognize different objects and features. These networks can then be composed into a pipeline to build an expert system, capable of performing a bigger task than each of the networks by themselves.

For example, a detection network can first locate the graphemes within a logogram, and then specialist networks used to classify each of the graphemes.

When using Quevedo in the command line, you can choose the pipeline to test or execute with the `-P` flag (equivalent to the `-N` flag for networks). In the web interface, the pipelines are available along the trained networks and user functions for the user to run and visualize.

Quevedo pipelines can be used to create expert systems for processing visually complex images. One such example, for the case of SignWriting, is described in Chapter 8.

10.3.1. Pipeline configuration

Pipelines are added to the `config.toml` file of the dataset. Each entry is the name of the pipeline, added to the "pipeline" table. There are a number of types of pipelines that can be used, each with their own configuration, but there are some common options.

Common options

Options for the pipelines often take as value the name of a neural network. Most of the time, instead of a neural network, the name of a pipeline can be used. Quevedo will search the configuration file for pipelines or networks with a matching name, so neither "network" nor "pipeline" need to be prepended.

If a "subsets" key is given to a pipeline, you will be able to use the `test` command to evaluate the performance of the pipeline on those sets. Testing pipelines follows the same rules as for networks, and uses the configured test folds. Training full pipelines is not yet supported, please train the networks individually.

Pipelines can also use the `extend` keyword to inherit configuration from another pipeline, making it easy to set the same subsets to test all pipelines, or share some of the options. This extension is recursive, so chains of pipeline configurations can be used.

10.3.2. Logogram recognizer

A logogram recognizer pipeline has two steps. The first step uses a detector network (Section 10.2.2) to find graphemes in an image. An optional second step uses a classifier network (Section 10.2.2) or another pipeline to then extract the tags for each of the graphemes. The end result is the detected logogram but with the graphemes augmented by the classifier.

List. 10.5. – Example logogram pipeline. It uses a network named `detector` to find the graphemes, and then further classifies them with a network named `classifier`

```
[pipeline.logograms]
detect = "detector"
classify = "classifier"
```

10.3.3. Sequence classifier

A sequence classifier uses many classifier networks or pipelines to iteratively augment the annotation of a grapheme. It has one single option, `sequence`, which is a list of the sub-systems to run.

Additionally to networks and pipelines, the steps in the sequence can be lambda functions to run on the grapheme, or longer functions defined in a user script. For this, place the script in the `scripts` directory, and use as step `script_name.py:function_name`.

List. 10.6. – Example sequence pipeline. It uses a network called `classifier1` to find a first possible set of tags for the grapheme. Then, a function `error_correction` in the `functions.py` user script fixes some common errors. A second network called `classifier2` makes use of the fixed grapheme to get a better prediction.

```
[pipeline.sequence]
extend = "defaults"
sequence = [
    "classifier1"
    "funtions.py:error_correction",
    "classifier2"
]
```

10.3.4. Branching classifier

A branching pipeline can serve to classify graphemes using different networks or pipelines according to some of the grapheme characteristics. A `criterion` option sets the value to use to choose the branch, and then the other options are the networks or sub-pipelines for each branch. The criterion can be the name of a tag, or a lambda function to call on the grapheme.

List. 10.7. – Example branching pipeline.

```
[pipeline.branching]
criterion = "lambda g: g.tags.get('TAG1')"
criterion = "TAG1" # equivalent to the previous one
value1 = "classifier_for_value1s"
value2 = "classifier_for_value2s"
```

10.4. Dataset Configuration

All configuration of a Quevedo dataset is found in its configuration file, which is found at the root of the dataset and named `config.toml`. This file is in TOML format, which makes it ideal for both human and machine editing. We recommend reading TOML documentation to really understand the format, but it is an intuitive enough language that you can understand the configuration file enough to modify it just by reading it.

As a convenience, Quevedo provides the `quevedo config` command to edit the configuration file, but this only launches the user's configured text editor with the `config.toml` file open.

10.4.1. Local configuration

Quevedo datasets are meant to be shared, and configuration is an essential part of the dataset. However, some options may be applicable only for the local environment, and others may be sensitive and best not distributed. For this, Quevedo also reads a `config.local.toml` if present. The options in the local configuration file are merged with those in the main file, overriding them when there is a conflict.

This can be useful for the configuration of darknet installation, which is likely different for different environments, and for the web interface, which may contain sensitive information like secrets and users' passwords (even if hashed).

10.4.2. Annotation schema

The annotation schema of a dataset is a complex set of information and decisions, but to Quevedo, the important information is the featuers that graphemes, logograms and edges can have. These are lists of strings, and each string represents a possible feature for an annotation object. Each concrete object, then, has a particular value (another string) for each of the appropriate features in its schema. There are four schemas:

- `g_tags`: Possible features for each grapheme, either bound or isolated.
- `l_tags`: Features for each of the logograms.
- `e_tags`: Features for the edges of the logogram graph.
- `meta_tags`: Additional information that can be stored for isolated annotation files, either graphemes or logograms, but not for bound graphemes.

Tags are represented as dictionary objects both in the annotation files (in `json` format) and in the code (python `dicts`). In the annotation file, apart from their own `tags`, logograms have a list of graphemes found within them. These bound graphemes have their own `tags` from the `g_tags` schema, and an additional piece of information: the coordinates where they can be found within the logogram image (a.k.a bounding boxes).

Note that in versions of Quevedo before v1.1, tags were stored as a list instead of a dictionary. Before v1.3, there was no logogram annotation schema. If your dataset is using an old structure, Quevedo will warn you. Please run the `migrate` command to upgrade the dataset.

10.4.3. Other options

- `darknet`: Configuration for using the darknet binary and library. See "Darknet installation" (Section 10.2.1).

- `network`: Configuration for training and using neural networks. See "Network configuration" (Section 10.2.2).
- `pipeline`: Configuration for pipelines which use many networks to solve a task. See "Pipeline configuration" (Section 10.3.1).
- `web`: Configuration for the web interface. See "Web interface configuration" (Section 10.5.1).
- `generate`: These options guide the process of artificial logogram generation used for data augmentation. See `generate`.
- `folds`, `train_folds`, `test_folds`: The `folds` option sets the default folds that the `split` will use to partition annotations. The `train_folds` option is a list of fold values that will be used to train, and the `test_folds` option respectively for testing. See "Splits and folds" (Section 10.6.6) for more.

10.4.4. Default configuration

When creating a dataset, Quevedo places a default configuration file with comments to ease personalization. The default file is included here for reference:

List. 10.8. – Default Quevedo dataset configuration file.

```
# This file is a Quevedo dataset configuration file. Find
  ↪  more at:
# https://www.github.com/agarsev/quevedo

# Local overrides can be written in `config.local.toml`

title = "The title"

description = """
The dataset description.
"""

tag_schema = [ "tag" ]
# For a multi-tag schema, use:
# tag_schema = [ "tag1", "tag2" ]

```

```
# Meta tags affect the whole annotation, rather than individual graphemes. The
# first one will be used as title for the annotation in the web listing.
meta_tags = [ "filename", "meaning" ]

# Flags are also meta tags, but can only be true/false. They are displayed in
# the web interface as checkboxes with the icon set here.
flags = { done = "✔️", problem = "⚠️", notes = "📝" }

annotation_help = """
Write here any help for annotators, like lists of graphemes or other
instructions. For example:

Make boxes slightly larger than the graphemes, not too tight.
"""

config_version = 1 # Version of quevedo dataset schema, not of dataset data

# Number of folds to split annotations into, and which to use for training and
# which to use for testing
folds = 10
train_folds = [0,1,2,3,4,5,6,7]
test_folds = [8,9]

[darknet]
path = "darknet/darknet"
library = "darknet/libdarknet.so"
options = [ "-dont_show", "-mjpeg_port", "8090" ]

[network.one]
default = true # Network to use if not specified
task = "detect"
# tag = "tag" # Uncomment and choose a tag name from tag_schema to use
# subsets = [ "default" ] # If not specified, all subsets will be used
subject = "Focus on grapheme type learning and recognition"

[network.two]
task = "classify"
# tag = "tag"
```

```
# tag = [ "tag1", "tag2" ] # A combination of tags can be used as the network "class"
subsets = [ "default" ]
subject = "Classify graphemes"

# A filter can be used to select the annotations to use for training this network
# [network.two.filter]
# criterion = "tag"  # Tag to use to decide whether to include or not each annotation
# include = [ "value" ]  # Values for the criterion to use
# # exclude = [ "value" ]  # Alternatively, values to exclude

# Automatic data augmentation can be configured here:
# [network.two.augment]
# angle = 10
# flip = 1  # yes/no, 1/0
# exposure = 0.8
# aspect = 0.8  # only for classify tasks

# Add more networks like so:
# [network.other]
# task = "detect" or "classify"
# tag = "tag"
# subject = "human readable description"
# ...

[web]
host = "localhost"
port = "5000"
mount_path = ""
lang = "en"
public = true # Set to false to require login
secret_key = "ce8c9cd0316faac773645648ac827ff6"

# [web.users.annotator]
# password = ""
# read = "ALL" # Can read all subsets
# # read = [ "public" ] # Can read all subsets that contain the string 'public'
# write = "NONE" # Can't write (modify) any subsets
# # write = [ "set1$$", "set2$$" ] # Can write to set1 and set2, both logogram or
#                                  # graphemes (they are regexes)
#
```

```
# [web.users.user2]
# ...

[generate]
# Configuration for the artificial logogram generation
count = 500
width_range = [ 200, 300 ]
height_range = [ 200, 300 ]
tag = "tag" # Tag to guide grapheme placement

[[generate.params]]
match = 'one' # Match graphemes tagged with class "one"
mode = 'one' # Only put one of these graphemes
freq = 0.4 # How often to add one of these
rotate = false

[[generate.params]]
match = 'excluded'
mode = 'none' # Don't put any of these graphemes

[[generate.params]]
match = '.*' # Match any grapheme (it's a regex)
mode = 'many' # Add potentially many of these graphemes
max = 3 # How many to potentially add
prob = 0.6 # Probability for a single grapheme to appear
  ↪ (times max = expected number)
rotate = true
```

10.5. Web Interface

The data that Quevedo aims to manage are highly visual, and therefore a visual interface can be very useful. Indeed, in the case of annotation, being able to see the target of annotation is fundamental. It is also a task which may be shared between a team, or conducted by people who are not data scientists or engineers and don't feel comfortable with code and a command line interface.

Quevedo provides a web interface which can be used to visualize, manage and annotate data in a Quevedo dataset. The web interface has the advantage that is graphical, and it can also be run on a server or some shared computer accesible via the internet so that collaborators can work on the dataset without any infrastructure needs on their part (beyond a modern browser).

To use the web interface locally, just run `quevedo web`. This will launch the server in a local port and open a browser window at the appropriate location. To quit the server, just press `Ctrl+C` in the terminal window. For more options see the rest of this document, and for usage of the web interface see Section 10.7.

10.5.1. Configuration

Some options for the web interface can be configured in the dataset configuration file (Section 10.4). Since some of these settings may be sensitive, they can be set in the local configuration file if the dataset is going to be distributed publicly.

The main configuration is set under the heading `web`, and the following options can be set:

10.5.2. Server options

- `host`, `port`: IP address and port to bind the server to.
- `mount_path`: path under which the application will be mounted.
- `secret_key`: secret string to sign session cookies. You can generate a random one for your installation with `python -c 'from secrets import token_hex; print(token_hex(16))'`.

10.5.3. Interface options

- `lang`: the language for messages in the web interface. Supported values for now are `en` (english) or `es` (spanish).
- `colors`: a custom list of colors (in hex notation) can be given for the logogram annotation interface to use. For example: `colors = ['##ff0000', '##00ff00', '##0000ff']`.
- `public`: If true, no login will be required. If false, access will only be provided to logged in users.

10.5.4. Users

To create a user (needed if the dataset interface is not set to `public`) add a heading `[web.users.<user_name>]`, with the following options:

- `password`: hex digest of the sha1 hash of the password for the user. You can generate the hash with the following code:

```
import hashlib
hashed = hashlib.new("sha1",
↪  "thepassword".encode("utf8"))
print(hashed.hexdigest())
```

- `read`: subsets that the user has *read* access to. Can be `ALL`, `NONE`, or a list of subsets to allow access to. These strings are actually *regexes*[5], so any subset matching them will be available.

- `write`: subsets that the user has *write* access to. Follows the same syntax of `read`. Write access means adding annotations to a subset, and *saving* modified annotations to the server (the annotation can be changed locally, but not saved).

Assuming we have the sets `spanish_a`, `spanish_b` and `english_a`:

- `read = [ "spanish", "english" ]` will allow access to all sets.
- `read = [ "spanish_a" ]` will only allow access to the `spanish_a` set.
- `read = [ "_a" ]` will allow access to the `spanish_a` and `english_a` sets.

You can use `^`, `$`, and other *regex* syntax to be more specific.

10.6. Building a Dataset

In this guide, we give an example of the commands and steps needed to create and use a Quevedo dataset. It might be helpful to have an environment available where you can test the different commands as you read the guide, and maybe some data that you can import. The process of creating a dataset is often not straightforward or works out on the first step, so don't worry about making mistakes and having to repeat the process (but please don't delete your original data! Keep those safe in some backup or cloud. Quevedo only

[5] `https://docs.python.org/3/library/re.html#regular-expression-syntax`

works with data it has copied, so deleting a Quevedo dataset is safe as long as you keep copies of your original data somewhere).

In this guide we use git and DVC to manage repository versions and workflows, but that is not necessary, so you can ignore those steps if you don't use them. Also, we assume Quevedo is installed and available as the `quevedo` command, if not, please follow the steps in Section 10.2.1.

10.6.1. Create repo

To initialize the directory where the data and annotations will live, use the `create` command:

List. 10.9. – Creating a Quevedo dataset.

```
$ quevedo -D dataset_name create
```

It will offer you the opportunity to customize the configuration file (Section 10.4) for the repository, and set your annotation schema (Section 10.4.2) and other information. You can modify it later by editing the `config.toml` file, or using the `config` command.

From this point on, we will run commands with the dataset directory as working dir, so change to it with `cd dataset_name`, and we won't need the `-D` flag anymore.

If you want to use git and/or DVC, initialize the repository with the commands:

List. 10.10. – Initialize the dataset with git and DVC.

```
$ git init
$ git add -A .
$ git commit -m "Created quevedo repository"
$ dvc init
$ git commit -m "Initialize DVC"
```

10.6.2. Add data

Once we have the structure, the first step is to import our data. This can be done using the `add_images` command, specifying both the source directory and target subset. To specify the target subset, use the `-l` flag if it's a logogram subset, or `-g` for graphemes. You can specify multiple source directories, which will be imported to the same subset.

List. 10.11. – Add images to a dateset.

```
$ quevedo add_images -i source_image_directory -l
  ↪  logogram_set
```

To track these data with DVC, run:

List. 10.12. – Track dataset data with DVC.

```
$ dvc add logograms/*
$ git add logograms/logogram_set.dvc logograms/.gitignore
$ git commit -m "Imported logograms"
```

10.6.3. Automatically process the data

After the images are imported, we may want to use some preliminary automatic processing, like adding some tags that can be determined by code, preprocessing the images, etc. Create a `scripts` directory if it doesn't exist, and write your code there according to the user script documentation (Section 10.8.2). Then you can run it on the appropriate subsets with the `run_script` command.

List. 10.13. – Run custom scripts on the dataset.

```
$ mkdir scripts
$ $EDITOR scripts/script_name.py
$ quevedo run_script -s script_name -l logogram_set
$ dvc add logograms
```

10.6.4. Annotate the data

Most of the important information in a dataset, apart to the source data, are the human annotations on these data (otherwise, why bother, right?). Since Quevedo deals with visual data, a graphical interface is needed for annotation, and is provided in the form of a web interface (Section 10.5). Remember to first set in the configuration file the annotation schema that you want to use, and then you can lanch the server with the `web` command. If using git and dvc, remember to add and commit any modifications.

List. 10.14. – Annotate logograms with the web interface, dvc and git.

```
$ quevedo web
$ dvc add logograms
$ git commit -m "Annotated logograms"
```

10.6.5. Augment the data

Once logograms are manually annotated, Quevedo can extract the graphemes included within them to augment the number of grapheme instances available to us, with the `extract` command. If what we have are graphemes, we can generate artificial examples of logograms with the `generate`. With these two commands, the data available for training increase, hopefully improving our algorithms.

List. 10.15. – Augment the dataset with artificial samples.

```
$ quevedo extract -f logogram_set -t extracted_grapheme_set
$ quevedo generate -f grapheme_set -t generated_logogram_set
```

These steps can be added to a DVC pipelines file [6] so that DVC tracks the procedure and the results, and when we distribute the dataset other people can reproduce the full process. To have dvc automatically fill the pipelines file, run the commands with `dvc run`[7]:

[6] https://dvc.org/doc/user-guide/project-structure/pipelines-files
[7] https://dvc.org/doc/start/data-pipelines

List. 10.16. – Record data augmentation as a DVC pipeline.

```
$ dvc run -n extract \
          -d logograms/logogram_set \
          -o graphemes/extracted_set \
          quevedo extract -f logogram_set -t extracted_set
```

10.6.6. Splits and folds

For experimentation, we often need to divide our files into a **train** split on which to train the algorithms, and a **test** or evaluation split which acts as a stand-in for "new" or "unknown" data. We may also want to do cross-validation, in which evaluation is done on different runs of the experiment using different train/test partitions. Or in other cases, we may want to exclude some data from all training and testing, making a **heldout** set which is only looked at in the very end to really evaluate performance.

To support different needs from the researchers, the strategy adopted by Quevedo is to assign annotations to "folds". Then, groups of folds can be defined either as being used for training, testing, or none. What folds to use, and which to assign to training or testing is set in the configuration file (Section 10.4.3).

Quevedo can assign the folds to your annotations randomly so that different folds have the approximate same number of annotations using the `split` command:

- Split all logograms into the default folds:

List. 10.17. – Split all logograms into folds.

```
$ quevedo split -l _ALL_
```

- Assign all graphemes in "some_set" to either fold 0 or 1:

List. 10.18. – Assign a particular set of graphemes to some folds.

```
$ quevedo split -g some_set -s 0 -e 1
```

10.6.7. Train and test the neural network

Now that our data are properly organized and annotated, we can try training a neural network and evaluating its results. The first step is to `prepare` the files needed for training, then calling the darknet binary with the `train` command. Finally, the `test` command evalates some quick metrics on the trained neural networks, and can also print all predictions so you can use your statistical software to get a more in-depth analysis.

The commands can also be chained, so it is enough to run (but it will probably take some time):

```
$ quevedo prepare train test
```

Remember that first you must have installed darknet (Section 10.2.1) and configured the neural network (Section 10.2.2) in the Quevedo configuration file. If you have more than one network, specify which one to use with the `-N` flag:

```
$ quevedo -N other_network prepare train test
```

To keep track of the neural network in DVC, we recommend setting preparation, training and test as different stages in the pipeline, so that intermediate artifacts can be cached and the expensive process of training only performed if necessary, and letting DVC track the produced metrics. If you have different networks, they can be set up as template parameters[8] in the pipelines file to keep things *DRY*[9].

List. 10.19. – Train and test neural networks with DVC pipelines.

```
$ dvc run -n prepare_detect \
          -d logograms \
          -o networks/detect/train.txt \
          quevedo -N detect prepare
$ dvc run -n train_detect \
          -d networks/detect/train.txt \
          -o networks/detect/darknet_final.weights \
          quevedo -N detect train
```

[8] https://dvc.org/doc/user-guide/project-structure/pipelines-files#templating

[9] https://en.wikipedia.org/wiki/Don%27t_repeat_yourself

```
$ dvc run -n test_detect \
          -d networks/detect/darknet_final.weights \
          -m networks/detect/results.json \
          quevedo -N detect test --results-json
```

10.6.8. Exploitation

When doing data science, sometimes it is enough to stop at this step. Data are annotated, neural networks trained, experiments run and conclusions obtained. But often the results are actually useful beyond the science, and we want to somehow peruse them. The trained neural network weights are stored in the network directory (Section 10.2.3), and can be used with darknet in other applications or loaded by OpenCV[10] for example. If access to the dataset data is needed, and not only the training results, Quevedo can also be used as a library (Section 10.8) from your own code.

10.7. Using the web interface

10.7.1. Dataset overview

The main page of Quevedo's web interface presents an overview of the dataset, with the different subsets of data listed as folders. On top, general dataset information (from the configuration file) is listed. Data subsets are divided into logogram and grapheme sets, and the number of images in each is noted.

Icons are actually Unicode Emojis, so they may differ between platforms. To browse a subset, click on the folder icon. A new (empty) subset can be created with the "Create new" button.

10.7.2. Subset listing

Each subset has its own listing, where the images contained can be quickly previewed. To go back to the general overview, click on the dataset title on the top left or on the "back to list" icon. At the end of the listing, an "Upload

[10] https://docs.opencv.org/3.4/d6/d0f/group__dnn.html#gafde362956af949cce087f3f25c6aff0d

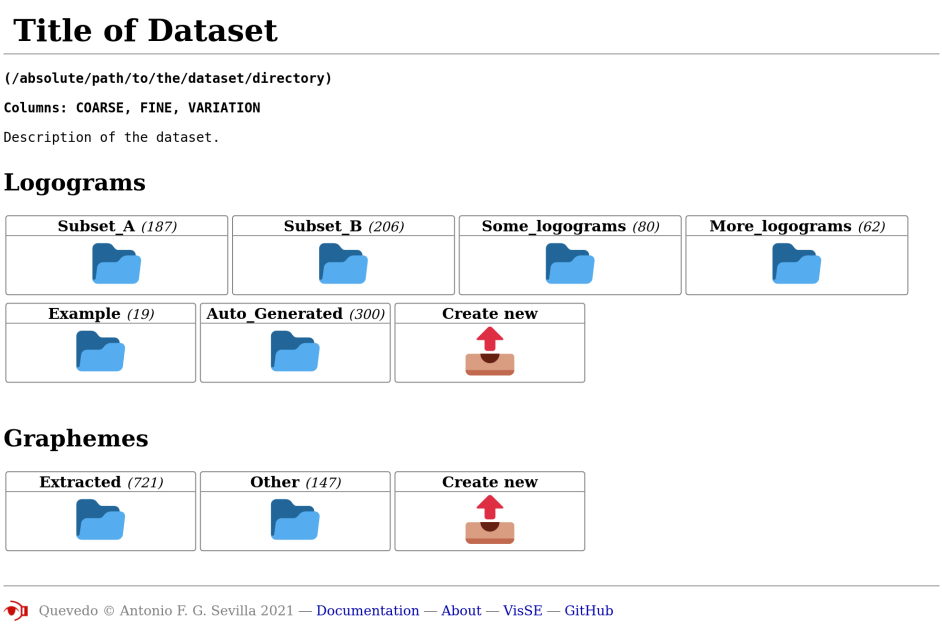


Fig. 10.2. – Dataset overview in the web UI.

new" button allows adding images to the dataset (for a quicker way, see the command `add_images`. Clicking on an image, or the "edit" icon underneath, takes you to the annotation page.

10.7.3. Grapheme annotation

The grapheme annotation page shows, under the heading "Annotation", the image to annotate to the left, and the tags to the right. The tag headers are the ones set in the dataset configuration file under the option `tag_schema`, and the values are to be input by the user. On top of this, the metadata associated with this annotation can be edited.

Below the annotation, any quick guide text set for annotators in the option `annotation_help` is displayed.

The header contains a number of buttons for navigation and access to annotation functions. The links allow you to navigate up to the overview or subset listing, and the arrows after the annotation id navigate to the previous or next annotation.

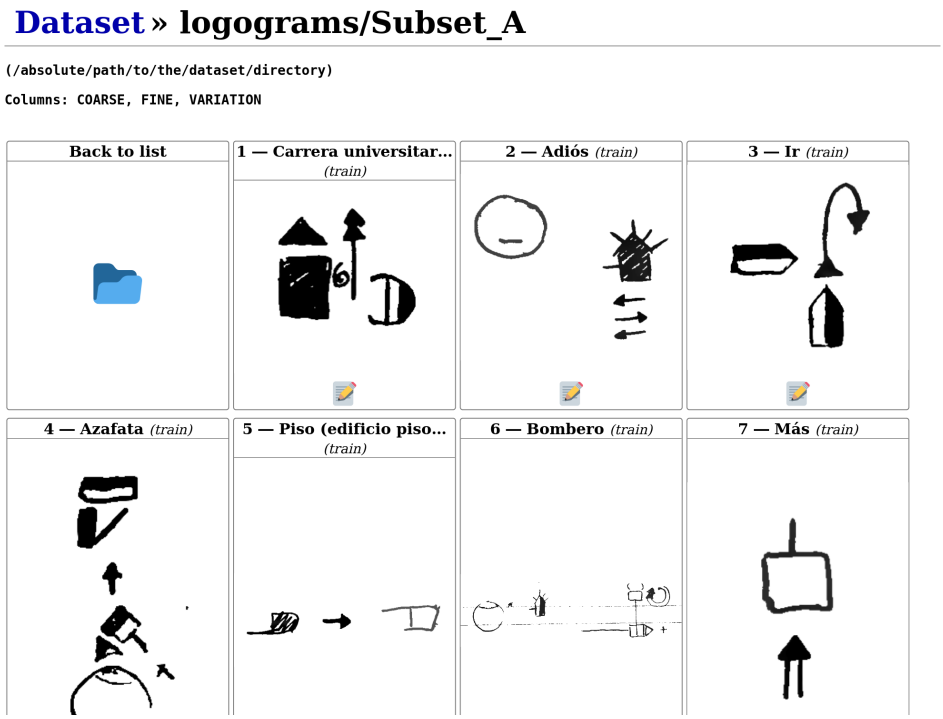


Fig. 10.3. – Subset listing in the web UI.

There is also an undo button that lets you revert any changes from this session (even after saving, but not after navigating away). The save icon sends your changes to the server to be stored in the dataset.

In the top right, a list of functions can be selected, and then run using the gears button. The functions will do some transformation on the annotation, and send it to you to be previewed. If the changes are OK, you can click the save button to store them permanently.

The functions available are of two kinds:

Trained networks When editing graphemes, any neural networks which have been already trained and which have the task `classify` will be listed. When run, the annotation image will be fed to the neural network, the prediction decoded, and the new tags sent to the web interface. This can be used to both visually check the networks, and to bootstrap manual annotation by using the networks output as a first step.

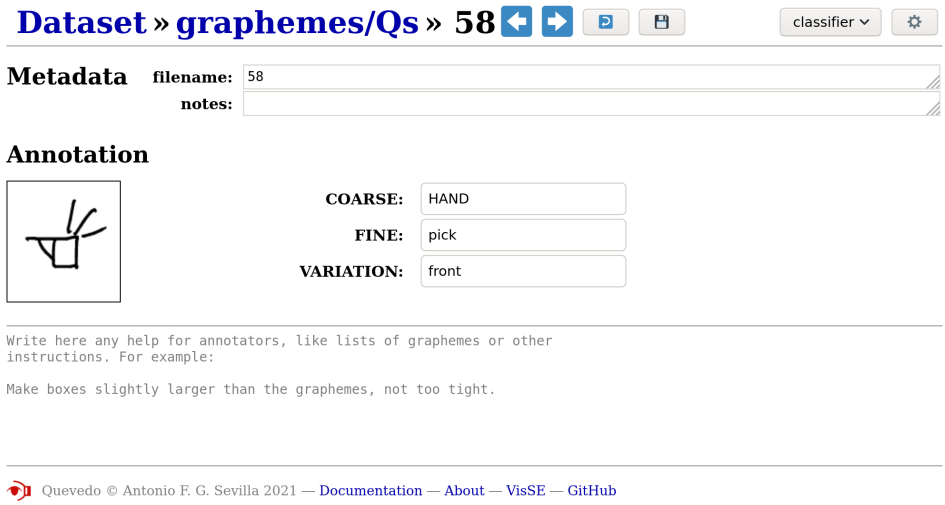


Fig. 10.4. – Grapheme annotation in the web UI.

User scripts Any user script (Section 10.8.2) under the `scripts` directory which has a filename starting with *grapheme* will be listed and available for collaborators to use. Please note that any code in the script will be able to be run by collaborators, so if there are potentially dangerous operations or modifications in the script, don't make it available in the web interface (by changing the name) or properly advise your collaborators.

10.7.4. Logogram annotation

The page for logogram annotation is very similar to the grapheme annotation page, but the actual annotation area is more complicated.

On the right, instead of a list of tags, there is a table. Each row corresponds to a different grapheme found within the logogram, and the columns are the tags from the `tag_schema`. You can move right and left with `Tab` and `Shift+Tab`, and up and down with the arrow keys in the keyboard.

To add a grapheme, click on the logogram image and drag the appearing rectangle until it covers the full grapheme area. You can redo the rectangle while this grapheme is selected, and you can always modify the rectangle for a grapheme selecting it again by clicking on its row. To finish the rectangle and deselect, click outside of the logogram image or press `Enter`. You can

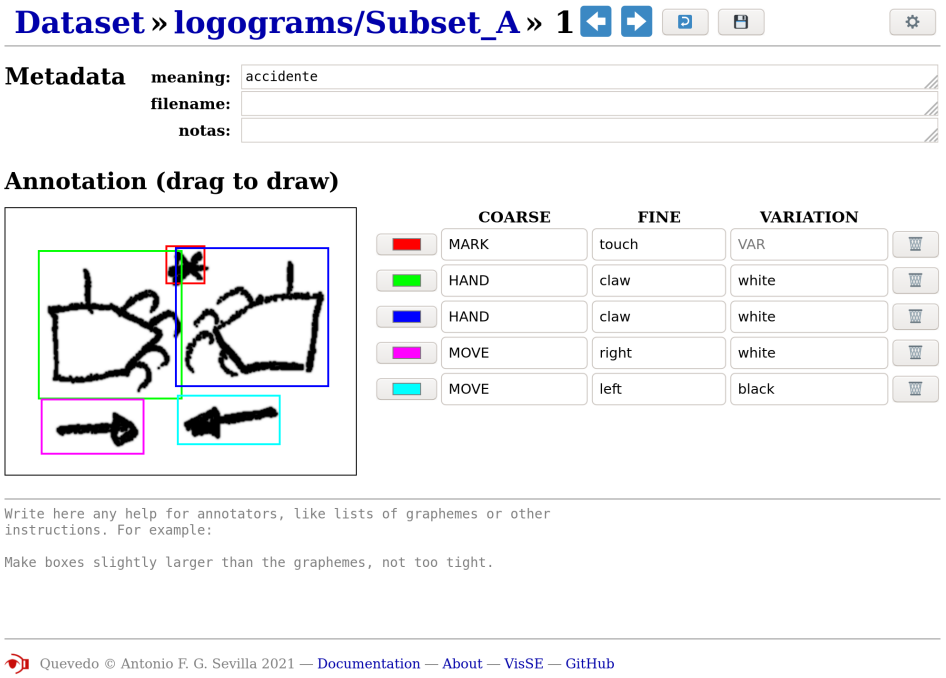


Fig. 10.5. – Logogram annotation in the web UI.

remove graphemes by clicking on the Trash button at the right of each row. You can also change the color of the grapheme rectangle with the button on the left of each row, but please note that these colors are not stored by Quevedo, and are only a visual aid to annotation while on the web interface.

As with graphemes before, changes can be undone, and must be sent to the server to be stored with the "Save" button.

The networks offered in the functions list on the top right will be those which have the task `detect`, and the scripts available will be those with filenames starting with *logogram*.

10.8. Quevedo as a library

Quevedo can be used as a command line application to manage a dataset, but it can also be used from other Python code to make programatic access to

the dataset more convenient, or in user scripts run by Quevedo on the dataset objects.

10.8.1. Call from python

To use Quevedo from other python code, you can import it with `import quevedo`, or it may be more useful to directly import the Dataset class: `from quevedo import Dataset`. This class has most of the functionality to deal with Quevedo datasets, including managing the data and the neural networks. There are some examples of use in the examples directory (`https://github.com/agarsev/quevedo/tree/master/examples`) of the repo, and we try to keep the code readable to help understand the library. The full API reference with the different classess and methods can be read at `https://agarsev.github.io/quevedo/latest/api`.

10.8.2. User scripts

Every dataset is different, and dealing with data often needs to have custom procedures and algorithms, specific to the problem at hand. We suggest to place these scripts in the `scripts` directory of the dataset, to keep them organized. Quevedo can also help run scripts in this directory using the command `run_script`.

With `run_script`, you don't need to write the boilerplate code of accessing all the annotations, loading their data and image, and saving them. Just provide a `process` function, which receives an annotation object and the dataset, and process the annotation with your custom logic. The `run_script` command then lets you specify, using syntax similar to the other commands, what subsets to run the script in.

List. 10.20. – Python script that uses Quevedo as a library. This script automatically adds some metadata to annotations, like the date and the original filename, but only if there is no `author` metadata field defined in order not to overwrite manually added metadata.

```python
from datetime import date
from quevedo import Annotation, Dataset

# Our custom logic to get tags from the filename
def tags_from_filename(filename: str):
    tags = filename.split('_')
    if tags[0] == 'something':
        tags[0] = 'some other thing'
    return tags

def process(a: Annotation, ds: Dataset):
    if a.meta['author'] is not None:
        return False  # We don't want to modify this annotation
    a.meta['annotation_date'] = date.today()
    a.meta['author'] = 'automatic'
    # The original filename is kept by `add_images`
    a.tags = tags_from_filename(a.meta['filename'])
    # Return True for Quevedo to save the updated annotation
    return True
```

Another advantage of user scripts is that Quevedo makes them available on the web interface (Section 10.7.3). The top right corner of the annotation page has a listing of functions, including trained neural networks and user scripts, that can be run, allowing annotators to access this functionality directly from the web interface. If the script is used from the web interface, the annotation won't be automatically saved, allowing the user to review the results before clicking save.

10.8.3. Modifying Quevedo

Quevedo is open source! You can modify and extend it by forking it on GitHub: `https://github.com/agarsev/quevedo`. If you use Quevedo for your research and have ideas for improvement, please do get in touch via GitHub discussions or email.

III

Appendices

A. Otras aportaciones

Además de las aportaciones documentales, incluidas en la parte II de este documento, durante la tesis he realizado otras publicaciones o aportes que no encajan con el modelo tradicional de documento escrito, y por tanto no se pueden (o no tiene sentido) incluir de forma integrada aquí. Quiero recogerlas de todos modos, en parte también para dar visibilidad a distintos formatos de obra que se pueden realizar en la ciencia contémporanea, especialmente en las disciplinas digitales. Las describo a continuación con enlaces a su "casa" en internet.

A.1. Pósters de congresos

Programados en markdown y HTML, usando Pandoc, para poder producir versiones equivalentes imprimible y web. La versión web es "responsiva", adaptándose a distintos dispositivos.

A.1.1. Quevedo: Annotation and Processing of Graphical Languages

Versión web del póster presentado en LREC 2022 en Marsella (capítulo 9).

- `https://garciasevilla.com/2022/06/30/Quevedo-Annotation-and-Processing-of-Graphical-Languages/Quevedo_Poster.html`

A.1.2. Tools for the use of SignWriting as a Language Resource

Versión web del póster enviado al noveno Taller de Representación y Procesamiento de las Lenguas de Signos (2020, capítulo 5), presentado también en Marsella en 2022 debido a la pandemia.

- `https://garciasevilla.com/2020/05/31/Tools-for-the-use-of-SignWriting-as-a-Language-Resource/Poster_VISSE_signlang22.html`

A.2. Resultados de VisSE

A.2.1. Código fuente de Quevedo

Código fuente de Quevedo (capítulos 9 y 10), escrito principalmente en lenguaje Python. Publicado en abierto bajo la licencia Open Software License, en el repositorio internacional de código GitHub.

- `https://github.com/agarsev/quevedo`

A.2.2. Meta repositorio de la aplicación para el proyecto VisSE

Repositorio que incluye a su vez los repositorios con el código de las distintas partes de la aplicación desarrollada para el proyecto VisSE (capítulos 3 y 8). Programados en Python, HTML, Javascript, CSS, entre otros. Publicados en abierto bajo la licencia Open Software License, de nuevo en GitHub.

- `https://github.com/agarsev/visse-app`

A.2.3. Corpus VisSE de SignoEscritura Española

Conjunto de datos y manual de anotación (en inglés y español, formato imprimible) publicados en abierto e indexados en OpenAire. DOI: 10.5281/zenodo.633. Publicado en el repositorio científico internacional Zenodo (capítulos 6 y 7).

- `https://zenodo.org/record/6337885`

A.2.4. Vídeo divulgativo del proyecto VisSE

Vídeo que describe las distintas partes del proyecto "Visualizando la SignoEscritura", su motivación y resultados (capítulo 3). Realizado en Lengua de Signos Española y doblado y subtitulado en castellano.

- `https://garciasevilla.com/2022/06/11/Proyecto-VisSE/Proyecto_VISSE_video_subtitulado.mp4`

A.3. Aplicaciones interactivas de LSE

Estas aplicaciones utilizan la teoría fonológica desarrollada por el equipo para entender la LSE desde el punto de vista de la lingüística computacional (Sevilla y Lahoz-Bengoechea 2019; Lahoz-Bengoechea y Sevilla 2022a; Lahoz-Bengoechea y Sevilla 2022b).

El (sencillo) avatar tridimensional, el código de interactividad y las ilustraciones de las manos han sido creados por mí y su uso es libre bajo la licencia de Creative Commons CC BY-SA 4.0[1].

A.3.1. Juego para aprender dactilología

El avatar signa una palabra del español usando el alfabeto dactilológico, y el jugador debe responder correctamente.

- `https://garciasevilla.com/signos/juego/`

A.3.2. Deletreador

Versión inversa. El usuario escribe una palabra del español, o elige una letra, y el avatar la interpreta usando el alfabeto dactilológico.

- `https://garciasevilla.com/signos/mano/dactilo.html`

A.3.3. Cuadro fonológico de configuraciones de la mano

Distribución de configuraciones de la mano, reminiscente de los cuadros de fonemas de la lengua oral, utilizando los rasgos fonológicos para distribuir las distintas configuraciones de manera ordenada.

- `https://garciasevilla.com/signos/tabla/`

A.3.4. "PICAM"s interactivos

Visualización interactiva del sistema "Signotación" para transcribir de manera fonológica las configuraciones de la mano.

- `https://garciasevilla.com/signos/interactive/`

[1] `https://creativecommons.org/licenses/by-sa/4.0/`

Bibliografía

Aarons, Debra (1996). «Topics and topicalization in American sign language». En: *Stellenbosch Papers in Linguistics* 30.1, págs. 65-106.

Amraee, Somaieh, Maryam Chinipardaz, Mohammadali Charoosaei y Mohammad Amin Mirzaei (2022). «Handwritten Logic Circuits Analysis Using the YOLO Network and a New Boundary Tracking Algorithm». En: *IEEE Access* 10, págs. 76095-76104. DOI: 10.1109/access.2022.3192467.

Aronoff, Mark (1993). *Morphology by itself: Stems and inflectional classes*. MIT press.

Aung, Ni Htwe, Ye Kyaw Thu, Su Su Maung, Swe Zin Moe y Hlaing Myat Nwe (2020). «Transfer Learning Based Myanmar Sign Language Recognition for Myanmar Consonants». En: *Journal of intelligent informatics and smart technology* 4, pág. 10.

Barberà Altimira, Gemma (2015). *The meaning of space in sign language: reference, specificity and structure in Catalan sign language discourse*. Sign languages and deaf communities 4. De Gruyter Moutkr Ishara Press. ISBN: 978-1-61451-866-2.

Benbasat, Ari Y. y Joseph A. Paradiso (2002). «An inertial measurement framework for gesture recognition and applications». En: *Gesture and Sign Language in Human-Computer Interaction: International Gesture Workshop, GW 2001 London, UK, April 18–20, 2001 Revised Papers*. Springer, págs. 9-20.

Bouzid, Yosra y Mohamed Jemni (2014). «A Virtual Signer to Interpret SignWriting». En: *Computers Helping People with Special Needs*. Vol. 8548. Springer International Publishing, págs. 458-465. ISBN: 978-3-319-08598-2.

Bragg, Danielle, Oscar Koller, Naomi Caselli y William Thies (2020). «Exploring Collection of Sign Language Datasets: Privacy, Participation, and Model Performance». En: *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, págs. 1-14. ISBN: 978-1-4503-7103-2. DOI: 10.1145/3373625.3417024.

Branchini, Chiara y Lara Mantovan (2020). *A Grammar of Italian Sign Language (LIS)*. Fondazione Università Ca' Foscari. ISBN: 978-88-6969-474-5. DOI: 10.30687/978-88-6969-474-5.

Brentari, Diane (2019). *Sign Language Phonology*. Cambridge University Press.

Camgoz, Necati Cihan, Simon Hadfield, Oscar Koller, Hermann Ney y Richard Bowden (2018). «Neural Sign Language Translation». En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, págs. 7784-7793.

Carmen Cabeza-Pereiro, María del, José Mª Garcia-Miguel, Carmen García Mateo y José Luis Alba Castro (2016). «CORILSE: a Spanish Sign Language Repository for Linguistic Analysis». En: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. European Language Resources Association (ELRA), págs. 1402-1407.

Carneiro, Alvaro Leandro Cavalcante, Lucas de Brito Silva y Denis Henrique Pinheiro Salvadeo (2021). «Efficient sign language recognition system and dataset creation method based on deep learning and image processing». En: *arXiv:2103.12233 [cs]*. arXiv: 2103.12233.

Cassidy, Steve, Onno Crasborn, Henri Nieminen, Wessel Stoop, Micha Hulsbosch, Susan Even, Erwin Komen y Trevor Johnston (2018). «Signbank: Software to Support Web Based Dictionaries of Sign Language». En: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Cooper, Helen, Brian Holt y Richard Bowden (2011). «Sign language recognition». En: *Visual Analysis of Humans*. ISBN: 9781538609309, págs. 1122-1126.

Crespo Vidal, Ana Belén (2020). «Análisis lingüístico de elementos prosódicos de la lengua de signos mediante hand tracking». (no publicado). Tesis de mtría. Universidad Complutense de Madrid, Facultad de Filología.

Deacon, Terrence William (1997). *The symbolic species: The co-evolution of language and the brain*. 202. WW Norton & Company.

Deng, J., K. Li, M. Do, H. Su y L. Fei-Fei (2009). «Construction and Analysis of a Large Scale Image Ontology». En: Vision Sciences Society.

Di Renzo, Alessio, Luca Lamano, Tommaso Lucioli, Barbara Pennacchi y Luca Ponzo (2006). «Italian Sign Language (LIS): can we write it and transcribe it with SignWriting?» En: *Proceedings of the 2nd Workshop on the Representation and processing of Sign Languages: Lexicografic matters and didactic scenarios – LREC*, págs. 11-16.

Eccarius, Petra y Diane Brentari (2008). «Handshape coding made easier: A theoretically based notation for phonological transcription». En: *Sign Language & Linguistics* 11.1, págs. 69-101. ISSN: 1387-9316, 1569-996X. DOI: 10.1075/sll.11.1.11ecc.

Elliott, Ralph, John RW Glauert, JR Kennaway, Ian Marshall y Eva Safar (2008). «Linguistic modelling and language-processing technologies for Avatar-based sign language presentation». En: *Universal access in the information society* 6, págs. 375-391.

Fang, Gaolin, Wen Gao, Xilin Chen, Chunli Wang y Jiyong Ma (2002). «Signer-independent continuous sign language recognition based on SRN/HMM». En: *Gesture and Sign Language in Human-Computer Interaction: International Gesture Workshop, GW 2001 London, UK, April 18–20, 2001 Revised Papers*. Springer, págs. 76-85.

Filhol, Michael y Annelies Braffort (2007). «Description lexicale des signes, intérêts linguistiques d'un modèle géométrique à dépendances». En: *Revue TAL* 48, 27p.

Filhol, Michael, John McDonald y Rosalee Wolfe (2017). «Synthesizing sign language by connecting linguistically structured descriptions to a multi-track animation system». En: *Universal Access in Human–Computer Interaction. Designing Novel Interactions: 11th International Conference, UAHCI 2017, Held as Part of HCI International 2017, Vancouver, BC, Canada, July 9–14, 2017, Proceedings, Part II*. Springer, págs. 27-40.

Forster, Jens, Christoph Schmidt, Oscar Koller, Martin Bellgardt y Hermann Ney (2014). «Extensions of the Sign Language Recognition and Translation Corpus RWTH-PHOENIX-Weather.» En: *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'14)*, págs. 1911-1916.

Fragkiadakis, Manolis y Peter van der Putten (2021). «Sign and Search: Sign Search Functionality for Sign Language Lexica». En: *arXiv:2107.13637 [cs]*. arXiv: 2107.13637.

Freeman, William T y Michal Roth (1995). «Orientation histograms for hand gesture recognition». En: *International workshop on automatic face and gesture recognition*. Vol. 12, págs. 296-301.

Galea, Maria (2014). «SignWriting (SW) of Maltese Sign Language (LSM) and its development into an orthography: Linguistic considerations». Tesis doct. University of Malta.

Goldsmith, John Anton (1976). «Autosegmental phonology». Tesis doct. Massachusetts Institute of Technology.

Granlund, Gösta H. y Hans Knutsson (1995). *Signal processing for computer vision*. Kluwer Academic Publishers.

Gutierrez-Sigut, Eva, Brendan Costello, Cristina Baus y Manuel Carreiras (2016). «LSE-Sign: A lexical database for Spanish Sign Language». En: *Behavior Research Methods* 48, págs. 123-137. ISSN: 1554-3528.

Hadjadj, Mohamed, Michael Filhol y Annelies Braffort (2018). «Modeling French Sign Language: a proposal for a semantically compositional system». En: *International Conference on Language Resources and Evaluation*.

Hanke, Thomas (2004). «HamNoSys – Representing Sign Language Data in Language Resources and Language Processing Contexts». En: *Proceedings of the Workshop on Representation and Processing of Sign Language, Workshop to the fourth International Conference on Language Resources and Evaluation (LREC'04)*. ISSN: 17913721, págs. 1-6.

Hanke, Thomas, Marc Schulder, Reiner Konrad y Elena Jahn (2020). «Extending the Public DGS Corpus in size and depth». En: *sign-lang@ LREC 2020*. European Language Resources Association (ELRA), págs. 75-82.

Hassan, Saad, Larwan Berke, Elahe Vahdani, Longlong Jing, Yingli Tian y Matt Huenerfauth (2020). «An isolated-signing RGBD dataset of 100 American Sign Language signs produced by fluent ASL signers». En: *Proceedings of the LREC2020 9th Workshop on the Representation and Processing of Sign Languages: Sign Language Resources in the Service of the Language Community, Technological Challenges and Application Perspectives*, págs. 89-94.

Herrero Blanco, Ángel (2003). *Escritura alfabética de la Lengua de Signos Española: once lecciones*. Publicaciones de la Universidad de Alicante.

– (2009). *Gramática didáctica de la Lengua de Signos Española (LSE)*. Ediciones SM.

Hilzensauer, Marlene y Klaudia Krammer (2015). «A multilingual dictionary for sign languages: "SpreadTheSign"». En: *ICERI2015, the 8th annual International Conference of Education, Research and Innovation*.

Hofstadter, Douglas (2018). *The Shallowness of Google Translate*. `https://www.theatlantic.com/technology/archive/2018/01/the-shallowness-of-google-translate/551570/`.

Holler, Anke y Markus Steinbach (2018). «Sign language agreement: A constraint-based perspective». En: *Proceedings of the 25th International Conference on Head-Driven Phrase Structure Grammar*, págs. 49-67.

Hulst, Harry van der (2022). «The (early) history of sign language phonology». En: *The Oxford Handbook of the History of Phonology*, Chapter 14.

Isaac Caswell y Bowen Liang (2020). *Recent Advances in Google Translate*. `http://ai.googleblog.com/2020/06/recent-advances-in-google-translate.html`.

Jimoh, Kudirat O, Anuoluwapo O Ajayi e Ibrahim K Ogundoyin (2020). «Template Matching Based Sign Language Recognition System for Android Devices». En: *FUOYE Journal of Engineering and Technology* 5, pág. 7.

Kennaway, Richard (2004). «Experience with and Requirements for a Gesture Description Language for Synthetic Animation». En: *Gesture-Based Communication in Human-Computer Interaction*. Vol. 2915. Springer Berlin Heidelberg, págs. 300-311. ISBN: 978-3-540-21072-6. DOI: 10.1007/978-3-540-24598-8_28.

Kimmelman, Vadim (2015). «Topics and topic prominence in two sign languages». En: *Journal of Pragmatics* 87, págs. 156-170. ISSN: 03782166. DOI: 10.1016/j.pragma.2015.08.004.

Kipp, Michael, Alexis Heloir y Quan Nguyen (2011). «Sign Language Avatars: Animation and Comprehensibility». En: *Intelligent Virtual Agents*. Vol. 6895. Springer Berlin Heidelberg, págs. 113-126. ISBN: 978-3-642-23973-1 978-3-642-23974-8. DOI: 10.1007/978-3-642-23974-8_13.

Kocab, Annemarie, Ann Senghas y Jennie Pyers (2022). «From Seed to System: The Emergence of Non-Manual Markers for Wh-Questions in Nicaraguan Sign Language». En: *Languages* 7.2, pág. 137. ISSN: 2226-471X. DOI: 10.3390/languages7020137.

Koller, Oscar, Hermann Ney y Richard Bowden (2013). «May the force be with you: Force-aligned SignWriting for automatic subunit annotation of corpora». En: *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*. DOI: 10.1109/fg.2013.6553777.

Krebs, Julia, Gerda Strutzenberger, Hermann Schwameder, Ronnie B Wilbur, Evie Malaia y Dietmar Roehm (2021). «Event visibility in sign language motion: Evidence from Austrian Sign Language». En: *Proceedings of the annual meeting of the Cognitive Science Society*. Vol. 43.

Krizhevsky, Alex, Ilya Sutskever y Geoffrey E. Hinton (2017). «ImageNet classification with deep convolutional neural networks». En: *Communications of the ACM* 60.6, págs. 84-90.

Kruger, Norbert, Peter Janssen, Sinan Kalkan, Markus Lappe, Ales Leonardis, Justus Piater, Antonio J. Rodríguez-Sanchez y Laurenz Wiskott (2013). «Deep Hierarchies in the Primate Visual Cortex: What Can We Learn for Computer Vision?» En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8, págs. 1847-1871. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2012.272.

Kuprieiev, Ruslan, Dmitry Petrov, Saugat Pachhai, Pawel Redzyński, Casper da Costa-Luis, Alexander Schepanovski, Peter Rowlands, Ivan Shcheklein, Jorge Orpinel, Batuhan Taskaya, Fábio Santos, Aman Sharma, Gao, Zhanibek, Dani Hodovic,

Andrew Grigorev, Earl, Nikita Kodenko, Nabanita Dash, George Vyshnya, Maykulkarni, Max Hora y Vera (2021). *DVC: Data Version Control - Git for Data & Models*. DOI: 10.5281/zenodo.5037865.

Lahoz-Bengoechea, José María y Antonio F. G. Sevilla (2022a). «Parámetros fonológicos y uso de variables para buscar formas flexivas en un diccionario de Lengua de Signos Española». En: *39º Congreso internacional de la Asociación Española de Lingüística Aplicada (AESLA)*, págs. 243-244.

– (2022b). «Signotación: una transcripción fonológica para el tratamiento computacional de la Lengua de Signos Española». En: *L Simposio y IV Congreso Internacional de la Sociedad Española de Lingüística*, pág. 29.

Langer, Jiří, Jan Andres, Martina Benešová y Dan Faltýnek (2020). *Quantitative linguistic analysis of Czech sign language*. 1.ª ed. Univerzita Palackého v Olomouci. DOI: 10.5507/pdf.20.24457277.

Ley 27/2007, de 23 de octubre, por la que se reconocen las lenguas de signos españolas y se regulan los medios de apoyo a la comunicación oral de las personas sordas, con discapacidad auditiva y sordociegas (2007). Boletín Oficial del Estado. BOE núm. 255, de 24 de octubre de 2007. Gobierno de España.

Li, Yuncheng, Zehao Xue, Yingying Wang, Liuhao Ge, Zhou Ren y Jonathan Rodríguez (2022). *End-to-End 3D Hand Pose Estimation from Stereo Cameras*. Inf. téc. arXiv:2206.01384. arXiv. DOI: 10.48550/arXiv.2206.01384.

Liddell, Scott K. y Robert E. Johnson (1989). «American Sign Language: The Phonological Base». En: *Sign Language Studies* 1064.1, págs. 195-277. ISSN: 1533-6263. DOI: 10.1353/sls.1989.0027.

Lin, Tsung-Yi, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick y Piotr Dollár (2015). *Microsoft COCO: Common Objects in Context*. arXiv: 1405.0312.

Liu, Xiaodong, Luc Vermeylen, David Wisniewski y Marc Brysbaert (2020). «The contribution of phonological information to visual word recognition: Evidence from Chinese phonetic radicals». En: *Cortex* 133, págs. 48-64.

Lombardo, Vincenzo, Cristina Battaglino, Rossana Damiano y Fabrizio Nunnari (2011). «An avatar-based interface for the Italian sign language». En: *2011 International Conference on Complex, Intelligent, and Software Intensive Systems*. IEEE, págs. 589-594.

Lu, Shan, Seiji Igi, Hideaki Matsuo y Yuji Nagashima (1998). «Towards a dialogue system based on recognition and synthesis of Japanese sign language». En: *Lecture*

Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 1371. ISBN: 3540644245, págs. 259-271. ISSN: 16113349. DOI: 10.1007/BFb0053005.

Lucas, Ceil y Clayton Valli (1989). «Language contact in the American deaf community». En: *The sociolinguistics of the Deaf community*. Elsevier, págs. 11-40.

Ludeña, Verónica López (2014). «Diseño, desarrollo y evaluación de sistemas de traducción automática para reducir las barreras de comunicación de las personas sordas». Tesis doct. Universidad Politécnica de Madrid.

Makarov, Ilya, Nikolay Veldyaykin, Maxim Chertkov y Aleksei Pokoev (2019). «Russian Sign Language Dactyl Recognition». En: *2019 42nd International Conference on Telecommunications and Signal Processing (TSP)*, págs. 726-729. DOI: 10.1109/TSP.2019.8768868.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu y Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*.

Matsuo, Hideaki, Seiji Igi, Shan Lu, Yuji Nagashima, Yuji Takata y Terutaka Teshima (1998). «The recognition algorithm with non-contact for Japanese Sign Language using morphological analysis». En: *Gesture and Sign Language in Human-Computer Interaction*, págs. 273-284. DOI: 10.1007/bfb0053006.

Miller, Christopher (2001). «Section I: Some reflections on the need for a common sign notation». En: *Sign Language & Linguistics* 4.1-2, págs. 11-28.

Miyazaki, Taro, Yusuke Morita y Masanori Sano (2020). «Machine translation from spoken language to Sign language using pre-trained language model as encoder». En: *Proceedings of the LREC2020 9th workshop on the representation and processing of sign languages: sign language resources in the service of the language community, technological challenges and application perspectives*, págs. 139-144.

Moreno, David Sánchez (2012). «Proyecto DILSE III: primer diccionario normativo de la Lengua de Signos Española». En: *Estudios sobre la Lengua de Signos Española: III congreso nacional de Lengua de Signos Española: hacia la normalización de un derecho*

lingüístico y cultural, Madrid 2009. UNED, Universidad Nacional de Educación a Distancia, págs. 297-310.

Moulton, Ryan (2020). *Why Deep Learning Works Even Though It Shouldn't*. https://moultano.wordpress.com/2020/10/18/why-deep-learning-works-even-though-it-shouldnt/.

O'Mahony, Niall, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan y Joseph Walsh (2020). «Deep Learning vs. Traditional Computer Vision». En: *Advances in Computer Vision*. Springer International Publishing, págs. 128-144. DOI: 10.1007/978-3-030-17795-9_10.

Ong, Sylvie C W y Surendra Ranganath (2005). «Automatic sign language analysis: A survey and the future beyond lexical meaning». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.6. ISBN: 01628828 (ISSN), págs. 873-891. ISSN: 01628828. DOI: 10.1109/TPAMI.2005.112.

Parkhurst, Stephen J. y Dianne D. Parkhurst (2001). *Variación de las Lenguas de Signos usadas en España*. Revista Española de Lingüística de las Lenguas de Signos.

Parvini, Farid, Dennis McLeod, Cyrus Shahabi, Bahareh Navai, Baharak Zali y Shahram Ghandeharizadeh (2009). «An approach to glove-based gesture recognition». En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5611 LNCS.PART 2. ISBN: 3642025765, págs. 236-245. ISSN: 03029743. DOI: 10.1007/978-3-642-02577-8_26.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai y Soumith Chintala (2019). «PyTorch: An Imperative Style, High-Performance Deep Learning Library». En: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., págs. 8024-8035.

Peñalba Acitores, Alicia, Carlos Herminio Moriyón Mojica y Sonia Luque Perea (2018). «Más que sonido: interpretación de música instrumental en lengua de signos para las personas sordas». En: *Tabanque. Revista Pedagógica* 31, págs. 94-107. ISSN: 2530-6766. DOI: 10.24197/trp.31.2018.94-107.

Plann, Susan (1997). *A silent minority: Deaf education in Spain, 1550-1835*. University of California Press.

Porta-Lorenzo, Manuel, Manuel Vázquez-Enríquez, Ania Pérez-Pérez, José Luis Alba-Castro y Laura Docío-Fernández (2022). «Facial Motion Analysis beyond Emotional Expressions». En: *Sensors* 22.10. ISSN: 1424-8220. DOI: 10.3390/s22103839.

Quer, Josep, Roland Pfau y Annika Herrmann, eds. (2021). *The Routledge Handbook of Theoretical and Experimental Sign Language Research*. ISBN: 9781138801998.

Redmon, Joseph (2013). *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/.

Redmon, Joseph, Santosh Divvala, Ross Girshick y Ali Farhadi (2016). «You Only Look Once: Unified, Real-Time Object Detection». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*, págs. 779-788.

Redmon, Joseph y Ali Farhadi (2018). *YOLOv3: An Incremental Improvement*. arXiv: 1804.02767.

Rocha Costa, Antônio Carlos da y Graçaliz Pereira Dimuro (2002). «SignWriting-Based Sign Language Processing». En: *Gesture and Sign Language in Human-Computer Interaction*, págs. 202-205.

Rodríguez, Jefferson y Fabio Martínez (2021). «How important is motion in sign language translation?» En: *IET Computer Vision* 15.3, págs. 224-234. ISSN: 1751-9640. DOI: https://doi.org/10.1049/cvi2.12037.

Rodríguez Redondo, Ana Laura, Sarah Díaz-Wengelin, Gemma Píriz, Mª José Carmona, Carmen González y Gloria Fernández (2008). «Cognitive Strategies for the Development of Visuo-Gestural and Motor Abilities in Hearing Adult Learners of Spanish Sign Language as Second Language». En: *Series A: General & Theoretical Papers* 726. ISSN: ISSN 1435-6473.

Rumi, Roisul Islam, Syed Moazzim Hossain, Ahmed Shahriar y Ekhwan Islam (2019). «Bengali Hand Sign Language Recognition Using Convolutional Neural Networks». Bachelor's thesis.

Sambasivan, Nithya, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Kumar Paritosh y Lora Mois Aroyo (2021). «"Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI». En: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, págs. 1-15.

Sánchez Jiménez, John Byron, Samuel López Prieto y José Ángel Garrido Montoya (2019). «Reconocimiento de lenguaje Signo-Escritura mediante Deep Learning». Trabajo de fin de grado. Universidad Complutense de Madrid, Facultad de Informática.

Sandler, Wendy, Mark Aronoff, Carol Padden e Irit Meir (2014). «Language Emergence: Al-Sayyid Bedouin Sign Language». En: *Cambridge handbook of linguistic anthropology*. Cambridge Handbooks of Linguistics, págs. 250-284. ISBN: 978-1-139-34287-2.

Santiago, Darío Tilves, Carmen García Mateo, Soledad Torres Guijarro, Laura Docío Fernández y José Luis Alba Castro (2019). «Estudio de bases de datos para el reconocimiento automático de lenguas de signos». En: *Hesperia: Anuario de filología hispánica* 22, págs. 145-160.

Senghas, Ann y Marie Coppola (2001). «Children creating language: how Nicaraguan Sign Language acquired a spatial grammar». En: *Psychological Science* 12.4, págs. 323-328.

Senghas, Richard J, Ann Senghas y Jennie E Pyers (2005). «The emergence of Nicaraguan Sign Language: Questions of development, acquisition, and evolution». En: *Biology and knowledge revisited: From neurogenesis to psychogenesis*. Publisher: Lawrence Erlbaum Associates Mahwah, NJ, págs. 287-306.

Sevilla, Antonio F. G. (2015). «An online collaborative platform for the development of empirical grammars». Tesis de mtría. Univerzita Karlova, Matematicko-fyzikální fakulta y University of Malta, Faculty of ICT.

Sevilla, Antonio F. G., Alberto Díaz y José María Lahoz-Bengoechea (2022). «Quevedo: Annotation and Processing of Graphical Languages». En: *Proceedings of the Workshop on Representation and Processing of Sign Language, Workshop to the fourth International Conference on Language Resources and Evaluation (LREC'22)*.

– (2023). «Automatic SignWriting Recognition: Combining Machine Learning and Expert Knowledge to Solve a Novel Problem». En: *IEEE Access* 11, págs. 13211-13222. DOI: 10.1109/ACCESS.2023.3242203.

Sevilla, Antonio F. G. y José María Lahoz-Bengoechea (2019). «A different description of orientation in sign languages». En: *Procesamiento del Lenguaje Natural, 62* 62, págs. 53-60. ISSN: 1135-5948.

Sevilla, Antonio F. G., José María Lahoz-Bengoechea y Alberto Díaz (2022). *VisSE corpus of Spanish SignWriting*. https://zenodo.org/record/6337885. Ver. 2.0.0.

Shterionov, Dimitar, Vincent Vandeghinste, Horacio Saggion, Josep Blat, Mathieu De Coster, Joni Dambre, Henk Van den Heuvel, Irene Murtagh, Lorraine Leeson e Ineke Schuurman (2021). «The signon project: a sign language translation framework». En: *the 31st Meeting of Computational Linguistics in the Netherlands*.

Slevinski, Steve (2016). *The SignPuddle Standard for SignWriting Text*. https://datatracker.ietf.org/doc/html/draft-slevinski-signwriting-text. Internet-Draft.

Smith, Ray W. (2013). «History of the Tesseract OCR engine: what worked and what didn't». En: *Document Recognition and Retrieval XX*. Vol. 8658. Spie, págs. 1-12. DOI: 10.1117/12.2010051.

Stamp, Rose, Svetlana Dachkovsky y Wendy Sandler (2020). «Time will tell: Time and discourse as 'motion through space' in early Israeli Sign Language (ISL)». En: *Our Lives–Our Stories*. De Gruyter Mouton, pág. 28.

Starner, T. y A. Pentland (1995). «Real-time American Sign Language recognition from video using hidden Markov models». En: *Proceedings of International Symposium on Computer Vision - ISCV*, págs. 265-270. DOI: 10.1109/ISCV.1995.477012.

Starner, Thad, Joshua Weaver y Alex Pentland (1998). «Real-Time American Sign Language Recognition Using Desk and Wearable Computer Based Video». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.12, págs. 1371-1375.

Stokoe, William C. (1960). «Sign Language Structure: An Outline of the Visual Communication Systems of the American Deaf». En: *Studies in linguistics: Occasional papers* 8.

Sutton, Valerie (1995). *Lessons in SignWriting: Textbook*. Center for Sutton Movement Writing.

Sutton, Valerie y Adam Frost (2008). *SignWriting: sign languages are written languages!* Center for Sutton Movement Writing.

Sutton, Valerie y Stephen Slevinski (2010). *International SignWriting Alphabet, HTML Reference*. http://www.signbank.org/iswa/.

Sutton-Spence, Rachel y Bencie Woll (1999). *The linguistics of British Sign Language: an introduction*. Cambridge University Press.

Technical Committee: ISO/IEC JTC 1 Information technology (2012). *Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 2: Superstructure*. Standard ISO/IEC 19505-2:2012. International Organization for Standardization.

The Unicode Consortium (2021). *The Unicode Standard: Version 14.0.0*. Unicode Consortium. ISBN: 978-1-936213-29-0.

Thiessen, Stuart M. (2011). «A Grammar of SignWriting». Tesis de mtría. University of North Dakota.

Tkachman, Oksana (2016). «The status of third person pointing signs in American Sign Language (ASL)». En: *University of British Columbia Working Papers in Linguistics* 44.

Trettenbrein, Patrick C y Emiliano Zaccarella (2021). «Controlling video stimuli in sign language and gesture research: The OpenPoseR package for analyzing OpenPose motion-tracking data in R». En: *Frontiers in Psychology* 12, pág. 628728.

Tyrone, Martha E, Hosung Nam, Elliot Saltzman, Gaurav Mathur y Louis Goldstein (2010). «Prosody and movement in American Sign Language: A task-dynamics approach». En: *Speech Prosody 2010-Fifth International Conference*.

«Una herramienta para traducir la SignoEscritura, la lengua de signos escrita» (2022). En: *La Vanguardia*.

Utray, Francisco y Esther Gil (2014). «Diversidad cultural, lengua de signos y televisión en España». En: *Fonseca: Journal of Communication*, págs. 118-143. ISSN: 2172-9077.

Van der Hulst, Harry y Rachel Channon (2010). «Notation systems». En: *Sign languages*. Cambridge University Press, págs. 151-172.

Vázquez-Enríquez, Manuel, Jose L Alba-Castro, Laura Docío-Fernández y Eduardo Rodríguez-Banga (2021). «Isolated sign language recognition with multi-scale spatial-temporal graph convolutional networks». En: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, págs. 3462-3471.

Vegas Cañas, Sara, Miguel Rodríguez Cuesta y Alejandro Torralbo Fuentes (2020). «Text2LSE: Traductor de texto a Lengua de Signos Española (LSE)». Trabajo de fin de grado. Universidad Complutense de Madrid, Facultad de Informática.

Verdu Perez, Elena, Begoña Cristina Pelayo García-Bustelo, María Ángeles Martínez Sánchez, Rubén González Crespo et al. (2017). «A system to generate SignWriting for video tracks enhancing accessibility of deaf people». En: *International Journal of Interactive Multimedia and Artificial Intelligence, 4 (6)*, págs. 109-115.

Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht y Oriol Vinyals (2021). «Understanding deep learning (still) requires rethinking generalization». En: *Communications of the ACM* 64.3, págs. 107-115. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3446776.