Automatic SignWriting Recognition

Antonio F. G. Sevilla[a,b,*], Alberto Díaz Esteban[a], José María Lahoz-Bengoechea[b]

[a] *Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid, Facultad de Informática, c/ Profesor José García Santesmases, 9 28040 Madrid, Spain*
[b] *Spanish Linguistics and Literary Theory, Universidad Complutense de Madrid, Facultad de Filología edificio D, c/ Prof. Aranguren s/n, 28040 Madrid, Spain*

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 automatic processing poses a novel challenge for computational linguistics. We identify as first and fundamental step to overcome this challenge the extraction of a computational representation of the semantics represented by SignWriting transcriptions. We propose a data-based modelization of the problem, construed from real handwritten SignWriting instances. We then propose two solutions involving state of the art machine learning techniques combined with expert analysis. The first solution is direct application of an existing deep neural network. Our second proposal exploits the expert knowledge codified in the data annotation scheme that we present, in order to craft a system that improves on the straight-forward solution's accuracy by 30%. This improved system uses a number of different neural networks to divide the necessary processing, progressively constructing the prediction in an iterative pipeline that combines deep learning and domain knowledge in a mixed solution.

Keywords: Sign Language, SignWriting, Deep Learning, Expert Knowledge, Neural Networks, Computer Vision

*Corresponding author.
Email addresses: afgs@ucm.es (Antonio F. G. Sevilla), albertodiaz@ucm.es (Alberto Díaz Esteban), jmlahoz@ucm.es (José María Lahoz-Bengoechea)

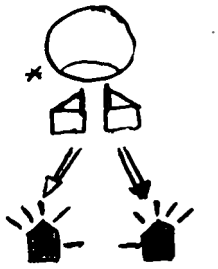


Figure 1: SignWriting transcription for the sign "History", showing the hands near the chin in an initial configuration and moving down and sideways to a finishing configuration. A video of this sign can be found online at `https://www.spreadthesign.com/es.es/search/?q=historia` (Hilzensauer & Krammer (2015)).

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. 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. 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 (see Stokoe (1960), Herrero Blanco (2003), Hanke (2004)). 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 & Frost (2008)).

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. 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 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 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 colocation 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. To be able to computationally process SignWriting, techniques from artificial intelligence and computer vision are required, so that pixels and colours are converted to abstract symbols along their symbolic and spatial meaning.

First, a formal representation of the contents of a SignWriting image is needed. We present one such representation, going beyond location information of the symbols, also classifying them with sets of features that codify their meaning. This representation is realized in an annotation schema, which we apply to real-world handwritten transcriptions of signs.

To be then able to automatically extract this representation, we also present

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

two possible solutions, and comparatively evaluate their performance. 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. The first solution uses a single such neural network to solve the full task. Improving on the issues that this first solution has, a second system further utilizes the expert knowledge encoded in our data annotation. In this second solution, neural networks are arranged in a pipeline, with rules coming from domain knowledge used in between to reduce the complexity of the problem and make it more tractable.

The rest of the paper is structured as follows: in section 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 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 4 discusses the underlying data engineering, and section 5 explicates our two proposed solutions. These are evaluated and compared in section 6, and finally, section 7 gives our conclusions and outlines future lines of research and development that can be followed.

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

[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.

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 (see for example Liddell & Johnson (1989), Herrero Blanco (2009), Branchini & Mantovan (2020) or Langer et al. (2020)).

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 & Frost (2008)) is a naturalistic system based on iconicity rather than linguistic categories. 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 & 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 & 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. Figure 2 shows a few hand shapes and orientations along their SignWriting representation.

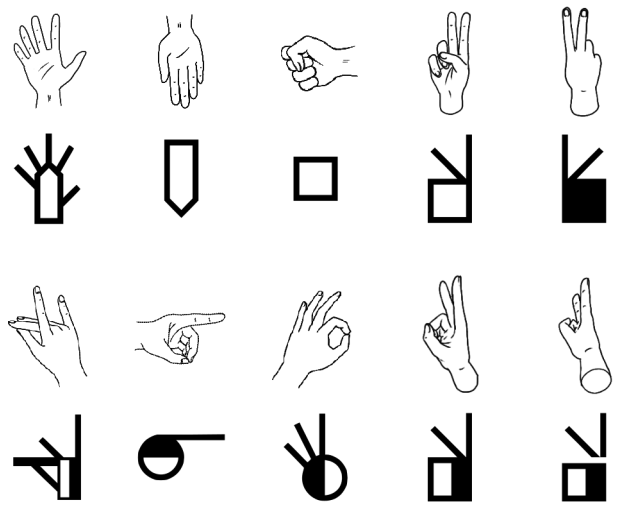


Figure 2: Some hand graphemes, comprising different hand shapes and orientations. Based on Sevilla & Lahoz-Bengoechea (2019).

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 2.
3. The hand grapheme can then be rotated, to iconically represent the rotation as observed from the chosen point of view.
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

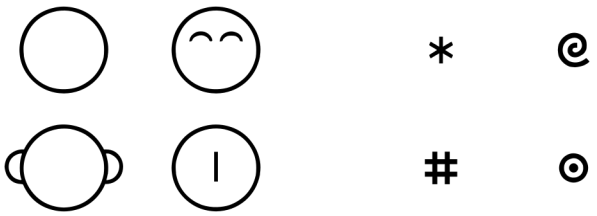


Figure 3: Examples of non-rotating graphemes. On the left, some body parts (on the head) are shown. Clockwise from top left: generic face, eyes, nose and ears. On the right, diacritics marking different types of contact are show. Clockwise from top left: touch, rub, brush and strike.

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 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 3.

Lastly, hands are often not static, but rather move in space, and these movements can have meaning in themselves. In SigWriting, 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

[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.

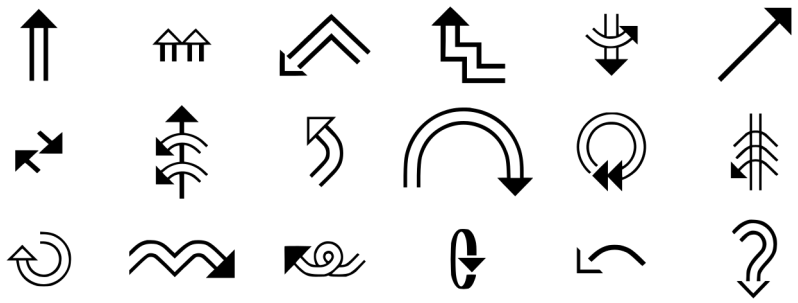


Figure 4: A selection of movement markers in SignWriting. Each of these is a different character in SignWriting fonts, but we can think of it as composed of smaller units: straight or curved segments, single or double stems, and black, white or empty arrow heads.

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 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 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 (2014), or online at `https://www.signwriting.org/`.

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.

So instead of using a ready-made OCR system, or OCR technologies that can be adapted to our problem, we need to use the underlying technology: computer vision.

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 state of the art before the uprise of deep learning can be read in Granlund & 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.

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 et al. (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.

YOLO networks (You Only Look Once) 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 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.

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.

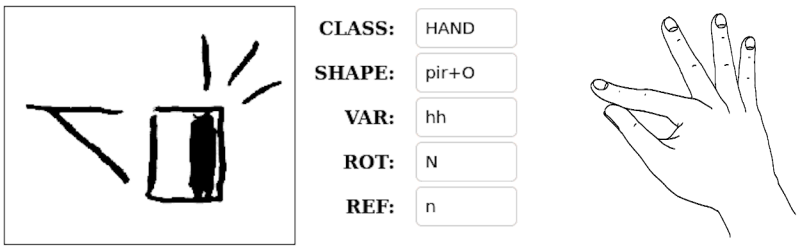


Figure 5: Annotation for a hand grapheme, showing the different labels and values that need to be assigned. To its right, an illustration of the depicted hand shape is shown.

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 for us. The Deaf community is small relative to the general population, and the majority don't use SignWriting at all. Furthermore, data is generally not readily prepared for scientific use. 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 processable 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 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.

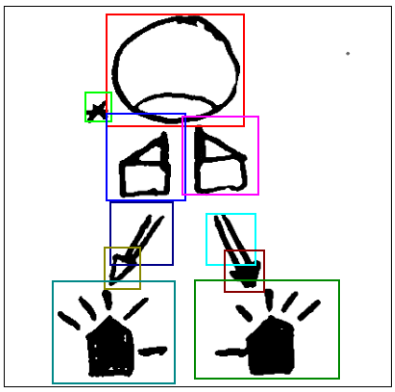
A first label, “CLASS”, divides graphemes into 6 classes of symbols, with related graphical properties and similar semantic information: “HEAD” for head symbols, “DIAC” for small diacritics, often used for contact information; “HAND” for graphemes which represent hands, and “ARRO”, “STEM” and “ARC” for 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 5, have an additional label, “VAR”, which represents the wrist rotation (whether the hand is white, black, or half and half, see section 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 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.

We have collected and annotated 608 handwritten logograms, within which 4232 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 6.



	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

Figure 6: Annotation of the logogram for the sign "History".

5. Proposed solutions

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 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.

We propose two solutions. The first one uses YOLO, the detector and classifier neural network architecture, which can solve the full task with one single network. The second solution addresses some limitations of the first, combining additional networks into a pipeline, which allow us to exploit the expert knowledge encoded into the corpus annotations.

5.1. One-shot Solution

YOLO networks, as introduced in section 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 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 & Farhadi (2018), as implemented in `https://github.com/AlexeyAB/darknet`.

We will examine the results later, in section 6, but this one-shot solution will prove to be not enough for our problem. Essentially, there is too much complexity in our problem for the single neural network to solve it successfully.

5.2. Pipeline Solution

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 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 data. 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 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 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,

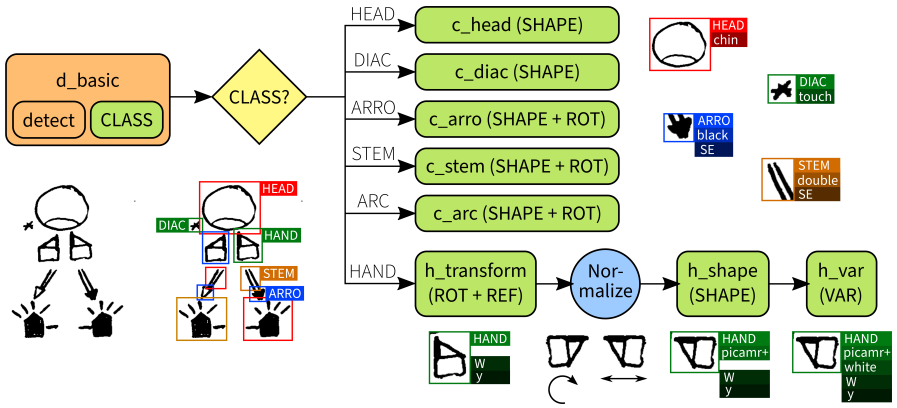


Figure 7: Full pipeline architecture, including the different networks, decisions and processes. An example logogram and some of the detected graphemes are shown as they progress through the pipeline.

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.

The full pipeline for SignWriting can be seen in Figure 7, with an example logogram for reference.

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

$$\text{Detection Accuracy} = \frac{\text{correct detections}}{\text{total detections} + \text{missed detections}} \tag{1}$$

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 2).

$$\text{Classification Accuracy} = \frac{\text{correctly classified}}{\text{correct detections}} \tag{2}$$

Finally, a combined measure is computed, overall accuracy (equation 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.

$$\text{Overall Accuracy} = \frac{\text{correctly classified}}{\text{total detections} + \text{missed detections}} \tag{3}$$

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

Table 1: Performance of our two proposed solutions for the task of SignWriting recognition: grapheme detection and classification within logograms.

Solution	Detection	Classification	Overall
Single YOLO	0.45	0.83	0.37
Pipeline	0.73	0.65	0.48

measurements are evaluated on this test set. There are 496 logograms in the training set, and 112 in the test set, resulting in a total of 3491 graphemes in the training set and 741 in the test one.

The results of computing these metrics for our two proposed solutions are shown in Table 1. The overall accuracy of the single YOLO network is of 0.37, which may seem low but is impressive for the complexity of the problem. The pipeline solution manages to increase performance by 30%, to an overall accuracy of 0.48. This number may seem low, since it means slightly less than half of the graphemes are being correctly predicted, but human analysis of the pipeline results paints a slightly more optimistic view. Often, the pipeline errs in ways 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, 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.

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 most of the 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 low 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 the 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

Table 2: Detection, classification and overall accuracy of the two solutions, computed for each grapheme CLASS.

	One-shot			Pipeline		
CLASS	Det.	Class.	Overall	Det.	Class.	Overall
ARRO	0.56	0.84	0.47	0.72	0.80	0.58
HEAD	0.60	0.58	0.35	0.90	0.63	0.56
DIAC	0.61	0.89	0.54	0.71	0.77	0.55
STEM	0.58	0.84	0.49	0.68	0.71	0.48
HAND	0.21	0.48	0.10	0.85	0.39	0.33
ARC	0.18	0.50	0.09	0.49	0.11	0.05

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 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 0.21 by the single YOLO network to 0.85 by the pipeline, and the overall accuracy improvement from 0.10 to 0.33.

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.

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 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 in numerical positions of graphemes within a logogram and key-value pairs of features for each of them.

To automatically reproduce this representation for new instances of SignWriting, the best approach is using 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.

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 two systems for automatically recognizing SignWriting. A simple, direct approach using a single YOLO network can deliver some useful results, and serves as a useful baseline with which to compare. A second system uses a number of deep learning networks combined in a branching pipeline. Further steps utilize knowledge extracted from previous steps to facilitate further processing, possible thanks

to the annotation schema decided for the data, and significantly improving over the first solution's performance.

The main difficulty of our problem lies in the sparseness and complexity of the data. There are many different graphemes 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 graphemes 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.

The annotation of data into a hierarchical scheme, however, lets us partition the problem into a series of sub-tasks, each easier than the full task. By reducing the features to extract at each step, the search space is made more dense, and entropy of the problem lowered. Not only are each of these sub-tasks consequently easier to solve, but extracted information can be processed, informing the next step in the pipeline so that it does not have to start from zero.

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.

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. A second direction to go is downstream, putting the recognized SignWriting representation to use in 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, or of downstream applications.

Acknowledgements

The development of the expert system described in this paper is part of the project "Visualizando la SignoEscritura" (Visualizing SignWriting), 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.

References

Branchini, C., & Mantovan, L. (2020). *A Grammar of Italian Sign Language (LIS).* Fondazione Università Ca' Foscari. doi:10.30687/978-88-6969-474-5.

Eccarius, P., & Brentari, D. (2008). Handshape coding made easier: A theoretically based notation for phonological transcription. *Sign Language & Linguistics*, *11*, 69–101. doi:10.1075/sll.11.1.11ecc.

Granlund, G. H., & Knutsson, H. (1995). *Signal processing for computer vision.* Kluwer Academic Publishers.

Hanke, T. (2004). HamNoSys – Representing Sign Language Data in Language Resources and Language Processing Contexts. In *Proceedings of the Workshop on Representation and Processing of Sign Language, Workshop to the forth International Conference on Language Resources and Evaluation (LREC'04)* (pp. 1–6). ISSN: 17913721.

Herrero Blanco, Á. (2003). *Escritura alfabética de la lengua de signos española: once lecciones.* Publicaciones de la Universidad de Alicante.

Herrero Blanco, Á. (2009). *Gramática didáctica de la lengua de signos española (LSE).* Ediciones SM.

Hilzensauer, M., & Krammer, K. (2015). A multilingual dictionary for sign languages: "SpreadTheSign". *ICERI2015 Proceedings*, (pp. 7826–7834).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*, 84–90. doi:10.1145/3065386.

Langer, J., Andres, J., Benešová, M., & Faltýnek, D. (2020). *Quantitative lingustic analysis of Czech sign language.* (1st ed.). Univerzita Palackého v Olomouci. doi:10.5507/pdf.20.24457277.

Liddell, S. K., & Johnson, R. E. (1989). American Sign Language: The Phonological Base. *Sign Language Studies*, *1064*, 195–277. doi:10.1353/sls.1989.0027.

O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., Riordan, D., & Walsh, J. (2020). Deep Learning vs. Traditional Computer Vision. In K. Arai, & S. Kapoor (Eds.), *Advances in Computer Vision* (pp. 128–144). Springer International Publishing. doi:10.1007/978-3-030-17795-9_10.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. (pp. 779–788). URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. arXiv:1804.02767.

Sambasivan, N., Kapania, S., Highfill, H., Akrong, D., Paritosh, P. K., & Aroyo, L. M. (2021). "Everyone wants to do the model work, not the data work": Data Cascades in High-Stakes AI. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1–15).

Sevilla, A. F. G., & Lahoz-Bengoechea, J. M. (2019). A different description of orientation in sign languages. *Procesamiento del Lenguaje Natural, 62*, (pp. 53–60).

Smith, R. W. (2013). History of the Tesseract OCR engine: what worked and what didn't. In *Document Recognition and Retrieval XX* (pp. 1–12). SPIE volume 8658. doi:10.1117/12.2010051.

Stokoe, W. C. (1960). Sign Language Structure: An Outline of the Visual Communication Systems of the American Deaf. *Studies in linguistics: Occasional papers*, *8*.

Sutton, V. (2014). *Lessons in sign writing: Textbook*. (4th ed.). Center for Sutton Movement Writing. URL: `https://www.signwriting.org/archive/docs2/sw0116-Lessons-SignWriting.pdf`.

Sutton, V., & Frost, A. (2008). *SignWriting: sign languages are written languages!*. Center for Sutton Movement Writing.