Bachelor thesis
Computing Science

Radboud University

Detecting Fingerspelling in Sign Language Videos Using Pose Information

Author:
Tobias van der Werff
t.n.vanderwerff@student.ru.nl
s1011178

First supervisor/assessor:
Prof. Martha Larson
m.larson@cs.ru.nl

[Second supervisor:]
MSc. Simge Ekiz
s.ekiz@student.ru.nl

Second assessor:
Prof. Onno Crasborn
o.crasborn@let.ru.nl

June 28, 2020

Abstract

In this research, we present an approach for detecting fingerspelling in sign language videos using pose information. Given a sign language video, pose information for each frame is extracted using OpenPose, an open-source pose detection library. We use template matching to compare the hand area of each frame to reference images for fingerspelling signs. Using the template matching results, density-based clustering is used to predict the time slots at which fingerspelling occurs. Our approach is primarily aimed at assisting language researchers who work with annotated sign data. Using our approach, the time slots at which fingerspellings occur could be found automatically, after which an annotator could verify the predicted time slots and annotate them. At the same time, our approach can also be applied for other purposes, such as helping to build educational tools for sign language learners, or helping to annotate sign language datasets more quickly for machine learning systems within the domains of sign language recognition, generation and translation. Using a challenging dataset, the Corpus NGT, we are able to detect an average of 74% of fingerspelling segments, using a lower bound on the average precision of 10%.

Contents

Chapter 1

Introduction

Sign language is a mode of communication used primarily by individuals who are deaf or hard of hearing. Its usage is widespread and many countries have their own native sign language, such as American Sign Language, British Sign Language, or Nederlandse Gebarentaal (NGT) — the Dutch native sign language. These languages are very much independent languages. For example, NGT does not have a direct connection to the Dutch language and does not have a standard written form. Since approximately 5% of the world population is deaf or hard of hearing[1], sign language can provide an important alternative to spoken language for a large amount of people.

Fingerspelling is a part of sign language where letters from the spoken language orthography are signed using hand gestures. For example, in the case of NGT, fingerspelling involves spelling out words from written Dutch. Fig. 1.1 shows the hand alphabet for Dutch sign language. One common reason for using fingerspelling is to spell out words that do not have their own signs.

For language researchers who wish to annotate sign language videos, finding and labeling fingerspellings in such videos can be costly and time-consuming. This is particularly the case for NGT, where fingerspelling is relatively rare, as opposed to other sign languages such as American Sign Language, where fingerspelling makes up anywhere from 12 to 35 percent of signed interaction [20]. Therefore, automatic fingerspelling detection, which involves identifying when a signer is signing letters, as opposed to other types of signs, such as word or number signs, could potentially speed up the annotation process. However, current methods for recognition of fingerspelling "in the wild", i.e. using videos containing naturally occurring sign language rather than sign language in a controlled studio environment, are still lacking. [23].

Another way in which automatic fingerspelling detection can be useful is

[1] https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss Accessed 29-02-2020.

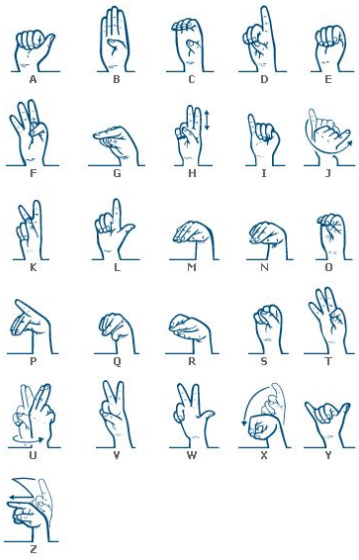


Figure 1.1: The NGT hand alphabet[2]. Note that the letters H, J, U, X and Z involve motion, whereas the other letters are static.

for building educational tools. People that learn NGT at a later age, such as sign language translators in training, often find fingerspelling easy to learn, but hard to read and understand [9]. Creating a collection of fingerspelling examples by detecting them in sign language videos could provide such sign language learners with more practice material to learn from.

What's more, within the fields of sign language recognition, generation and translation, faster annotation of datasets could help to improve existing machine learning systems that are dependent on high quantities of training data to work well. This is relevant given the current lack of large-scale annotated sign language data within these fields [2]. Existing public sign language datasets often have shortcomings that limit the power and generalizability of systems trained on them; these shortcomings include small size of the datasets, lack of signer variety and a lack of "real-life" applicability. One of the reasons why it is hard to create a simple sign language datasets is that variation exists in the execution of sign language, not only between different languages, but also within languages. Just as differences in dialects and articulation exist in spoken language, differences in execution of sign language exist among signers. There are many sources of variation within sign language: "sociolinguistic factors" such as region, dialect, age, but also "fonetic variables" such as placement of words in sentences, and the signs preceding and/or following signs [9]. Fluency can also play a role

[2]Image from: `https://geocachen.nl/geocaching/geocache-puzzels-oplossen/gebarentaal/`

— sign language users who learn sign language at a later age are typically less fluent than those who acquired it from a younger age [17].

In this research, we propose a method for detecting fingerspellings in sign language videos that requires only a minimal amount of labeled data. This is done using templates based on pose information, i.e. the position and orientation of a person in a video. The part of the frame that contains the signing hand is extracted, normalized and compared to a set of letter templates. Based on this comparison, a prediction can be made as to whether or not the frame potentially contains a fingerspelling sign. By clustering together potential fingerspelling frames, we attempt to detect fingerspelling segments — sequences of frames that contain one or multiple fingerspelling signs. Our memory-based approach works around the problem of limited data by using a comparatively small set of templates to detect fingerspellings. Another way in which we address the issue of limited sign language data is by using OpenPose, an open-source pose detection library, in combination with template matching, which allows us to achieve a domain transfer from the field of pose recognition to that of sign language recognition. We focus on detecting fingerspelling in NGT, but the approach may also be applied to other sign languages. We analyze the way in which a template set can be formed efficiently, along with an analysis of the results that different template sets produce.

There are three main reasons why the template matching approach may be preferable to other approaches:

1. *Smaller need for labeled data.* Template sets only require a handful of labeled data in order to be effective, as shown in Chapter 5. This is one of the driving factors of the approach laid out in this research. In contrast with more data-driven approaches, we consider an approach that uses only a small set of labeled data as a reference for what fingerspelling looks like.

2. *Speed.* Template matching is a computationally cheap operation, which means it can provide high performance in terms of computing cost. If the OpenPose output for a sign language video can be computed in advance, our system can provide a way to detect fingerspellings quickly.

3. *Transparency.* Finding the cause of failure cases based on template matching is often relatively easy, in comparison to model-based approaches such as neural networks, which provide less insight into the decision-making process.

The research question that is central in this thesis is: *How can fingerspelling be detected in sign language videos using templates based on pose information?*. In order to answer this question, we consider the following sub-questions:

- Under which conditions does the template matching approach work?
- What is an efficient way to create templates?

The chapters in this research are laid out as follows. Chapter 2 provides background information regarding sign language and also an overview of the OpenPose system. In Chapter 3 we lay out the approach that is used for detecting fingerspelling. Then, the experimental framework that is used for testing the approach is laid out in Chapter 4. Chapter 5 explains the experimental tests that are performed in order to answer the central research questions, including a discussion of the results. Chapter 6 contains an overview of related work, and finally, Chapter 7 contains the conclusions that are drawn based on the results.

Chapter 2

Background

In this chapter we provide background information for the subjects that are central in this research. First, an introduction to sign language and specifically fingerspelling is given in Section 2.1. Then, the OpenPose keypoint detection system is elaborated in Section 2.2, along with the OpenPose hand keypoint detection in Section 2.3. The sections on OpenPose are not required reading to understand this research, but provide a technical understanding of the inner workings of the pose detection system that is used as part of a template matching pipeline in this research.

2.1 Sign language

Sign languages such as NGT are natural languages used by, but not limited to, people who are deaf or hard of hearing. Communication via sign language is not limited to the hands — it can involve a wide range of bodily expressions using for example the eyes and facial expressions.

For sign language, it is possible to make a distinction between manual and non-manual ways of articulation — the former are mainly used for producing words, the latter for producing intonation [9]. Manual features include all gestures that are made with the hands by using hand shape and motion. Non-manual features describe features such as body posture, head posture and facial expression [5]. This distinction can be applied to all sign languages. Besides language-specific words, many sign languages also have a way to spell out words from spoken language by using manual ways of articulation, also known as fingerspelling. Fig. 1.1 shows the hand alphabet for NGT and Fig. 2.1 shows an example of a fingerspelling sign. As shown in Fig. 1.1, there are two types of fingerspellings: static and non-static, which means there are signs based on motion and signs based on fixed hand formations. For example, the letter Z consists of drawing the shape of a Z with the index finger, whereas the letter L is formed only by forming an L-shaped hand sign. The NGT alphabet contains twenty-one static, and five

Figure 2.1: The NGT fingerspelling sign for the letter W formed with the right hand, shown in a YouTube instruction video for fingerspelling [1].

non-static fingerspelling signs. Due to the limited scope of this research, our main focus concerns static fingerspellings.

There are several reasons why fingerspelling may be used in sign language; we list some of these use cases here. It should be noted that this list is not exhaustive, especially considering that relatively little research exists on the usage of fingerspelling in NGT [9]. When a signer wants to use a word that does not have its own sign, she can spell out the word using fingerspelling. In this sense fingerspelling plays a compensatory role since it fills a gap in sign language vocabulary. Another common use of fingerspelling is spelling names. For example, a person introducing herself can use fingerspelling to spell out her own name. However, fingerspelling is not always necessary for spelling names, since names can also have their own signs. Fingerspelling can also be used to disambiguate signs that are not clearly understood. Finally, another, although less common reason for fingerspelling, is to put emphasis on specific words. In this case, a lexical sign is often combined with fingerspelling in order to specifically emphasize a word [10].

In NGT, fingerspelling seems to take a comparatively small role. Based on observations of a subset of the NGT sign language dataset that is used in this research (introduced in Section 4.1), we note a low occurrence of fingerspelling, making up roughly 1% of the frames in the videos (the actual number may be even lower since the subset of the data that is used is specifically selected to only contain videos in which fingerspelling occurs). The frequency of fingerspelling may also vary based on context: a congenial conversation between signers about the weather would probably require less fingerspelling than a sign language news translator talking about international organisations that do not have their own signs.

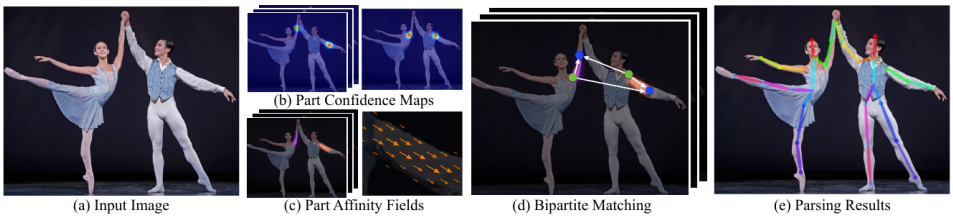


Figure 2.2: OpenPose pipeline, taken from [4].

2.2 OpenPose

For detecting pose information, we use OpenPose in this research. OpenPose is an open-source library for multi-person 2D pose detection, which includes body, foot, hand and facial keypoints [3]. Instead of employing a top-down approach where a person detector is used after which single-person pose estimation is performed, OpenPose employs a bottom-up approach using Part Affinity Fields (PAFs), a set of 2D vector fields that encode the location and orientation of limbs over the image domain [4]. The overall pipeline of OpenPose is illustrated in Fig. 2.2. In order to properly understand the pipeline, we first discuss the concept of Part Affinity Fields, after which we discuss the pipeline further.

2.2.1 Part Affinity Fields

Part Affinity Fields provide a representation for limbs that encodes both location and orientation information. This representation provides a way to encode the association between body parts, such that individually detected body parts can be linked together as limbs, which consist of specific body part combinations. An example of a Part Affinity Field that links the left shoulder and left elbow together into its corresponding limb (the upper arm) is shown in Fig. 2.2c. In this way, PAFs can be used to associate body part combinations with individuals in an image.

Specifically, Part Affinity consists of 2D vector fields for each limb. Each limb is formed by specific body part combinations. For each pixel in the area belonging to a particular limb, a 2D vector encodes the direction that points from one of the limb's body part to its associated other body part.

In order to predict PAFs on new images, a set of groundtruth PAFs needs to be formed as training data for the pose detection system. To provide an example of the way such a PAF is formed, consider the limb shown in Fig. 2.3. Let $\mathbf{x}_{j_1,k}$ and $\mathbf{x}_{j_2,k}$ be the groundtruth body parts j_1 and j_2 from limb c for person k in the image. Then let $\mathbf{L}^*_{c,k} \in \mathbb{R}^{w \times h \times 2}$ be the Part Affinity Field, consisting of 2D vectors for each pixel in the image, where w and h denote width and height, respectively. Whenever a point $\mathbf{p}$ lies on the area covering limb c of person k, the value at $\mathbf{L}^*_{c,k}(\mathbf{p})$ is a unit vector pointing

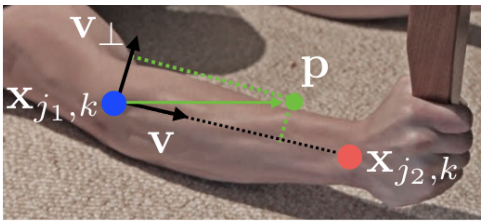


Figure 2.3: An example of a limb connected by two body parts at $\mathbf{x}_{j_1,k}$ and $\mathbf{x}_{j_2,k}$. The PAF indicates the direction the limb is pointing in. Taken from [4].

from j_1 to j_2. For points not covering the limb, the vector has a value of zero. Thus:

$$\mathbf{L}^*_{c,k}(\mathbf{p}) = \begin{cases} \mathbf{v} & \text{if } \mathbf{p} \text{ on limb } c,k \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{2.1}$$

The unit vector $\mathbf{v} = (\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k}) / \|\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k}\|_2$ indicates the direction of the limb. The set of points on the limb is defined as those within a distance threshold of the line segment, based on the width of the limb.

The groundtruth Part Affinity Field used for training is then defined as the average of the Affinity Fields of all people in the image:

$$\mathbf{L}^*_c(\mathbf{p}) = \frac{1}{n_c(\mathbf{p})} \sum_k \mathbf{L}^*_{c,k}(\mathbf{p}), \tag{2.2}$$

where $n_c(\mathbf{p})$ is the number of non-zero vectors at point $\mathbf{p}$ across all k people. This takes into account the parts of the image where limbs of different people overlap.

2.2.2 OpenPose pipeline

The system takes as input a color image of size $w \times h$ (Fig. 2.2a). After running the image through the pipeline, the system produces the 2D locations of body parts for each person in the image. The process can be roughly divided into two parts:

1. Simultaneous detection and association using convolutional neural networks;

2. Multi-person parsing using bipartite graph matching.

Simultaneous detection and association

In order to predict body part locations, a neural network is used. Neural networks are computational models, roughly based on the organizational structure of the brain. More specifically, a convolutional neural network (CNN) is used, a neural network often used for image processing. The CNN is used to predict a set of confidence maps $\mathbf{S}$ of body part locations and a set of part affinities $\mathbf{L}$ (Fig. 2.2b and 2.2c). The confidence map $\mathbf{S}_j \in \mathbb{R}^{w \times h}$, $j \in \{1, \ldots, J\}$ encodes the degree of confidence that a given pixel contains body part j, for each pixel in the image. This can also be seen as a representation of the belief that a particular body part occurs at each pixel location. The Part Affinity Field $\mathbf{L}_c \in \mathbb{R}^{w \times h \times 2}$, $c \in \{1, \ldots, C\}$ consists of 2D vector fields encoding the degree of association between parts, for every limb c.

Instead of predicting the confidence maps and Part Affinity Field separately, they are predicted simultaneously using a two-branch multi-stage CNN, shown in Fig. 2.4. This is based on the architecture laid out in [25], a sequential architecture using multiple CNNs that captures long-range spatial dependencies between variables by using a large receptive field. The upper and lower branches are responsible for predicting the confidence maps and Affinity Fields, respectively. This is an iterative procedure, where predictions from the CNN are refined over multiple stages, $t \in \{1, \ldots, T\}$. At each stage, predictions from both branches are concatenated along with the image features $\mathbf{F}$ for the next stage. In this manner, predictions of the confidence maps and PAFs are formed over multiple stages:

$$\mathbf{S}^t = \rho^t(\mathbf{F}, \mathbf{S}^{t-1}, \mathbf{L}^{t-1}), \forall t \geq 2, \tag{2.3}$$

$$\mathbf{L}^t = \phi^t(\mathbf{F}, \mathbf{S}^{t-1}, \mathbf{L}^{t-1}), \forall t \geq 2, \tag{2.4}$$

where ρ^t and ϕ^t are the CNNs for inference at stage t.

Multi-person parsing using bipartite graph matching

After obtaining the part confidence maps and Affinity Fields, the detected part candidates still need to be paired together with other part candidates to represent connected limbs. An example of detected body parts is shown in Fig. 2.5a.

In order to measure the association between different body parts, the line integral over the corresponding PAF is computed, along the line segment connecting the candidate parts. This means the alignment between the PAF and the candidate limb that is formed by connecting the two candidate body parts is measured. For two candidate part locations $\mathbf{d}_{j_1}$ and $\mathbf{d}_{j_2}$, the confidence in their association is measured as follows:

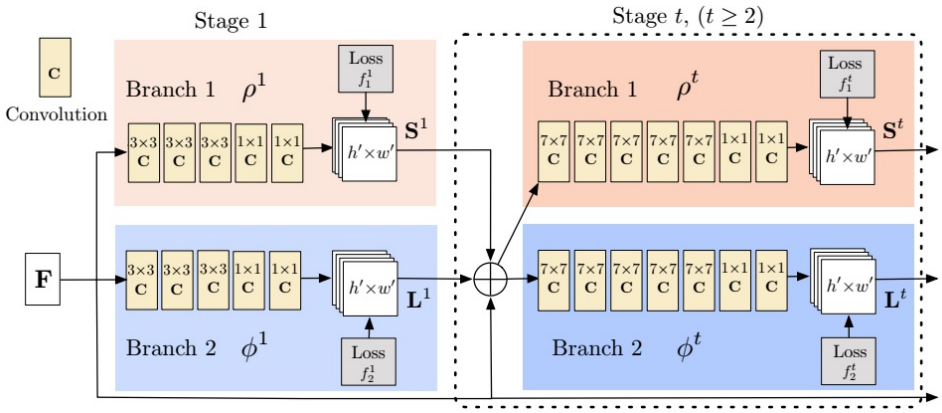


Figure 2.4: Architecture of the two-branch multi-stage CNN, where predictions from both branches are concatenated along with the image features **F** for the next stage. The first branch (in beige) predicts confidence maps, the second branch (in blue) predicts PAFs. Taken from [4].

$$E = \int_{u=0}^{u=1} \mathbf{L}_c(\mathbf{p}(u)) \cdot \frac{\mathbf{d}_{j_2} - \mathbf{d}_{j_1}}{\|\mathbf{d}_{j_2} - \mathbf{d}_{j_1}\|_2} du, \tag{2.5}$$

where $\mathbf{p}(u)$ interpolates the body parts $\mathbf{d}_{j_1}$ and $\mathbf{d}_{j_2}$.

Formally, a set of body part detection candidates $\mathcal{D}_\mathcal{J}$ is obtained for multiple people, where $\mathcal{D}_\mathcal{J} = \left\{\mathbf{d}_j^m : \text{for } j \in \{1, \ldots, J\}, m \in \{1, \ldots, N_j\}\right\}$, with N_j the number of candidates of part j, and $\mathbf{d}_j^m \in \mathbb{R}^2$ the location of the m-th detection candidate of body part j. To represent matchings between body part candidates, let $z_{j_1,j_2}^{mn} \in \{0, 1\}$ indicate whether two detection candidates $\mathbf{d}_{j_1}^m$ and $\mathbf{d}_{j_2}^n$ are connected.

Pair-wise matching of all detected body part candidates can be formulated as a maximum weight bipartite graph matching problem, where body part detection candidates represent the nodes of the graph and the edges represent all possible connections between pairs of detection candidates. A matching indicates a subset of edges chosen in such a way that no two edges share a node. A maximum matching is a matching of maximum cardinality, i.e. a matching M such that for any matching M', $|M| \geq |M'|$ [6]. Additionally, the edges are weighted by Eq. 2.5, representing a pair-wise association score. The goal is to find the optimal assignment for the set of all possible connections, $\mathcal{Z} = \{z_{j_1,j_2}^{mn} : \text{for } j_1, j_2 \in \{1, \ldots, J\}, m \in \{1, \ldots, N_{j_1}\}, n \in \{1, \ldots, N_{j_2}\}\}$. This means finding a matching with maximum weight for the chosen edges,

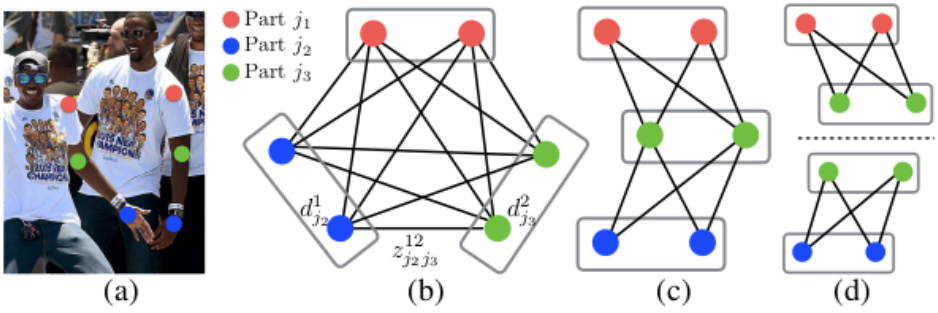


Figure 2.5: Graph matching. Taken from [4].

$$\max_{\mathcal{Z}_c} E_c = \max_{\mathcal{Z}_c} \sum_{m \in \mathcal{D}_{j_1}} \sum_{m \in \mathcal{D}_{j_2}} E_{mn} \cdot z^{mn}_{j_1,j_2}, \tag{2.6}$$

$$\text{s.t. } \forall m \in \mathcal{D}_{j_1}, \sum_{n \in \mathcal{D}_{j_2}} z^{mn}_{j_1,j_2} \leq 1, \tag{2.7}$$

$$\forall n \in \mathcal{D}_{j_2}, \sum_{m \in \mathcal{D}_{j_1}} z^{mn}_{j_1,j_2} \leq 1, \tag{2.8}$$

where E_c is the overall weight of the matching from limb type c, $\mathcal{Z}_c$ is the subset of $\mathcal{Z}$ for limb type c, and E_{mn} is the Part Affinity between parts $\mathbf{d}^m_{j_1}$ and $\mathbf{d}^n_{j_2}$ defined in Eq. 2.5. Equations 2.7 and 2.8 enforce that no two edges share a node, ensuring that no two limbs of the same type share a part. The optimal matching is obtained using the Hungarian algorithm [16]. Some domain specific details make this matching less computationally expensive than it usually is. First of all, instead of considering the complete graph, a minimal number of edges can be chosen to obtain a spanning tree skeleton of human pose. This is shown in Fig. 2.5c. Furthermore, the matching problem can be decomposed into a set of bipartite matching sub-problems which can be solved independently, as shown in Fig. 2.5d.

The optimization then comes down to:

$$\max_{\mathcal{Z}} E = \sum_{c=1}^{C} \max_{\mathcal{Z}_c} E_c. \tag{2.9}$$

Using the obtained set of pair-wise assignments, the full-body poses of multiple people can be constructed.

2.3 OpenPose hand keypoint detection

OpenPose combines both hand and body keypoint detection. The hand keypoint detection system is based on [24]. In order to predict hand keypoints,

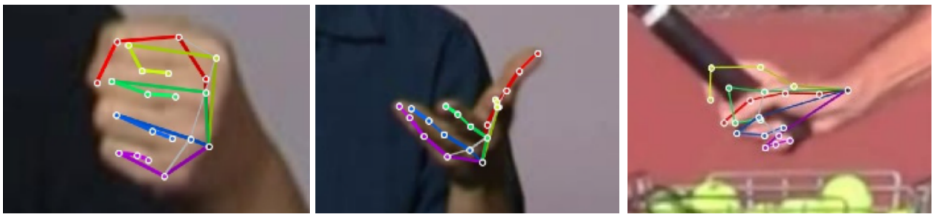


Figure 2.6: Hand keypoint detection in 2D images is made more difficult by occlusions, e.g. by (a) articulations of other parts of the hand, (b) a particular viewing angle, or (c) objects that the hand is grasping. Taken from [24].

the location of the hand must first be predicted. An important thing to note here is that in order to predict the hand region, wrist and elbow position are used as references for the hand position. The wrist and elbow position is estimated using the body keypoint detector laid out in Section 2.2.2. This means the elbow and wrist must be visible in the input image in order for the following method to work.

Hand keypoint detection can be hard to get right due to the fact that often times, parts of a hand in a 2D image are occluded. Some examples of this are shown in Fig. 2.6. This means certain keypoints that are not visible have to be estimated, which can result in a loss of accuracy. To address this problem of self-occlusion of the hand in 2D images, the OpenPose hand keypoint detector is created using 3D keypoint information. The underlying assumption is that even when some parts of a hand are not visible in an image, there is a high chance that there exists an alternative view of the hand that does show the hidden parts of the hand. An initial classifier d_0 produces hand keypoint estimations for 2D images of a hand, using multiple views of the same hand from different angles. These images are created using a multi-camera setup. 3D triangulation is then used to find the keypoints of the hand that were not always visible in some of the 2D images. The initial keypoint annotations can thus be improved by including keypoint detections of the same hand from a different angle. The newly generated annotations are used to train an improved classifier d_1. This process is shown in Fig. 2.7.

The hand keypoint detector $d(\cdot)$ takes as input an RGB image patch $\mathbf{I} \in \mathbb{R}^{w \times h \times 3}$, mapping P keypoint locations $x_p \in \mathbb{R}^2$, each with an associated detection confidence c_p:

$$d(\mathbf{I}) \mapsto \{(x_p, c_p) \text{ for } p \in [1 \dots P]\}. \tag{2.10}$$

Where each point p corresponds to a different part of the hand (e.g. base of the hand, tip of the index finger; see Fig. 3.2 for all the hand keypoints).

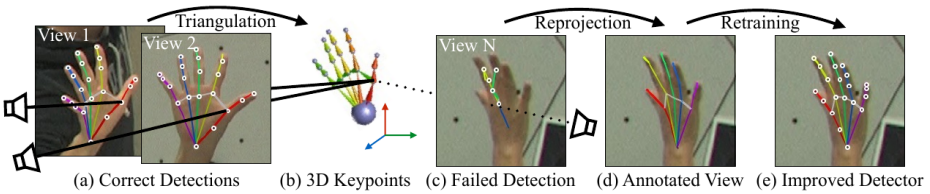


Figure 2.7: 3D triangulation to improve keypoint annotations. (a) A multi-view system provides views of the hand where keypoint detection can be done reliably, used to triangulate (b) the 3D position of the keypoints. Difficult views with (c) failed detections can be (d) improved using the 3D keypoints, and used to retrain (e) an improved detector. Taken from [24].

The keypoint detector is trained using images that contain corresponding keypoint annotations, $(\mathbf{I}^f, \{y_p^f\})$, where f denotes an image frame and the set $\{y_p^f \in \mathbb{R}^2\}$ includes all labeled keypoints for the image $\mathbf{I}^f$. The initial training set $\mathcal{T}_0$, having N_0 training pairs:

$$\mathcal{T}_0 := \{(\mathbf{I}^f, \{y_p^f\}) \text{ for } f \in [1 \dots N_0]\}, \tag{2.11}$$

is used to train the first classifier d_0, using stochastic gradient descent, a popular optimization algorithm that iteratively finds optimal values for the model parameters such that a given cost function is minimized:

$$d_0 \leftarrow \text{ train}(\mathcal{T}_0\}. \tag{2.12}$$

The keypoint detector d_0 produces a set of labeled images $\mathcal{T}_1$, which is then used to train an improved second classifier:

$$d_1 \leftarrow \text{ train}(\mathcal{T}_0 \cup \mathcal{T}_1\}. \tag{2.13}$$

This process can be repeated to iteratively create an improved classifier for hand keypoint detection, using annotated data generated from all of the previous iterations.

Chapter 3

Approach

In this chapter we lay out the template matching approach that is used to detect fingerspelling segments. In Section 3.1 we lay out our fingerspelling detection approach, illustrating the overall pipeline and the details of each step of this pipeline. In Section 3.2 we lay out the corresponding hyperparameters that have to be set.

3.1 Template matching for fingerspelling detection

The fingerspelling detection approach outlined in this research is based on rudimentary fingerspelling recognition at the level of individual frames of a video using template matching, combined with density-based clustering[1]. The complete fingerspelling detection pipeline is shown in Fig. 3.1. The system takes a sign language video as input (Fig. 3.1a) and predicts the time slots at which fingerspelling occurs (Fig. 3.1e). First, OpenPose detects the hand keypoints of the signing hand (Fig. 3.1b). After parsing out frames that are not suitable fingerspelling candidates, the signing hand is extracted by creating a bounding box around the hand and normalizing the keypoints (Fig. 3.1c). Then, the signing hand is compared to a set of templates; the hands that sufficiently resemble a template are marked (Fig. 3.1d). Finally, fingerspelling segments are predicted by applying density-based clustering to the marked frames (Fig. 3.1e). A fingerspelling segment refers to a time frame containing a minimum amount of closely grouped frames that matched a template. The underlying assumption of our method is that even though template matching is often not sufficient to accurately detect fingerspellings at the level of individual frames, we can combine template matching with density-based clustering to more accurately predict time slots at which fingerspelling occurs. In the following sections, we will discuss the details of the aforementioned pipeline.

[1]The code used for this research is available at `https://github.com/tobiasvanderwerff/fingerspelling-detection`

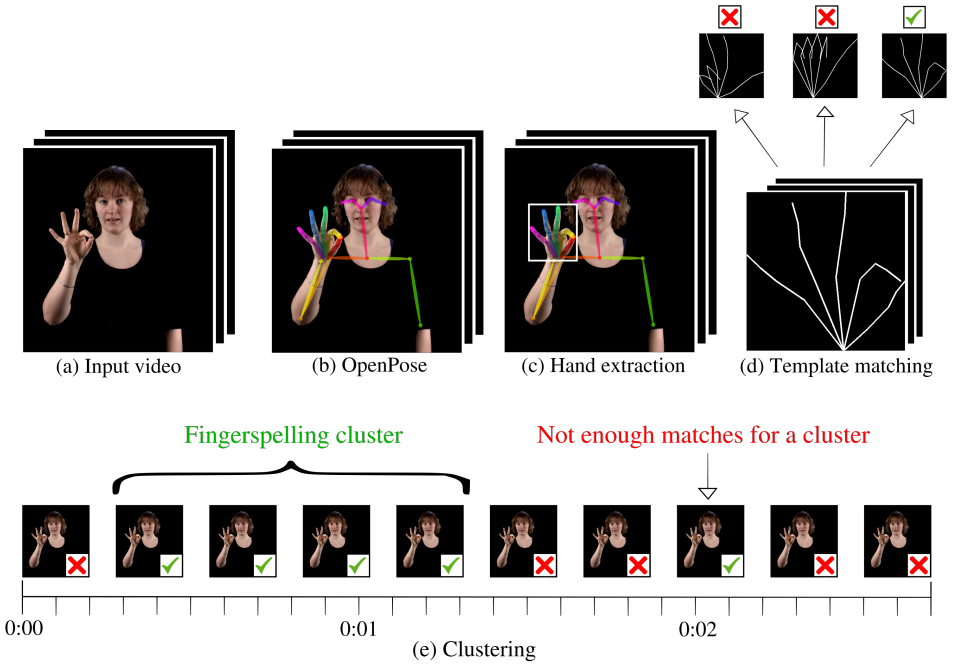


Figure 3.1: Overall pipeline. (a) Our method takes a video as input after which (b) OpenPose detects the body and hand keypoints for each frame. (c) After parsing out frames that are not suitable fingerspelling candidates, the hand is extracted and (d) compared to a set of templates to see if a match can be found. (e) Finally, clusters containing a minimum amount of template matched frames are marked as fingerspellings segments.

3.1.1 Detecting hand position and shape

We use OpenPose for detecting the position and shape of the signing hand. Given a frame of an input video, OpenPose estimates the body and hand keypoints of the signer in the video, extracting twenty-one keypoints per hand: one for the base of the hand and four for each finger, as shown in Fig. 3.2.

In this research we use OpenPose for detecting pose information because it produces state-of-the-art pose detection results and provides reliable results in most situations. One important caveat is that OpenPose relies on the wrist and elbow position to estimate the hand position. For example, when the wrist and elbow keypoints are not detected, the hand keypoints can also not be detected (see Section 2.3). Thus, in images where the wrist and/or elbow is not visible, OpenPose will not be able to detect hand keypoints. This affects the type of videos that can be analyzed using our method, e.g. OpenPose will fail on videos where only the hand is visible.

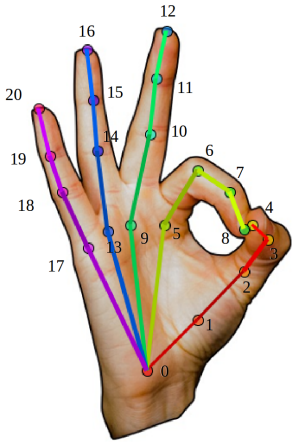


Figure 3.2: OpenPose hand keypoints, taken from [24].

3.1.2 Eliminating unsuitable frames using physical constraints

After detecting pose information for a given frame there are a number of ways in which a frame can be prematurely rejected as a potential fingerspelling. The first way in which a frame can be rejected pertains to a failure on the part of OpenPose to properly predict pose information. The remaining rules, which are listed below, can be seen as physical constraints on the part of the signer. The idea is that certain poses by the signer are unlikely to be fingerspellings and are therefore eliminated as possible fingerspelling candidates. Note that some of the physical constraints listed below may disregard some actual fingerspellings. The idea behind the physical constraints is to parse out as many unsuitable frames as possible, while removing a minimal amount of actual fingerspelling frames, if any. The remaining frames can then be processed for template matching. The utility of these constraints is shown in Section 5.4.

No hand keypoints detected

Due to a variety of reasons, OpenPose may not always be able to estimate the hand keypoints in the frame. One obvious example of this is when the signer moved his fingerspelling hand out of the frame; another might be that the wrist and/or elbow of the signing arm has not been detected, as explained in Section 2.3. Even when the elbow, wrist and hand are all visible in the frame, OpenPose may still not always be able to detect the hand keypoints properly. See [3] for some common OpenPose failure cases.

Fingerspelling hand is not above the other hand

Even though differences in fingerspelling execution exist across signers, there are some common patterns that can be discovered when analyzing fingerspelling. First, fingerspelling in NGT is performed with one hand (see Fig. 1.1). This means that when a signer is fingerspelling with the right hand, the left hand is commonly not used at the same time; because of this the left hand is generally in a resting position below the right hand. We include this constraint into the pipeline by rejecting frames whenever the fingerspelling hand is not higher than the other hand, or more specifically, when the lowest keypoint of the fingerspelling hand is not higher than the lowest keypoint of the other hand. This constraint can be especially useful for rejecting frames where a fingerspelling hand formation is used in conjunction with a sign in the left hand, as shown in Fig. 5.1.

Lowest keypoint of fingerspelling hand is below the elbow

Fingerspelling in NGT is commonly performed at shoulder height. Because of this, whenever the fingerspelling hand is relatively low in the frame, it is generally unlikely to be a fingerspelling. Therefore we reject all frames where the lowest keypoint of the fingerspelling hand is below its respective elbow.

Fingerspelling hand is not relatively still

Quick hand movements across the frame are unlikely to be fingerspellings, due to the static nature of most fingerspelling signs. This does not mean the fingerspelling hand is always perfectly still while fingerspelling, but we mainly want to reject those cases where hand movements are abrupt enough to significantly decrease the chance of fingerspelling signs occurring.

We measure the movement of the hand as Euclidean distance between the position of the hand base keypoint of the current frame and that of the previous frame. Let $b^t \in \mathbb{R}^2$ be the position of the hand base at frame t. Then,

$$m(t) = \left\| b^t - b^{t-1} \right\|_2 . \tag{3.1}$$

If $m(t)$ exceeds a movement threshold ϵ, we reject frame t. This threshold corresponds to the `movement_threshold` mentioned in Section 3.2, which is obtained by experimenting with different values for ϵ in order to see what produces the best results.

A disadvantage of this movement constraint is that repeated signs, meaning the same letter being signed twice in a row, may be missed. This is because signing the same fingerspelling sign twice in a row is done in NGT by moving the fingerspelling hand sideways while forming the fingerspelling

sign. On the other hand, since double letters in words are often used in combination with other letters, it is quite possible that this effect may be mitigated by the use of clustering as explained in 3.1.5.

3.1.3 Hand extraction

Extraction of the hand position and shape involves creating a bounding box around the fingerspelling hand and rescaling the keypoints to a 500×500 frame. Normalizing the hand keypoints allows us to more easily compare different hand formations, which is a prerequisite to the template matching performed in the following part of the pipeline.

3.1.4 Template matching

After the hand keypoints have been normalized, we perform template matching on the resulting keypoints. Template matching is a relatively straightforward way for detecting objects in images. The idea is to create a reference image (a template) for the object one wants to detect, and to measure the similarity between (parts of) an input image and the template image to see if the input image contains the object (i.e. if a part of the input image "matches" the template). Template matching is not computationally expensive and can be useful for detecting objects with a somewhat fixed structure.

Fingerspelling in NGT involves forming a particular one-handed sign, which generally involves no or very little movement of the hand. The mostly static nature of the fingerspelling signs leads us to believe that templates may be used for detecting them, by creating templates for each letter of the hand alphabet and comparing these templates to the hand region in an image. In combination with the OpenPose library for human pose detection with which we can reliably detect hand keypoints in images, templates might provide a simple and straight-forward way to detect fingerspelling.

Creating templates

The templates consist of twenty-one normalized hand keypoints that act as a reference for letter signs. Templates are created using pose information acquired by OpenPose. The template is formed by using the extracted hand keypoints to create a bounding box of the signing hand region, after which the keypoints are normalized. This corresponds to the steps shown in Fig. 3.1a-c, except that template creation is done for a single frame, not an entire video. The coordinates of the keypoints in the resulting bounding box illustrated in Fig 3.1c form the template. Note that even though the keypoints have been connected in Fig. 3.1c, the actual template consists only of the twenty-one hand keypoints as shown in Fig. 3.2. Also note that a one-to-one correspondence between the points of a template image and

that of the hand region of an input image has already been established by OpenPose, since OpenPose labels the hand keypoints as shown in Fig. 3.2. This greatly simplifies the template matching process.

Calculating template similarities

Given K keypoints, let $x, t \in \mathbb{R}^{K\times 2}$ represent the normalized input frame fingerspelling hand keypoint coordinates and a template, respectively. We derive a measure for dissimilarity using squared Euclidean distance. The lower the dissimilarity, the better the match between t and x is. This dissimilarity measure is weighted to address a practical issue that OpenPose does not always detect all hand keypoints properly. The OpenPose output includes confidence scores $s \in [0, 1]$ for every keypoint p, where 1 indicates the highest possible confidence and 0 the lowest. Whenever the confidence s_p for a keypoint p is too low ($s_p \leq 0.15$, which is the threshold maintained by OpenPose for showing a keypoint), we do not include it in the dissimilarity measure. The dissimilarity is then calculated by

$$D(x,t) = \frac{\sum_{k=2}^{K} c(x_k) \cdot \|x_k - t_k\|_2^2}{\sum_{k=2}^{K} c(x_k)}, \tag{3.2}$$

where $c(\cdot)$ is defined as

$$c(p) = \begin{cases} 1, & \text{if } s_p > 0.15. \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

We use squared Euclidean distance in order to give more weight to keypoint outliers. This is important since changes in only a few keypoints can determine the difference between a fingerspelling sign and a non-fingerspelling sign (see for example Fig. 5.3). We do not include the hand base keypoint in the score (keypoint 0 in Fig. 3.2), because this keypoint is generally not informative for calculating hand formation similarity, even though the normalization in the previous step can cause large differences in base keypoint position.

Given a set of templates S, the best matching template is selected based on the lowest dissimilarity:

$$s^* = \arg\min_{s \in S} D(x, s). \tag{3.4}$$

The resulting dissimilarity based on s^* is then compared to a predefined template match threshold τ, corresponding to the `template_match_threshold` mentioned in Section 3.2. This means s^* and x constitute a match when

$$D(x, s^*) \leq \tau. \tag{3.5}$$

3.1.5 Clustering

When each frame of an input video with n frames has been processed, we end up with a 1D array of size n, indicating for each frame whether it constituted a match with one of the templates from the template set. However, when a single frame has been marked as a template match, we do not yet conclude that a fingerspelling occurs in that frame. There are several reasons for this:

- *False positives.* A simple way to use template matching in order to detect fingerspelling would be to apply it to each frame of a video and conclude that fingerspelling occurs in each frame that matches a template. However, this approach is prone to false positives, i.e. predicting non-fingerspelling frames as fingerspellings. There are only a limited number of ways in which one can use the hand to form gestures, and there is bound to be some overlap between hand formations for fingerspelling and the gestures that are used for other non-fingerspelling gestures, or simply hand gestures that come about by accident. This can lead to an excessive amount of template matches for non-fingerspelling frames.

- *Templates easily miss small changes in fingerspelling.* Using templates for fingerspelling recognition means that only those gestures that match the ones used to create the templates are recognized, including the particular orientation of the hand when the template was made. Assuming the letter templates are based on a "standard" way of fingerspelling (e.g. the NGT alphabet shown in Fig. 1.1), fingerspellings containing small deviations from these standard letter signs may not be matched by a template. This can be seen as one of the main limitations of using templates for fingerspelling detection: fingerspellings that do not conform to any of the templates are missed. This can be mitigated in some sense by creating more templates such that variations of the fingerspelling signs are taken into account, but it is clear that there is a limit to the amount of variation that can be taken into account. For example, a fingerspelling sign may be turned towards the right in such a way that it does not form a match with the corresponding letter template. This means that a new template would have to be created to take into account this variation, where the hand is turned to the right. Including all these kinds of variations would take a lot of time and effort and may not lead to concrete improvements, due to the large number of possible variations. We explore the effect of adding more templates to cover possible variations in Section 4.4 and 5.1.

Due to the aforementioned reasons, template matching alone is generally insufficient as a method to detect fingerspelling. To mitigate the effect

of these inherent shortcomings of the template matching approach, we consider the context in which template matches have been detected by applying density-based clustering. Put simply, we look for a minimum number of template matches in a given time window in order to mark the time window as a fingerspelling segment. We then mark all of the frames within that segment as fingerspelling frames. At the same time, we do not mark template matches that are not part of a segment as fingerspelling frames. The underlying assumption is that the more template matches occur within a particular time frame, the higher the probability is that fingerspelling occurs in that time frame.

This clustering approach allows us to smooth out many of the false positives mentioned earlier, because we only mark frames as fingerspelling frames when they are part of a fingerspelling segment (see Section 5.5 for an illustration of how clustering reduces the amount of false positives). Note that such a segment can still be quite short and may only span a few frames, depending on the minimum cluster size that is chosen. At the same time, fingerspelling signs that do not match any letter template due to small differences in gesture can still be picked up if they fall within a longer fingerspelling segment that contains other signs that do match a template.

A disadvantage of this clustering approach is that very short fingerspelling segments that do not exceed the minimum amount of frames for a cluster may not be picked up. This could occur, for example, when only a single letter is signed very quickly, which may be too short to be considered by the clustering algorithm. However, considering the minimum cluster size established in Section 4.5 (minimum cluster size of 5), this would only be a problem if fingerspellings would occur in a mere fraction of a second.

DBSCAN

Given a 1D array of frames indicating for each frame whether it constituted a match with one of the templates from the template set, we use DBSCAN for cluster detection. In DBSCAN, clusters are defined as high-density groups of data points. For more details on the DBSCAN algorithm, see Appendix A.1. In contrast to other clustering algorithms such as K-means, DBSCAN includes a notion of outlier points. This means that not all points belong to a cluster; we consider these points "noise": template matches that can not be assigned to a cluster. Since we are working with data that contains a relatively large amount of noise (false positives) as a result of the template matching, it makes sense to consider a clustering algorithm that takes this into account. The clusters of fingerspellings found by DBSCAN form the final fingerspelling segment predictions.

Another advantage of DBSCAN is that the number of clusters does not have to be specified in advance, contrary to other clustering algorithms such as K-means. Since the number of fingerspelling segments in a video is not

known in advance, we cannot make any accurate estimation with regards to the total number of clusters.

We use the Scikit-learn DBSCAN implementation in Python for computing the clusters. Although this is not the most efficient way to compute clusters because additional optimizations could be made for working with one-dimensional data, it is fast enough for our purposes.

3.2 Hyperparameters

Our system has a number of hyperparameters: parameters that have to be set beforehand in order to run the system. These are the following:

- `Eps`. DBSCAN parameter, see Appendix A.1.
- `MinPts`. DBSCAN parameter, see Appendix A.1. This parameter defines the minimum number of frames that constitute a segment.
- `template_match_tresh`. Threshold that determines when a template matches an image, based on the dissimilarity measure in Equation 3.2. Note that a higher value means that less similarity is required for a template match.
- `movement_threshold`. Threshold that determines if the fingerspelling hand is moving too much to be considered a potential fingerspelling sign. See Section 3.1.2 for more information.

The values of these parameters are established by experimenting with the hyperparameters on our validation set and subsequently choosing the values that provide the best results. For the chosen values of these parameters, see Section 4.5 and also Chapter 5.

Chapter 4

Experimental framework

In this chapter we discuss the experimental framework that forms the basis for the experiments performed in Chapter 5. In Section 4.1 we discuss the dataset that we use for our experiments. In Section 4.2 we discuss how we split the data into a test and validation set. In Section 4.3 we discuss the metrics that we use to measure performance. In Section 4.4 we lay out two template sets that we will use for testing our approach. Finally, we discuss the process of selecting hyperparameters in Section 4.5.

4.1 Dataset

The Corpus NGT [7], [8], is a sign language dataset made between 2006 and 2008 collected from 100 native NGT signers of different ages and from various regions in the Netherlands[1]. The aim of the corpus is to "provide a large resource for NGT research in the shape of movies of native NGT signers" [7]. By including participants from different ages, the corpus includes both newer and older stages of NGT. As for the type of interactions it contains, it includes one-on-one interactions and individual tasks assigned to signers. The videos are in MPEG-1 format, with a frame rate of 25. The recordings were made at different locations; therefore certain conditions such as lighting and camera angle are not equal across all videos. For each interaction, the signers are filmed individually by using a multi-camera setup; conversations can be played back by synchronizing the two videos, as shown in Fig. 4.2. This allows us to easily analyze videos of individual signers. Another frame from one of the videos of the Corpus NGT is shown in Fig. 4.1.

The Corpus NGT is suitable for testing our approach because of the large amount of conversations that it contains, along with a large number of different signers. Moreover, the conversations resemble "real-life" situations in which NGT is used, due to their informal nature. The high number of

[1]Videos from the Corpus NGT can be downloaded from the RU website at `https://www.ru.nl/corpusngt/de_filmpjes/download-filmpje/` (Dutch)

Figure 4.1: A frame from the Corpus NGT (session CNGT0253).

signers provides a large amount of potential signing variation. This provides us with a challenging but also realistic sign language dataset for testing the efficacy of our fingerspelling detection approach.

The data is partly annotated, although not always completely accurate. This means fingerspellings do not always start or end exactly at the annotated begin and end frames. Moreover, some fingerspelling annotations do not correspond to properly identifiable fingerspelling signs. These problems are mentioned in the Corpus NGT paper: "Several other mistakes occur in the annotations, including misalignments between start and end of many signs and their annotations. [....] The current annotations should not be blindly relied on" [7].

The fact that the annotations for the fingerspellings are not always accurate is not such a problem for our purposes, since the focus of our approach is segment detection, not letter recognition. This means that as long as all of the fingerspellings time slots are annotated, be it correctly or incorrectly, we can use the annotations as our ground truth. This seems to be largely the case. With regards to the fact that the start and end times of the annotations are not always accurate, we take this into account when defining the metrics used for evaluating the results of our approach, as discussed in Section 4.3.

One of the biggest hurdles that the Corpus NGT poses is the fact that signers are not recorded facing the camera, but rather facing another person, meaning they are slightly turned to the side. Two signers are facing each other and because of the way the cameras are positioned, one of the signers is turned to the right and one is turned to the left, as shown in Fig. 4.2. When evaluating the experimental results in Chapter 5, we will be making a distinction between "S1" and "S2", where S1 indicates signers on the left and S2 indicate signers on the right. For example, in Figure 4.2 the signer on the left is S1 and the signer on the right is S2.

The main problem that this non-frontal perspective poses is that the fingerspelling hand is not always fully visible, making it harder to recognize

Figure 4.2: A conversation between two signers from the Corpus NGT (session CNGT0132). The cameras are positioned in such a way that the signers can be seen facing each other. Because one camera is used for each signer, we can analyze the video footage of the individual signers separately.

the signs made with this hand (see for example Fig. 5.2). This is not so much a problem for the OpenPose hand detection, since OpenPose does not need to be able to see all aspects of a body part in order to detect it properly, but it does make the task of creating generalizable templates harder, since the hand can be shown from multiple different perspectives. Also, somewhat more obviously, information about the hand shape is lost when the hand is not fully visible.

4.2 Test and validation set

For testing, we use a subset of videos from the Corpus NGT, specifically choosing those videos that contain fingerspellings. We create two disjoint video sets: a test and validation set. The validation set will be used for finding the optimal values for the hyperparameters and the test set is used for evaluating results of our approach using the hyperparameters found using the validation set. The test and validation set contain sixty-eight and sixty-four sign language videos with a total of fifty and forty-nine different signers, respectively. In every video, one signer is visible in the frame. The same signers were used for both sets. Since there is already quite a lot of variation in the dataset in terms of different signers, different fingerspelling signs, etc., we allow for an overlap between the signers that appear in both sets, because we do not expect this will bias the test results.

The videos range from twenty-six seconds to eleven minutes, with an average time of three minutes and fifty-nine seconds for the test set and three minutes and nineteen seconds for the validation set, as shown in Table 4.1. The average number of fingerspelling segments is 3.2 for the validation set

Video length	Avg.	Shortest	Longest
Test	3m 59s	26s	11m 8s
Eval	3m 19s	47s	8m 55s

Table 4.1: Length of videos in test and validation set.

No. of segments	Avg.	Minimum	Maximum
Test	5.5	1	52
Eval	3.2	1	12

Table 4.2: Statistics regarding the number of fingerspelling segments in test and validation set.

and 5.5 for the test set, as shown in Table 4.2. We select the videos based on the number of right hand fingerspellings since these are most common; because of this we will also be using templates that are based on fingerspellings in the right hand. The videos that were picked from the Corpus NGT for the test and validation set are those that contained the largest amounts of fingerspellings.

With regards to the test set, our main objective is to find out whether fingerspelling segments can be detected in videos containing as many segments as possible. This explains why the test set contains more fingerspelling segments per video. Since the template matching approach that we use only requires a set of letter templates to work and does not have to be trained, we omit a training set.

4.3 Metrics

For evaluation of the test results, we use two different metrics. The ground truth fingerspelling frames for a given video consists of all the frames of the time slots that are annotated as fingerspellings in the video that form a segment containing at least `MinPts` frames (see Section 3.2). For example, given a frame rate of 25 and a value of 5 for `MinPts`, this means segments must be at least $5/25 = 0.2$ seconds long. We use macro average precision (AP) and macro average recall (AR) over a set of videos V to measure performance, which we define as follows:

$$AP(V) = \frac{\sum_{v \in V} \text{Prec}(v)}{|V|}, \tag{4.1}$$

$$AR(V) = \frac{\sum_{v \in V} \text{Rec}(v)}{|V|}, \tag{4.2}$$

where Prec and Rec stand for the precision and recall, respectively, which can be defined as:

$$\text{precision} = \frac{tp}{tp + fp}, \tag{4.3}$$

$$\text{recall} = \frac{tp}{tp + fn}, \tag{4.4}$$

where tp, fp and fn stand for true positives, false positives and false negatives, respectively. Concretely, the true positives indicate frames or segments that are correctly predicted as fingerspellings, the false positives indicate frames or segments that have been falsely predicted as fingerspellings, and false negatives indicate frames or segments that have been falsely predicted as non-fingerspelling frames.

4.3.1 Metric 1: segments

The first metric that we use involves counting the number of segments that have been correctly detected. For this, we first have to define when a fingerspelling segment has been "detected". We base this notion of a detected segment on the idea that in a real-world scenario where a linguist is using the fingerspelling segment detection for annotation, he does not require an exact prediction of the time frame when a fingerspelling occurs. Rather, an estimation of the time frame in which fingerspelling occurs is generally sufficient. This is based on the assumption that during annotation, the annotator will not only look at the detected fingerspelling segments, but will also consider a few surrounding seconds such that the proper manual annotation can be made. This means that finding the exact start and end time of segments is not as important as finding good approximations of the locations of the different segments.

As a general rule, we denote a predicted segment as correct when a person can be seen fingerspelling in it and when the segment is not much longer than the corresponding ground truth segment. In a real-world scenario, this would result in further inspection of the surrounding frames, allowing the user to find the entire segment manually with minimal effort. More specifically, we consider a ground truth segment detected if a predicted segment exists that meets the following conditions. First, at least 20% of the ground truth segment has to be detected. Second, the start time of the predicted segment can at maximum fall two seconds before the start of the ground truth segment; for the end time this is at maximum two seconds after the end of the ground truth segment.

More formally, let r denote the frame rate for a video, i.e. the number of frames per second. For a ground truth fingerspelling segment s with length l, start frame f_i and end frame f_j, a predicted segment with start frame u

and end frame v, $u \leq v$, is considered valid if the following conditions are met:

1. $|[u, v] \cap [f_i, f_j]|/l \geq 0.2$
2. $u \geq f_i - 2r$ and $v \leq f_j + 2r$

A ground truth fingerspelling segment is considered detected if at least one valid segment exists for that ground truth segment. It should be noted that using this definition, it is possible that a segment is detected multiple times, for example when a long fingerspelling segment is detected as two different segments because some part of the middle of the segment was not detected as a segment. When this is the case, we count both predicted segments as correct.

The time frame within which a segment is considered valid for some ground truth segment is relatively broad, because of three reasons. First, it is not a problem for an annotator if the predicted segment ends before the fingerspelling sequence is completed, since the annotator will notice that it is incomplete and look at a few more seconds to find the actual end frame. Second, it is not a problem if the end frame falls after the end of the fingerspelling sequence, since an annotator will easily notice this and correct it when making the manual annotation for the fingerspelling sequence. Third, the ground truth fingerspelling annotations are not always completely accurate in marking the start and end frame of fingerspellings, as mentioned in Section 4.1. By using a tolerance window around the start and end frame, the effect of these imprecise time annotations can be largely mitigated.

4.3.2 Metric 2: frames

For the second, alternative metric, we calculate precision and recall on a per frame basis. When a fingerspelling segment is detected, all of the frames in that segment are marked as fingerspelling frames and compared to the ground truth fingerspelling frames. This metric is less informative since it does not take into account the difference in length between segments, which means shorter segments will be given less weight in this evaluation. Furthermore, a frame-by-frame analysis is not necessarily informative since the ground truth annotations are not always precise in marking the exact fingerspelling frames. Nevertheless, we include this metric for completeness.

4.4 Template sets

In order to get a better idea of the ways in which templates can be created in an efficient manner, we will be trying out different template sets when testing our approach. Since we will be analyzing sign language videos where NGT

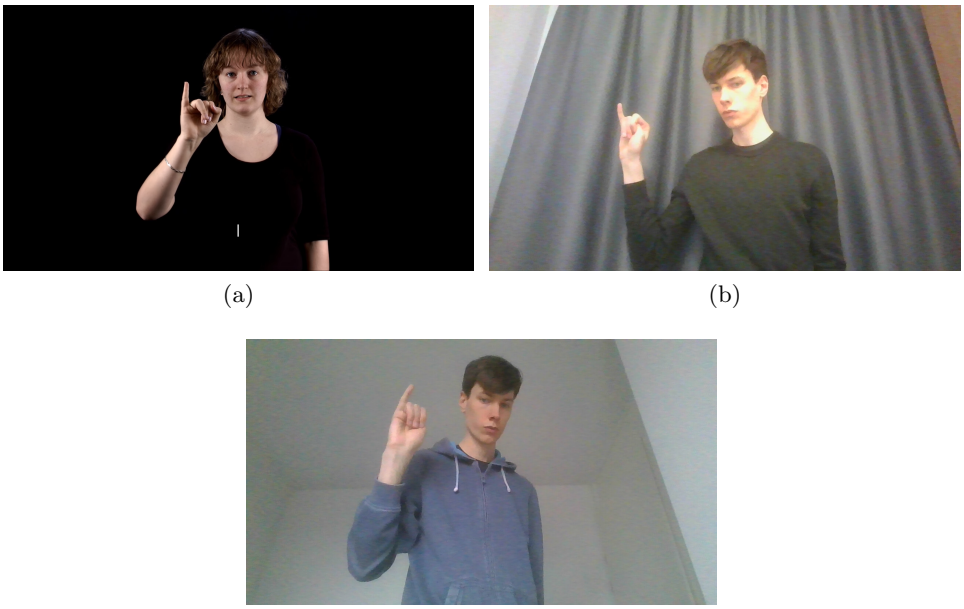
is used, we base the template signs on the NGT alphabet (see Fig. 1.1). It should be noted that five of the letters of the NGT alphabet are non-static, i.e. some letters involve a particular movement with the hand. Since these signs are not based on static hand formations, we do not include templates for these non-static signs. However, note that some of the hand shapes that are used for the non-static fingerspellings resemble static fingerspelling shapes. For example, the non-static fingerspellings H and U use a hand shape similar to the letter V. We are not explicitly modeling non-static fingerspellings in our template set, but due to the similarity between static and non-static fingerspelling hand shapes, we may still be able to detect non-static fingerspellings using the static templates. Therefore we also include non-static fingerspellings in our quantitative results in Chapter 5.

We try out two different template sets: one that is derived from an internet resource which will serve as a baseline, and another that is custom-made and more tailored to the data we are working with. The template sets are set up as follows:

- Set 1: one template per letter using a YouTube fingerspelling tutorial video [1] as basis for the templates . This leads to a total of twenty-one templates.
- Set 2: supplement/replace templates from set 1 with custom-made templates that include more variation in terms of the orientation of the signs. Concretely, this means we create a maximum of two additional templates per letter that show a left and right rotation of the letter sign, as shown in Fig. 4.3. This can be useful seeing as this rotation of the hand occurs often in the Corpus NGT (see for example Fig. 4.4). Creating the template images is done by taking webcam pictures of the signs. This leads to a template set with two to three templates per letter, with a total of sixty-one templates.

One of the main differences between the two sets is the number of templates per letter. One of the main reasons for creating multiple templates per letter is that templates are not rotation invariant, i.e. they capture only one particular orientation. By adding more templates that fit the way in which fingerspelling occurs in the data at hand, the higher the chance seems to be that fingerspellings will be detected, although it may also lead to an increase in falsely detected fingerspellings. At the same time however, it seems likely that a trade-off should be made between the amount of effort put into making a template set and the performance that the template set provides. Set 1 is relatively easy to create, whereas set 2 requires more effort since it involves manually creating template images.

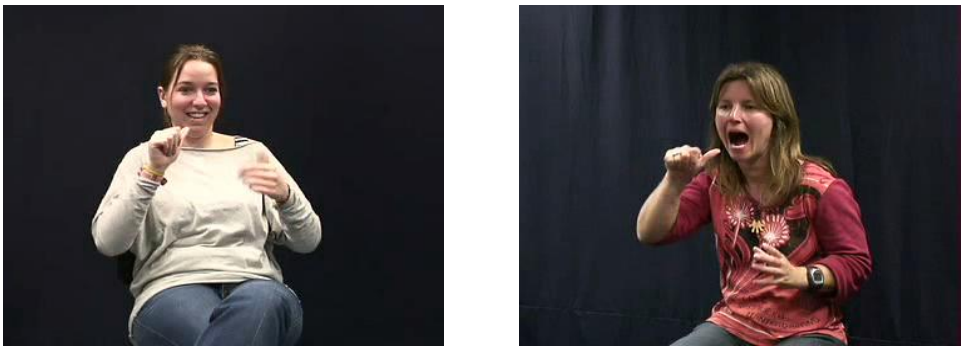
Set 1 aims to test how well a "standardized" fingerspelling reference is suited for detecting fingerspellings in real-life scenarios. Set 2 aims to test if custom templates can lead to better performance compared to the baseline



(a) (b)

(c)

Figure 4.3: Template images from set 2 for the letter I, which include: (a) a frontal view [1], (b) the sign turned to the right, (c) the sign turned to the left. Both (b) and (c) are created by taking webcam pictures.



(a) (b)

Figure 4.4: The fingerspelling sign for the letter A, in the Corpus NGT (session CNGT0065 and CNGT0259, from left to right). (a) The sign made by a signer sitting on the left. (b) The sign made by a person sitting on the right. Notice that even though the same sign is used in both images, a single template would most likely not be sufficient to detect both instances, indicating that multiple templates that take into account these different perspectives can be useful.

results established by set 1. Using set 2, we aim to test if knowledge about the dataset plays an important role when creating a template set, by looking at the way fingerspelling occurs in the data and creating templates based on this information.

4.5 Choosing hyperparameters

We use the validation set for determining the optimal values for the hyperparameters. The aim is choosing the hyperparameters in such a way that the recall is maximized, while trying to maintain a lower bound on the precision. Intuitively, this means we want to detect as many of the fingerspelling segments in a video as possible, even if this leads to a relatively high number of false positives. This makes sense in the context of the annotation process for sign language videos. When trying to annotate fingerspellings, it is easy to check whether or not a small segment contains fingerspelling, whereas it can be cumbersome to look for all the fingerspelling segments in a video. Therefore we prioritize recall over precision. We set a lower bound on the average precision of 0.1. To put this into context: fingerspelling frames make up $\pm 1\%$ of our dataset (1% for the validation set, 1.6% for the test set), thus by simply marking every frame as a fingerspelling frame would result in an accuracy of approximately 0.01.

We conclude that in general, the following setup works well for the validation set:

- `Eps` $= 7$
- `MinPts` $= 5$
- `movement_threshold` $= 0.025$

We use these parameter values for the experiments in Chapter 5.

The last hyperparameter, `template_match_thresh`, which determines the template matching threshold as used in Equation 3.5, is set based on the template sets that we will be using. For example, for a large template set it might make sense to set a stringent template matching threshold, such that frames must be quite similar to a template for there to be a match, whereas a small template set might benefit from a less stringent template matching threshold since the templates will most likely not cover all the different ways in which fingerspelling might occur.

By experimenting with the template matching threshold on the validation set, we find that setting the `template_match_thresh` hyperparameter to 29000 for set 1 and 24000 for set 2 produced optimal results. We will be using these values as part of our testing configuration for these sets in Chapter 5.

Chapter 5

Experimental results

In this chapter we discuss the experiments performed to answer the research questions laid out in Chapter 1. Quantitative results are laid out in Section 5.1. Following this, we discuss common failure cases in Section 5.2. In Section 5.3 we discuss the process of creating a new template set based on the previous results, containing significantly fewer templates. We discuss the effect of the physical constraints on signers in Section 5.4, along with the effect of DBSCAN in reducing false positives in Section 5.5. Finally, we provide some rough timing estimates for the individual components of our pipeline in Section 5.6.

5.1 Results

The results for template set 1 and 2 are shown in Table 5.1 and 5.2, respectively. We include the results for S1 and S2 signers separately (see Section 4.1). The results show that using set 2 on the test set produces an average recall in segment detection of 0.715, while set 1 gives an average recall of 0.735, where set 2 has a slightly higher precision of 0.118 compared to 0.102. Therefore, set 1 and set 2 seem to provide roughly the same performance.

It is interesting to note the discrepancy between the S1 and S2 results for both template sets. The results show that the recall for S2 signers is generally higher than that for S1, but at the cost of a lower precision. The most likely explanation for this difference in performance is the difference in which the S1 and S2 signers are filmed with regards to their orientation. When considering right hand fingerspellings, the fingerspellings for S1 are not always fully visible, because both the body and the fingerspelling hand are oriented towards the right. Compare for example Fig. 5.2b and Fig. 5.1, which show a S1 signer and a S2 signer, respectively. The fingerspelling hand of the S1 signer is somewhat occluded because of her orientation, whereas the fingerspelling hand of the S2 signer is not occluded. These occlusions of the fingerspelling hand seem to be more common for the S1 signers.

The results indicate that the the custom-made templates of set 2 do not lead to better performance compared to set 1. Considering the extra effort that custom-made templates require to make them, it seems reasonable to conclude that set 1 provides the best trade-off between performance and time investment required to make the templates. Intuitively, it seems to make sense that by adding more fingerspelling variations as templates, more corner cases could be detected, thus increasing overall performance. Nevertheless, the results show that there is no obvious benefit in using the extra templates. This is an indication that the size and scope of the template set can be kept relatively small.

	Validation set				Test set			
	Segments		Frames		Segments		Frames	
	AR	AP	AR	AP	AR	AP	AR	AP
S1	0.765	0.112	0.702	0.070	0.694	0.114	0.703	0.094
S2	0.856	0.086	0.869	0.054	0.780	0.088	0.860	0.056
Total	0.808	0.100	0.780	0.062	0.735	0.102	0.777	0.076

Table 5.1: Results for set 1, which contains one template per letter. AP and AR stand for Average Precision and Average Recall, respectively.

	Validation set				Test set			
	Segments		Frames		Segments		Frames	
	AR	AP	AR	AP	AR	AP	AR	AP
S1	0.771	0.115	0.689	0.070	0.647	0.136	0.625	0.119
S2	0.857	0.084	0.851	0.056	0.790	0.096	0.820	0.065
Total	0.811	0.100	0.765	0.063	0.715	0.118	0.717	0.093

Table 5.2: Results for set 2, which contains a maximum of three templates per letter. AP and AR stand for Average Precision and Average Recall, respectively.

5.2 Common failure cases

As part of a qualitative analysis of the results, we discuss some commonly observed failure cases.

- Some non-letter signs use the same hand formations as letter signs. This is one of the major causes of false positives when detecting fingerspellings using template matching. Fig. 5.1 shows a woman making

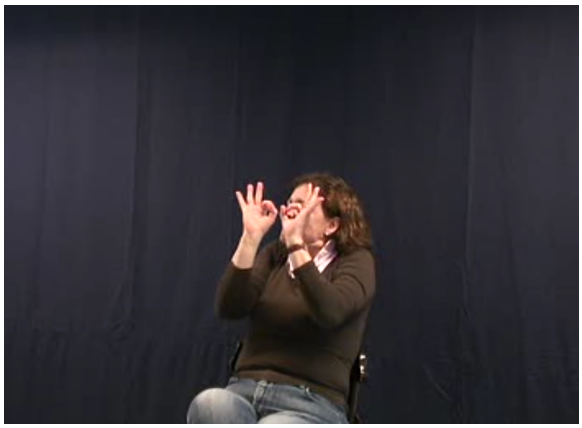


Figure 5.1: A woman making a sign that resembles the fingerspelling sign for the letter T. The right hand should not be marked as a fingerspelling sign, due to the fact that she is including the left hand for signing. Taken from the Corpus NGT (session CNGT2216).

a sign that resembles the fingerspelling sign for the letter T with both hands. However, due to the fact that she is including the left hand for signing, the right hand should not be marked as a fingerspelling sign. This means template matching is not sufficient in these kinds of situations; in this case the left hand should be taken into account in order to infer that the sign under consideration is not a fingerspelling sign.

- Bad visibility of fingerspelling hand due to signer orientation. Two examples of this are shown in Fig. 5.2. When signers are not directly facing the camera, the fingerspelling hand may not be fully visible at all times. This can can occur more frequently when a signer is sitting at a slight angle; this is the case for most of the videos in the NGT corpus: signers are often slightly shifted to the side, making it harder to see their fingerspelling hand completely, as discussed in Section 4.1. This is most notably the case when the signer is oriented towards the right side of the frame and signs with her right hand, or when facing the left side of the frame and signing with her left hand, since this often results in the side of the hand facing the camera (see for example Fig. 5.2a). This suggests that when only considering right hand fingerspelling, the S1 signers in the Corpus NGT will pose more problems than the S2 signers. This possibly explains some of the differences in S1 and S2 results in Table 5.1 and 5.2, although it is not clear that S2 performs better: the recall for S2 is consistently higher compared to S1, but this is combined with a lower precision.

- Some fingerspelling signs have few distinguishing features, making

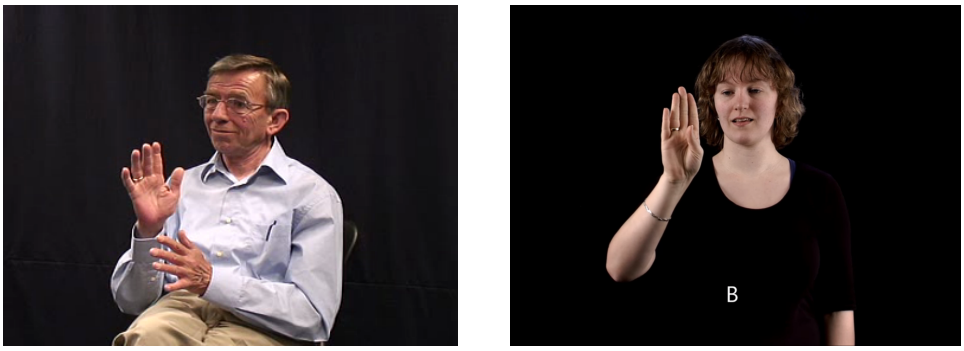
(a) (b)

Figure 5.2: Two examples of poor hand visibility. (a) The letter N viewed from a perspective that makes it hard to distinguish all the parts of the hand, e.g. the index finger is obscured by the middle finger. (b) The letter I signed from the side, obscuring the thumb and parts of the ring and index finger. Taken from the Corpus NGT (session CNGT0485 and CNGT0566, from left to right).

them susceptible to false positives. The sign for the letter B for example, consists of a forward-facing flat hand with a bent thumb (as shown in Fig. 5.3b). The only real distinguishing feature for this sign is the thumb; there is no conspicuous combination of features that make it uniquely identifiable, which can lead to false detections of this sign since it can easily occur by accident or as part of other signs. Fig. 5.3a shows an example of this where the letter B is mistakenly recognized based on the template image shown in Fig. 5.3b.

- Differences in execution of letter signs. Looking at the data, we observe that fingerspelling signs are often executed in slightly different ways. Such small differences might be as simple as turning the hand in a slightly different direction, or executing the signs in a quick and less precise manner. Some fingerspellings are executed more deliberately where the individual signs do not deviate much from the canonical letter signs, whereas other fingerspelling segments may contain signs that show only a vague resemblance to the canonical letter signs. Nevertheless, these fingerspelling segments may be perfectly understandable to the conversation partner, who might not need to see every sign clearly to infer the intended meaning, for example by relying on context.

- Individual fingerspellings that are part of a sequence of signs can be influenced by neighbouring signs, as shown in [15]; this co-articulation can make it more difficult to match letter templates to instances of these letters in fingerspelling segments consisting of multiple letters.



(a) (b)

Figure 5.3: An example of an incorrectly recognized letter. (a) A frame where no fingerspelling occurs, but since the right hand sufficiently resembles the letter B as shown in the template image (b), the hand in (a) is seen as a match with the template. (a) is from the Corpus NGT (session CNGT0299), (b) from [1].

5.3 Creating a more minimal template set

Given the results of our approach as discussed in Section 5.1, we want to explore the idea of a small but effective template set further. Ideally, we want to create a template set that contains a minimal amount of templates, while capturing as many fingerspellings as possible. Using fewer templates is preferred because it is generally less labor-intensive to create a small template set than it is to create a larger one, especially if the templates are tailored to the dataset. Furthermore, using fewer templates that capture more general features of fingerspelling may help our method to generalize better to other data. The question then rises: how can we create a small set of templates that captures the most common patterns across fingerspellings? In this section, we attempt to create a smaller, more general template set that can still capture a wide range of fingerspellings, instead of using a template set that contains templates for each letter of the NGT alphabet.

5.3.1 Learning from previous results

The failure analysis in Section 5.2 shows us that it can be hard to find suitable templates for fingerspelling signs, because of differences in execution of fingerspelling signs, different orientations of the hand, few distinguishing features for some signs, and co-articulation. Furthermore, the results in Section 5.1 indicate that template sets can perhaps be more minimal, since a more expansive template set does not necessarily provide an increase in performance.

This leads us to a slightly different approach for designing templates.

Instead of creating templates for each letter that needs to be detected, we want to create templates that are less specific, but similar enough to most fingerspelling signs such that most fingerspellings will still be detected. This has the advantage of having to design fewer templates, while also potentially making the templates less dependent on the dataset that we are using, thus making them more generally applicable.

5.3.2 Single template performance

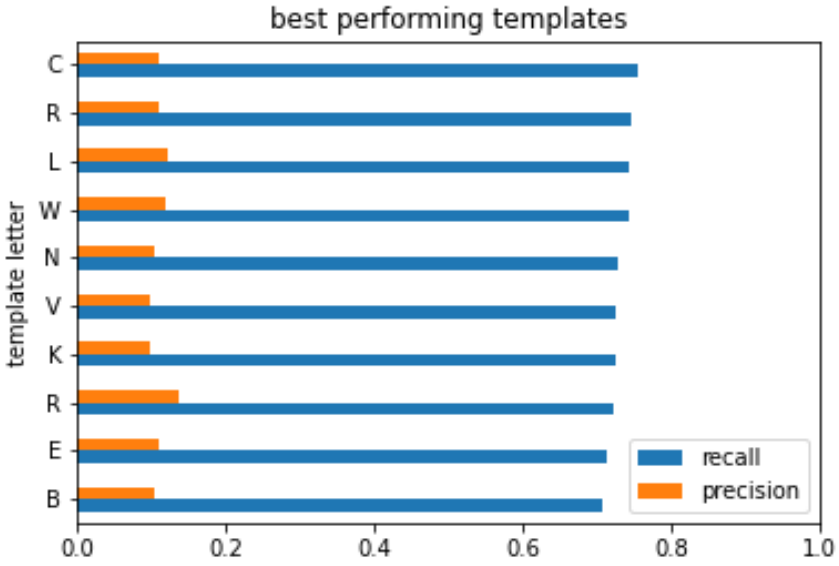
In order to get a better idea of the kinds of templates that may be used to capture a wide range of fingerspellings, we evaluate single template performance on the test set, meaning we test the fingerspelling detection system using template sets consisting of single templates. In this manner, we evaluate the performance of every template of set 2 (which contains sixty-one templates with a maximum of three templates per letter) individually. The ten best and the ten worst results are shown in Fig. 5.4.

One particularly salient detail is that the best performing templates reach a performance level similar to the performance for much larger template sets such as set 2, as shown in Table 5.2. At the same time, the worst performing templates show a noticeable difference in performance with the best performing templates in terms of recall. Moreover, When taking a closer look at the previous results from set 1 and set 2, we find that for set 1, 41% of the template matches are caused by two different templates; 30% of the template matches for set 2 are caused by three different templates. This suggests that only a small percentage of the templates is responsible for most of the template matches.

By inspecting the best and worst performing templates, there seems to be a correlation between orientation of the hand and performance. Most of the worst performing templates are based on images where the fingerspelling hand is oriented towards the left, whereas most of the best performing templates are based on images where the fingerspelling hand is oriented towards the right. This might be explained by the orientation of the fingerspelling hand in the Corpus NGT: often a right hand fingerspelling is oriented slightly towards the right; this is the case both for S1 and S2 signers. This means hand templates where the hand is slightly oriented towards the right may be better at detecting fingerspellings for this particular dataset. Furthermore, a common pattern among the best templates seems to be that they display a similarity to multiple different letters. When designing new templates, this seems to be something to strive for.

5.3.3 Creating a new template set

The results from the previous section indicate that possibly, only a small number of templates provide most of the information required to detect



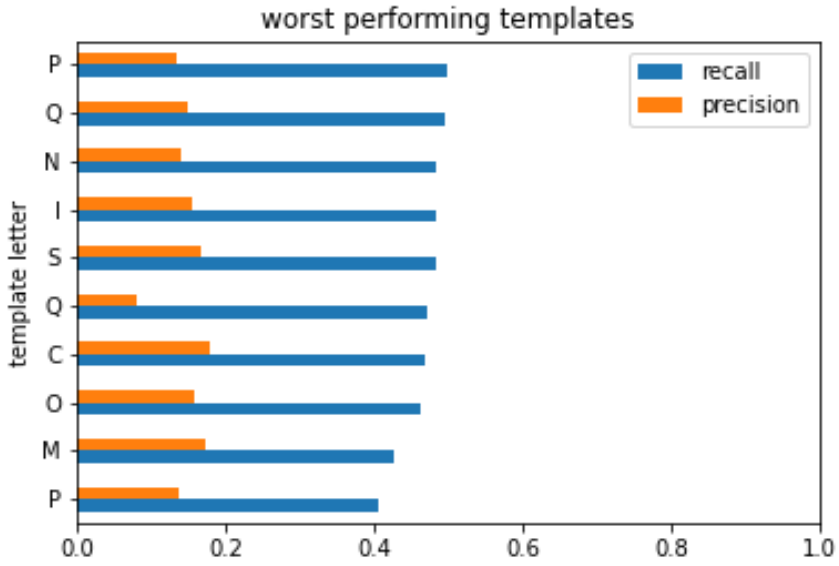


Figure 5.4: Test set results for one-item template sets.

most fingerspellings. Nevertheless, it is not yet clear what an adjusted, more minimal template set might look like. For example, if we were to select a subset of the best performing templates to use as a new template set, we run the risk of creating a template set that is highly tuned to the data, but does not necessarily perform well on new data. One thing that seems to distinguish the templates that perform well from those that do not, is that they bear a resemblance to most other fingerspellings. Therefore we aim to make the new templates as "generic" as possible, meaning they show resemblance to a large number of fingerspelling signs. Another guiding principle for designing the new template set is that the templates should complement each other, detecting particular fingerspelling occurrences that other templates in the same set might not detect, thus reducing the amount of redundant templates.

The new template set contains templates that are not based on letters, but on hand formations closely related to fingerspellings. In this way they should capture a set of general hand shapes that are common across fingerspellings. The templates complement each other by capturing different hand formations, but also different horizontal hand orientations. The resulting generic template set is shown in Fig. 5.6, consisting of four different templates.

5.3.4 Results for the generic template set

Since we are using a template set of much smaller size than the template sets used earlier, we set the `template_match_thresh` to a much higher value of 56000, which seems to produce optimal results on the validation set. The results achieved with the new template set are shown in Table 5.3. Surprisingly, the performance is up to par with set 2 as shown in Table 5.2, which uses sixty-one templates, whereas the generic template set uses four.

By comparing the results of set 1 and the generic template set, the generic templates seem to capture a wider range of hand gestures. By increasing the `template_match_thresh` hyperparameter (as explained in Section 3.2), the required similarity between template and input frame for a match is lower, allowing the templates to capture more variation in fingerspelling execution. This is made possible by the small number of templates in the template set. The effect seems to be that the templates are not simply detecting specific fingerspelling signs, but rather detect a more general fingerspelling hand formation, which is common across fingerspelling signs. This addresses the issue of differences in sign execution and different signer orientations, as explained in the common failure cases section, Section 5.2. As an example, Fig. 5.5 shows a fingerspelling sign that is oriented to the right, which can be problematic to detect, since it differs from its corresponding letter template used in set 1. The result is that the sign was detected using the generic template set, but not detected using set 1.

Figure 5.5: A fingerspelling sign in the right hand that was not detected using set 1, but which was detected using the generic template set. This is an indication that the generic set can capture a wider range of hand gestures. Taken from the Corpus NGT (session CNGT0821).

Figure 5.6: The images used for the "generic" template set.

There is also a possible downside to this: the generic templates can lead to more false positives. For example, the results show that sometimes, by using the generic set, very long fingerspelling sequences are detected, because the generic templates quickly match frames where the hand is in a "fingerspelling position". In this manner, situations where the fingerspelling hand is in a fingerspelling position but not actually fingerspelling can lead to false positives. Nevertheless, the results show that this does not lead to noticeable differences in precision compared to set 2, indicating that this effect is not significant.

	Validation set				Test set			
	Segments		Frames		Segments		Frames	
	AR	AP	AR	AP	AR	AP	AR	AP
S1	0.750	0.111	0.662	0.071	0.676	0.123	0.672	0.105
S2	0.935	0.097	0.833	0.061	0.812	0.097	0.824	0.065
Total	0.837	0.104	0.742	0.066	0.740	0.111	0.743	0.086

Table 5.3: Results for the generic template set. The performance is similar to the performance of set 2, while using only four different templates. AP and AR stand for Average Precision and Average Recall, respectively.

5.4 Effect of the physical constraints on the results

In Section 3.1.2 we introduced some physical constraints on signers for our fingerspelling detection pipeline: additional constraints meant to filter out some of the frames that are unlikely to be fingerspellings in an early stage. However, there are two important caveats that must be taken into account with regards to these constraints. First, it is possible that they parse out some of the actual fingerspelling frames. The assumption is that the constraints will parse out relatively more frames that do not contain fingerspellings, while removing only a relatively small amount of fingerspelling frames, if any. Second, one of the physical constraints leads to an additional hyperparameter that has to be set: `movement_threshold`. Although using a fixed value for this hyperparameter seemed to work well in our experiments, having to set more hyperparameters is generally not preferable since it can introduce more complexity in choosing the right combination of hyperparameters.

For these reasons, it makes sense to analyze whether or not the aforementioned constraints have a sufficiently positive effect on the results. For this, we will consider a version of the fingerspelling detection system that does not include the physical constraints in our pipeline. We will use set 1 as template set (see Section 4.4). The results are shown in Table 5.4.

Table 5.4 shows that for segment detection, removing the physical constraints provides a drop in recall of almost 0.25, from 0.735 to 0.503, while halving the precision, from 0.102 to 0.048. Therefore, removing the physical constraints leads to a significant drop in both precision and recall. This provides a strong indication that the physical constraints have a positive effect on the performance of the fingerspelling detection system.

	Validation set				Test set			
	Segments		Frames		Segments		Frames	
Detections	AR	AP	AR	AP	AR	AP	AR	AP
w constraints	0.808	0.100	0.780	0.062	0.735	0.102	0.777	0.076
w/o constraints	0.674	0.049	0.913	0.023	0.503	0.048	0.889	0.033

Table 5.4: Results for set 1 with and without the physical constraints laid out in Section 3.1.2. Removing the physical constraints results in a noticeable drop in performance. AP and AR stand for Average Precision and Average Recall, respectively.

5.5 Effect of DBSCAN in reducing false positives

As mentioned earlier, simply using template matching to detect fingerspelling in a video would most likely result in a large number of false positives and would therefore be insufficiently accurate as a way of detecting fingerspelling. To illustrate this, we provide an example, showing the strong effect the DBSCAN clustering algorithm has in reducing the amount of matched frames. Fig. 5.7 shows template matches for a video in our dataset. The ground truth indicates what frames contain fingerspellings and the predictions are the result of applying template matching to each frame as shown in Fig. 3.1d. In other words, this shows the predictions that would be made if template matching was used to predict fingerspelling frames without applying density-based clustering. Fig. 5.8 shows the new predictions after applying DBSCAN to the template matching results. As the figure shows, DBSCAN can significantly cut down the number of false positives.

5.6 Processing time

The pose detection part of our pipeline using OpenPose is the main bottleneck in terms of processing time. The videos from our dataset were processed by OpenPose using a single Nvidia Tesla M40 GPU and the resulting pose information was stored in JSON format, to speed up testing. Running OpenPose on the Corpus NGT videos takes approximately twenty times real-time (thus, one minute of video material takes approximately twenty minutes to process).

Much of the remaining processing time comes from IO operations that involve loading the saved pose information from disk. Every frame of a video that OpenPose processed has to be loaded from disk and passed through the pipeline. Now follow some very rough timing estimates for each of the components of the pipeline on a regular laptop. For processing a five minute video, parsing out unsuitable frames takes ± 0.5 seconds and hand extraction

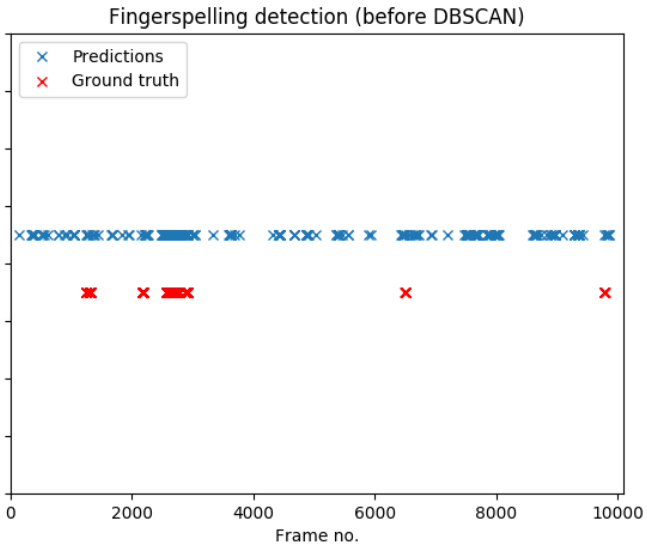


Figure 5.7: Template matching results for a video of the dataset. Without applying clustering, the predictions would show a large amount of false positives.

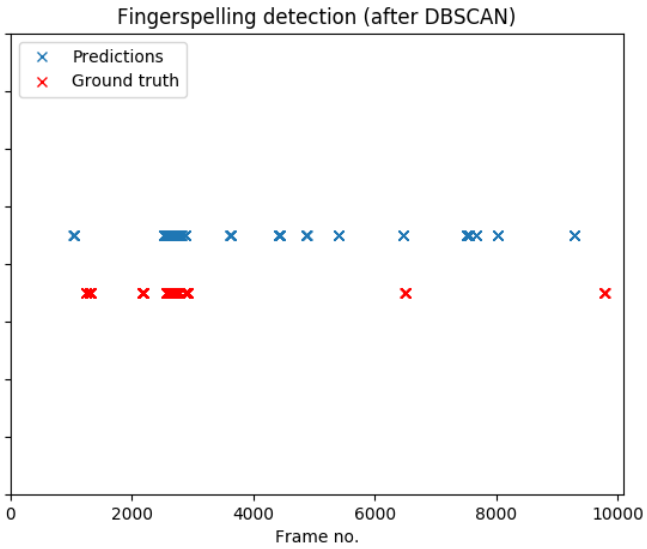


Figure 5.8: Fingerspelling predictions after applying DBSCAN to template matching results from Fig. 5.7. The number of false positives is significantly reduced.

takes ±0.05 seconds. Template matching takes ±0.4 seconds for the generic set, ±1.3 seconds for set 1 and ±3.7 seconds for set 2. Finally, DBSCAN takes ±0.7 seconds.

Chapter 6

Related work

A large part of the research concerning fingerspelling recognition is focused on fingerspelling classification as opposed to fingerspelling detection, i.e. recognizing fingerspelling signs in a frame as indicating specific letters, rather than determining whether or not fingerspelling occurs in a frame at all. For this reason, we highlight some research in this section that overlaps with different parts of our research.

In terms of data-driven approaches, Hidden Markov Models, widely used in speech recognition, are often used to capture the temporal aspect of non-static signs. Neural networks are also often employed for classifying hand shapes [19]. These methods rely on large amounts of representative training data in order to work well in the wild, in contrast with the approach in this research.

With regards to OpenPose based gesture evaluation systems, Qiao et al. used a human gesture grading system based on OpenPose to evaluate Tai Chi gestures [21]. The system captures poser's gesture and computes a score to estimate whether the real-time motion is precisely matched with the standard gesture. Different weights are assigned to groups of body parts according to the importance of these body parts in judging difference in gesture.

Several template matching approaches exists for recognizing sign language. Gupta et al. [12] classified number signs in ASL, based on contour information. Images for testing and training show only the signing hand and are created in fixed lighting conditions with a uniform background. A perfect classification precision is achieved using heavily constrained image conditions. Rahaman et al. [22] used skin segmentation in combination with contour analysis to create templates representing specific word signs in order to recognize 18 common Bengali sign words. For training, a data set of 1800 images (18 signers, 10 versions of 10 signs each) is used. The method achieves a mean accuracy of 90%. Unfortunately, the authors are not clear about the way in which the data is split, seemingly using the same data for

creating templates and testing them. Furthermore, the method is trained on a sizeable amount of templates, which means applying it to a different sign language would be cumbersome since this would involve finding a new similarly sized template set.

With regards to general template-based detection using "generic" templates that capture different variations of shapes, a similar idea is used by Nanda et al. [18], who use probabilistic templates for pedestrian detection in infrared images. 1000 different images of pedestrians in different poses and orientation are amalgamated into a probabilistic template, which is able to capture variation in human shape.

Another idea seemingly related to the generic templates in this research is that of "deformable" templates [14], which are able to deal with shape deformations and variations. Using a so called prototype-based model, shape classes possessing a characteristic structure, but also substantial variability, can be represented. The prototype deformable templates are defined around a "standard" or "prototype" template, describing the '"most likely", "average" or "characteristic' shape of a class of objects. Variation in shape is modeled using different parameter values for the templates. For example, Jain et al. [13] tackle general object matching by representing object shape using a hand-drawn prototype template, consisting of the representative contour/edges. Shape variation is included using probabilistic deformation transformations on the template image. Using a type of deformable templates as an extension of the generic templates used in this research may be a possibility for future research.

Chapter 7

Conclusions

7.1 Conclusion

In this research we laid out an approach for detecting fingerspelling by combining pose information with template matching in order to find fingerspelling segments. Given a sign language video, pose information for each frame is extracted using OpenPose. The part of the image that shows the fingerspelling hand is extracted and compared to a set of template images to see if a match can be found. Segments of fingerspelling are predicted by attempting to cluster matched frames together using DBSCAN, a density-based clustering algorithm.

We tested our approach on the Corpus NGT, a challenging sign language dataset containing realistic conversations between signers. Furthermore, we provided an in-depth evaluation of the ways in which a template set can be efficiently created. We found that creating a template set can be done in a number of ways, for example by using an internet resource or by taking pictures using a webcam. A large template set including different variations of fingerspelling signs turned out to provide no significant advantage over a simpler template set that includes one template for every static letter of the hand alphabet. What's more, we found that a simple template set consisting of only a few templates can be just as effective as a larger template set, using "generic" templates meant to resemble a wide range of fingerspelling signs. Using this generic template set, we managed to detect an average of 74% of the fingerspelling segments in our test set, with a precision of 11.1%.

7.2 Discussion and outlook

7.2.1 Left hand fingerspellings

The fingerspelling detection approach outlined in this research was tested on right hand fingerspelling, but could also apply to detecting left hand

fingerspelling, since this would merely involve adjusting the templates to left hand fingerspellings. This can be useful since some signers use their left hand for fingerspelling, instead of their right hand. A simple way of adjusting a template set to left hand fingerspellings might be to mirror right hand templates in order to create left hand templates.

7.2.2 Multiple people

Our approach works on the assumption that one person is present in a video frame, but could easily be extended to multiple people. OpenPose is designed to detect pose information for multiple people as well as for one person; because of this, our approach could be extended to work on sign videos that contain multiple people.

7.2.3 Towards generalizability

The results achieved with the generic template set show that a small template set consisting of only a few "generic" hand shapes may perform just as well for detecting fingerspelling segments as a larger template set consisting of templates for every fingerspelling letter that needs to be detected. This is a useful result, since it indicates that designing a template set does not need to be an arduous process; rather, a small number of templates can easily be made, such as those shown in Fig. 5.6. Going through the effort required to create a larger template set is not necessarily worth the effort, since a small template set can provide similar performance.

One of the most promising consequences of using a minimal template set such as the generic set used in Section 5.3.4, is that because of the small amount of assumptions that are made about the way fingerspelling occurs, it seems more likely that the approach will generalize to new data. The generic set merely contains a few generic hand shapes — the particular details of individual fingerspelling signs are largely ignored in favor of looking for a few general hand formations that are characteristic of fingerspelling. Since these hand formations remain largely the same across different sign language videos, it seems likely that the approach will also work for other sign language videos than those from the Corpus NGT.

An interesting detail to note, is that the generic templates are not explicitly based on the language of NGT. For this reason, it may even be possible to detect fingerspellings for other sign languages using our approach, given that the fingerspellings are at least somewhat similar to NGT (cf. British sign language, where two hands are used for fingerspelling, making it unsuitable for the current setup).

Nevertheless, when considering our complete fingerspelling detection pipeline, there are a number of factors that may prevent our approach from generalizing to new data:

- *OpenPose failure to detect hand keypoints.* Our template matching pipeline depends on OpenPose for detecting hand keypoints. Generally speaking the OpenPose keypoint detection works well, even for obscure viewing angles. However, most notably, hand keypoints cannot be detected if the corresponding elbow and/or wrist is not visible (see Section 2.3). Thus, when considering videos that, for example, only show the signing hand but not the rest of the arm, our approach will not work. In those cases where OpenPose is not able to detect pose information, our approach will not work.

- *Exceptions to physical constraints.* As explained in Section 3.1.2, we use three physical constraints that parse out unlikely fingerspelling frames before they are considered for template matching, for example by checking if the fingerspelling hand is moving too fast. We showed in Section 5.4 that these constraints noticeably improve performance. Nevertheless, these physical constraints are largely based on observation of the Corpus NGT dataset and are therefore not foolproof. Videos where, for some reason, the physical constraints do not apply, may result in skewed results, where a disproportionate amount of fingerspellings is rejected early on.

- *Different camera angles.* Although the results on the Corpus NGT dataset have shown that our approach can deal with non-frontal views to some degree, the templates are still at least somewhat dependent on a particular viewing angle. Thus, the templates used in this research may not work for videos where the signer is filmed from the side, or at some other angle that is quite different from a frontal view. A possible solution for dealing with such a different viewing angle is adjusting or adding the hand templates such that they are oriented in the same way as the signs in the videos under consideration. It should be noted though, that such a different viewing angle may prevent proper hand visibility, making it hard to recognize the signs regardless of the templates that are used.

- *Different frame rates.* The Corpus NGT consists of videos with a frame rate of 25. The hyperparameters for the DBSCAN clustering algorithm that is used to cluster together fingerspelling frames into segments are based on experimentation with the Corpus NGT videos. It seems reasonable to assume that when applying our approach to videos with a different frame rate, the chosen DBSCAN hyperparameters may not work as well. For example, given a video with a high frame rate, it seems reasonable to increase the `MinPts` DBSCAN hyperparameter, which indicates the minimum number of frames that can form a fingerspelling segment. This is because given a higher frame rate, a fingerspelling segment of the same length will contain

more frames, which should be taken into account. A simple way of dealing with this may be to simply scale the `MinPts` hyperparameter according to the new frame rate.

- *Very short fingerspelling sequences.* Our approach was designed to detect fingerspelling segments in the form of clusters of frames rather than individual frames. For this reason, very short fingerspelling sequences consisting of only a couple of frames such as single letters may not be detected. However, this should generally not be such an issue, considering that for our testing setup, we used a minimum of 5 frames per cluster, which translates to a minimum of $5/25 = 0.2$ seconds for a fingerspelling segment. It does not seem likely that many fingerspelling segments are shorter than this minimum time frame.

For future implementations of a similar template-based system, it might be possible to use an out-of-the-box implementation of the system described in this research, using a standard template set such as the one shown in Fig. 5.4. The generic template set in particular, might be applicable to other datasets than the Corpus NGT, considering its generic nature and the relatively few assumptions that are made about the fingerspelling signs. Nevertheless, the factors described above should be taken into account when considering new data.

Bibliography

[1] Petra Bouterse. Petra gebarentaal - handalfabet. `https://www.youtube.com/watch?v=fp6tTdEWVSI`, December 2014.

[2] Danielle Bragg, Oscar Koller, Mary Bellard, Larwan Berke, Patrick Boudreault, Annelies Braffort, Naomi Caselli, Matt Huenerfauth, Hernisa Kacorri, Tessa Verhoef, Christian Vogler, and Meredith R. Morris. Sign language recognition, generation, and translation: An interdisciplinary perspective. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*, pages 16–31, 2019.

[3] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In *arXiv preprint arXiv:1812.08008*, 2018.

[4] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[5] Helen Cooper, Eng-Jon Ong, Nicolas Pugeault, and Richard Bowden. Sign language recognition using sub-units. *Journal of Machine Learning Research*, 13:2205–2231, 2012.

[6] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[7] O. Crasborn, T. Hanke, E. Efthimiou, I. Zwitserlood, and E. Thoutenhoofd. The corpus ngt: an online corpus for professionals and laymen. In *Proceedings of the 3rd Workshop on the Representation and Processing of Sign Languages: Construction and Exploitation of Sign Language Corpora*, pages 44–49. Paris: ELRA, 2008.

[8] Onno Crasborn, Inge Zwitserlood, and Johan Ros. *The Corpus NGT. An open access digital corpus of movies with annotations of Sign Language of the Netherlands.* Centre for Language Studies, Radboud University Nijmegen, 2008. `http://hdl.handle.net/hdl:1839/00-0000-0000-0004-DF8E-6`. ISLRN: 175-346-174-413-3.

[9] Onno A. Crasborn. Personal communication, June 2020.

[10] Onno A. Crasborn. Personal communication, May 2020.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[12] L. Gupta and Suwei Ma. Gesture-based interaction and communication: automated classification of hand gesture contours. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(1):114–120, 2001.

[13] A. K. Jain, Yu Zhong, and S. Lakshmanan. Object matching using deformable templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):267–278, 1996.

[14] Anil K Jain, Yu Zhong, and Marie-Pierre Dubuisson-Jolly. Deformable template models: A review. *Signal Processing*, 71(2):109–129, 1998.

[15] Jonathan Keane, Diane Brentari, and Jason Riggle. Coarticulation in ASL fingerspelling. In *Proceedings of the North East Linguistic Society*, volume 42, 2012.

[16] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[17] Rachel I. Mayberry and Robert Kluender. Rethinking the critical period for language: New insights into an old question from american sign language. *Bilingualism: Language and Cognition*, 21(5):886–905, 2018.

[18] H. Nanda and L. Davis. Probabilistic template based pedestrian detection in infrared videos. In *Intelligent Vehicle Symposium, 2002. IEEE*, volume 1, pages 15–20, 2002.

[19] Sylvie CW Ong and Surendra Ranganath. Automatic sign language analysis: A survey and the future beyond lexical meaning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):873–891, 2005.

[20] Carol A. Padden and Darline Clark Gunsauls. How the alphabet came to be used in a sign language. *Sign Language Studies*, 4(1):10–33, 2003.

[21] S. Qiao, Y. Wang, and J. Li. Real-time human gesture grading based on openpose. In *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–6, 2017.

[22] M. A. Rahaman, M. Jasim, M. H. Ali, and M. Hasanuzzaman. Computer vision based bengali sign words recognition using contour analysis. In *2015 18th International Conference on Computer and Information Technology (ICCIT)*, pages 335–340, 2015.

[23] B. Shi, A. M. Del Rio, J. Keane, J. Michaux, D. Brentari, G. Shakhnarovich, and K. Livescu. American sign language fingerspelling recognition in the wild. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 145–152, 2018.

[24] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017.

[25] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Appendix A

Appendix

In this appendix we provide more detail about DBSCAN, the clustering algorithm used in our fingerspelling detection pipeline.

A.1 DBSCAN

Density-based spatial clustering of applications with noise, or DBSCAN, is a clustering algorithm relying on a density-based notion of clusters, designed to discover clusters of arbitrary shape [11]. The central idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points. For points p and q, a distance function $dist(p, q)$ determines the shape of a neighborhood.

There are two important parameters for the DBSCAN algorithm: Eps and MinPts, where Eps denotes the radius of the neighborhood of a point and MinPts denotes the minimum number of points in a group for it to be considered a cluster. Given a set of points D, the ϵ-neighborhood of a point p, denoted by $N_\epsilon(p)$, is defined by

$$N_\epsilon(p) = \{q \in D | dist(p, q) \leq \epsilon\}\,. \tag{A.1}$$

We differentiate three types of points: core points and border points, as shown in Fig. A.1, and noise points. Any point with at least MinPts in its neighborhood is considered a core point. Border points are those points that have less than MinPts neighbours, but are within the ϵ-neighborhood of some core point. Any point that is not either a core point or a border point is considered a noise point. In the following, we define three terms to understand the DBSCAN algorithm.

Definition 1. *(directly density-reachable) A point p is directly density-reachable from a point q wrt. Eps, MinPts if*

- $p \in N_\epsilon(q)$;

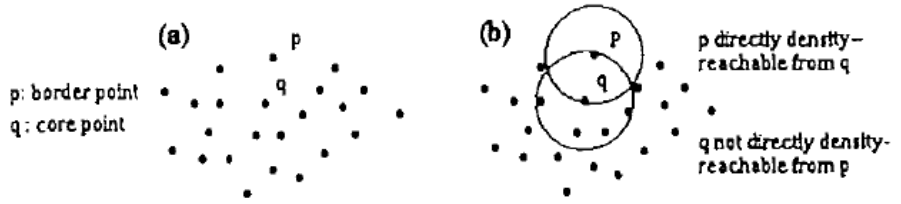


Figure A.1: Core points and border points. Taken from [11].

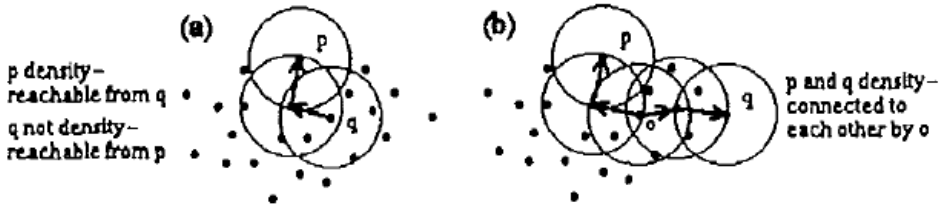


Figure A.2: Density-reachability and density-connectivity. Taken from [11].

.

- $|N_\epsilon(q)| \geq MinPts.$

Definition 2. *(density-reachable) A point p is density-reachable from a point q wrt. Eps and MinPts if there is a chain of points* $p_1, \ldots, p_n$, $p_1 = q$, $p_n = p$ *such that* p_{i+1} *is directly density-reachable from* p_i.

Definition 3. *(density-connected) A point p is density-connected to a point q wrt. Eps and MinPts if there is a point o such that both p and q are density-reachable from o wrt. Eps and MinPts.*

Fig. A.2 shows examples of Definition 2 and 3. Using the previous definitions, clusters can be defined as groups of density-connected points. The basic algorithm for finding all clusters in a set of points is as follows:

1. Find all core points by marking all points that have at least MinPts in their ϵ-neighborhood.

2. For each core point, if it is not already part of a cluster, create a new cluster. Then, find all its density-reachable points and assign them to the same cluster.

This will find all clusters in the dataset; all data that is not part of any cluster is considered noise.