UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Tracking and Recognition of Fingerspelling from Videos

Diploma Thesis

Dimos Alexandros

Supervisor: Potamianos Gerasimos

Volos 2022

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Tracking and Recognition of Fingerspelling from Videos

Diploma Thesis

Dimos Alexandros

Supervisor: Potamianos Gerasimos

Volos 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανίχνευση και Αναγνώριση Δακτυλο-συλλαβισμού από Βίντεο

Διπλωματική Εργασία

Αλέξανδρος Δήμος

Επιβλέπων: Ποταμιάνος Γεράσιμος

Βόλος 2022

Approved by the Examination Committee:

Supervisor	**Potamianos Gerasimos** Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Daskalopoulou Aspasia** Assistant Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Stamoulis Georgios** Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Gerasimos Potamianos for his guidance and support through the course of this Thesis. I would also like to thank Katerina Papadimitriou for offering me valuable advice and insight in many occasions. Moreover, I would like to thank all of the people that accompanied me through these years. My parents and my family who always supported and believed in my efforts and choices. Finally, I cannot thank enough all of the friends I have made through this journey. Each and everyone of them contributed in their own way to help me accomplish what I have so far.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Abstract

In this Thesis we examine how machine learning techniques can be utilized in sign language recognition. Sign languages are expressed through manual articulation in combination with non-manual elements such as movements of the body, hands, mouth, cheeks, eyes etc. to convey linguistic information. There exist various sign language categories, each with its own purpose and functionality. In this Thesis we focus specifically on fingerspelling, which employs manual articulations alone (hand and finger arrangements and movements) to spell words that lack dedicated signs, forming a visual representation of their sequence of letters.

We will be using the frame sequences from online videos, provided by the ChicagoFSWild dataset. The videos depict people facing a camera and performing fingerspelling in the American sign language. We focus mainly on the hand and finger motion and try to distinguish the signing hand. To track the hands we use MediaPipe and OpenPose, two multipurpose object tracking libraries. They are able to produce 3D and 2D skeleton coordinates of specified body parts such as hands, body, head etc. We proceed by comparing the performance of these libraries and then use the provided coordinates to determine the signing hand. Both 2D and 3D skeleton coordinates are used as input to our machine learning models in various combinations. The models use the aforementioned coordinates within a bidirectional recurrent neural network. Our architectures are able to recognize the fingerspelled words with satisfactory accuracy.

Περίληψη

Στην παρούσα διπλωματική εργασία θα εξετάσουμε πως τεχνικές μηχανικής μάθησης μπορούν να χρησιμοποιηθούν για την αναγνώριση νοηματικής γλώσσας. Οι νοηματικές γλώσσες εκφράζονται με κίνησεις των χεριών σε συνδυασμό με κινήσεις του σώματος, του προσώπου, των ματιών κλπ. έτσι ώστε να αποδώσουν διάφορες κατηγορίες πληροφοριών. Υπάρχουν διάφορες υποκατηγορίες νοηματικών γλωσσών, η καθεμία με το δικό της σκοπό και λειτουργικότητα. Στην παρούσα διπλωματική θα ασχοληθούμε συγκεκριμένα με τον δακτυλοσυλλαβισμό, ο οποίος χρησιμοποιεί μόνο κινησιακά μέσα (κινήσεις και διατάξεις των χεριών και των δακτύλων) ώστε να αποδώσει λέξεις που δεν έχουν καθορισμένα νοήματα, σχηματίζοντας μια οπτική αναπαράσταση των γραμμάτων τους.

Θα χρησιμοποιήσουμε τις ακολουθίες καρέ από βίντεο στο διαδίκτυο, που μας παρέχει το σύνολο δεδομένων ChicagoFSWild. Τα βίντεο απεικονίζουν ανθρώπους να εκτελούν δακτυλο-συλλαβισμό στην Αμερικανική νοηματική γλώσσα μπροστά σε μια κάμερα. Εστιάζουμε κυρίως στην κίνηση των χεριών και των δακτύλων και προσπαθούμε να ξεχωρίσουμε το χέρι που νοηματίζει. Για να ανιχνεύσουμε τα χέρια χρησιμοποιούμε τα εργαλεία MediaPipe και OpenPose, δύο βιβλιοθήκες ανίχνευσης αντικειμένων πολλαπλών σκοπών. Είναι ικανές να παράγουν τρισδιάστατες και δισδιάστατες συντεταγμένες σκελετών για συγκεκριμένα μέρη του σώματος όπως χέρια, κορμός, κεφάλι κλπ. Στη συνέχεια εξετάζουμε πόσο αποδοτικές είναι οι βιβλιοθήκες αυτές και χρησιμοποιούμε τις συντεταγμένες που μας δίνουν ώστε να προσδιορίσουμε το χέρι που νοηματίζει. Τόσο οι δισδιάστατες όσο και οι τρισδιάστατες συντεταγμένες των σκελετών χρησιμοποιούνται σαν είσοδος στα μοντέλα μηχανικής μάθησης σε διάφορους συνδυασμούς. Τα μοντέλα χρησιμοποιούν τις προαναφερθείσες συντεταγμένες σε ένα αμφίδρομο επαναλαμβανόμενο νευρωνικό δίκτυο. Οι αρχιτεκτονικές αναγνωρίζουν δακτυλο-συλλαβισμένες λέξεις με ικανοποιητική ακρίβεια.

Table of contents

List of figures

List of tables

Abbreviations

ASL	American Sign Language
ANN	Artificial Neural Network
BRNN	Bidirectional Recurrent Neural Network
BGRU	Bidirectional Gated Recurrent Unit
BLSTM	Bidirectional Long Short-Term Memory
CER	Character Error Rate
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
IoU	Intersection Over Union
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Minimum Square Error
RNN	Recurrent Neural Network
ROI	Region Of Interest
SLR	Sign Language Recognition

Chapter 1

Introduction

Sign languages are a visual method of communication mostly common between deaf people, employing hand gestures, facial expressions, and body movements. Nowadays there exist over 300 sign languages in use. In this Thesis we focus on fingerspelling in the American sign language (ASL). It is relatively simpler to perform, as it only uses hand gestures and in most occasions only one hand is needed. Nevertheless it still comprises a significant part of ASL, covering words that lack dedicated signs.

1.1 Thesis Focus

In the past decades, there has been significant progress in machine learning (ML) both in the theoretical field as well as in practical applications. An area of increasing interest is recognition of human communication, such as speech recognition or handwritten text recognition. Another problem that falls in this category is sign language recognition (SLR). Although a bit overlooked in comparison to its counterparts, SLR has caught the attention of many researchers over the last few years. With its popularity growing recently, many have started developing applications and models able to translate sign language to text. This task has proven to be a challenging one though. The sheer number of sign language types and variations, along with the fact that each signer performs hand gestures in a unique style, add complexity to the problem.

SLR requires two main modules. The first one concerns the extraction of informative features from the video frames. This necessitates the detection of the manual articulators. In our case these are the hands of the signers. For this purpose, in this Thesis we exploit some state-of-the-art object tracking tools, trained for body part recognition, that will provide us with the required information. Based on this, we obtain skeleton points of the signing hands and use them in our models as our main features. The second SLR module concerns the sequence learning approach, namely how to produce text from the feature sequence. Recurrent neural networks (RNNs) have proven quite capable in such problems, hence we utilize them in our work.

1.2 Related Work

Overall there exists a significant number of works on fingerspelling recognition in the literature. One of the earliest solutions proposed [7] dates back to 1995. There, hidden Markov models (HMMs) were used to make predictions on a set of randomly generated five word ASL sentences following a predetermined grammatical structure. Other studies that relied upon the abilities of HMMs are [8, 9, 10]. In their research, Kim et al. [8] proposed that an HMM recognizer be used in tandem with multilayer perceptron (MLP) classifiers. The MLPs use seven features representing the ASL hand shapes and make predictions both on fingerspelled letters as well as phonological features of fingerspelling. The predictions are then fed as observations to the HMM-based recognizer. In an alternative approach, [9] proposes a semi-Markov conditional random field. The difference of a typical generative HMM lies in the fact that the labels are predicted by maximizing the conditional probability of the label sequence with the visual observations sequence. In other work, [10] utilizes the fingerspelling recognizers from [8] and [9], but tries to also mitigate issues like limited training data and signer dependence. Researchers there adjust deep neural network adaptation techniques, borrowed from speech recognition, to the fingerspelling recognition problem.

In more recent years though, artificial neural networks (ANNs) have drawn a significant amount of attention. Problems such as SLR and hand gesture recognition use ANNs increasingly. Since SLR inevitably requires ASL images or videos, convolutional neural networks (CNNs) appear to be ideal for such a task. RNNs have also been utilized to cover the temporal aspect of the problem presented by many corpora with videos. Such models have been used in [11, 12, 13]. In [11] researchers propose a hand detection mechanism that crops the hand region of interest (ROI) from the original frame. The cropped ROIs are then used as input to a CNN to extract the visual features. The sequence of features is finally fed to a long short-term memory (LSTM) network. Shi et al. [12] later proposed an attention model based on a convolutional recurrent architecture, where they apply a fully convolutional network to compute an attention map. This attention map represents the importance of features at different spatial locations to the letter sequence. The frames are then processed by an RNN that unveils the temporal relationships between them. Apart from the translation side of the problem, researchers in [14] noticed the significance of a detection module. This detection module finds intervals corresponding to fingerspelling in the frame sequence, classifying each frame as positive (fingerspelling) or negative (non-fingerspelling). It is interesting to mention here that even though CNNs are great at extracting image features, simpler methods can still be used. In [13] instead of CNNs, a vanilla encoder consisting of MLPs has been constructed in order to extract the hand features.

Another significant issue many researchers might face in SLR is what kind of input data to choose for their models or how to acquire them. In [12] and [15] the input is raw images, which are processed by the network itself to extract important features. Another typical approach is to use as features the skeletal coordinates of the manual articulators via object tracking libraries. For example in [16] OpenPose is used to extract body part skeletons, and their coordinates are fed as features to the ML system. In [17] the authors propose a novel method to create both real and synthetic images generated on the basis of a 3D hand model. In subsequent research [18], the authors test their synthetic images as input to a CNN for a Japanese fingerspelling recognition task.

1.3 Our Contribution

As already indicated, our work lies along two axes. The first concerns the use of object tracking tools to drive our feature extraction process. These are MediaPipe [1] and OpenPose [19], two powerful multipurpose object tracking libraries. To our knowledge, MediaPipe has not been tested to such an extent as OpenPose in fingerspelling. Specifically we focus on:

1. Describing the two tools we chose for our experiments.
2. Using these tools to extract hand skeletons and ROIs from the image sequences.
3. Treating cases where hand detection fails.
4. Deciding automatically on the signing hand.

5. Discarding false detections.
6. Comparing the performance of the two tools.

Our second axis of work concerns sequence modeling. Specifically we focus on:

1. Describing the ML models and tools we will be using.
2. Developing a series of models using various combinations of hand skeleton data within a RNN architecture.
3. Presenting our results and conclusions.

1.4 Thesis Structure

The rest of this Thesis is structured as follows:

- Chapter 2 describes the ChicagoFSWild dataset, presenting its structure along with statistics.
- Chapter 3 includes a detailed description of MediaPipe and OpenPose along with the other algorithms we developed.
- Chapter 4 provides analytics on the performance of MediaPipe and OpenPose as well as the hand classification algorithms from Chapter 3.
- Chapter 5 describes the 2D and 3D coordinates used as visual features along with normalization schemes.
- Chapter 6 describes the ML modules and techniques.
- Chapter 7 presents our recognition results.
- Chapter 8 draws our conclusions.

Chapter 2

Dataset

In this Thesis we employ the ChicagoFSWild dataset [20]. This corpus contains fingerspelling video sequences in ASL obtained from online videos, where sign language is naturally performed. This means that no professional equipment has been used to record these videos and signers perform the sign language gestures as they personally deem better, providing natural and diverse data in the wild.

2.1 Database Description

The size of the dataset is about 14.6 Gigabytes, making it one of the largest available. It contains the online ASL videos, stored as video frame sequences. Figure 2.1 includes part of such a sequence so we can observe the movement of the signer's right hand.

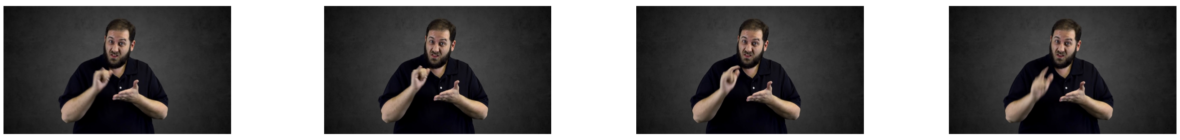


Figure 2.1: The initial frames of a video of the ChicagoFSWild dataset

The corpus consists of 7034 ASL fingerspelling videos performed by 160 different people. In our experiments we also use a subset of this dataset, named BBox, that includes the annotated bounding boxes of the ROI around the signer's hands and a label designating the signing hand(s). This subset contains:

1. 150 videos.

2. 1871 jpeg images.

3. 1919 signing hands.
4. 1870 signing hands (if one of the hands in videos where both hands sign is excluded).
5. 4 videos with 2 signing hands.
6. 7 videos with more than one person in the frame.
7. A folder of annotated hand bounding boxes.

The ChicagoFSWild.csv spreadsheet is also included, annotating the translation of each video to English text. These are the target labels that helped us train and evaluate our neural networks. Regarding the lexicon of the target labels, it consists of 8692 English words in total, with a vocabulary of 3551 unique words. We also performed a character level search and discovered that apart from the 26 alphabet letters, the characters "@&'." also occur in a small percentage of target labels.

2.2 Some Interesting Statistics

We observed that sequence lengths vary in size, as some sequences can have fewer than 10 frames, while others reach a few hundreds. A distribution of the different lengths is shown in the graph of Figure 2.2. Clearly the vast majority of videos are less than 50 frames, which is reasonable since we are dealing with individual words or small phrases.

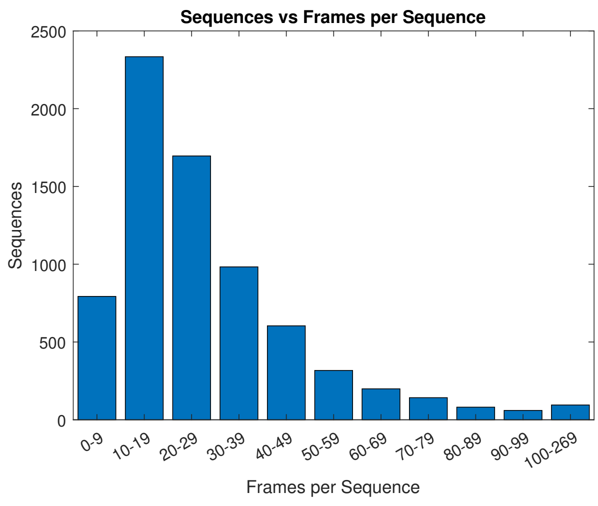


Figure 2.2: Video length histogram in the ChicagoFSWild dataset

When examining the dataset we noticed that apart from various sequence lengths, the frame resolutions also vary from video to video. Different resolutions should be taken under consideration when computer vision tools are used in the experiments. Figure 2.3 gives us an idea of the image resolution variety in the data with their frequency. Frames with a size of 640x360 pixels appeared to be the most dominant.

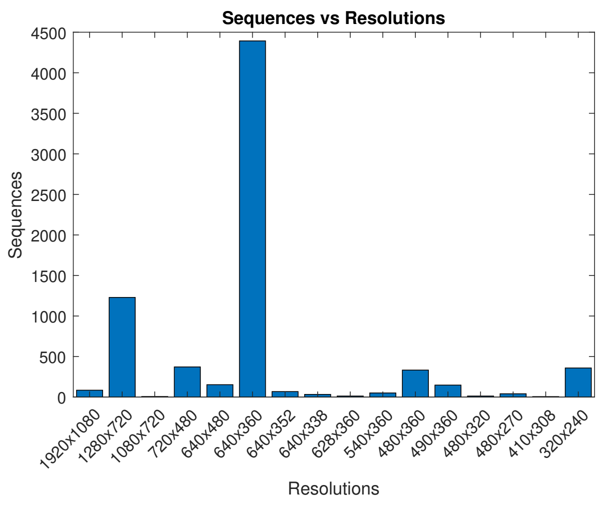


Figure 2.3: Video resolutions in the ChicagoFSWild dataset

Finally in Figure 2.4 we display the number of occurrences of the ten most frequent fingerspelled words in the dataset.

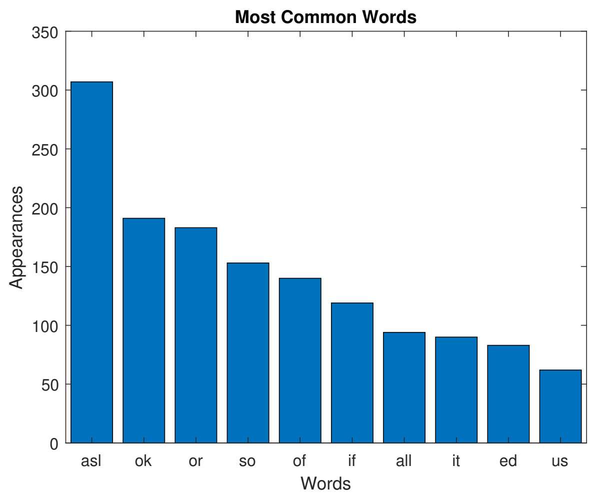
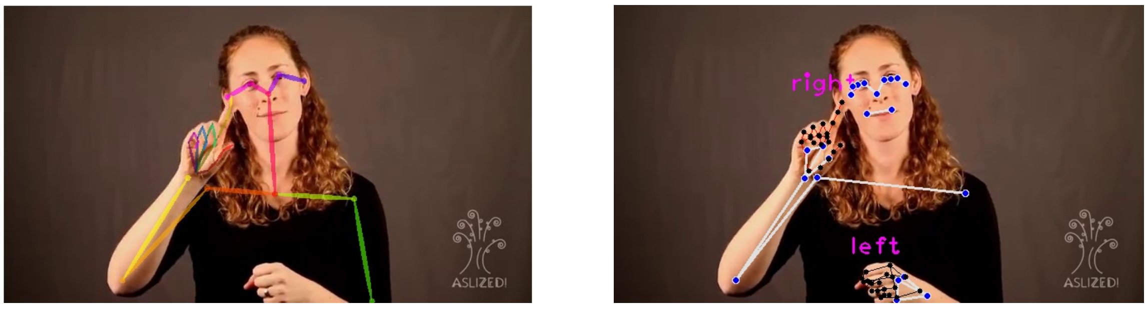


Figure 2.4: The ten most common words in the ChicagoFSWild dataset with their number of appearances

Chapter 3

Signing Hand Keypoint Detection

In this chapter we present the two computer vision libraries that we employ to detect the hand and body pose keypoints. We describe their abilities and the way each library performs detection. Both libraries extract similar keypoints, i.e. skeleton coordinates of humans, both from videos and static pictures.



(a) (b)

Figure 3.1: Examples of generated keypoints on a sample frame of the ChicagoFSWild dataset, employing (a) OpenPose and (b) MediaPipe

Such body parts help us determine the body and more significantly the hand position of the signers. In Figure 3.1 we display the skeletons generated by the libraries. It is obvious that they both focus on similar body parts, such as eyes, shoulders etc. When it comes to the hand skeletons, that we are specifically interested in, the choice of keypoints is identical. After we acquire those we make use of certain algorithms to extract and refine information. The most important issue we faced was deciding on the signing hand among the pair of hands returned by the library. Another issue was discontinuities of hand skeletons between frames, meaning that in some cases a library failed to return a hand skeleton. Finally we had to mitigate erroneous skeleton detections produced primarily by OpenPose.

3.1 MediaPipe

MediaPipe [21] has been developed by Google and provides a variety of different solutions regarding object recognition and detection such as hand detection, body pose detection, object tracking etc. It has become very popular in recent years mainly due to the fact that it is easy to set up and can run on a variety of operating systems such as Android, macOS, Windows etc. Furthermore, it supports Python, C++, and Javascript, while it does not necessarily require a GPU or a Cuda environment to run on.

As mentioned, MediaPipe offers many solutions to object tracking problems. Here, we choose to proceed with the holistic pipeline that integrates three separate models to track pose, face and hand landmarks. According to the library developers [1] this is not an easy task as the input of one model might not be compatible to another, because each one requires a different resolution. Thus, the tracking task takes place in three different stages corresponding to the three different models. More specifically, the body pose is estimated first and then the inferred landmarks help the algorithm determine ROIs around the hands and the face. The ROIs are then cropped from the original image to be fed to the hand and face MediaPipe modules. One interesting characteristic of MediaPipe is that it uses estimation from the previous frame as a guide to the object region at the current one. Therefore the whole tracking process becomes faster. In case a hand moves significantly from one frame to another it will be noticed, because of the fact that pose is the first to be determined at every frame. We depict part of the multistage process in Figure 3.2.

The pipeline is implemented as a MediaPipe graph that uses a holistic landmark sub-graph from the holistic landmark module and renders using a dedicated holistic renderer sub-graph. The holistic landmark sub-graph internally uses a pose landmark module, a hand landmark module and a face landmark module.

MediaPipe generates (if body parts are not occluded):

- 33 body pose landmarks.
- 468 face landmarks.
- 21 hand landmarks per hand.

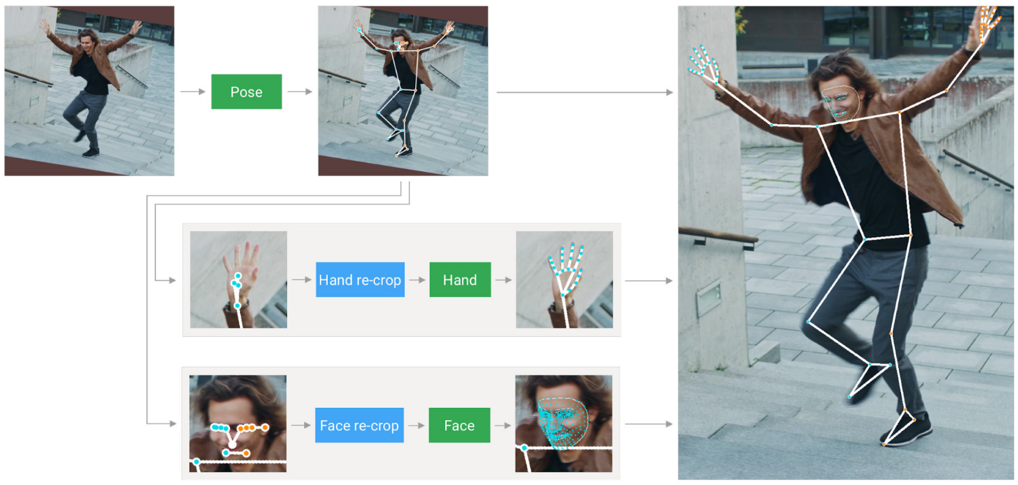


Figure 3.2: Body, hands and face fed to the MediaPipe holistic pipeline separately and re-combined to form a complete detection (image taken from MediaPipe website [1])

We mostly care about the hand and pose landmarks, because these are most relevant to our problem. All inferred landmarks are 4-dimensional and consist of *x, y, z* and visibility values, where:

- *x* represents the landmark horizontal coordinate and it is normalized between [0.0, 1.0].
- *y* represents the landmark vertical coordinate and it is normalized between [0.0, 1.0].
- *z* represents the landmark depth with the wrist considered as origin for the hand landmarks. Note that the smaller the value of *z*, the closer the landmark is to the origin. The *z* coordinate also exists for the pose landmarks but creators advice against using it, as it needs further training.
- visibility is a value between [0.0, 1.0] indicating the likelihood of the landmark being visible in the image. Note that we chose not to use that value.

The depth coordinate *z* is particularly interesting, as it is quite challenging to reproduce a depth estimate without requiring auxiliary aid, such as depth sensors, and only using RGB video as input. In Figure 3.3 we depict the 3D skeleton of the right hand of the signer in Figure 3.1b. Comparing it against the hand of that frame, it appears that MediaPipe managed to accurately capture the depth of each skeletal landmark.

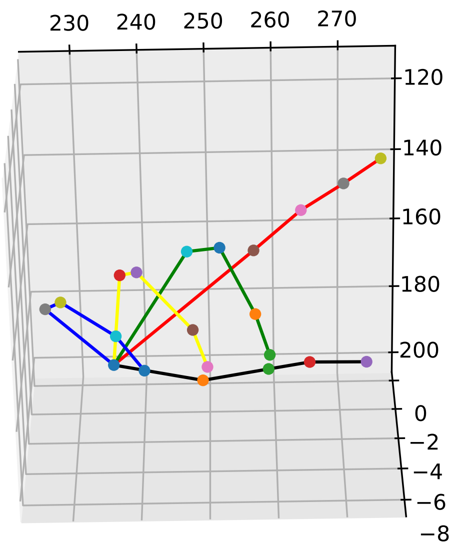


Figure 3.3: 3D hand skeleton (*x, y* and *z* coordinates are scaled) of the right hand of the signer in Figure 3.1b

The holistic solution of MediaPipe also offers a variety of configuration options. These options are set according to the input given to the pipeline, the detection quality we want to achieve, the computational cost etc. The configurations described by the developers [1] are:

- Static image mode: If set to false, MediaPipe treats the input images as a video stream. It will try to detect the most prominent person in the initial image and upon success it further localizes the hand landmarks. Then, in subsequent images it simply tracks these landmarks without invoking another detection until it loses track of them. If set to true, person detection runs on every input image.

- Model complexity: There are three levels of model complexity (0, 1, 2). Landmark accuracy as well as inference latency generally increase for higher values of model complexity.

- Minimum detection confidence: This designates the minimum confidence value (within [0.0, 1.0]) of the person detection model for the detection to be considered successful.

- Minimum tracking confidence: This designates the minimum confidence value (within

[0.0, 1.0]) of the landmark tracking model for the pose landmarks to be considered tracked successfully, or otherwise person detection will be invoked automatically at the next input image. Setting it at higher values can increase robustness of the solution at the expense of a higher latency.

We must note though that MediaPipe has its limitations. The most significant is the fact that it can not track more than one person simultaneously.

3.2 OpenPose

OpenPose was developed by the Perceptual Computing Laboratory at Carnegie Mellon University and has been the first real-time multi-person system to jointly detect human body, hand, facial, and foot keypoints. In total, it provides up to 135 keypoints for every person detected in single images or videos.

OpenPose also utilizes a multi-stage pipeline to filter the input images, although it works in a different way from MediaPipe [2]. As depicted in Figure 3.4a, an RGB image is fed as input into a two-branch multi-stage CNN, each producing two different outputs. Multi-stage simply means that the networks are stacked one on top of the other at every stage. This step is analogous to simply increasing the depth of the neural network in order to capture more refined outputs towards the later stages.

More specifically, the top branch predicts the confidence maps (Figure 3.4b) of the location of different body parts such as the right eye, left eye, right elbow and others. The bottom branch, predicts the affinity fields (Figure 3.4c), which represent a degree of association between different body parts.

Concerning their multi-stage nature, at the first stage, the network produces an initial set of detection confidence maps and a set of part affinity fields. Then, at each subsequent stage, the predictions from both branches of the previous stage, along with the original image features, are concatenated and used to produce more refined predictions. At the final stage the confidence maps and affinity fields are processed by greedy inference (Figure 3.4d) to output the 2D keypoints for all persons in the image (Figure 3.4e).

According to its documentation [19] OpenPose provides:

- 15, 18 or 25 2D body keypoints, including 6 foot keypoints.
- 21 2D hand keypoints, for each hand.

- 70 2D face keypoints.

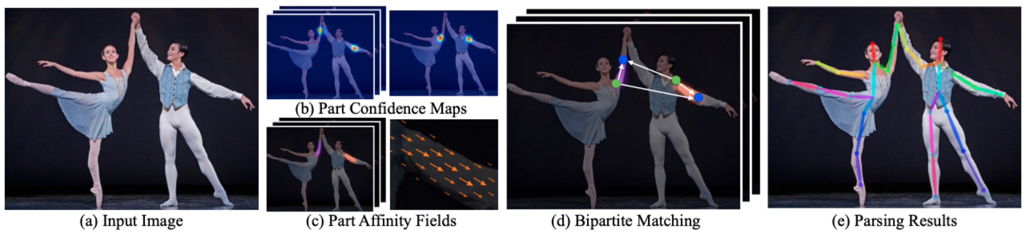


Figure 3.4: OpenPose pipeline (image taken from [2])

The produced keypoints are enclosed in a JSON file. More specifically, the contents of the JSON file are:

- Person id: Set automatically to -1.
- Pose, face and hand keypoints in separate lists of 2D keypoints representing the horizontal and vertical coordinate with a confidence.
- 3D keypoints: In future updates, OpenPose will be able to also produce 3D coordinates.

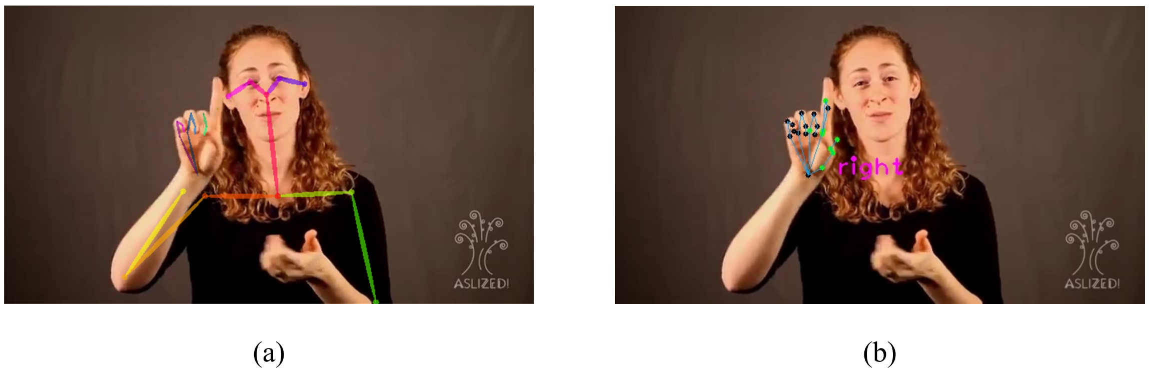


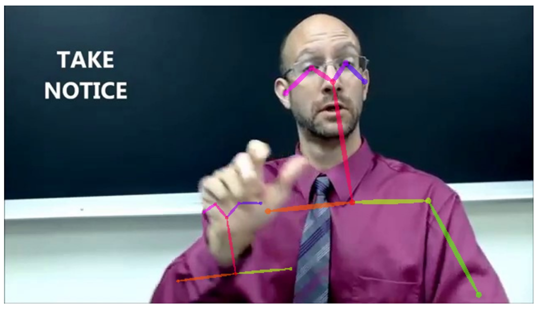
Figure 3.5: (a) Keypoints returned by OpenPose with high confidence; (b) all OpenPose derived keypoints, regardless of their confidence

In OpenPose the horizontal and vertical coordinates are not normalized as in MediaPipe. The confidence score coordinate expresses how confident the library is for a certain keypoint and ranges between [0.0, 1.0]. If it is less than 0.2, the keypoint will not be rendered on the image. We observed though that a number of keypoints with scores below 0.2 were not far from truth, so we decided to ignore the confidence factor. The image in Figure 3.5a is rendered automatically by OpenPose with low-score keypoints missing. In Figure 3.5b those keypoints are coloured with green and even though some are clearly off, others correctly form the hand skeleton.

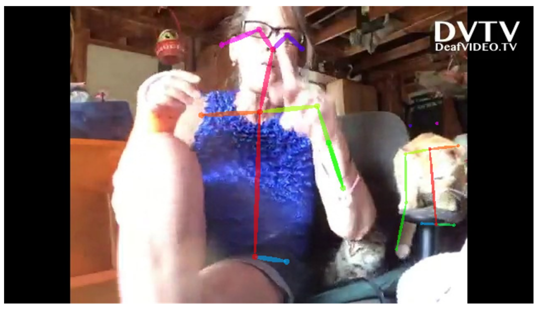
OpenPose unlike MediaPipe has the ability to track multiple human skeletons, but that ability also turned out to be a hindrance sometimes. By being eager to find human poses the library is prone to mistakes, meaning it will make detections on objects that might not be human but merely resemble a body. These can be objects in the background, animals, or even body parts of the real human.

3.3 Elimination of Falsely Detected Skeletons

We have already mentioned that it is possible for OpenPose to detect multiple persons simultaneously. Our dataset includes such frame sequences (Figure 3.7), where we need to track all the people and decide which one is the most probable signer. Besides detecting people, OpenPose can invoke body pose detections on random background objects, or body parts that are definitely not a human. The reason behind this can be image background, bizarre body positions etc. Figures 3.6a and 3.6b depict such examples, where the eagerness of the library to extract body keypoints produced erroneous results. Such detections can be problematic in the ML stage. This forces us to develop an algorithm to detect and eliminate additional skeletons, either them being real people not signing or objects misclassified as humans.



(a)



(b)

Figure 3.6: Erroneous skeletons on body part (a) and random background object (b)

Our algorithm works by using spatial characteristics of the body and hand skeletons. Generally false detections tend to have smaller skeletons, because the signer covers the largest part of the image most of the times. Furthermore, false skeletons typically do not have hands. Finally, when more than one person are present the signer's hands tend to rise higher than the

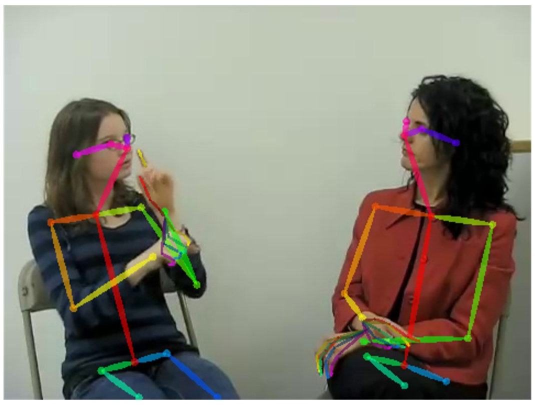


Figure 3.7: Frame of a sequence with a signer (left) and another person remaining idle (right)

non-signer's. These led us to develop the following algorithm. For each person in a frame, we extract the following info:

- Person's id: A unique number given to each detected skeleton.
- Maximum height of hand: This is the height of the hand's higher keypoint. If a hand does not exist, it is set to a large number (e.g. 10000).
- Skeleton height: We calculate it by subtracting the body skeleton's highest and lowest point. We then express it as a percentage of the image height, so that we can use a global threshold.
- Skeleton area: This is the total area that a skeleton covers. We calculate it as the area of the rectangle defined by the highest and lowest points along with the left-most and right-most points of the skeleton.

To decide on which person signs, we pass all of our detected skeletons through a number of filters. Initially, we check the frame area each detection covers. We can safely assume that in most cases the skeleton with the largest area probably belongs to the signer. Having that in mind we can eliminate false detections with smaller skeletons. Since we know the largest area, we then need to calculate the ratios of the other detected skeleton areas to this area. If this ratio is above 30%, we keep the skeleton for further evaluation. If not we discard it as disproportional, meaning that most likely it is a false detection. We then sort the list of remaining skeletons based on the maximum height of their hands from highest to lowest. Subsequently, we check the skeleton heights to estimate if a skeleton is above the threshold. Here we set that threshold at 40%, meaning that a legitimate skeleton's height should cover more than 40% of the image height. Keep in mind though, sometimes skeletons can be smaller than the threshold resulting in zero eliminations. This can happen as the number of people in the image increases. In both cases the first skeleton in the final list is the one we designate as the signer. In our subset we encounter 206 cases of images with multiple skeletons, many of them being false detections. Our algorithm succeeds in finding the correct skeleton 100% of the times.

3.4 MediaPipe and OpenPose Collaboration

Here we suggest a collaborative way to use the libraries. We set MediaPipe as our main tracking library, with OpenPose having the role to supplement the former in frame sequences with multiple people. That means OpenPose needs to first estimate how many people to expect and MediaPipe will then process the frames of the video. After we have counted the number of people in every frame and stored these values in a list, we calculate which value has the most appearances. This is the most probable number of people. This is necessary, if we want to eliminate sporadic false detections by OpenPose:

- If the most probable number of people in a sequence equals one, we can rely on MediaPipe.
- If that number is higher than one, we must consult OpenPose.

If the second case holds, we need to first determine the most probable signer of the specific frame. We find the signer with the help of our false skeleton elimination algorithm. It was observed that in some cases with more than one person MediaPipe will invoke a detection on the right person for part or even for the whole frame sequence. If that is the case, we check whether MediaPipe's detection matches the most probable signer returned by the aforementioned algorithm. To do that we rely on the intersection over union (IoU) metric, described in Section 4.1.1. Initially, we create bounding boxes for the two body skeletons, using their body pose keypoints for the IoU calculation. If the score is above the threshold of 40%, we are confident that the skeletons belong to the same person and proceed with MediaPipe's result. If this is not the case, we rely on OpenPose. In the BBox dataset OpenPose supplemented MediaPipe in 31 images.

3.5 Signing Hand Classification Algorithm

A fundamental part of this Thesis is the signing hand classification algorithm, which determines the hand performing the fingerspelling. Our dataset includes videos where signers use only one or both of their hands to sign. Moreover, one video may include multiple people but always a single signer. When our signers perform gestures with one hand the other can remain still or move arbitrarily inside the frame. We also came upon cases, where signers use both hands. We observed though that their movement and positioning is almost identical and concluded one hand is enough for our problem.

To determine if a hand signs, we propose an algorithm that uses both spatial and temporal characteristics of the hand skeleton sequence. We start with the fact that the overall motion of the fingers in a signing hand is larger throughout the video. Furthermore, the signing hands tend to be higher in the frame than non-signing hands. To calculate the height of each hand we take the average sum of the hand centroid's heights:

$$\text{Average\_Hand\_Height} = \frac{1}{N} \cdot \sum_{n=0}^{N-1} \text{hand\_height}(n)$$

To measure the total finger motion we use the distance between each pair of neighbouring fingertips. This means we calculate the distance between thumb and index, index and middle, middle and ring, ring and pinkie fingers for *N* frames. To decide the amount of finger motion, we need a way to determine how much each distance fluctuates through the frame sequence. The need of a fluctuation metric led us to consider linear regression. By performing linear regression on each distance we obtain a line that best fits the finger distances of the frames. If fingers constantly change positions the mean square error (MSE) of the regression will be higher, because the line will not be able to fit the data. If fingers remain stable through the video, their distances will form an almost linear curve and their MSE significantly decreases. We calculate the MSE for a finger in a sequence of *M* frames using the equation below, where *Y(m)* is the value of the actual fingertip distance and *Y'(m)* is the value returned by linear regression:

$$\text{MSE} = \frac{1}{M} \cdot \sum_{m=0}^{M-1} (Y(m) - Y'(m))^2$$

Based on the above, the approach to detect the signing hand is shown as Algorithm 3.1, below. First and foremost, we need to check the total MSE value for both hands. The total

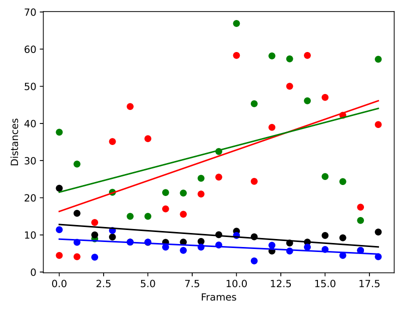
MSE is calculated by summing the MSEs of the fingers. If the total MSE of a hand equals zero, we can safely assume that only one hand is present in the video. This makes the final decision obvious. If that is not the case, we need to check the ratio of the total right and left MSEs. If the ratio is above a certain threshold, we choose based on a hand's total MSE only. If that same ratio is below that threshold, we assume that the hands move similarly, enhancing the possibility of two signing hands. As a final check we use the height difference. In the event of the absolute difference being below a threshold, we conclude that both hands sign, if not we choose the one with the highest MSE.

Algorithm 3.1: Hand classification algorithm

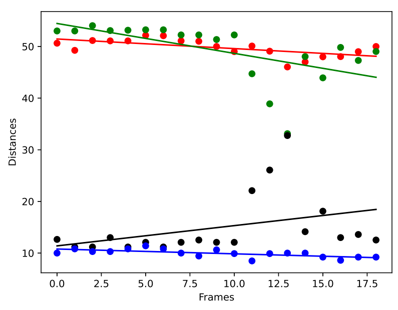
```
if right_hand_MSE = 0:
    Mark the left hand as signing
else if left_hand_MSE = 0:
    Mark the right hand as signing
else:
    max_MSE = max(right_hand_MSE, left_hand_MSE)
    min_MSE = min(right_hand_MSE, left_hand_MSE)
    if max_MSE/min_MSE < 2.5:
        Calculate the height_difference
        if height_difference < 50:
            Probably both hands sign
        else:
            if right_hand_MSE > left_hand_MSE:
                Probably the right hand signs
            else:
                Probably the left hand signs
    else:
        if right_hand_MSE > left_hand_MSE:
            Probably the right hand signs
        else:
            Probably the left hand signs
```

In the following examples we applied our algorithm on two frame sequences. In the first sequence (case of Figure 3.8) the signer uses only the right hand to fingerspell, while in the second (case of Figure 3.9) both hands are used. In the graphs below we have:

- The distance between thumb and index in red.
- The distance between index and middle in green.
- The distance between middle and ring in black.
- The distance between ring and pinky in blue.

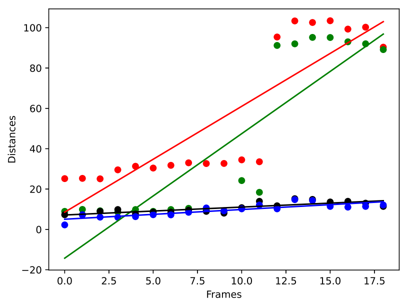


(a)

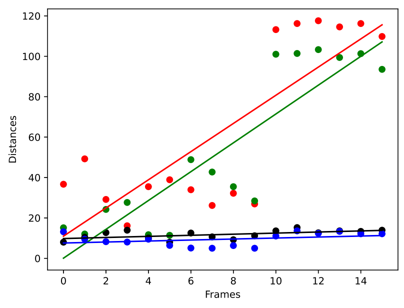


(b)

Figure 3.8: Right (a) and left hand distances (b) with the lines that best fit them in a case where the signer uses only the right hand to fingerspell



(a)



(b)

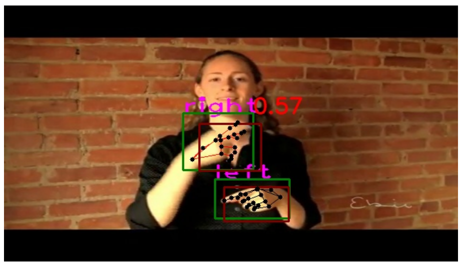
Figure 3.9: Right (a) and left hand distances (b) with the lines that best fit them in a case where the signer uses both hands to fingerspell

Figures 3.8a and 3.8b demonstrate how much distances diverge from their best fitting lines. It becomes obvious that the right hand produced larger MSEs, leading us to believe it is signing. The right hand's MSE of 444.32 in comparison with the left's 49.59 confirms our speculation. If we run our algorithm on these results, the calculated ratio is much higher than the threshold, leading to a decision based on the two MSEs alone.

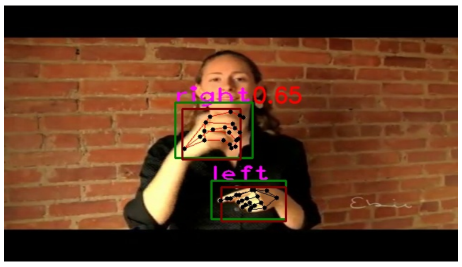
In the second example, where both hands sign, we notice some similarity between the graphs of Figure 3.9a and 3.9b. The total MSE is 698.85 for the right hand and 919.99 for the left hand. The calculated ratio is smaller than 2.5, leading to the case of two possible signing hands. We also confirmed that their height difference is below 50, leading us to decide that this is a two signing hand scenario.

3.6 Missing Hand Recovery

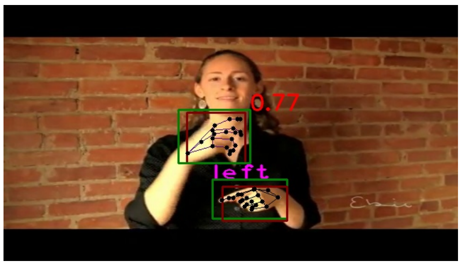
Sometimes an object tracking library might fail to detect the hand skeleton in a frame. This phenomenon can be caused due to image noise, motion blur, poor video resolution, background color being almost identical with skin color or unusual hand positioning (hands being too close to the camera). There is not much to be done, if the library fails in consecutive frames, although we are able to calculate isolated missing hands. We do that by averaging the horizontal, vertical and depth coordinates of each hand skeleton keypoint from the previous and the next frame.



(a)



(b)



(c)

Figure 3.10: Right hand skeleton in frame (c) is calculated by averaging skeletons in frames (a) and (b)

We demonstrate our technique in the frame of Figure 3.10c, where MediaPipe failed to return the right hand skeleton. The keypoints drawn on Figure 3.10c represent the missing right hand skeleton accurately enough. They seem to fit the hand shape well, reaching an IoU of 0.78 with their respective ground truth bounding box.

3.7 Region Of Interest Extraction

In order to proceed to the next chapter, we need to explain how the detected hand's ROI is extracted. The ROI represents the image area that contains the detected hand. Usually ROIs are enclosed in bounding boxes, which are essentially rectangular areas that determine the boundaries of the ROI. In Figure 3.10 the green rectangles are such boxes. In order to determine the rectangular area around the detected hand we begin by organizing the vertical and horizontal coordinates of its keypoints in separate lists. We now need to pinpoint the starting corner of the rectangle. Our starting corner is the point located on the lowest horizontal and vertical coordinate (*min_x*, *min_y*). Subsequently, we calculate the width and height of our bounding box. The width is calculated by subtracting *min_x* from the maximum horizontal coordinate. We perform the same action with the vertical coordinates and *min_y* to calculate the height. The starting corner along with the width and height values enables us to draw the bounding box. To be certain that the bounding box contains the whole hand, we expand it by a certain margin.

Chapter 4

Performance Evaluation of Hand Detection Algorithms

In this chapter we report the results of MediaPipe and OpenPose on the BBox subset of the ChicagoFSWild dataset and we determine which one works better for our problem. We also display results on the efficiency of the hand classification algorithm. It should be noted that OpenPose has been available for a longer time than MediaPipe and has already been tested in experiments similar to ours [16]. MediaPipe on the other hand has not been tested much on our problem, so it is of interest to provide a comparison between the two.

4.1 Performance Metrics

To draw concluions on the performance of MediaPipe and OpenPose in hand tracking, we need to decide on the best configuration for each. After this is determined, we proceed by making the final comparison between the two based on the scores of each library. These scores are drawn from the following metrics: IoU, recall, precision and F1-Score. Te be more efficient in our work we organize the library results in a hand dictionary containing information about:

- The keypoint list, arranging the 21 3D or 2D coordinates in a list.
- The hand bounding box, storing the starting corner, the width and the height needed to draw the rectangular ROI.
- The hand centroid, that is the center of the bounding box.

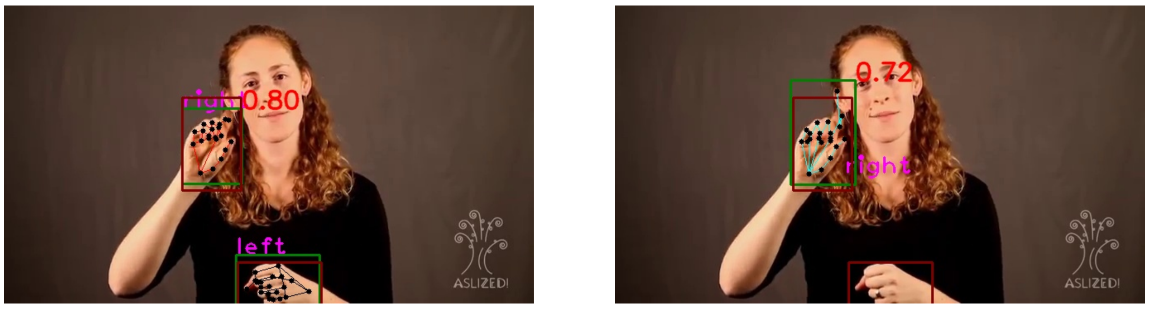
- The hand type, categorized by the library as right or left.
- The signing value, originally set as 0. The signing hand decision algorithm later sets it to 1 for signing or 2 for non-signing.

It is also important to note that the hand keypoints and ROIs were scaled to a resolution of 640x360 after the detection. The ground truth bounding boxes also correspond to this resolution.

4.1.1 Intersection Over Union

The IoU is an evaluation metric used to measure the accuracy of an object detector, ranging from 0.0 to 1.0. It uses the ground truth bounding box, which is normally annotated by the creators of the dataset and a bounding box inferred by an algorithm. IoU is calculated as a ratio between the overlapping area and the total area of the two bounding boxes:

$$\text{IoU} = \frac{\text{Area\_Of\_Overlap}}{\text{Area\_Of\_Union}}$$



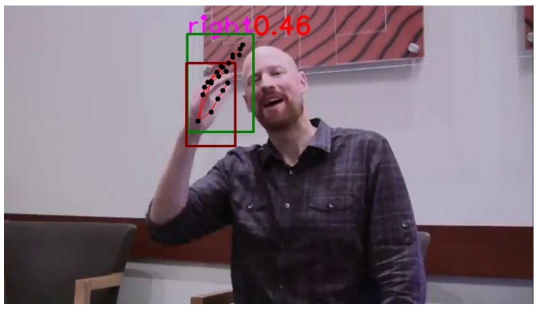
(a) (b)

Figure 4.1: Signer image with the predicted hand skeletons, bounding boxes and IoU value for MediaPipe (a) and OpenPose (b)

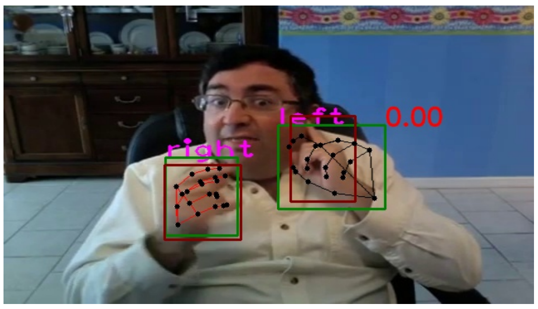
In Figure 4.1a and Figure 4.1b we display the bounding boxes and their IoU scores. The ground truth bounding boxes are colored in red, while the green ones are those estimated by the library keypoints. In each figure we see the calculated IoU on the top right corner of the bounding box. After we have determined the bounding box of the signing hand, we choose the ground truth bounding box annotated with one and calculate their IoU. The total IoU of the dataset is calculated as the average of the IoUs of every frame.

4.1.2 Precision, Recall and F1-Score

In order to understand precision and recall we need to explain how we use the terms true positive, false positive and false negative. In general, a positive prediction is when we classify a hand as a signing one. A true positive is when the IoU between the predicted bounding box and ground truth bounding box is above 0.5, on the other hand a false positive is when the IoU is less than 0.5. A false negative represents the failure to invoke any detection, where there should be one. In Figure 4.2 we demonstrate two false positive cases. In the first image (Figure 4.2a) the false positive was caused because of inaccurate detection results, resulting in IoU less than 0.5. In Figure 4.2b we have accurate bounding boxes for both hands but still get an IoU of 0.0. The problem here is that the algorithm classifying the signing hand did not make a correct decision, marking the wrong hand as signing.



(a)



(b)

Figure 4.2: Cases of false positives with a low IoU (a) and false hand classification (b)

To generate scores for our metrics we count the cases of false positives, true positives and false negatives. The first metric we calculate is precision. Precision deduces how many, out of all the predictions that our model terms positive, are actually positive:

$$\text{Precision} = \frac{\text{True\_Positives}}{\text{True\_Positives} + \text{False\_Positives}}$$

Precision represents the detection quality of MediaPipe and OpenPose. An increased number of true positives means more accurate hand skeletons. Precision can give us an idea of how adept our signing hand classification algorithm is, but recall is more appropriate for that purpose. Recall deduces out of all the positive cases, how many positive cases the algorithm was able to identify:

$$\text{Recall} = \frac{\text{True\_Positives}}{\text{True\_Positives} + \text{False\_Negatives}}$$

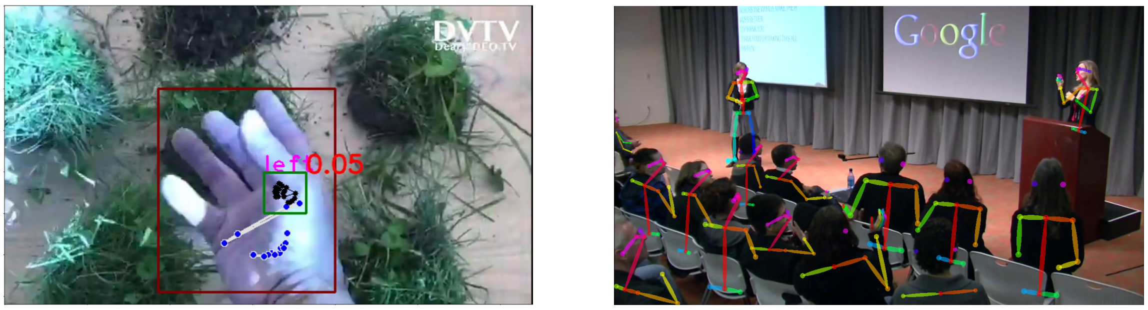
For the above reason, it is preferable to use recall to later check how efficient we are in determining the signing hand. Finally, the F1-Score is the weighted average of precision and recall:

$$\text{F1}-\text{Score} = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$$

Along with the total IoU, F1-Score is a cumulative indicator for the efficiency of both libraries and algorithms.

4.2 Hand Classification Performance

To measure the performance of the signing hand classification algorithm on the BBox subset, we use the recall statistic. We found a total recall of 89.73%, something not representative of the algorithm's efficiency. The reason behind this are poor detection results or no detections at all accumulating false negative and false positive cases. This raises the need of a different evaluation approach. We decided to rely on the recall of each individual frame sequence. If a recall is above a certain threshold, we consider the hand classification to be successful. For our experiments we set the threshold at 0.6 and found eleven sequences with a recall less than 0.6. In eight of these sequences, only one hand is present making the final classification obvious. Out of the three remaining two-hand cases, it is worth discussing two videos.



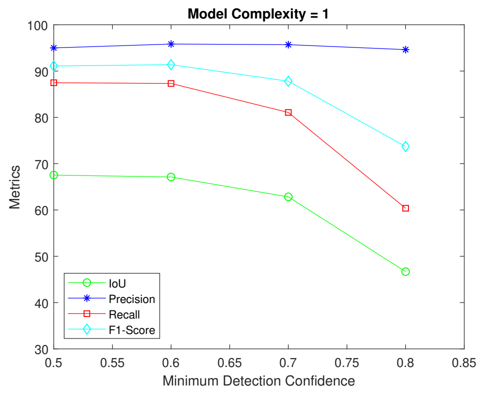
(a) (b)

Figure 4.3: Atypical frames found in the ChicagoFSWild dataset

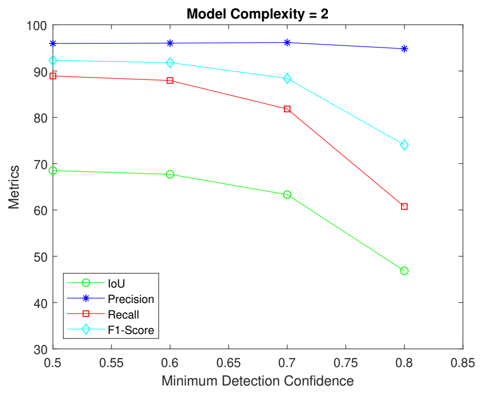
The first is a ten-frame sequence with nine false negatives and one false positive, which collectively result to a recall of 0. The false positive frame is shown in Figure 4.3a. It is important to notice here that both libraries failed when processing this sequence. The second sequence is again a case not representative of our dataset. From Figure 4.3b it becomes obvious that the video contains approximately fourteen people and the signer stands far from the camera. These factors challenge both the object tracking libraries and the classification algorithm. In conclusion, the algorithm fails completely in detecting the correct signing hand in 3 occasions, which means it succeeds in 98% of the frame sequences.

4.3 MediaPipe and OpenPose Performance Comparison

In this section we present our results and conclusions for MediaPipe and OpenPose. MediaPipe has many configuration options and since it has not been tested much in datasets like ChicagoFSWild, we had to check different combinations to be certain about our results. This included tuning the model's complexity, minimum detection confidence and minimum tracking confidence.



(a)



(b)

Figure 4.4: MediaPipe results with model complexity 1 (a) versus model complexity 2 (b)

In Figures 4.4a and 4.4b we display how MediaPipe performed on the BBox dataset. We have concluded that the two most influential parameters are minimum detection confidence and model complexity. Regarding minimum tracking confidence, it was observed that it is not as influential, yielding no significant changes in the final results. It has become obvious that by gradually increasing the minimum detection confidence all of our metrics deteriorate. Thus, it is preferable to keep its default value of 0.5. Model complexity does not induce the same amount of change, but still setting it to 2 results in improvements. Furthermore, MediaPipe's static image mode should be set to false, as in this case more detections are produced. Since setting image mode to false implies that MediaPipe should expect a video input, we are concerned on how to warn MediaPipe for the end of the video. The reason behind this concern is that the model will not invoke another detection until it loses track of the current keypoints. This could cost us a detection, if it occurred in the first frame of the next sequence. So, after each sequence of frames has been processed, we present a black image to the model. By adding the black image, we force this event and prepare the library for the new frame sequence.

OpenPose also offers a wide variety of configurations, but with a more standard setup that we could not tune much. The OpenPose configuration and variables of our choice were:

- –model_pose_BODY_25: The fastest model for CUDA version with high accuracy, also including foot keypoints.
- –image_dir: The sequence frame directory.
- –hand: To enable hand keypoint detection.
- –write_json: To write OpenPose output in JSON format.
- –write_images: Directory to write the rendered frames.

Table 4.1: Performance of the MediaPipe and OpenPose libraries on the BBox set of the ChicagoFSWild dataset

Library	IoU	Precision	Recall	F1-Score
MediaPipe	68.50%	95.96%	88.93%	92.31%
OpenPose	45.19%	82.00%	56.00%	66.55%
MediaPipe and OpenPose	69.22%	95.55%	89.73%	92.53%

Table 4.1 shows that MediaPipe produced better results. The final row demonstrates how a collaborative effort between the two, described in Section 3.4, leads to further improvements.

Chapter 5

Feature Extraction

In this chapter we describe the postprocessing of the landmarks returned by MediaPipe and OpenPose. We explain how we convert these landmarks into feature vectors, which are subsequently fed to our ML models. We then list the different types of feature vectors as well as their dimensions. The second part of this chapter includes the normalization schemes we have selected for our input vectors.

5.1 Extracted Features

We base our visual features on the 2D and 3D coordinates extracted from MediaPipe's and OpenPose's hand skeletons. We use the algorithms from Chapter 3 to create a pipeline which processes these hand skeletons in order to produce visual feature sequences. Our pipeline is shown below:

1. Extract the 2D and 3D coordinates from the hand skeletons in each frame. Organize them in right and left hand sequences.

2. Apply the collaborative algorithm between MediaPipe and OpenPose to mitigate multiple people and false detections. This step is not necessary, if we only use MediaPipe.

3. Recover any isolated discontinuities (missing detections) in the hand sequences.

4. Use the hand classification algorithm to determine which hand performs the fingerspelling.

5. Return the final right, left and signing hand sequences.

We can now create the feature vectors from each hand sequence. Based on the hand sequence as well as the library used, we can distinguish the following types:

- Feature vectors containing the 2D and 3D MediaPipe coordinates of the signing hand sequence.
- Feature vectors containing the 2D and 3D MediaPipe coordinates of both right and left hand sequences.
- Feature vectors containing the 2D coordinates of the signing hand sequence produced by the collaborative effort between MediaPipe and OpenPose.
- Feature vectors containing the 2D coordinates of both right and left hand sequences produced by the collaborative effort between MediaPipe and OpenPose.

As mentioned before, MediaPipe produces 21 3D keypoints for each hand. For each time step of the sequence we create a feature vector by flattening the hand keypoints into a 63-dimensional vector. If we now discard the depth coordinate from our keypoints, what remains are the 2D coordinates. This results in dimensionality reduction and we now have 42-dimensional feature vectors. We also performed experiments, where the feature vectors were a product of MediaPipe's and OpenPose's collaboration. OpenPose also extracts 21 3D keypoints for each hand, only this time the third coordinate represents visibility instead of depth. So, during the collaboration phase we are forced to work exclusively with the horizontal and vertical coordinates. This results again in 42-dimensional feature vectors. It is worth mentioning here that both libraries order the hand keypoints the same way, making unnecessary any reordering by our part.

5.2 Normalization Schemes

One final operation on the feature vectors before they are used as input to the ML stage is to normalize them. By applying normalization schemes we expect to develop better fitting models as well as more accurate results. In this Thesis we test three normalization techniques. We also consider the case where no normalization is applied.

5.2.1 Wrist Normalization

Inspired by other work [16], we normalize every hand skeleton keypoint based on the wrist keypoint. This means that we consider the wrist as the respective origin for the whole hand. We apply this normalization technique by subtracting the wrist keypoint's values from the other keypoints. This action occurs at the frame level, so that every feature vector begins with the origin.

5.2.2 Min-Max Scaling

Another normalization technique we test is Min-Max scaling. The idea behind this technique is to scale all our input data into the range [0.0, 1.0]. This type of scaling is mainly used when variables that are measured at different scales do not contribute equally to the model fitting. This situation could end up creating a bias. Although this does not apply much to our hand coordinates, we still thought it would be interesting to test this type of scaling. To perform the scaling we apply the following formula to our coordinates:

$$\text{coordinate\_sc} = \frac{\text{coordinate} - \text{min\_coordinate}}{\text{max\_coordinate} - \text{min\_coordinate}}$$

The *min_coordinate* and *max_coordinate* are acquired at the sequence level where *max_coordinate* is the largest coordinate in the whole feature vector sequence and *min_coordinate* the lowest.

5.2.3 Hand Size Normalization

Finally, we decided it would be interesting to try and scale the feature vectors in a sequence with the average hand size of that sequence. We estimate the hand size by employing the distances between the wrist and the four keypoints of the hand where the main fingers start. The average of the sum of these distances gives us an idea of the hand's size in a frame. To apply this rationale in a sequence of N frames we use the formula:

$$\text{sc\_svalue} = \frac{\sum_{n=0}^{N-1} \text{sum\_of\_hand\_distances}(n)}{4 \cdot N}$$

We divide the values of our feature vectors with the *sc_value*. This technique is used in addition to wrist normalization.

Chapter 6

Recognition Module: Basic Neural Network Components and Adopted Architecture

In this chapter we describe the ANN components we use in our ML models. We have already established in previous chapters that our input data are sequential. Due to their efficiency with this type of data, we decided to use RNNs as our core architectural component. We also explain the rationale behind choosing connectionist temporal classification (CTC) as the loss function for our problem, and we describe how we handle the output of RNNs using CTC. Finally, we present the adopted architecture of our model.

6.1 Recurrent Neural Networks

RNNs specialize in temporal and ordinal problems that usually include sequential or time series data. Fields such as speech recognition, language translation and natural language processing are already using RNN models in numerous applications. The unique characteristic that makes them excel in such fields is their ability to take information from prior inputs to influence the current output by employing hidden states. The hidden state is propagated forward through time to help the network decide which characteristics of the input sequence should be kept. Another unique feature of RNNs is the fact that they share parameters across each layer of the network. These parameters are still adjusted by backpropagating gradients, only this time the algorithm differs a little. The difference lies in the fact that backpropagation in RNNs sums errors at each time step therefore calling the algorithm backpropagation through time.

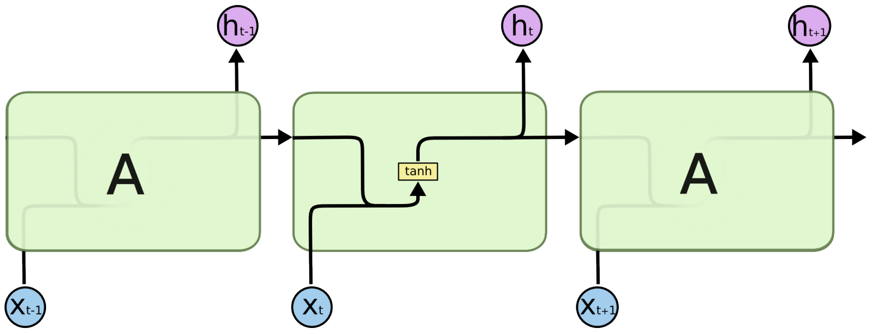


Figure 6.1: Consecutive time steps of a simple RNN (image taken from [3])

In Figure 6.1 we display three time steps of a typical RNN architecture. The repeating modules have a simple structure with only one *tanh* layer. Notice how in each time step the hidden state h_t is both the current output and the input to the next time step. The number of input and output features also determines the network type. We can categorize RNNs in four different types:

- One to one, where a single output is inferred from a single input.
- One to many, where multiple outputs are inferred from a single input.
- Many to one, where a single output is inferred from multiple inputs.
- Many to many, where multiple outputs are inferred from multiple inputs.

For example our problem is a case of "many to many", since our input is composed of multiple feature vectors and our output includes sentences containing multiple alphabetical characters.

6.1.1 Bidirectional Recurrent Neural Networks

In this Thesis we select a more advanced type of RNN, namely the bidirectional RNN (BRNN). The typical RNN relies only on past and present events to make predictions, however many real world problems also rely on future events. For example in many language prediction problems the meaning of a sentence is not complete without looking or waiting for future events. These events could be words at the end of the sentence that complete its meaning. These linguistic dependencies are very common in spoken and written speech, making

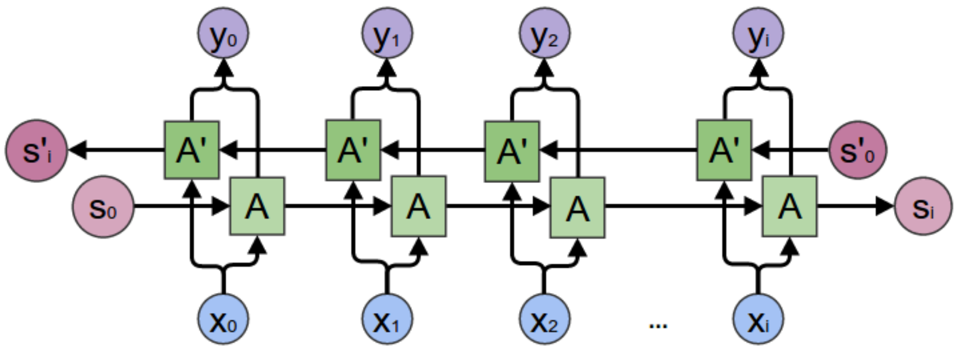


Figure 6.2: Example of the forward and backward RNN structure (image taken from [4])

capturing and analyzing both past and future events very helpful. BRNNs are able to capture these time dependencies by combining two separate RNNs. The first RNN moves forward through time, beginning from the start of the data sequence while the second RNN moves backwards in time, beginning from the end of the data sequence. Each RNN works as an individual layer producing outputs for the forward and backward pass. The outputs are then concatenated to form a single output as shown in Figure 6.2.

6.1.2 Long Short-Term Memory and Gated Recurrent Unit Models

RNNs have proven very competent in capturing the dependencies between sequential data, but the length of an input sequence could affect their performance. To counter this, a special kind of cell was devised and integrated to RNNs, yielding the LSTM networks [22] – see also Figure 6.3.

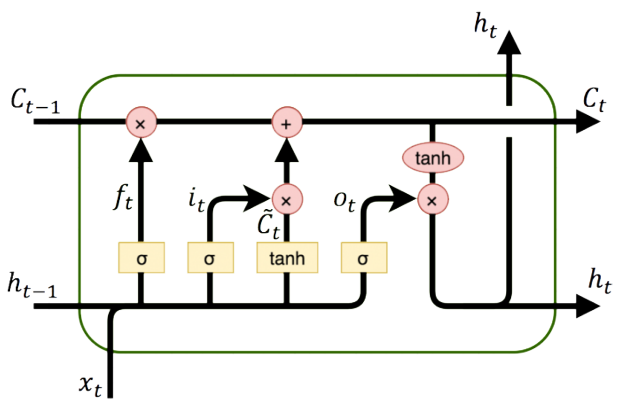


Figure 6.3: Layer architecture of LSTM cell (image taken from [5])

More recently, the gated recurrent unit (GRU) was introduced by Cho et al. [23], as a modification to the LSTM. It was discovered that the results produced by the newly introduced

GRU architecture were comparable to the already established LSTM. Since then, GRUs have been used extensively in many practical applications where they proved very competent in solving the vanishing gradient problem introduced by simple RNN cells. They do not differ much from their LSTM counterparts, although they use a simpler design (see also Figure 6.4).

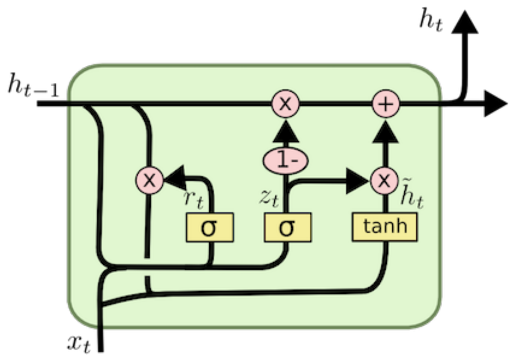


Figure 6.4: Layer architecture of GRU cell (image taken from [3])

6.1.3 Layer Normalization

Another type of layer we tested in our RNN architectures is the normalization layer. Introduced by Hinton et al. [24], layer normalization is inspired by batch normalization. Batch normalization computes a mean and a variance by using a distribution of the summed input to a neuron over a mini-batch of training cases. The mean and variance of each training case are used to normalize the summed input to that neuron. Because of its batch dependency this type of normalization is rather impractical for RNNs. So, instead of computing mean and variance over a batch, layer normalization calculates them using the summed inputs to the neurons in a layer. Layer normalization is said to reduce training time, as well as making the RNN more stable.

6.2 Connectionist Temporal Classification

We next describe the loss function we used in our experiments. Problems like ours are similar to speech recognition or hand writing symbol recognition and in general situations where the input and output alignment is ambiguous. Our problem's dataset comprises of video frame sequences and corresponding transcripts, but there is no alignment rule to match a character to certain frames. Devising some alignment rule for the data, e.g. a slice of the video frames to correspond to an output character could be a solution. Keep in mind though that applying such a strict rule to videos with different frame rates and different fingerspelling rates could lead to unpredictable results.

6.2.1 Alignment

CTC [6] provides us with an alternative for problems where input and output alignment is unclear. To give a better understanding of CTC we rely on a simple recognition problem with no alignment rules. There, we map the input sequences $X = [x_1, x_2, \dots, x_N]$ to corresponding output sequences $Y = [y_1, y_2, \dots, y_M]$, i.e. transcripts. The obstacles lying ahead are:

- X and Y usually have various lengths.
- The ratio of lengths between the input X and the corresponding output Y varies.
- The possibility of an accurate alignment rule between X and Y is low.

When it comes to the alignment of input X and output Y, the input length will probably be longer or equal than the corresponding transcript's. This forces the algorithm to produce an output sequence equal to the input in length. Consider an input X of length 6 and the target word is "cat", Y = [c,a,t]. A naive approach is to assign one output character to each input step and collapse repeating characters in the output to acquire the desired word.

Input	x_1	x_2	x_3	x_4	x_5	x_6
Alignment	c	c	a	a	a	t
Output		c		a		t

Table 6.1: Naive alignment rule between input and output sequence

Table 6.1 demonstrates this technique where the output Y = "ccaaat" collapses to the word "cat". Although this approach worked here, it is rather simplistic for real world applications mainly due to the following reasons:

- In many cases forcing every input step to align to some output is not realistic. Long stretches of silence are such a case. Usually these parts of speech do not correspond to some output.

- Many words contain repeating characters, such as “hello” or “soon”. Our initial approach will therefore delete correct characters.

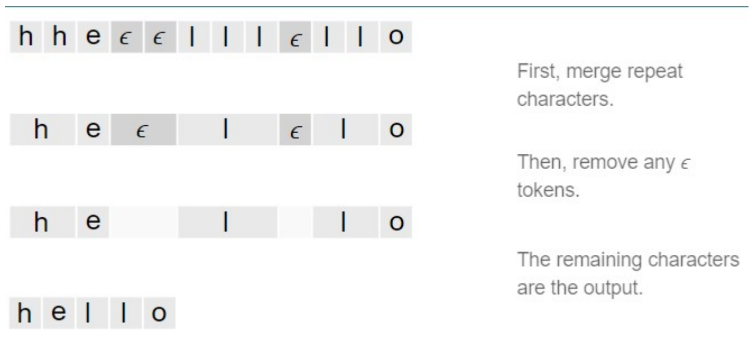

h h e ϵ ϵ l l l ϵ l l o
First, merge repeat characters.
h e ϵ l ϵ l o
Then, remove any ϵ tokens.
h e l l o
The remaining characters are the output.
h e l l o

Figure 6.5: Example of the CTC alignment technique (image taken from [6])

To overcome such difficulties, the CTC algorithm introduces a new token to the possible output characters. This token is called the blank token and is represented by the letter ϵ. It corresponds to nothing, so we simply discard it from the output sequence. That characteristic also solves the repeating character problem because in order to have a valid alignment there must be a blank token between the repeating characters. Figure 6.5 demonstrates how the blank token contributes to the alignment method, dealing with cases of long stretches of silence as well as repeating characters.

6.2.2 Loss Function

RNNs using CTC as a loss function produce a probability distribution matrix, containing the probabilities of each possible element at every time step. The possible elements are predetermined and in our case are alphabetical characters. Acquiring this distribution enables us to infer a likely output or to assess the probability of a given output. Our current goal is to train the model to maximize the probability it assigns to the right answer, namely a character alignment. To do this, we need to calculate the conditional probability *p(Y|X)* of the possible alignments for a number of T time steps:

$$p(Y|X) = \prod_{t=0}^{T-1} p_t(a_t|X)$$

where $p_t(a_t|X)$ expresses the per time step probability of a possible output character. A straightforward approach would be to calculate the probability of every possible alignment between all elements over the entire sequence and keep the one with the highest probability. Despite returning a correct result this method is computationally expensive. The problem is the sheer number of possible alignments that could end up slowing the whole process. That is why a cost effective method is needed when calculating the CTC loss. Dynamic programming algorithms are a good practice for such problems where the goal is to maximize the probability of the labelling given a sequence.

6.3 Output Decoding

It has become clear from the previous section that an efficient way of decoding the output in order to get the alignment with the maximum probability is crucial. We already mentioned dynamic programming algorithms as a solution to these types of problems. However since dynamic programming covers a wide variety of algorithms, there is no standard decoder for this purpose. We will later observe that different decoders produce different results for the same output, therefore affecting the final accuracy of our models. In our applications we tested two of the most popular decoders for such problems. Initially, we tested the more straightforward greedy decoder and then proceeded with the more complex beam search decoder. The greedy decoder algorithm performs the following steps:

1. Find and keep the maximum probability element at each time step.

2. Collapse blanks and all repeating characters not separated by blank.

3. Remove blanks from the sequence.

4. Concatenate the remaining elements.

The greedy approach has the advantage of being fast and computationally inexpensive. Despite its advantages, greedy decoder is adept at overlooking alignments that could possibly produce a higher probability. Beam search might not be as simple and fast, but it can produce more sophisticated alignments.

The beam search algorithm creates possible character alignments or beams and calculates their total probabilities. A beam is a character sequence with a corresponding probability score. The higher the score the higher the possibility we have located a good character alignment. The biggest difference between the two decoders lies in the fact that beam search is able to score multiple alignments per time step. Essentially beam search resembles the greedy decoder when we force it to only check one alignment per time step. Furthermore, in beam search if more than one beams are equal, they are merged and their probabilities are added. For example both a beam containing the text "aϵ" and the beam containing "ϵa" produce "a". Our beam search implementation follows the methodology used in [25].

6.4 Adopted Model Architecture

In this section we present the two neural network architectures we developed for the purposes of this Thesis. Figure 6.6 displays the complete pipelines with their main blocks. The difference between the two lies in the kind of input they accept. The pipeline in Figure 6.6a uses the signing hand feature vectors as input, whereas the pipeline in Figure 6.6b accepts feature vectors from both right and left hand. The data for the two hands are fed separately into the model. Other than that, both pipelines use the same building blocks and output the same type of probability distribution.

1. The first blocks of the pipelines are embedding layers. They are essentially fully connected layers and their functionality is to increase the dimension of the input feature vectors before they proceed to the next block. The pipeline in Figure 6.6b yields two embedding layers, one for each hand. Their respective outputs are then concatenated in a single vector.

2. The recurrent layers are the core component of our models. We rely on them to capture the temporal dependencies of the sequential data. They are comprised of a normalization layer and a BRNN layer. We tested both GRUs and LSTMs in our BRNNs.

3. The time distributed layer is constructed by a number of stacked fully connected layers. We use the term "time distributed" because we apply the same fully connected layers to every temporal slice of the input. This layer produces the final probability distribution of the possible characters.

4. The dropout layers between the main layers are used to improve model generalization.

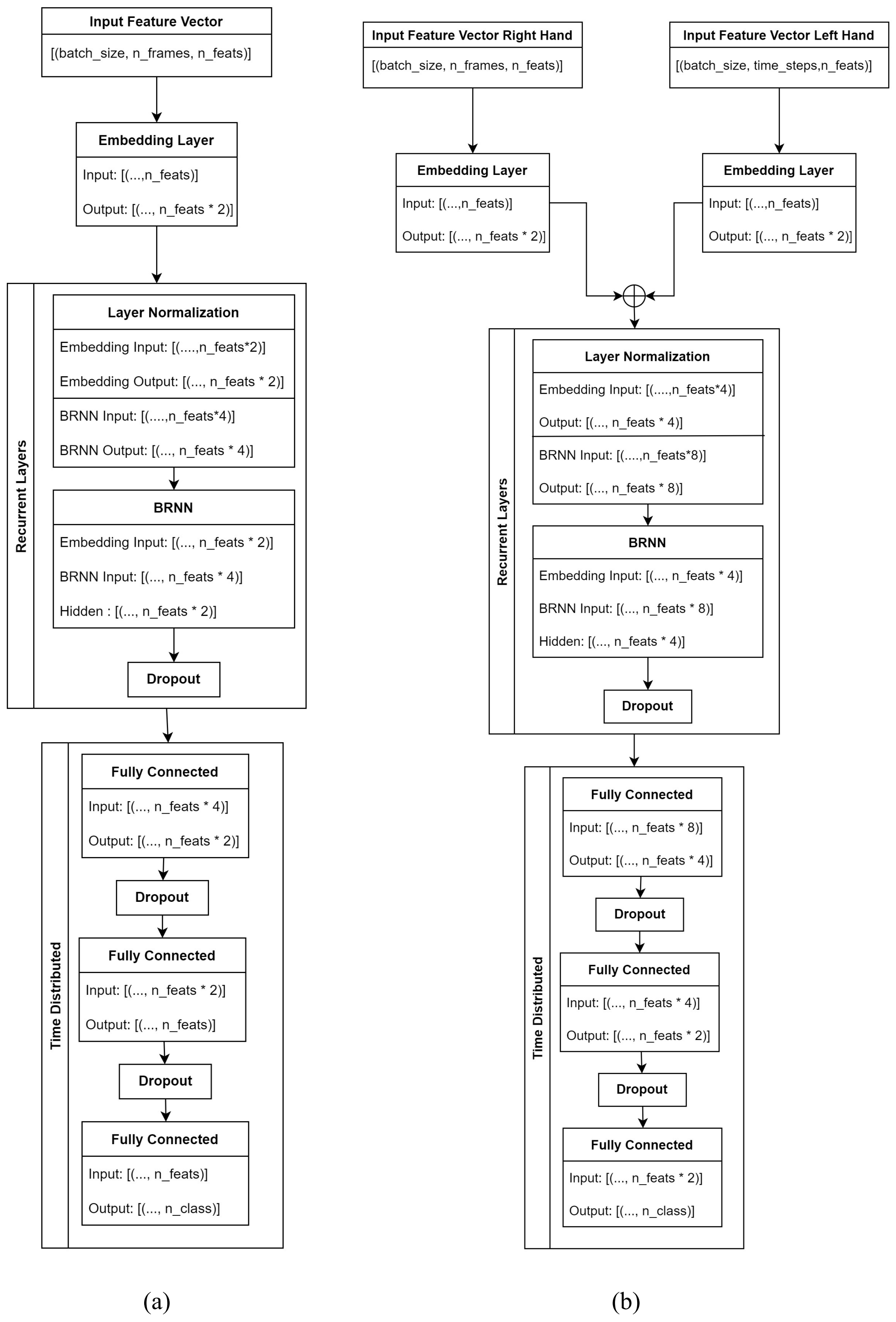

Input Feature Vector
[(batch_size, n_frames, n_feats)]
Embedding Layer
Input: [(...,n_feats)]
Output: [(..., n_feats * 2)]
Recurrent Layers
Layer Normalization
Embedding Input: [(....,n_feats*2)]
Embedding Output: [(..., n_feats * 2)]
BRNN Input: [(....,n_feats*4)]
BRNN Output: [(..., n_feats * 4)]
BRNN
Embedding Input: [(..., n_feats * 2)]
BRNN Input: [(..., n_feats * 4)]
Hidden : [(..., n_feats * 2)]
Dropout
Time Distributed
Fully Connected
Input: [(..., n_feats * 4)]
Output: [(..., n_feats * 2)]
Dropout
Fully Connected
Input: [(..., n_feats * 2)]
Output: [(..., n_feats)]
Dropout
Fully Connected
Input: [(..., n_feats)]
Output: [(..., n_class)]
(a)
Input Feature Vector Right Hand
[(batch_size, n_frames, n_feats)]
Input Feature Vector Left Hand
[(batch_size, time_steps,n_feats)]
Embedding Layer
Input: [(...,n_feats)]
Output: [(..., n_feats * 2)]
Embedding Layer
Input: [(...,n_feats)]
Output: [(..., n_feats * 2)]
Recurrent Layers
Layer Normalization
Embedding Input: [(....,n_feats*4)]
Output: [(..., n_feats * 4)]
BRNN Input: [(....,n_feats*8)]
Output: [(..., n_feats * 8)]
BRNN
Embedding Input: [(..., n_feats * 4)]
BRNN Input: [(..., n_feats * 8)]
Hidden: [(..., n_feats * 4)]
Dropout
Time Distributed
Fully Connected
Input: [(..., n_feats * 8)]
Output: [(..., n_feats * 4)]
Dropout
Fully Connected
Input: [(..., n_feats * 4)]
Output: [(..., n_feats * 2)]
Dropout
Fully Connected
Input: [(..., n_feats * 2)]
Output: [(..., n_class)]
(b)

Figure 6.6: The pipelines of the neural network architectures and their building blocks.

Another important part of Figure 6.6 is the dimensions of the building blocks:

- The input feature vector is a 3-dimensional vector of dimensions [*batch_size*, *n_frames*, *n_feats*]. The *batch_size* determines the number of sequences inside a mini-batch, while *n_frames* is the number of frames in a sequence. Finally, *n_feats* expresses the size of the feature vector, which is either 63 or 42 depending on the input data.
- Since we decided to generate batches for our sequences, we must ensure that each sequence inside the batch has the same number of *n_frames*. We accomplish that by padding the sequences inside the batch based on the length of the longest sequence. The padding is done by appending feature vectors filled with zeros to the sequence.
- Each block includes the dimensions of its input vector and the dimensions of the vector it outputs. We can observe that these dimensions are affected by *n_feats*. By using *n_feats* to determine the output of each layer, we force the whole architecture to adjust to the input.
- In the recurrent layers block there exist two categories of input and output vectors. This happens because we wanted to cover the case of stacked recurrent layers. In that case, the first recurrent layer accepts its input from the embedding layer, whereas the rest from the recurrent layers before them.
- The output probability distribution at each time step is a vector of size *n_class*, which in our case is 28. We assigned the alphabetical characters to integers, i.e. space is 0, characters between [a,z] correspond to integers 1 to 26 and the blank character is 27.

Finally, we use the CTC loss function in all our tests. The CTC blank character is expected to deal with areas of silence and padded parts in the input sequence. We should note here that the target character sequence should not exceed the input sequence in length, otherwise CTC might behave unpredictably. Furthermore, we provide CTC with the lengths of the input and target sequences apart from the sequences themselves. Since we created and trained our models with Pytorch, we also used the default CTC loss function offered by the Pytorch neural network solutions.

Chapter 7

Fingerspelling Recognition Results

We now proceed with our fingerspelling recognition experiments. We first describe the accuracy metric we use, followed by a categorization of the different types of experiments we conducted and the setup of the neural network training and testing phase. We then present the results of our experiments, including information on the accuracy, the architectural variations, normalization schemes etc.

7.1 Accuracy Metric

In order to determine the performance of our models in SLR we needed an accuracy metric suited to our problem. In our case this is the Levenshtein distance that reveals how different two character sequences are. More specifically, the Levenshtein distance is the number of edits required to transform one word into another. The possible edits are to insert, delete or substitute a character and each of these operations increases the number of edits by one.

Having the Levenshtein distance at our disposal allows us to estimate the character error rate (CER). The CER can be calculated for a dataset of N predictions and sequences as:

$$CER = \frac{1}{N} \cdot \sum_{n=0}^{N-1} \frac{\text{edit\_distance}(n)}{\text{sequence\_length(n)}}$$

The CER should gradually drop during the training phase of the ANN models, as more characters are predicted correctly and the edit distance therefore decreases.

7.2 Experimental Overview and Setup

We organize the experiments and their results based on the input data we used. The input data also determine the architecture we choose for each case. Based on the type of feature vectors we can distinguish the following types of experiments with:

1. Feature vectors containing the 3D and 2D coordinates of the signing hand produced by MediaPipe.

2. Feature vectors containing the 3D and 2D coordinates of both right and left hand produced by MediaPipe.

3. Feature vectors containing the 2D coordinates of the signing hand or the 2D coordinates of both right and left hand. This time the coordinates result from the synergy of both MediaPipe and OpenPose.

We also tested different normalization schemes, presented in Chapter 5, on our input data. Furthermore, all of our experiments had a batch size of 5 and a network learning rate of 1e-4. All of our models use the Adam optimizer during the training phase and more specifically the AdamW variation [26]. Apart from the optimizer we also set a cycling learning rate scheduler that accelerates the training of the network. This effect is called super-convergence [27]. To apply the scheduling technique we need to set an initial maximum learning rate, in our case 1e-4. Over the course of a training epoch, the learning rate will be inversely scaled from a very low value to its maximum value and then back to a learning rate much lower than the initial during the end of the epoch. We choose the one-cycle variation, where the cyclic process is applied over one epoch and the learning rate changes after every batch. During the training phase we used our greedy decoder to observe the CER behaviour. We tested both beam search and greedy decoders during the testing phase and acquired their respective CERs. Our beam search algorithm used a beam width of 3. The models were trained on 5451 video sequences and tested on 867 sequences.

7.3 Experiment 1: Signing Hand MediaPipe Landmarks

In this section we present the results from the experiments where we used the 3D (Tables 7.1 and 7.2) and 2D coordinates (Tables 7.3 and 7.4) of the signing hand as input to our model. The model uses the architecture of Figure 6.6a.

Table 7.1: Experiment 1 – Results for the 3D coordinates of the signing hand

BRNN	FCs	Normalization	Epochs	Train Loss	Test Loss	CER Greedy	CER Beam
2 BGRU	3	Wrist	130	0.9562	2.0411	42.70	41.38
2 BLSTM	2	Wrist	130	0.6516	2.3505	43.85	45.07
2 BGRU	3	Wrist & Min-Max	130	1.1742	1.9067	43.82	42.91
2 BLSTM	3	Wrist & Min-Max	130	1.0768	2.0915	45.56	46.42
2 BGRU	3	Wrist & Hand size	130	0.9332	2.0648	42.44	42.14
2 BLSTM	2	Wrist & Hand size	130	0.6450	2.3051	44.03	44.38
2 BGRU	3	None	130	1.2490	1.8750	45.01	44.58
2 BLSTM	2	None	130	0.9706	1.9737	45.66	45.07

Table 7.2: Experiment 1 – Results for the 3D coordinates of the signing hand without layer normalization

BRNN	FCs	Normalization	Epochs	Train Loss	Test Loss	CER Greedy	CER Beam
2 BGRU	2	Wrist	130	1.1005	2.1537	51.42	55.46
2 BLSTM	3	Wrist	130	1.2484	2.1847	53.38	52.98

Table 7.3: Experiment 1 – Results for the 2D coordinates of the signing hand

BRNN	FCs	Normalization	Epochs	Train Loss	Test Loss	CER Greedy	CER Beam
2 BGRU	2	Wrist	130	1.2305	1.8165	43.81	43.26
2 BLSTM	2	Wrist	130	1.1593	1.8949	45.35	44.22
2 BGRU	2	Wrist & Min-Max	130	1.3714	1.7975	46.25	44.31
2 BLSTM	2	Wrist & Min-Max	130	1.3248	1.8819	46.55	45.91
2 BGRU	2	None	130	1.5029	1.7827	48.24	45.42
2 BLSTM	2	None	130	1.5249	1.8238	49.99	47.46

Table 7.4: Experiment 1 – Results for the 2D coordinates of the signing hand without layer normalization

BRNN	FCs	Normalization	Epochs	Train Loss	Test Loss	CER Greedy	CER Beam
2 BGRU	2	Wrist	130	1.5067	2.0689	61.50	57.42
2 BLSTM	2	Wrist	130	1.5271	2.1801	64.24	57.92

We observe that by using 3D coordinates the models were able to produce better results. Discarding the depth coordinate did not inflict dramatic changes to the accuracy scores though. Regarding the choice of the neural network layers, the models with the bidirectional GRU layer performed better than their LSTM counterparts. Tables 7.2 and 7.4 reveal how much the normalization layer improved accuracy. When it comes to normalization schemes on the input data, the wrist normalization and the wrist with hand size normalization schemes produced the best scores. Finally, the beam search decoder appears to have an advantage over the greedy decoder.

7.4 Experiment 2: MediaPipe Landmarks of Both Hands

In this section we present the results from the experiments where we used the 3D (Table 7.5) and 2D (Table 7.6) coordinates of both right and left hand as input. The model uses the architecture of Figure 6.6b.

Table 7.5: Experiment 2 – Results for the 3D coordinates of both hands

BRNN	FCs	Normalization	Epochs	Train Loss	Test Loss	CER Greedy	CER Beam
2 BGRU	3	Wrist	50	0.8524	2.0406	45.65	44.67
2 BLSTM	3	Wrist	50	0.7329	2.3499	46.44	47.38
2 BGRU	3	Wrist & Min-Max	70	0.7901	2.1421	46.62	46.55
2 BLSTM	2	Wrist & Min-Max	70	0.4972	2.6152	48.78	53.62
2 BGRU	3	Wrist & Hand size	50	0.8928	2.0672	47.58	46.50
2 BLSTM	3	Wrist & Hand size	50	0.7599	2.4326	48.57	47.24
2 BGRU	3	None	50	1.4221	1.9658	51.42	50.84
2 BLSTM	3	None	50	1.6645	2.1473	60.66	57.14

Table 7.6: Experiment 2 – Results for the 2D coordinates of both hands

BRNN	FCs	Normalization	Epochs	Train Loss	Test Loss	CER Greedy	CER Beam
2 BGRU	2	Wrist	50	0.7145	2.2711	46.01	46.25
2 BLSTM	2	Wrist	50	1.0788	1.9365	47.72	46.69
2 BGRU	2	Wrist & Min-Max	70	1.0634	1.9597	49.32	49.02
2 BLSTM	2	Wrist & Min-Max	70	1.0127	2.0904	49.04	47.31
2 BGRU	2	None	50	1.7165	1.9713	56.93	53.38
2 BLSTM	2	None	50	1.9058	2.0897	65.70	59.25

When it comes to coordinate dimensions, neural network layers, normalization schemes and decoding, we observe the same trend as with the signing hand experiments. In comparison to Experiment 1, the CER degrades. This could imply that idle or arbitrarily moving hands inside the images insert noise to the models.

7.5 Experiment 3: MediaPipe and OpenPose Synergy Landmarks

In this section we present the results from the experiments where we used the 2D coordinates of the signing hand (Table 7.7) and both right and left hand (Table 7.8) as input. The coordinates were produced by the collaboration of MediaPipe and OpenPose and our models again use the architecture of Figure 6.6a for the signing hand and of Figure 6.6b for the two hand case.

Table 7.7: Experiment 3 – Results for the 2D coordinates of the signing hand

BRNN	FCs	Normalization	Epochs	Train Loss	Test Loss	CER Greedy	CER Beam
2 BGRU	2	Wrist	130	1.2087	1.6738	41.44	40.51
2 BLSTM	2	Wrist	130	1.1556	1.7725	42.45	41.28
2 BGRU	2	Wrist & Min-Max	130	1.3623	1.7005	42.58	41.22
2 BLSTM	2	Wrist & Min-Max	130	1.3152	1.7734	44.59	43.33

Table 7.8: Experiment 3 – Results for the 2D coordinates of both hands

BRNN	FCs	Normalization	Epochs	Train Loss	Test Loss	CER Greedy	CER Beam
2 BGRU	3	Wrist	70	0.8453	1.8893	44.08	43.07
2 BLSTM	3	Wrist	70	1.0155	2.0451	45.95	46.09
2 BGRU	3	Wrist & Min-Max	70	1.2810	1.8493	46.42	45.22
2 BLSTM	3	Wrist & Min-Max	70	1.2849	2.0797	50.37	50.44

Once more, neural network layers, normalization schemes and decoding algorithms have the same effect on the error rates. Unfortunately, we cannot make a comparison between 2D and 3D coordinates for this type of experiment. We can compare the current results with the respective results from MediaPipe experiments though and observe that 2D coordinates are able to produce better results this time.

7.6 Summary of Best Results

In Table 7.9 and Figure 7.1 we provide a summary of the best results of each experiment. As we have already shown, CER is affected by a variety of different factors. The origin and type of input feature vectors are the most decisive. In MediaPipe experiments we observed that models using 3D coordinates could produce better scores than those using 2D coordinates. It is interesting though that 2D coordinates from the MediaPipe and OpenPose synergy resulted in the best scores overall, with a CER of 40.51%. This could be attributed to the more accurate hand sequences produced by this approach. Furthermore, we observe that our most successful experiments were those with the signing hand.

Table 7.9 also demonstrates how GRUs surpassed LSTMs overall. The number of fully connected layers in the time distributed block was not a very decisive factor, as it varied between 2 or 3 stacked layers. When it comes to normalization schemes, plain wrist normalization was the most effective. Finally, beam search decoding helped to slightly improve CERs in most of Table 7.9's experiments.

Table 7.9: Cumulative table of the most successful experiments.

Index	Library	Input	BRNN	FCs	Normalization	CER Greedy	CER Beam
1	MediaPipe	2D-2 Hands	2 BGRU	2	Wrist	46.01	46.25
2	MediaPipe	3D-2 Hands	2 BGRU	3	Wrist	45.65	44.67
3	MediaPipe & OpenPose	2D-2 Hands	2 BGRU	3	Wrist	44.08	43.07
4	MediaPipe	2D-Signing	2 BGRU	2	Wrist	43.81	43.26
5	MediaPipe	3D-Signing	2 BGRU	3	Wrist	42.70	41.38
6	MediaPipe & OpenPose	2D-Signing	2 BGRU	2	Wrist	41.44	40.51

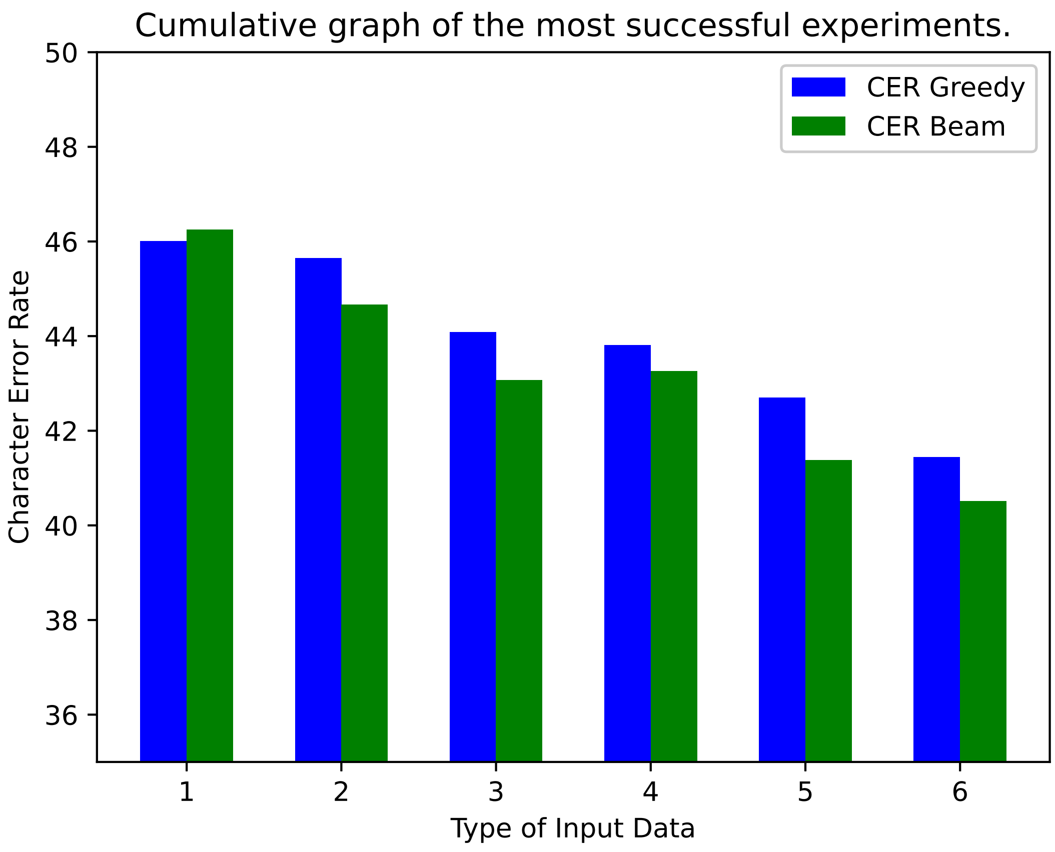


Figure 7.1: The CERs of each experiment from Table 7.9

Chapter 8

Conclusions

In this Thesis we tackled the problem of fingerspelling recognition from ASL videos. Since fingerspelling signers use their hands, we focused on how to extract the hand features from the video frames. For that purpose we resorted to object tracking tools, more specifically MediaPipe and OpenPose. Despite MediaPipe not being very widely used in SLR tasks, it managed to surpass OpenPose by invoking superior hand detection.

The ML models in this Thesis relied heavily on the postprocessing of the input features in order to produce better error rates. Regarding the models based on MediaPipe landmarks, the depth coordinate improved their accuracy. Models using the signing hand as input produced lower error rates than those using both hands. This might be due to the hand classification algorithm discarding the possibly useless information of the idle hands. Despite that, the error rates with both hands as input were not much higher. This possibly indicates our models' ability to learn and distinguish the signing hand on their own.

We performed the experiments with the signing hand and both hands again, but this time the visual features were generated by the collaboration of both MediaPipe and OpenPose. We only used 2D coordinates, and again models using the signing hand performed better than those using both hands. Despite not having the depth coordinate we managed to develop our best model yet with a CER of 40.51%. We attribute this score to the improved hand detection produced by the collaboration between the two libraries.

Regarding the normalization schemes, wrist normalization was the most effective. Applying hand size normalization along with wrist normalization is also promising for further research. Min-Max scaling generally increased CERs by a small percentage, thus indicating that it is not well suited for our problem. In the experiments where no normalization scheme was applied the CERs increased.

Finally, when it came to the ANN components, the only ones that significantly affected our accuracy were the BRNN and normalization layers. It was made clear that GRUs always performed better than LSTMs, with the latter contributing to increased error rates in many occasions. The absence of the normalization layer dramatically increased CERs. We also believe that beam search is worth testing in problems using CTC, as it improved CER in many of our experiments. Unfortunately, we did not have time to develop other variations of beam search and test different beam widths. Generally, we are satisfied with the models we developed, although we would have liked to achieve even lower CERs.

Bibliography

[1] MediaPipe Holistic. https://google.github.io/mediapipe/solutions/holistic.html. Accessed: 14-03-2022.

[2] Understanding OpenPose (with code reference)-part 1. https://medium.com/analytics-vidhya/understanding-openpose-with-code-reference-part-1-b515ba0bbc73. Accessed: 14-03-2022.

[3] Understanding LSTM networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 17-04-2022.

[4] Neural networks, types, and functional programming. https://colah.github.io/posts/2015-09-NN-Types-FP/. Accessed: 17-04-2022.

[5] LSTM networks. https://apmonitor.com/do/index.php/Main/LSTMNetwork. Accessed: 27-05-2022.

[6] Sequence modeling with CTC. https://distill.pub/2017/ctc/. Accessed: 18-03-2022.

[7] T. Starner and A. Pentland. Real-time American sign language recognition from video using hidden Markov models. In *Proceedings of International Symposium on Computer Vision - ISCV*, pages 265–270, 1995.

[8] T. Kim, K. Livescu, and G. Shakhnarovich. American sign language fingerspelling recognition with phonological feature-based tandem models. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 119–124, 2012.

[9] T. Kim, G. Shakhnarovich, and K. Livescu. Fingerspelling recognition with semi-markov conditional random fields. In *2013 IEEE International Conference on Computer Vision*, pages 1521–1528, 2013.

[10] T. Kim, J. Keane, W. Wang, H. Tang, J. Riggle, G. Shakhnarovich, D. Brentari, and K. Livescu. Signer independent fingerspelling recognition with deep neural network adaptation. *CoRR*, abs/1602.04278, 2016.

[11] B. Shi, A. Martinez Del Rio, J. Keane, J. Michaux, D. Brentari, G. Shakhnarovich, and K. Livescu. American Sign Language fingerspelling recognition in the wild. *CoRR*, abs/1810.11438, 2019.

[12] B. Shi, A. Rio, J. Keane, D. Brentari, G. Shakhnarovich, and K. Livescu. Fingerspelling recognition in the wild with iterative visual attention. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5399–5408, 2019.

[13] B. Shi and K. Livescu. Multitask training with unlabeled data for end-to-end sign language fingerspelling recognition. *CoRR*, abs/1710.03255, 2019.

[14] B. Shi, D. Brentari, G. Shakhnarovich, and K. Livescu. Fingerspelling detection in American sign language. *CoRR*, abs/2104.01291, 2021.

[15] N. Camgoz, S. Hadfield, O. Koller, H. Ney, and R. Bowden. Neural sign language translation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7784–7793, 2018.

[16] K. Papadimitriou, M. Parelli, G. Sapountzaki, G. Pavlakos, P. Maragos, and G. Potamianos. Multimodal fusion and sequence learning for cued speech recognition from videos. In *Universal Access in Human-Computer Interaction. Access to Media, Learning and Assistive Environments (HCII)*, pages 277–290. Springer, 2021.

[17] N.N. Tu, S. Sako, and B. Kwolek. Fingerspelling recognition using synthetic images and deep transfer learning. In *Thirteenth International Conference on Machine Vision*, pages 528–535, 2020.

[18] B. Kwolek, W. Baczynski, and S. Sako. Recognition of JSL fingerspelling using deep convolutional neural networks. *Neurocomputing*, 456, 2021.

[19] OpenPose. `https://github.com/CMU-Perceptual-Computing-Lab/openpose`. Accessed: 16-03-2022.

[20] Chicago Fingerspelling in the Wild Data Sets (ChicagoFSWild, ChicagoFSWild+). `https://home.ttic.edu/~klivescu/ChicagoFSWild.htm#download`. Accessed: 27-03-2022.

[21] MediaPipe. `https://google.github.io/mediapipe/`. Accessed: 14-04-2022.

[22] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[23] K. Cho, J. Chung, C. Gulcehre, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, 1412.3555, 2014.

[24] G. Hinton, J. Lei Ba, and J.R. Kiros. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[25] CTC Decoder. `https://github.com/githubharald/CTCDecoder`. Accessed: 03-05-2022.

[26] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *CoRR*, abs/1711.05101, 2019.

[27] L. N. Smith and N. Topin. Super-convergence: Very fast training of neural networks using large learning rates. *CoRR*, abs/1708.07120, 2017.