UNIVERSITY OF CAMBRIDGE

Ronan Ragavoodoo
rdr32

Sign Language Recognition

Computer Science Tripos – Part II

Homerton College

April 16, 2023

Declaration

I, Ronan Ragavoodoo of Homerton College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

I, Ronan Ragavoodoo of Homerton College, am content for my dissertation to be made available to the students and staff of the University.

Ronan Ragavoodoo	April 16, 2023
Signed	Date

Acknowledgements

Writing this dissertation required not only hard work and dedication but also help and encouragement from many individuals. First of all, I would like to express my gratitude to Mr. Filip Svoboda for supervising the project and pointing me to all the right resources. His encouragement and academic excellence have helped greatly whilst writing my dissertation. I am also grateful to Dr. John Fawcett for providing feedback on drafts of this dissertation and for helping me grow both academically and personally throughout my years at University. Lastly, I would like to thank my friends and family for reading drafts of this dissertation and for supporting my studies in Cambridge.

Proforma

Name:	**Ronan Ragavoodoo**
College:	**Homerton College**
Project Title:	**Sign Language Recognition**
Examination:	**Computer Science Tripos – Part II, May 2023**
Word Count:	**10205**[1]
Code Line Count:	**999**[2]
Project Originator:	The Author
Supervisor:	Mr. Filip Svoboda

Original Aims of the Project

The original aim of this project was to create a model that can help recognize British Sign Language (BSL), specifically isolated signs. This model would be able to accurately interpret and translate BSL gestures and movements into written or spoken language, making communication with individuals who use BSL more seamless and accessible. The goal was to develop a program that can assist individuals who are deaf or hard of hearing, as well as improve overall accessibility in society. The project would be focused on utilizing machine learning and computer vision techniques to train the model to recognize BSL gestures with high accuracy.

Work Completed

I have successfully implemented a model that can recognise with high-accuracy a variety of isolated BSL expressions in real-time. Different models have been evaluated to compare performances and determine which is more adequate for real-time recognition.

Special Difficulties

None

[2]This word count was computed using `texcount -inc rdr32.tex`

[2]This word count was computed using `loc`

Contents

Chapter 1

Introduction

1.1 Motivation

The motivation behind this project is to improve communication accessibility for individuals who use British Sign Language (BSL). BSL is a visual language that is used by around 150,000 people in the UK, many of whom are deaf or hard of hearing. Despite its prevalence, BSL is not widely understood or recognized by the general population, making communication with individuals who use BSL difficult and often frustrating. Additionally, the project aims to raise awareness and understanding of BSL, and to demonstrate the power of technology to promote accessibility and inclusivity.

1.2 Related Work

Sign language recognition can be grouped into two distinct parts: isolated sign language and continuous sign language. While isolated sign language involves distinct gestures being performed one at a time, continuous signing involves many gestures performed in quick succession, often merging some signs together.

Work has been conducted on the BOBSL[1] dataset, which is the largest continuous BSL dataset. This work aimed at improving real-time translation of BSL, which is usually performed with continuous signing.

Chapter 2

Preparation

This chapter gives a background of current research on various sign-language recognition based models, as well as existing datasets, libraries, and techniques. This forms part of the research conducted prior to writing the implementation code for the project, but will also include further research that has been done in response to challenges encountered during the implementation.

2.1 Requirements Analysis

The project requires the following to be implemented in order to have a working program:

- **Gesture Detection and Tracking**: This is what will determine the shape of the training data and its nature. It consists of identifying and tracking the different body parts present in the given frame, especially hands.
- **Gesture Recognition**: There also needs to be a framework which can accurately predict what gesture has been displayed, from a set of different gestures.
- **Real-time Detection**: The program should allow the gestures to be detected in real-time, giving immediate results.

An appropriate dataset is used to perform any form of training or comparison on. Specifically, we target an isolated BSL dataset. However, the nature of sign language in general means that we can expand on different variations of sign language in order to infer how well the model will perform.

2.2 Datasets

2.3 Model Selection

2.3.1 Neural Networks

Studies have been conducted on many different neural network architectures, focused on mainly on image classification. The best performing networks include 3D Convolutional

Neural Networks (3D CNNs), such as I3D, and Recurrent Neural Networks (RNNs), such as Long Short-Term Memory (LSTM) networks. I3D has shown remarkable performance in sign language recognition tasks due to its ability to capture spatiotemporal information from videos. It has been used in various sign language recognition projects, such as recognizing American Sign Language (ASL) gestures, notably on the MS-ASL dataset [2]. RNNs, particularly LSTMs, are also commonly used for sign language recognition due to their ability to handle sequential data. These networks have been used in projects aimed at recognizing both isolated gestures [3] and continuous sign language sentences [4].

2.3.2 Dynamic Time Warping

Dynamic Time Warping (DTW) is a well-known algorithm for comparing sequences that may have different lengths and may be slightly misaligned. It was first introduced by Sakoe and Chiba in 1978 [5] and has since been widely used in many areas such as speech recognition, pattern recognition, bioinformatics, and finance.

The basic idea of DTW is to find the optimal alignment between two sequences by warping one of the sequences in the time dimension. The algorithm constructs a distance matrix between the two sequences, where each element represents the distance between a point in the first sequence and a point in the second sequence. The optimal path through this matrix is then found using dynamic programming, which minimizes the total distance between the two sequences. The algorithm runs in $\mathrm{O}(m \times n)$.

Let $X = x_1, x_2, ..., x_n$ and $Y = y_1, y_2, ..., y_m$ be two sequences of length n and m, respectively. The algorithm aims to find the optimal path $\pi = (i_1, j_1), (i_2, j_2), ..., (i_k, j_k)$ that warps one of the sequences (usually the shorter one) to align with the other sequence. The optimal path π is the one that minimizes the total distance between the two sequences, which is defined as:

$$DTW(X, Y) = \min_{\pi} \sqrt{\sum_{(i,j)\in\pi} (x_i - y_j)^2} \tag{2.1}$$

where (i, j) represents a point in the distance matrix.

To find the optimal path π, the DTW algorithm constructs a distance matrix D of size $n \times m$, where each element $D_{i,j}$ represents the distance between x_i and y_j. The matrix is initialized with large values such that $D_{i,j} = \infty$ for all i and j. The first element is set to $D_{1,1} = (x_1 - y_1)^2$. Then, for each element $D_{i,j}$, the algorithm finds the minimum distance among the three neighboring elements:

$$D_{i,j} = (x_i - y_j)^2 + \min(D_{i-1,j}, D_{i,j-1}, D_{i-1,j-1}) \tag{2.2}$$

The path through the distance matrix that corresponds to the optimal path π is then found by backtracking from the bottom-right corner of the matrix to the top-left corner, following the minimum distance at each step. The resulting path is the optimal path π, which can be used to align the two sequences.

Algorithm 1 is a simple implementation of the DTW algorithm in pseudocode.

Algorithm 1: Simple DTW algorithm

Input : X: array $[x_1, x_2, \ldots, x_m]$, Y: array $[y_1, y_2, \ldots, y_n]$
Output: Distance: float
dp = matrix of size $m + 1 \times n + 1$
for $i \leftarrow 1$ **to** m **do**
  for $j \leftarrow 1$ **to** n **do**
    $dp[i, j] = \text{inf}$
  end for
end for
$dp[0, 0] = 0$
for $i \leftarrow 1$ **to** m **do**
  for $j \leftarrow 1$ **to** n **do**
    $cost = abs(X[i-1] - Y[J])$
    $dp[i, j] = cost + min(dp[i-1, j], dp[i, j-1], dp[i-1, j-1])$
  end for
end for
return $dp[m, n]$

DTW has been shown to be an effective algorithm for comparing sequences in many applications, but it has some limitations. One of the main limitations is its computational complexity, which is quadratic in the length of the sequences. This can be a problem for very long sequences or for applications that require real-time processing. In recent years, there have been many extensions and variations of DTW that aim to overcome some of these limitations [6].

2.3.3 Feature Extraction

Feature extraction is a crucial step in sign language recognition systems as it enables the conversion of raw video data into meaningful and informative representations that can be used for classification purposes. In this section, we discuss the feature extraction process used in our sign language recognition project, which involves the extraction of hand and body pose landmarks and the calculation of hand angles and relative distance features.

The project employed Mediapipe Holistic, an open-source library developed by Google, to extract landmarks from the hand and body poses in the sign language videos. This library provides a robust and accurate solution for the detection of body and hand keypoints, including wrist, elbow, shoulder, and finger joints. Specifically, we used the pretrained Holistic model, which uses a deep neural network to detect the keypoints and estimate their positions in real-time.

From the extracted landmarks, angles between each of the 21 hand landmarks were calculated for each hand. This resulted in 441 angles, which captured the intricate hand movements and positions that are characteristic of sign language gestures. These angles were calculated using the law of cosines and were normalized to ensure consistency across different signers and videos.

In addition to hand angles, the relative distance between the wrist and shoulder landmarks was also calculated for each hand. This feature provides information about the arm extension, which is a critical aspect of sign language gesture recognition. The distance feature was calculated as the Euclidean distance between the wrist and shoulder landmarks and was also normalized to ensure consistency.

These methods were used instead of relying on the raw landmark data as distance and positioning greatly affect the accuracy of the results. Angles and relative distances are less sensitive to these factors, thus allowing the model to be accurate even if users are positioned differently or are further away.

2.4 Tools and Libraries

Python

Python is a popular high-level programming language that is widely used in scientific computing, data analysis, and machine learning applications. Python was chosen as the main language for this project for the following reasons:

Python has a large and active community of developers, which has contributed to a vast ecosystem of libraries and frameworks that can be used for machine learning and computer vision applications. Some of the popular libraries for our project include NumPy, Pandas, Matplotlib, OpenCV, and Scikit-learn. These libraries provide a range of functionalities, such as data manipulation, visualization, and model development, which can significantly simplify and speed up the development of our project.

Moreover, Python is known for its simplicity and ease of use, making it an ideal language for rapid prototyping and experimentation. The syntax of Python is straightforward and easy to learn, making it an excellent choice for beginners who are new to programming. Additionally, Python's rich standard library provides a wide range of built-in functions and modules that can be used to perform various tasks, such as file I/O, networking, and regular expressions.

2.4.1 Visual Studio Code

Visual Studio Code (VSCode) is a popular and versatile code editor that is widely used for software development, including for Python-based projects. I chose this editor for the following reasons:

VSCode has excellent support for Python programming. It comes with a wide range of built-in features and extensions that facilitate Python development, including code highlighting, linting, debugging, and auto-completion. Additionally, VSCode has a rich and active extension marketplace, where users can access and install a wide range of third-party extensions that enhance its capabilities for Python development.

VSCode provides an intuitive and user-friendly interface that makes it easy to navigate and manage the project code. The built-in file explorer allows for quick navigation through project files and directories, while the integrated terminal provides a convenient way to

execute commands and scripts directly within the editor, without needing an external terminal.

Another advantage of VSCode is its support for Git, a version control system that is commonly used in software development projects. With VSCode, we can easily integrate Git into our project workflow, allowing us to manage and track changes to the codebase, collaborate with other developers, and maintain project versions, all done in a very user-friendly environment.

Furthermore, VSCode has an extensive ecosystem of plugins and extensions that can be used to enhance its functionality and customize the development environment to suit specific project requirements. For example, we can install extensions for machine learning libraries, such as TensorFlow, and language-related ones, like for Python.

2.4.2 OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning library that provides a wide range of functions for image and video processing, object detection, feature extraction, and more. OpenCV was chosen for the following reasons:

OpenCV supports various programming languages, including Python, C++, and Java, making it a versatile library that can be used in different environments and applications. The project is implemented in Python, and OpenCV provides an intuitive and easy-to-use Python interface that simplifies the development process and speeds up the implementation of the sign language recognition model.

It also has built-in support for hardware acceleration, which can be useful given that the project requires real-time performance as a requirement for the success criteria.

Finally, it is one of the most popular libraries for this purpose, being used by a lot of projects in the field of computer vision. It is well-documented and has a lot of support available on its website.

2.4.3 Mediapipe-Holistic

Mediapipe Holistic is an open-source library developed by Google that provides a suite of real-time, multi-person pose estimation and tracking solutions. It uses machine learning models to detect and track multiple key points on the human body, including facial landmarks, hand landmarks, and body pose, which can be used for a wide range of applications, including sign language recognition. In this section, we discuss why Mediapipe Holistic is suitable for this project.

Mediapipe Holistic provides a simple and easy-to-use API that abstracts away the complexity of training and deploying machine learning models for pose estimation and tracking. It comes with pre-trained models that can accurately detect and track facial landmarks, hand landmarks, and body pose, which significantly reduces the development time and cost of building a detection model from scratch.

The model is also designed for real-time performance, which is crucial for this project. It provides efficient algorithms that can process video streams in real-time and accurately

track multiple body parts simultaneously, which is essential for detecting and recognizing sign language gestures in real-world scenarios.

Additionally, Mediapipe Holistic is an open-source library, which means that it is free to use, distribute, and modify. This open-source nature provides flexibility and enables us to customize and extend the library to suit the specific requirements of the recognition project.

2.4.4 dtw-python

The `dtw-python` library is a Python implementation of the DTW algorithm, which provides a set of functions for computing the DTW distance between two time series data sequences. The library provides an efficient implementation of the DTW algorithm that can handle sequences of different lengths and allows for the customization of the distance function used to compute the DTW distance.

The algorithm used by dtw-python is similar to the basic DTW algorithm, but it differs in the way it computes the DTW distance. The basic DTW algorithm computes the DTW distance by computing the full distance matrix and finding the optimal path through it. In contrast, the dtw-python algorithm uses dynamic programming to compute the DTW distance by computing only the necessary elements of the distance matrix. This results in a more efficient implementation.

2.4.5 FastDTW

FastDTW is a variant of the DTW algorithm that trades accuracy for faster computation times. It works by reducing the number of elements that need to be computed in the distance matrix, resulting in a smaller memory footprint and faster computation times.

The FastDTW algorithm consists of two main steps: a coarse-to-fine grid search and a local search. A high-level overview of the algorithm is as follows:

1. **Coarse-to-fine grid search:** The first step is to compute the distance between the two sequences at a coarse resolution. This is done by dividing each sequence into non-overlapping windows of length w, and computing the distance between the centroids of the windows. The centroids are defined as the average of the elements within each window. The result of this step is a coarse distance matrix D_{coarse} of size $\lceil n/w \rceil \times \lceil m/w \rceil$, where n and m are the lengths of the two sequences.

2. **Local search:** The second step is to refine the distance matrix computed in the first step by performing a local search. The local search algorithm starts at the bottom-right corner of the coarse distance matrix D_{coarse}, and iteratively computes the minimum distance between the current element and its three neighbors: the element to its left, the element above it, and the element diagonally above it. This results in a refined distance matrix $D_{refined}$ of size $n \times m$.

3. **Backtracking:** Finally, the optimal warping path is found by backtracking from the bottom-right corner of the refined distance matrix $D_{refined}$ to the top-left corner, using the same rules as the standard DTW algorithm.

The FastDTW algorithm has a time complexity of $O(w \cdot n \cdot log_2(n))$, where w is the window size and n is the length of the longer sequence. This is faster than the standard DTW algorithm, which has a time complexity of $O(n^2)$, but it also results in slightly less accurate alignment of the sequences.

The `fastdtw` library in Python provides an implementation of the FastDTW algorithm, as well as other variants of the DTW algorithm. It is useful for this project given that there will be real-time detection involved, so a faster algorithm would speed up the predictions.

2.4.6 Git and Github

Git was used for version control as this is the system that I am most comfortable with using and familiar with. The repository is hosted on Github and changes were pushed regularly. A copy of this dissertation is also available on Github.

2.5 Software Engineering Techniques

1. **Code Reviews:** Code reviews involve a systematic examination of the source code by one or more team members. Code reviews can help identify bugs, improve code quality, and ensure adherence to coding standards and best practices.

2. **Unit Testing:** Unit testing involves testing individual units or modules of the software to ensure they function as intended. Unit testing can help catch bugs early in the development process and improve the overall quality of the software.

3. **Continuous Integration/Continuous Delivery (CI/CD):** CI/CD is a set of practices that automate the building, testing, and deployment of software. CI/CD pipelines can help catch bugs early, ensure the software is always in a working state, and make it easy to deploy updates to production environments.

4. **Documentation:** Documentation is important for ensuring that others can understand and use the software effectively. It can include documentation of the code, user manuals, and technical specifications.

These software engineering

Chapter 3

Implementation

In this chapter, I will describe how I implemented my project, describing challenges faced along the way. The implementation consists of the following:

- **Recording signs:** This includes the code which uses OpenCV to capture frames. They also integrate most of the functionality for determining when to process a given sequence of frames, when to reset the recording, and displaying visual and textual output to the user.
- **Pre-processing datasets:** They help process the datasets into a suitable format for the program to read them. LSA64 has one script which does the entire processing step, whereas BSLDict has an additional script to filter labels that have less than a desired amount of data.
- **Extracting features from the datasets:** They are used to get the relevant features that will be used to train our model. We use mediapipe holistic to first extract landmarks from video frames, then we convert these to angles and relative distances. This is discussed further in the relevant following section.
- **Comparing signs:** They focus on the computation using the DTW algorithm. They include the code which return the distances between any two compared signs and also the functionality to determine if any given sign is output with a certain confidence.

3.1 Data Pre-processing

The model needs the data to be in a a required format in order to process it. As such, we need to pre-process existing datasets in order to make them suitable.

We order the videos by having a directory for each label, containing all videos which fall under that corresponding label.

BSLDict

BSLDict is a large dataset of isolated BSL gestures [7]. The data has been obtained from BSL sign aggregation platform *signbsl.com*, with a total of 14k video clips for a vocabulary

Figure 3.1: Examples from the BSLDict dataset

of 9k labels.

The data provides two main issues:

- **Lack of examples per label:** A lot of the labels have only one or two videos. This is a problem as there is not enough data for testing or to use deep learning for efficient training.

- **Different signs per label:** Many labels contain ambiguous sign gestures. Some of the labels have different ways to represent them and some overlap with other labels as well. This leads to poor accuracy when trying to train on them.

The issue of lacking examples was one reason why DTW was used as an algorithm for this project. As deep learning models require a sufficient amount of training data to be sufficiently accurate, we use a model based on DTW instead as it can retain accuracy with less data.

LSA64

LSA64 is a sign database for isolated Argentian Sign Language (LSA) gestures [8]. It was created to produce a dictionary for LSA and to be used for training automatic sign recognition models. The dataset contains 3200 videos, with 10 signers executing 5 repetitions of 64 different types of signs.

This dataset was used mostly for initial testing purposes given the issues that were encountered with BSLDict. It provides both sufficient data and consistent signs per label.

Signers wore coloured gloves to aid hand recognition and segmentation, as well as wearing dark clothing. This does not affect this project due to the use of mediapipe to recognise signs.

Figure 3.2: Snapshot from the LSA64 dataset

3.2 Mediapipe Extraction

Using our datasets, we now need to extract information from them. This is done by processing each frame in a given video individually.

3.2.1 Mediapipe Holistic

Mediapipe Holistic is the model that outputs the needed data to start the feature extraction process.

Mediapipe Holistic utilizes a deep neural network model that is trained on a large dataset of labeled images and videos. The model is designed to recognize various human poses, hand gestures, and facial landmarks, and it achieves this by analyzing the pixels of the input image or video stream [9]. Mediapipe Holistic provides three tracking pipelines, namely for the face, hands, and pose. The latter two are of relevance to this project.

3.2.2 Hands Detection Pipeline

Detecting hands with articulated fingers is a complex task, due to factors such as varying hand spans relative to image frames. Unlike faces, hands lack high-contrast areas, increasing the complexity of detecting them. As such, usage of other features such as arms and the torso can help localise hand positions more accurately.

Mediapipe employs palm detection, as boundary-estimation is relatively easier compared to hands with articulated fingers. A Single-Shot Detector model called BlazePalm is used to detect initial hand locations. An encoder-decoder feature extractor is used for bigger scene context awareness, an approach similar to RetinaNet. Further explanation about these methods are described below.

Single-Shot Detection

Single Shot MultiBox Detector (SSD) is a state-of-the-art object detection algorithm that was proposed by Liu et al. in 2016 [10]. SSD is a single-shot detector, which means that

it can detect objects in a single forward pass through the network. This makes SSD much faster than two-stage detectors like R-CNN and Fast R-CNN, which require two forward passes through the network.

SSD works by predicting a set of bounding boxes and confidence scores for each object in an image. The bounding boxes are predicted using a set of anchor boxes, which are predefined boxes that are placed at different locations and scales in the image. The confidence scores are predicted for each anchor box, indicating how likely it is that the anchor box contains an object.

After the bounding boxes and confidence scores have been predicted, they are used to generate a set of final detections. This is done by first applying non-maximum suppression (NMS) to the bounding boxes. NMS is a technique that is used to remove duplicate detections. In NMS, the bounding boxes are sorted by their confidence scores, and then the boxes that have a high overlap with other boxes are removed.

After NMS has been applied, the remaining boxes are then classified using a softmax classifier. The softmax classifier outputs a probability distribution over the set of object classes. The class with the highest probability is then assigned to the box.

SSD has been shown to be very effective for object detection. It has achieved state-of-the-art results on a number of benchmark datasets, including the PASCAL VOC dataset and the COCO dataset. SSD is also very fast, making it suitable for real-time object detection applications.

Non-Maximum Suppresion

Non-maximum suppression (NMS) is a technique used in computer vision to eliminate duplicate or redundant detections of the same object in an image. It is commonly used in object detection tasks such as pedestrian detection, face detection, and vehicle detection. The algorithm works by selecting the detection with the highest confidence score and suppressing any overlapping detections that have a lower confidence score.

The NMS algorithm can be described mathematically as follows:

Given a set of bounding boxes, $B = b_1, b_2, ..., b_n$, and their corresponding confidence scores, $S = s_1, s_2, ..., s_n$, where n is the number of bounding boxes, the goal is to select a subset of bounding boxes, $B^* \subseteq B$, that have a high confidence score and do not overlap significantly with each other.

First, we sort the bounding boxes in descending order of their confidence scores. Let b_i be the bounding box with the highest confidence score, i.e., $s_i = \max_{j=1}^{n} s_j$. We add b_i to the subset B^* and remove it from the set B.

Next, we calculate the Intersection over Union (IoU) between b_i and each of the remaining bounding boxes $b_j \in B$, where $j \neq i$. The IoU is defined as:

$$\text{IoU}(b_i, b_j) = \frac{\text{area}(b_i \cap b_j)}{\text{area}(b_i \cup b_j)}$$

If $\text{IoU}(b_i, b_j) >$ threshold, where threshold is a predetermined threshold value (e.g., 0.5), we remove b_j from the set B. We repeat this process until all the bounding boxes in B have been processed.

The final output is the set B^* of selected bounding boxes.

The NMS algorithm helps to reduce the number of false positives and improves the accuracy of object detection models by removing redundant detections.

3.2.3 Pose Detection Pipeline

The process for pose detection, BlazePose, is similar to that used for hands [11]. The BlazePose detection pipeline consists of a lightweight pose detector followed by a pose estimation component.

Pose Detection

The first stage of the BlazePose pipeline is body part detection. Unlike the hands detection pipeline, BlazePose does not use the NMS algorithm as it breaks down in cases where poses are highly articulated. This is because multiple bounding boxes satisfy the IoU threshold. Instead, a face detector is used as a proxy for the person detector. This is augmented with additional person alignment parameters such as the middle point between the person's hip. The CNN used in the face detector is based on the BlazeFace architecture [12], which uses a modified MobileNetV1 model with depth-wise separable convolutions.

Pose Estimation

The pose estimation component predicts the location of the 33 landmarks, using the bounding boxes that were obtained from the first stage.

To achieve accurate pose estimation, a combined heatmap, offset, and regression approach is adopted. During training, the heatmap and offset loss are used to supervise the model, but the corresponding output layers are removed before running inference. This approach effectively uses the heatmap to supervise a lightweight embedding, which is then used by the regression encoder network. This approach is inspired by the Stacked Hourglass approach of Newell et al. [13], but in this case, a tiny encoder-decoder heatmap-based network is stacked with a subsequent regression encoder network.

To achieve a balance between high- and low-level features, skip-connections are actively utilized between all the stages of the network. However, the gradients from the regression encoder are not propagated back to the heatmap-trained features. This approach not only improves the heatmap predictions but also substantially increases the coordinate regression accuracy.

3.2.4 Mediapipe Pipeline

One problem with using multiple separate models is that each is optimised for their specific purposes [9]. For example, the pose estimation model requires a 256×256 input video frame. However, cropping the hand and face from this frame to feed their respective models results in an input which has a too low resolution to obtain accurate landmarks.

To solve this, Mediapipe Holistic uses a multistage pipeline. First, the human pose is estimated using the pose pipeline. Using the inferred pose landmarks, it then derives the regions of interest (ROI) for the face and hands, employing a recrop model along the

way to improve it. The ROIs are then passed into the respective models to obtain the remaining landmarks.

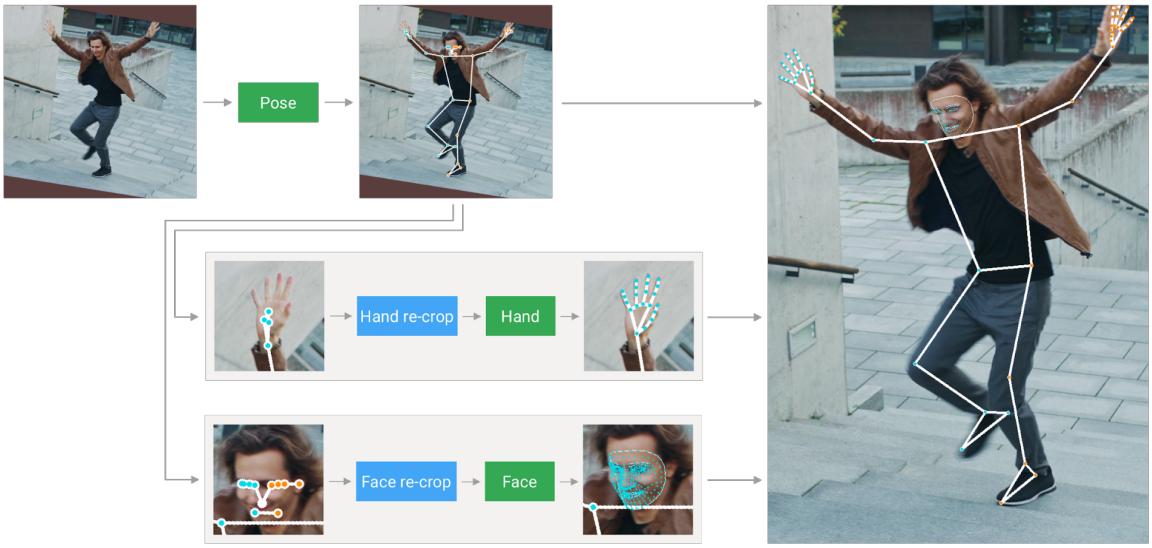

Figure 3.3: Pipeline Overview

3.2.5 Mediapipe Implementation

The goal here is to obtain the position of the different connections between joints. `mediapipe_utils` provides methods which help in locating the landmarks given a frame. The detection is relatively straightfoward to implement, using the provided Holistic model that is imported as part of the library. We can just use the model's `process` method. We also have to do color conversion because frames from OpenCV use BGR color rather than RGB. To add visual representation to the frames, the method `draw_landmarks` can be used to add drawing specifications to display the landmarks and the connections between them.

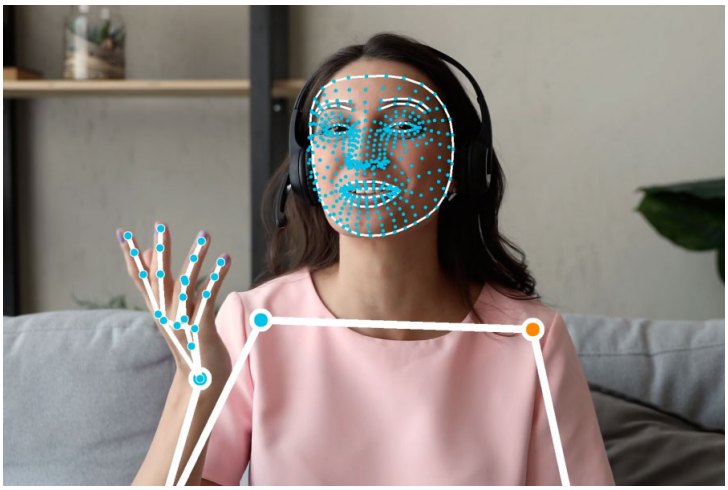

Figure 3.4: Visual display of landmarks of Mediapipe Holistic (Image from mediapipe.dev)

3.3 Feature Extraction

Once a list of connections from mediapipe has been obtained, we want to store this data in a suitable format and being able to reload them. The `feature_extraction` script provides a range of methods to do so. The initial training stage involves obtaining a collection of reference signs. These are what the recorded signs will be compared to to evaluate similarity.

3.3.1 Extract to Pickle Files

Initially, the `extract_features` method is used to obtain the landmarks from the data in our training folder. Every video is scanned and the relevant data extracted by mediapipe is stored in pickle files.

We use this opportunity here to also extract the data of the laterally-inverse of the frames. The reason for this is to obtain the features of the sign performed in a left-handed manner. In BSL, the leading hand used to perform the signing determines the handedness of the sign. Thus, flipping the image is sufficient to obtain the relevant data.

Each training video has two entries corresponding to it: its own extracted data and its flipped data. Each entry contains a pickle file for each hand and the pose, which is essentially an array of landmarks.

3.3.2 Loading Reference Signs

To perform predictions, the reference data needs to be loaded for the session. The `load_reference_signs` method is used to iterate through the entries containing the pickle files. For each entry, we initialise a new `sign_model` object, which is itself initialised with the respective `hand_model` and `pose_model` objects constructed from the data stored in the pickle files.

For efficiency, we use a `pandas.DataFrame` to store the name of the sign, its respective `sign_model`, and a distance initialised to 0. This will be overwritten when computing the distances.

3.4 Models

The project uses three models to represent features extracted from the vectors provided by Mediapipe Holistic.

Hand Model

The `hand_model` stores the feature vector representing all the angles between the connections provided by mediapipe on an arbitrary hand. Mediapipe provides 21 connections per hand, resulting in a feature vector of $21 * 21 = 441$ angles. We use the dot product formula to obtain the angle given two vectors (See Figure 3.2):

$$\theta = cos^{-1}\left(\frac{a \cdot b}{|a||b|}\right), \text{ where } a \text{ and } b \text{ represent connections}$$

The model only represents one particular frame, given a sequence of them. The final sign gesture is thus represented by a sequence of hand models.

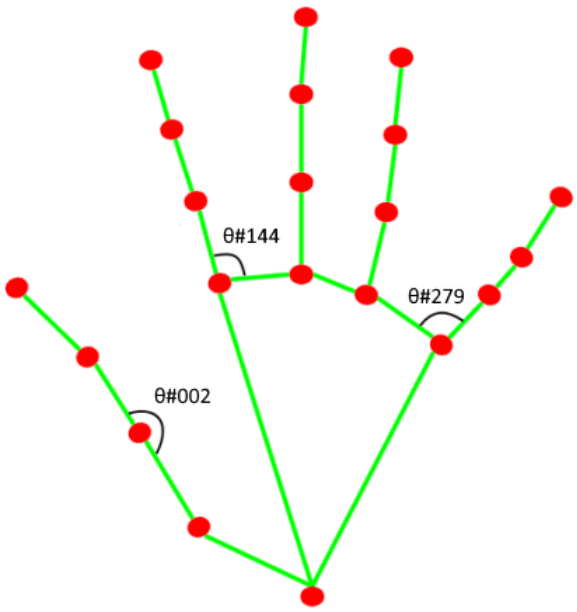

Figure 3.5: The angles on the mediapipe connection drawing

Pose Model

The `pose_model` takes a different approach. Instead of using angles, we use the distance between the shoulder and elbow as a normalisation distance, normalising coordinates provided by mediapipe. This is because the pose landmarks also include those of the

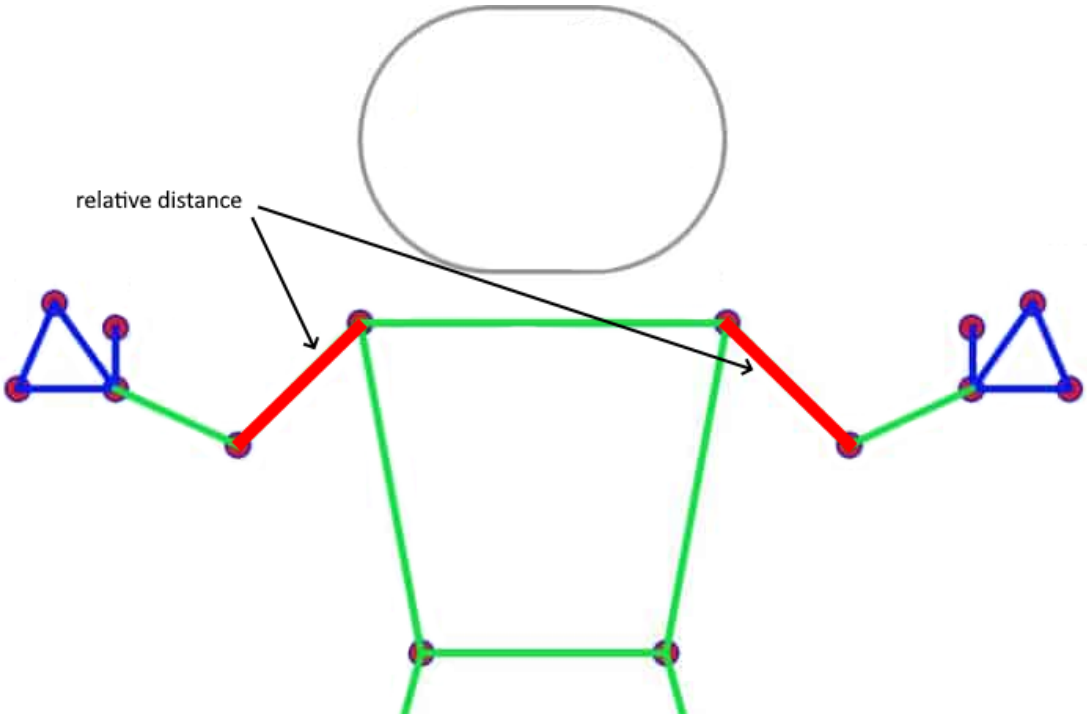

Figure 3.6: The relative distance used to normalise the landmarks

lower body, including the legs. Hence, we only use the landmarks of the shoulder, elbows and wrist, as they are more likely to be present in the frames. Taking a and b as the position vectors representing the shoulder and elbow respectively, we obtain the new coordinates as follows:

$$v_i = \frac{u_i}{|b - a|}, \text{ for every coordinate } u \text{ in the pose landmarks}$$

Sign Model

The `sign_model` object stores the hand and pose models for each frame in the sequence representing the sign, essentially being the representation of the sign as its features. This is what is compared when computing the similarity between a reference sign and a recorded sign.

Each object contains the following attributes:

- `has_left_hand` and `has_right_hand`, which are booleans which are true if such hand is present in the video.
- `lh_embedding` and `rh_embedding`, which are the lists of feature vectors for each frame for each hand.
- `left_arm_embedding` and `right_arm_embedding`, which are the lists of feature vectors for each frame for each arm.

3.5 Computing Similarity

The project provides two scripts for computing similarity: `compute_dtw` and `compute_fastdtw`, which use the `dtw-python` and `fastdtw` libraries respectively. The project provides an easy way to toggle between the two algorithms.

3.5.1 dtw-python

The `dtw_distances` method is what fills the `pandas.DataFrame` with distances. It takes the DataFrame as an argument along with the sign model representing the recorded sign. The embeddings of the sign model are extracted to variables as they are the sequences to be compared.

The model starts iterating through every sign present in the DataFrame. It discards those which do not have the same hands present in the recorded sign and sets their distances as inf in the distance column of the DataFrame. The `dtw` method from `dtw-python` [14] is then called with the respective embeddings of the reference and recorded sign as arguments. The distance obtained is then set in the corresponding entry of the DataFrame.

Challenges Faced

The `dtw` method handles the case of having a sequence of length 1 differently. In such a case, it transposes the array. This happened in a few instances on training due to some signs registering a particular hand in only one frame. This is dependent on the confidence tracking of mediapipe. As such, this specific case is handled by duplicating the singular entry present in the sequence to obtain an array of length 2.

3.5.2 fastdtw

The `fastdtw_distances` method works similarly to `dtw_distances`. It uses the `fastdtw` instead to calculate the distances. This implementation does not suffer from the same issue of handling sequences of length 1. This implementation runs in linear time, rather than in $O(m \times n)$.

3.6 Sign Prediction

Given the `pandas.DataFrame` with all distances computed, a prediction can now be made. The DataFrame is first sorted in ascending order of distances, with the top row being the sign with the lowest distance. Lower distances indicate higher similarity.

3.6.1 Batch Size and Threshold

The `get_sign_predicted` method takes an argument for `batch_size`. This indicates how many signs we need to consider in order to make a reliable prediction. For example, we can take a batch size of 5, which means we take the 5 signs with the lowest distances

and output the most common one. This increases the confidence in the prediction made. We choose a label l as follows:

$$l = \max\left(\frac{l_i}{N}\right), \text{ for } 0 \leq i < N$$

N corresponds to the chosen `batch_size`. A reasonable upper bound for N can be obtained by considering the label in the training data with the least amount of examples. This ensures that every label has the possibility of achieving a confidence of 100%.

The threshold argument is used to determine if a sign is common enough to be output with a certain minimum confidence. It is defined as the minimum ratio of appearances to the batch size that a sign needs to achieve in order to be output. The model outputs "Unknown Sign" if the threshold is not met.

3.7 Usage

The implementation provides a record button to start a capture of a determined number of frames. Once pressed, the feed will capture that amount of frames and process them. To indicate that the feed is recording, a circle is used as an indicator. It turns red while the feed is recording.

A hand and pose model will be generated for each frame and they are all stored in a sign model. Once compared, the program will output the most probable label as a prediction, which will be displayed on the bottom of the feed.

Chapter 4

Evaluation

Chapter 5

Conclusion

5.1 Further Work

Bibliography

[1] S. Albanie, G. Varol, L. Momeni, H. Bull, T. Afouras, H. Chowdhury, N. Fox, B. Woll, R. Cooper, A. McParland, *et al.*, "Bbc-oxford british sign language dataset," *arXiv preprint arXiv:2111.03635*, 2021.

[2] H. R. V. Joze and O. Koller, "Ms-asl: A large-scale data set and benchmark for understanding american sign language," *arXiv preprint arXiv:1812.01053*, 2018.

[3] T. Liu, W. Zhou, and H. Li, "Sign language recognition with long short-term memory," in *2016 IEEE international conference on image processing (ICIP)*, pp. 2871–2875, IEEE, 2016.

[4] D. Guo, W. Zhou, H. Li, and M. Wang, "Hierarchical lstm for sign language translation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[5] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.

[6] C. A. Ratanamahatana and E. Keogh, "Everything you know about dynamic time warping is wrong," in *Third workshop on mining temporal and sequential data*, vol. 32, Citeseer, 2004.

[7] L. Momeni, G. Varol, S. Albanie, T. Afouras, and A. Zisserman, "Watch, read and lookup: learning to spot signs from multiple supervisors," in *Proceedings of the Asian Conference on Computer Vision*, 2020.

[8] F. Ronchetti, F. Quiroga, C. A. Estrebou, L. C. Lanzarini, and A. Rosete, "Lsa64: an argentinian sign language dataset," in *XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016).*, 2016.

[9] I. Grishchenko and V. Bazarevsky, "Mediapipe holistic — simultaneous face, hand and pose prediction, on device," Dec 2020.

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pp. 21–37, Springer, 2016.

[11] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, "Blazepose: On-device real-time body pose tracking," *arXiv preprint arXiv:2006.10204*, 2020.

[12] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann, "Blazeface: Sub-millisecond neural face detection on mobile gpus," *arXiv preprint arXiv:1907.05047*, 2019.

[13] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*, pp. 483–499, Springer, 2016.

[14] T. Giorgino, "Computing and visualizing dynamic time warping alignments in r: the dtw package," *Journal of statistical Software*, vol. 31, pp. 1–24, 2009.

Appendix A

Project Proposal

Computer Science Tripos – Part II – Project Proposal

Sign Language Recognition

Ronan Ragavoodoo, Homerton College
Originator: The Author
14 October 2022

Project Supervisor: Mr. Filip Svoboda
Director of Studies: Dr. John Fawcett
Project Overseers: Dr. Ferenc Huszar & Dr. Andreas Vlachos
University Teaching Officer: Professor Nicholas Lane

Introduction

Sign Language has been adopted by many people who are unable to speak, or hear. While technology has been used to help this affected minority group to a certain degree, there are still a lot of facilities that can be improved to help them communicate. For example, even Windows does not support a form of sign language recognition. This is despite the advances that we have concerning digital assistants, which could help even further. Furthermore, there are different variations of sign languages, and most of the research focus has always exclusively been done on American Sign Language (ASL), leading to less inclusion. Thus, it is of interest to develop reliable software which could help in including variations of sign language as inputs to modern day devices.

Work to be done

The goal of this project is to implement a reliable detection model that can recognise a certain subset of sign language expressions displayed in front of a camera. The particular

variation of sign expressions that will be focused on is British Sign Language. Given that there is a possibility of a lack of reliable raining data, there should also be a way of adding new labelled signs to the training data.

I will first start by performing image labelling on data gathered. Collection will be done either on an existing dataset if a suitable one is available, or manually created if none are. This will be followed by training the model on the collected dataset, and then perform testing. The model will be evaluated with respect to other similar implementations.

Python seems like a suitable language for this project due to its popularity and variety of machine learning libraries available for it. It is a high-level language which has simple and readable code, making the implementation part of the project less tricky. OpenCV is a popular computer vision library that could be used for the project and which is popular for these applications. The method of recognising hand gestures is not confirmed and will be decided after the research and preparation phase. A good starting point for that would be contour detection.

Success Criteria

The project will be considered a success if I have done all of the following:

- Implemented an image learning model that successfully recognises a subset of British sign language with reliable accuracy and efficiency.
- Implemented a reliable way of capturing training data for extending the vocabulary of the model.
- Evaluated the efficiency and accuracy of the model, comparing it to the results of similar sign language recognition models.
- Explore possibility of using external devices other than webcams, such as devices equipped with depth-sensors (the Kinect[1]for example).

Extensions

- Expand the training data to cover other variations of sign languages.
- Optimise the algorithms to achieve faster processing.
- Evaluate the model with languages that involve more movement, rather than more static expressions.

Starting Point

I have some experience using python at an intermediate level.

I have never used OpenCV or any other machine learning libraries. Most of my experience with machine learning comes from coursework

[1]See https://en.wikipedia.org/wiki/Kinect

Resources required

A list of required resources:

- My own machines:
  - Laptop (Intel i7-10750H, 16GB RAM, Windows 11)
  - Backup Laptop (Intel i5-7200U, 8GB RAM, Ubuntu 21.04)
- OpenCV and other machine learning libraries, which will be settled on upon further research and noted in the reports.
- Github will be used to store regularly backups of the code along with a copy of the dissertation and proposal. Additionally, they will be kept on OneDrive and Google Drive.

Timetable and milestones

Work Package 1 : 13 Oct - 27 Oct

Task: Research about machine learning models and of similar implementations.

Milestone: Have run basic detection scripts on machine which can do basic contour detection or hand tracking if possible.

Work Package 2 : 27 Oct - 10 Nov

Task: Perform data collection and start implementing the sign language expression data capture component.

Milestone: Successfully obtain a reasonable amount of data for training and testing, along with a component that can successfully store training data from displayed sign expressions.

Work Package 3 : 10 Nov - 24 Nov

Task: Start implementing the base model, along with conducting some more research if needed to fill in the gaps.

Milestone: Model successfully able to differentiate hand and background, along with skeleton code for recognising signs set up.

Work Package 4 : 24 Nov - 08 Dec

Task: Finish implementation to recognise different sign expressions when displayed.

Milestone: Successfully able to differentiate different sign expressions shown on camera.

Work Package 5 : 08 Dec - 22 Dec

Task: Slack time / Start drafting implementation chapter and catch up on late deliverables. Start to get structure of dissertation written down.

Milestone: Finalised implementation of project.

Work Package 6 : 22 Dec - 05 Jan

Task: No Work on Project
Milestone: Finalised implementation of project and outline of dissertation setup.

Work Package 8 : 05 Jan - 19 Jan
Task: Write implementation of draft dissertation
Milestone: Draft Implementation chapter prepared and sent to supervisor for feedback.

Work Package 9 : 19 Jan - 02 Feb
Task: Write Progress Report and prepare report presentation
Milestone: Submit Progress Report (by 12:00 03 February)
Milestone: Presentation Ready

Work Package 10 : 02 Feb - 16 Feb
Task: Slack Time / Work on late deliverables
Milestone: Project has successfully been implemented and implementation feedback has been incorporated into draft dissertation.

Work Package 11 : 16 Feb - 02 Mar
Task: Conduct evaluation of implementation. Setup possible optimisations if needed.
Milestone: Demonstrate a convincing argument of the efficiency and possible optimisations of the implementation to supervisor.

Work Package 12 : 02 Mar - 16 Mar
Task: Write Evaluation chapter for draft dissertation
Milestone: Draft Evaluation chapter prepared and sent to supervisor for feedback.

Work Package 13 : 16 Mar - 30 Mar
Task: Finish draft dissertation and add missing chapters.
Milestone: Draft dissertation completed and sent to supervisor

Work Package 14 : 30 Mar - 13 Apr
Task: Incorporate draft dissertation feedback into final dissertation and resent to supervisor for a second round of feedback.
Milestone: Dissertation completed with feedback incorporated.

Work Package 15 : 13 Apr - 27 Apr
Task: Slack Time / Finalise Project. Catch up on any late deliverables.
Milestone: Final Project and Dissertation Ready for submission

Work Package 16 : 27 Apr - 11 May
Task: Slack Time
Milestone: Dissertation submitted (by 12:00 12 May)