Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Projektarbeit Elektrotechnik

AI-Based Translation of Sign Language

Autoren	Gianluca Pargätzi Michael Wäspe
Hauptbetreuung	Martin Loeser
Nebenbetreuung	Tobias Welti

Datum	23.12.2022

Abstract

Sign language recognition is a difficult challenge due to the complex relationships between motion, gestures, and facial expressions. However, advances in computer vision simplify this task by using neural networks with the help of supervised learning. This work involves the development of a recognition model for static gestures based on the ML solution "MediaPipe Hands" and its real-time implementation in the context of a test application on an iOS device. The proposed network structure of the classifier with skip connections shows an accuracy of 94.7% for the ASL alphabet (excl. letters J and Z) with data augmentation. By optimizing the network using Core ML, the inference with an iPhone 14 Pro is only $784\mu s$, which is about 100 times faster than a desktop app with a Python interpreter. The potential of the solution is high, as the first version of an additionally proposed prototype with LSTM structure for dynamic gestures also achieves promising results for the entire alphabet with 83.7% accuracy even on a small training dataset. With a diversified dataset, this approach could therefore be used for recognition of more complex gestures.

Keywords: Artificial intelligence, gesture recognition, sign language, MediaPipe

Zusammenfassung

Die Erkennung von Gebärdensprache ist aufgrund der komplexen Zusammenhänge zwischen Bewegung, Gestik und Mimik eine schwierige Herausforderung. Fortschritte in Computer Vision vereinfachen aber diese Aufgabe durch Einsatz von neuronalen Netzen mithilfe von Supervised Learning. Diese Arbeit beinhaltet die Entwicklung eines Erkennungsmodells für statische Gesten auf Basis der ML-Lösung "MediaPipe Hands" und dessen Echtzeit-Implementation im Rahmen einer Testapplikation auf einem iOS-Gerät. Die ausgearbeitete Netzstruktur des Classifiers mit Skip Connections zeigt mit Datenaugmentierung eine Genauigkeit von 94.7% für das ASL Alphabet (exkl. Buchstaben J und Z). Durch Optimierung des Netzes mittel Core ML beträgt die Inference auf einem iPhone 14 Pro nur $784\mu s$, welches damit ca. 100-mal schneller ist als eine Desktop-App mit Python-Interpreter. Das Potenzial der Lösung ist hoch, denn auch die erste Version eines zusätzlich vorgeschlagenen Prototyps mit LSTM-Struktur für dynamische Gesten erzielt für das gesamte Alphabet mit 83.7% Genauigkeit bereits bei einem kleinen Trainingsdatensatz vielversprechende Ergebnisse. Dieser Ansatz kann deshalb mithilfe eines diversifizierten Datensatzes auch für die Erkennung komplexerer Gesten eingesetzt werden.

Schlagwörter: Künstliche Intelligenz, Gestenerkennung, Gebärdensprache, MediaPipe

Contents

Chapter 1

Introduction

1.1 Initial Position and Use Case

Sign language is an important tool for communication with people who have a hearing impairment. The interrelationships of gestures and facial expressions must be learned, as with any foreign language. Due to the complexity and variety of gestures and symbols and their variations, automated sign language recognition brings challenging hurdles.

A robust gesture recognition solution opens the doors to new application possibilities. Sensor-based solutions such as sensor gloves were used in the past but are now increasingly replaced by AI-based computer vision approaches. Due to the availability of conventional camera systems in mobile devices, this image-based concept for gesture recognition can be adapted to the use-case of recognizing expressions in sign language. The biggest challenge lays undeniably in the generation of a suitable dataset representing all gestures in a compact number of meaningful, distinguishable features.

1.2 Project Scope

The first phase of this project involves the implementation of a desktop-compatible program, which is able to recognize simple static sign language gestures using a webcam. The focus lays on the design and implementation of a neural network for the classification of user gestures and the subsequent network training with the help of collected training data. The work is limited to the recognition of a compact number of useful static hand poses. The goal is set to recognize the alphabet of American Sign Language (figure 1.1). The letters "J" and "Z" are excluded in the first step because they are based on dynamic gestures.

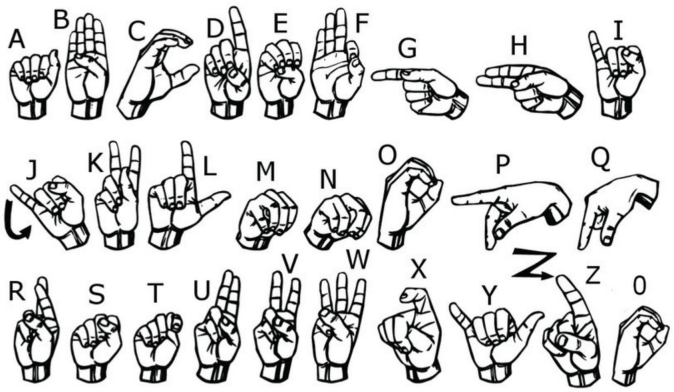


Figure 1.1: American Sign Language Alphabet

In a next project step, an iOS-compatible smartphone application has to be developed based on the neural network for static gestures. The realization of a mobile device application mainly focuses on the simplification and conversion of the trained network and its implementation in a real-time capable user environment. Packaged in an intuitive user interface, the application should enable the user to spell the alphabet letters similar to the model of the first project phase.

In addition to a smartphone app, the static model will be extended to dynamic gestures. This includes the expansion to the complete alphabet including the dynamic gestures "J" and "Z". For this purpose, the training data must be adapted to image sequences and the network must be extended to the recognition of hand movements. In a subsequent evaluation, the dynamic model will be examined based on its performance and accuracy after training with a small test dataset.

1.3 Project Management and Planning

Due to the scope of the project, meticulous planning and documentation of the project steps is necessary. The progress of the project is continuously discussed in weekly project meetings and roughly recorded in the corresponding meeting minutes.

For time management purposes, individual project steps were broken down in advance in a detailed schedule (appendix A.5). Milestone 1 is the completion of the research and data set creation. With milestone 2 the minimal part of this work (proof-of-principle for recognition of static gestures) is fulfilled. Subsequently, the project team will split up and tackle project phases 3 (iOS app) and 4 (recognition of dynamic gestures) simultaneously. As project completion, the last two weeks will be invested in the project documentation.

Chapter 2

Theory

2.1 Neural Network Structure

As a branch of supervised learning, the usage of neural networks has become more common and is broadly researched. Even though the structure of a neural network can be quite complex, it mainly consists of a set of hidden functional layers with trainable parameters to optimize for a specified problem (Classification, Segmentation, Regression, Forecasting).

2.1.1 Fully Connected Neural Network

A Dense Layer mainly consists of m neurons $a_1^{(k)}, ..., a_m^{(k)}$ each connected to all data points $a_1^{(k-1)}, ..., a_n^{(k-1)}$ from the layer before (either another layer or input data). The value of a single neuron is a linear combination of given data points $\boldsymbol{a}^{(k-1)}$ with certain weights $\boldsymbol{w}$ and an offset (bias) $\boldsymbol{b}$. Biases and weights are "trainable Parameters" and thus determined during training process. To allow the system to learn more complex dependencies, it is common to add non linearity by using an activation function f (e.g. σ, tanh, or *swish*) for the value of a single neuron.

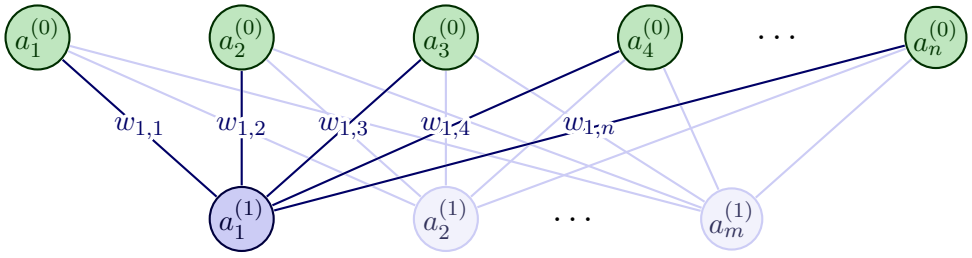


Figure 2.1: Structure of a fully connected layer

Summing up the information, a single Dense Layer can be described as followed with a single matrix multiplication (same as figure 2.1):

$$\begin{pmatrix} a_1^{(k)} \\ \vdots \\ a_m^{(k)} \end{pmatrix} = \boldsymbol{f}\left(\boldsymbol{W} \cdot \boldsymbol{a}^{(k-1)} + \boldsymbol{b}\right) = \boldsymbol{f}\left(\begin{bmatrix} w_{1,1} & \dots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \dots & w_{m,n} \end{bmatrix} \cdot \begin{pmatrix} a_1^{(k-1)} \\ \vdots \\ a_n^{(k-1)} \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}\right) \tag{2.1}$$

Note:
Nowadays, neural network training is increasingly outsourced to GPUs because the parallel processor structure is more efficient for float matrix computations than sequential CPU technology. For this work, network training was executed on a powerful Nvidia GeForce RTX 3090.

2.1.2 Convolutional Neural Network

A convolutional layer basically is a set of m learnable filters $\boldsymbol{w}$, creating m output features $g_1^{(k)}(x,y), ..., g_m^{(k)}(x,y)$. The filters are trained to recognize meaningful spatial information, which is why convolutional layers are mostly used in computer vision applications. For a total of n features in the previous layer and an output of filters with a kernel-size of $k \times k$ and a bias b each, the amount of trainable parameters is $(k^2 \cdot n + 1) \cdot m$. In addition to a simple convolution (linear transformation) a selected non-linear activation function f can be applied (same as in fully connected layers).

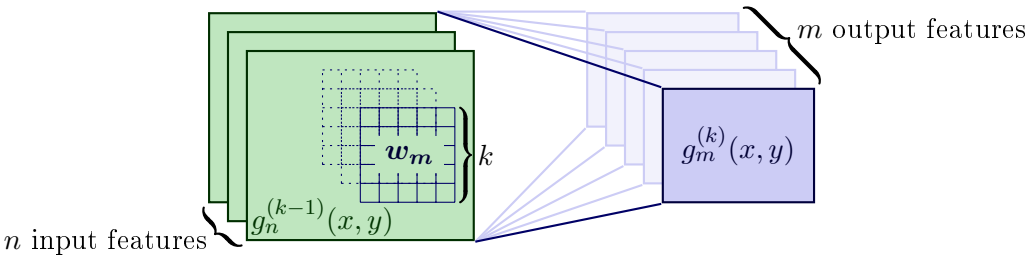


Figure 2.2: Structure of a convolutional layer

Summing up the information above, a single convolutional layer can be described as followed:

$$\boldsymbol{g}^{(k)}(x,y) = \begin{pmatrix} g_1^{(k)}(x,y) \\ \vdots \\ g_m^{(k)}(x,y) \end{pmatrix} = \begin{pmatrix} f\left(\boldsymbol{w_1} * \boldsymbol{g}^{(k-1)}(x,y) + b_1\right) \\ \vdots \\ f\left(\boldsymbol{w_m} * \boldsymbol{g}^{(k-1)}(x,y) + b_m\right) \end{pmatrix}, \quad \boldsymbol{w_1}, ..., \boldsymbol{w_m} \in \mathbb{R}^{k \times k \times n} \tag{2.2}$$

To classify the extracted features, the information has to be compressed using additional layer types, mostly pooling layers (e.g. max or mean pooling filter). Classification itself then happens in fully connected structures using the flattened pooling layer output.

Note:
The influence of the structure of a very deep convolutional neural network is a complex research question. A broad selection of established computer vision networks is available with pre-trained filters, which are optimized for generalized feature extraction. These networks can easily be customized to a specific problem using transfer learning (retraining certain network layers).

2.1.3 Recurrent Neural Networks

A special application of neural networks is forecasting and predictions based on temporal information in given data point sequences. Recurrent networks learn temporal dependencies of sequences by not only considering the current input data, but also the output of the network from previous features. For each "frame" $1, ..., t, ..., T$ the reccurent cell gets sequentally updated by using the frame's input features (data vector) $\boldsymbol{x_t} \in \mathbb{R}^n$ and the cell state one timestep $t-1$ before.

A special type of recurrent neural networks are LSTM layers. The most basic form has a structure as shown in figure 2.3. The dimension of cell state $\boldsymbol{C}$ and output state $\boldsymbol{h}$ can be specified with the "unit" argument m in Tensorflow. The layer output in general is the output state $\boldsymbol{h_T}$ after processing the last sequence frame.

An LSTM cell is designed to learn long-time dependencies, which normal RNNs struggle with. The structure can vary slightly for different use cases and can be split up in three fundamental processing mechanisms (forget, input and output gate).

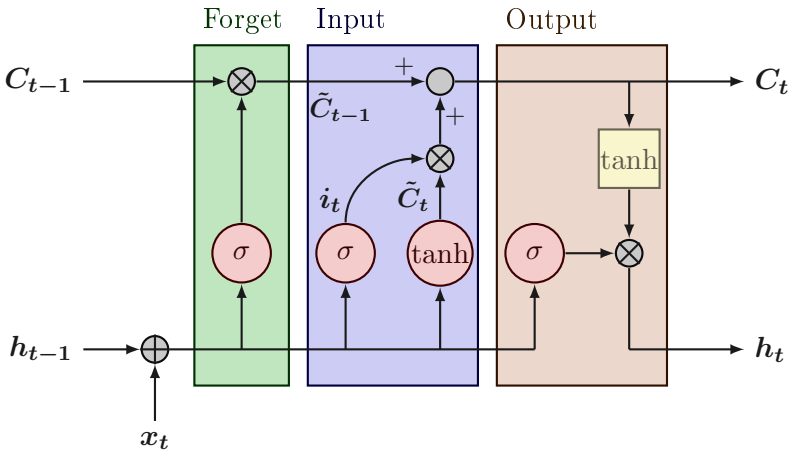


Figure 2.3: Basic structure of an LSTM cell ($\oplus$ = concatenation, $\otimes$ = element-wise multiplication)

All gates are updated based on the concatenated vector $[\boldsymbol{x_t}, \boldsymbol{h_{t-1}}] \in \mathbb{R}^{n+m}$.

Forget Gate
The forget gate is a fully connected layer with a sigmoid activation function and "decides" how important the previous cell state $\boldsymbol{C_{t-1}}$ is ($\circ$ = element-wise multiplication).

$$\tilde{\boldsymbol{C}}_{\boldsymbol{t-1}} = \boldsymbol{C_{t-1}} \circ \sigma(\boldsymbol{W_f} \cdot [\boldsymbol{x_t}, \boldsymbol{h_{t-1}}] + \boldsymbol{b_f}), \qquad \boldsymbol{W_f} \in \mathbb{R}^{m \times (n+m)}, \boldsymbol{b_f} \in \mathbb{R}^m \tag{2.3}$$

Input Gate
The input gate consist of two fully connected layers and adds the current input state based on its importance to the last cell state $\boldsymbol{C_{t-1}}$ to determine the next cell state $\boldsymbol{C_t}$

$$\tilde{\boldsymbol{C}}_{\boldsymbol{t}} = \tanh(\boldsymbol{W_C} \cdot [\boldsymbol{x_t}, \boldsymbol{h_{t-1}}] + \boldsymbol{b_C}), \qquad \boldsymbol{W_C} \in \mathbb{R}^{m \times (n+m)}, \boldsymbol{b_C} \in \mathbb{R}^m \tag{2.4}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W_i} \cdot [\boldsymbol{x_t}, \boldsymbol{h_{t-1}}] + \boldsymbol{b_i}), \qquad \boldsymbol{W_i} \in \mathbb{R}^{m \times (n+m)}, \boldsymbol{b_i} \in \mathbb{R}^m \tag{2.5}$$

Combined with the output of the forget gate, the cell state gets updated to

$$\boldsymbol{C_t} = \tilde{\boldsymbol{C}}_{\boldsymbol{t-1}} + \boldsymbol{i_t} \circ \tilde{\boldsymbol{C}}_{\boldsymbol{t}} \tag{2.6}$$

Output Gate
The final output gate is used to decide how important the internal cell state $\boldsymbol{C_t}$ is for the cell output $\boldsymbol{h_t}$ based on the current input.

$$\boldsymbol{h_t} = \tanh(\boldsymbol{C_t}) \circ \sigma(\boldsymbol{W_o} \cdot [\boldsymbol{x_t}, \boldsymbol{h_{t-1}}] + \boldsymbol{b_o}), \qquad \boldsymbol{W_o} \in \mathbb{R}^{m \times (n+m)}, \boldsymbol{b_o} \in \mathbb{R}^m \tag{2.7}$$

All four fully connected layers result in a total of $4 \cdot m \cdot (m + n + 1)$ trainable parameters.

Note:
For a more detailed explanation of the LSTM network, we would like to refer to "Understanding LSTM Networks", walking through each aspect of an LSTM cell as well as some variations in vivid detail.

2.2 Training Neural Networks

2.2.1 Gradient Descent

Deep neural networks are usually trained using supervised learning, where a network is given a set of training data and the corresponding desired network outputs. Training basically means adjusting the q network parameters $p_1, ..., p_q$ (weights $\boldsymbol{w}$ and biases $\boldsymbol{b}$), so that the difference between the prediction from a dataset and its solution, also called loss L, is minimal:

$$\min L \underbrace{(w_1, ..., w_m, b_1, ..., b_n)}_{m+n=q \text{ free Parameters}} \tag{2.8}$$

Note:
The structure of the loss function depends mainly on the application type, a broad selection of functions is available in machine-learning platforms like Tensorflow or Pytorch.

To minimize the loss function (find local minimum in loss landscape $L : \mathbb{R}^q \to \mathbb{R}$), a wide range of optimizers is available with adjustable learning rates γ, which optimize the parameter values based on iterative gradient descent starting from a point $\boldsymbol{p_0}$:

$$\boldsymbol{p_{k+1}} = \boldsymbol{p_k} - \gamma \cdot \nabla L(\boldsymbol{p_k}), \qquad \boldsymbol{p_k}, \boldsymbol{p_{k+1}} \in \mathbb{R}^q \tag{2.9}$$

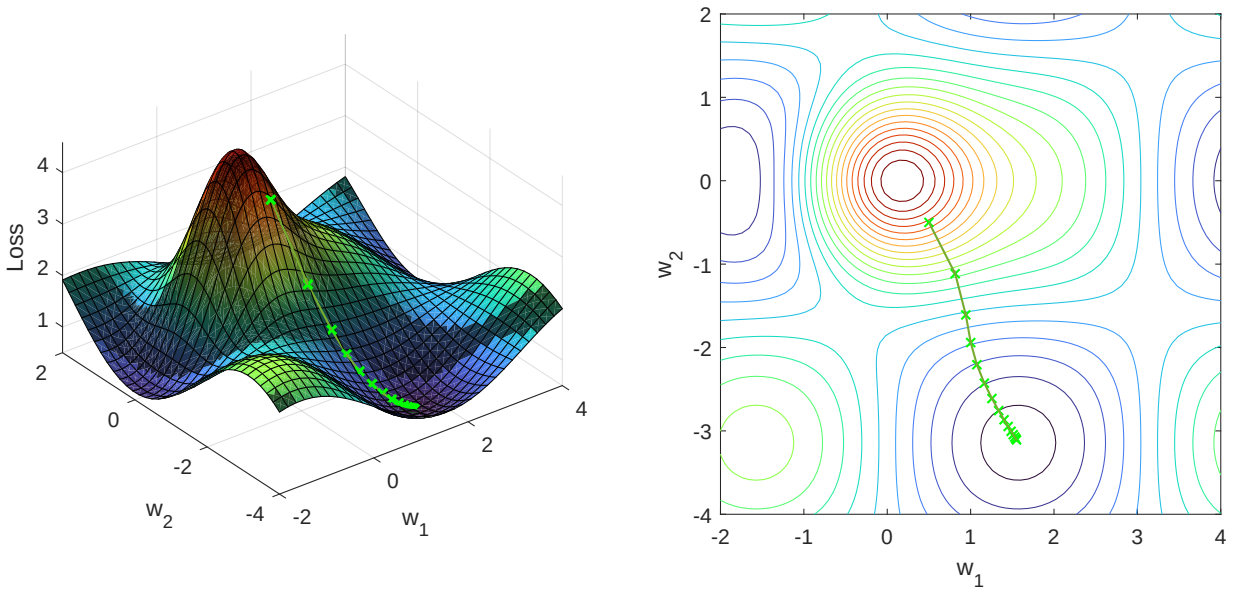


Figure 2.4: Gradient descent in loss landscape $L : \mathbb{R}^2 \to \mathbb{R}$ (two weights w_1, w_2)

Unlike a naive direct gradient computation, as of today more advanced algorithms such as backpropagation, popularized in [1] (introduced in the 1960s) are used to calculate gradients more efficiently and thus optimize training in multilayer networks.

2.2.2 Overfitting

Overfitting can occur when neural networks are too specificly fitted to the training dataset. In this case, the AI-Application will perform perfect on the training data, but is not able to correctly predict unseen data. For this reason, preventing overfitting is necessary to generalize a model with a meaningful relationship between input and output.

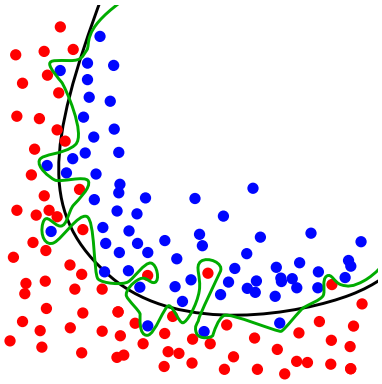


Figure 2.5: Ideal border (black), overfitting (green) [2]

A general way of detecting overfitting is to use unseen data for a validation of the network during training. A small loss for training data and at the same time high loss for validation data signalizes overfitting.

The following methods are used to prevent overfitting:

- Early Stopping: Pausing the training process at the sweet spot between over- and underfitting
- Generalise Dataset: With a bigger dataset, the meaningful input features are easier to isolate. Additional data either has to be recorded or created using data augmentation.
- Feature Selection: A smaller selection of meaningful features sometimes performs better than big data.
- Regularization: Identify and reduce the meaningless data features (noise) by using dropout or batch normalization layers.

2.2.3 Skip Connections

Skip connections are a special way of interconnecting layers by skipping some layers in between and are used to tackle the vanishing gradient and degradation problem, which often occurs in complex and deep neural networks (Figure 2.6: High amount of local minimas in complex loss landscape). The model training and convergence can be stabilized, which is why a broad selection of well-known pre-trained computer vision networks (DenseNet [3], ResNet [4] and Unets[5]) use skip connections.

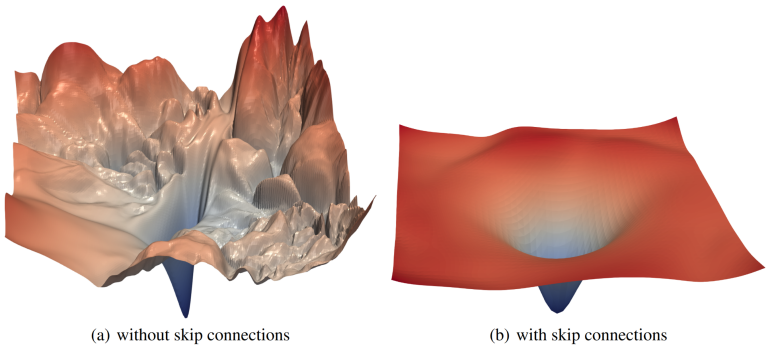


Figure 2.6: Loss landscape simplification using skip connections [6]

Chapter 3

Procedure and Methodology

3.1 Processing Procedure Requirements

The requirements for the gesture recognition process are the main factors influencing the design of a processing chain for automatic sign language recognition. For best possible results, solutions need to be optimized regarding the following criterias:

- Robustness against environmental factors such as image noise, exposure and dynamic backgrounds
- Position, rotation and scaling invariant hand pose recognition
- Different hand shapes and skin colors
- Efficient processing with low latency for real-time potential

3.2 Evaluation of Processing Steps

The task of digitally assigning input data to a specific gesture is complex and was split up in different processing steps.

The state of the art for gesture recognition is diverse and a great deal of different implementation approaches were pursued in the past. Available research papers e.g. [7][8] and [9] offer a quick review of different approaches. The following summed up overview takes existing solutions based on artificial intelligence into account and neglects the countless other machine learning and image processing approaches (kNN-Classifier, SVM, Random Forests, SIFT, Fourier Descriptors, ...).

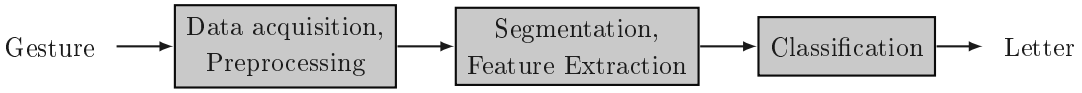


Figure 3.1: Processing steps for gesture recognition

3.2.1 Data Acquisition & Preprocessing

In the first step, the gesture information has to be digitized in order to process it correspondingly.

Sensor-based Methods

The first approaches of gesture recognition as human-computer interaction were sensor-based solutions. A gesture was detected for example with curvature or angle sensors mounted on special sensor gloves (figure 3.2). While these approaches provide good results, they become unwieldy due to numerous limitations and the need for special hardware. Thus, this work realizes a sensorless approach.

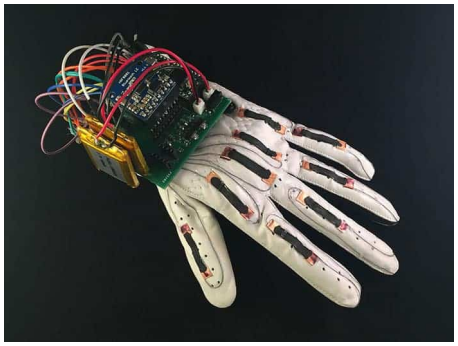


Figure 3.2: Smart glove used to translate sign language to digital text [9]

Depth Images

Due to the complexity of gesture recognition, approaches using depth images as generated for example with a "Microsoft Kinect" camera were further investigated. However, the hardware required for this (for example LIDAR or ToF sensors) is still not state of the art in conventional smartphones, which is why such approaches are also excluded from this project.

Camera-based Approaches

Today's solutions for gesture recognition are mostly camera-based. The use of optical approaches is extremly simple and more convenient from the user's point of view because basic camera systems are broadly available. The disadvantage is that in comparison to other solutions the gesture-specific information has to be elaborately extracted from a conventional image in a more complex manner. The progress in machine learning and AI however simplifies this aspect, which is why such a computer vision approach is pursued.

3.2.2 Segmentation and Feature Extraction

After collecting the input data representing a gesture, in a next step the gesture specific information has to be extracted and isolated. Herefore, mostly machine learning or neural network solutions are pursued.

Convolutional Neural Networks

The brute-force approach for feature extraction would be a model-free solution using a CNN. The disadvantage is that comparatively complex tasks such as pose or gesture recognition also require rather deep and complex network structures like InceptionResNetV2 as proposed in [10]. Due to the high amount of network parameters, training with image data is consequently also costly and a large dataset is necessary.

Model-based Approaches

Model-based approaches estimate a defined feature group (e.g. skeleton or volume mesh), but their design and training is time-consuming. This work uses a pre-trained skeleton model for the proof of principle of sign language recognition. Many open source solutions described in [11] such as DeepPrior++ [12] are not applicable because they rely on RGB-D images with depth information or multi-camera systems. MediaPipe [13], OpenPose [14], AlphaPose [15] are cross-platform open-source machine learning approaches of pose recognition for conventional RGB camera images. AlphaPose extracts only body poses, would need retraining to hand skeletons and is therefore discarded as a solution. MediaPipe outperforms OpenPose, is far more robust and gives better results in low-light performance, motion blur, and computational power used. [16]

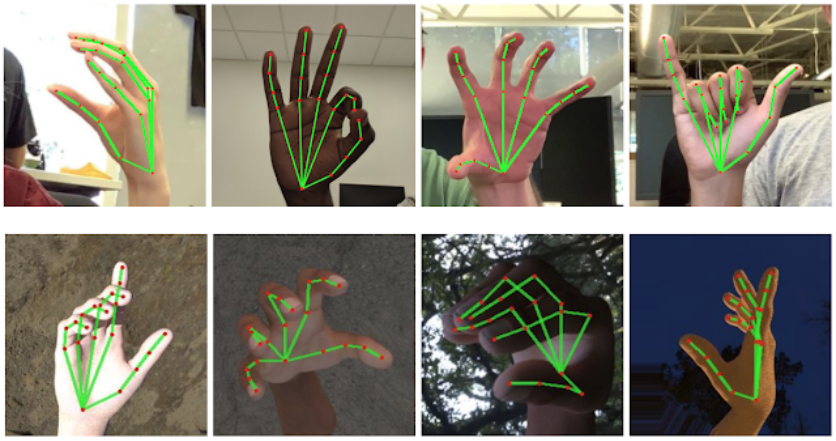


Figure 3.3: MediaPipe Hands: A ML Solution for Hand Pose Esimtation [17]

3.2.3 Classification

The last processing step is the classifier, which connects features to a predicted gesture.

Fully-connected neural networks

The easiest way of implementing classification is to use a network consisting of fully connected layers as proposed e.g. in [18]. Additional skip connections as described in section 2.2.3 can simplify training and stabilize the classification.

LSTM

If not only static gestures, but also dynamic gestures need to be classified, it makes sense to use recurrent networks to additionally consider temporal information while predicting. The usage of LSTM was researched in the past in related papers like [19][20][21] and [22] and seems to deliver promising results.

Chapter 4

Recognition of Static Gestures with MediaPipe Hands and Skip Connections Network

4.1 Functional Tests of MediaPipe Hands

MediaPipe was developed by google and offers customizable machine learning solutions for a broad spectrum of application types (Face recognition, body pose estimation, object tracking, ...).

One pipeline is the MediaPipe Hands solution, which allows the user to extract 21 3D-Keypoints representing all joints in a hand. The model consists of a single-shot hand detector localizing the hand and a regression-based tracker model estimating the hand's keypoints using a cropped image (region of interest) of the predicted hand position.

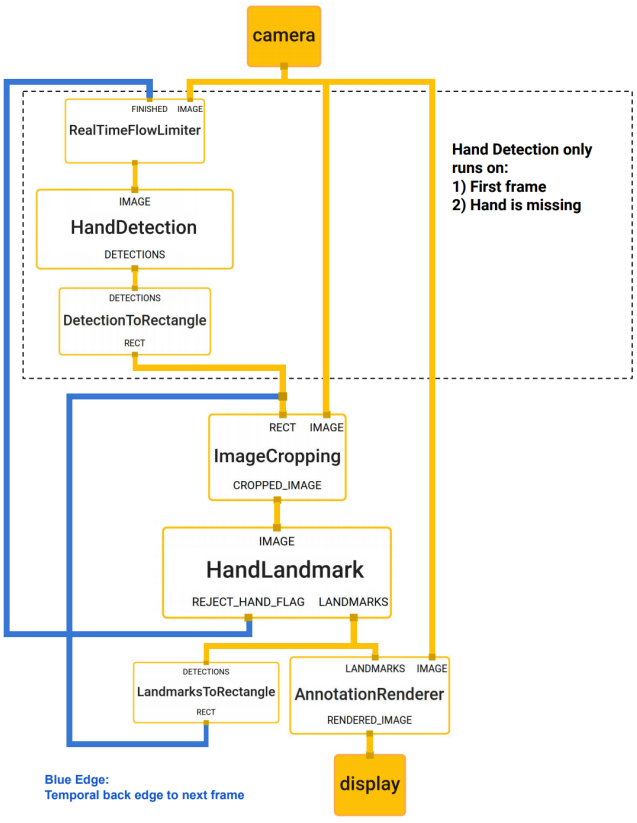


Figure 4.1: Internal structure of two-stage MediaPipe Hands solution [17]

In the first step of realising a gesture recognition network, MediaPipe Hands was evaluated and intensively tested to find its weaknesses. The following findings limit the use of the solution:

Color Sensitivity
It was discovered that the recognition of a hand by MediaPipe depends essentially on the skin color and was trained on "normal" skin colors (skin tones 1 - 6 based on the Fitzpatrick scale). If the RGB color channels are swapped, the skeleton estimation is poor, most of the time no hand was recognized.

Handedness: None Handedness: 99.85%

Figure 4.2: Color sensitivity of MediaPipe Hands module

Incomplete body features
Additionally a limitation was found while processing hand images only consisting of an "isolated" hand without arm and wrist, which were not recognized. After artificially adding an arm, the results were very good, even when the skin color of the inserted arm was significally different.

Handedness: None Handedness: 99.36%

Figure 4.3: Added arm results in better perfomance on "isolated" hand pictures

Low Contrast and Bad Image Quality
In low contrast and with low image resolution, the amount of false hand pose estimations increased significantly. In some cases, especially with cheap built-in computer cameras where the color fidelity is poor, hands were randomly detected even when there was no hand in the picture.

Figure 4.4: Wrongly estimated skeletons for low resolution and noisy images

4.2 Dataset

The main work of implementing a gesture recognition was to gather gesture images and build a problem-oriented dataset. Since the classifier should predict the alphabet letter based on the keypoint coordinates ($3 \cdot 21 = 63$ data points) provided by MediaPipe Hands, the dataset for supervised learning has to also consist of skeletons and the corresponding label (= alphabet letter).

4.2.1 Available Datasets

For best possible classification results, a diverse dataset for each sign language letter is critical. The basis for the skeleton dataset is built by gesture images from existing online data [23][24][25][26][27][28][29] from related work in sign language recognition. Additionally, single frames from youtube videos were captured to enlargen the dataset.

4.2.2 Skeleton Extraction

After reviewing the mentioned datasets, the conclusion was made that the data was far from perfect containing wrong gestures, low resolution images, similiar pictures and even duplicates. The skeleton dataset was therefore created by optimizing the existing dataset using the following workflow:

Figure 4.5: Workflow keypoint dataset

Select base dataset

Because the online dataset quality is insufficient, the first step consists of removing unusable images as followed:

1. Wrongly performed gestures were removed and the remaining images were processed using MediaPipe Hands. Images where no hand was recognized were deleted.
2. The keypoints of all other images were checked visually by hand and images with wrongly estimated skeletons were discarded.

Because most datasets contain a lof of similiar images, only a handful of the remaining "candidate pictures" was selected for further processing (Around 240 Images per letter).

Image augmentation

Neural Networks are heavily reliant on big data to prevent overfitting and maximize their performance.[30] Because the base dataset from the previous step is limited, data augmentation is used to diversify the images such that a better classification network can be built.

To enlargen the image dataset, an image augmentation tool was built, which allows the user to select and apply random geometrical transformations (zoom, rotation, shear and mirroring) and therefore vary the resulting keypoints. Augmentations where no skeletons are recognized were discarded.

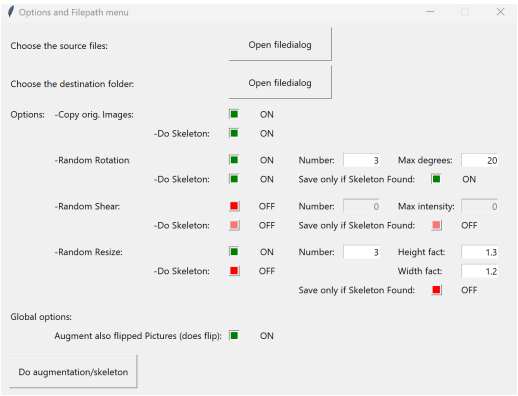


Figure 4.6: Self-programmed image augmentation tool

With 30 Augmentations per image, the total dataset amount was expanded to around 175'000 images. In the diagrams of section 4.4 both training with augmented and unaugmented data is shown. The improving effect on validation accuracy stems solely from image augmentation.

Skeleton extraction

In the last step, every resulting augmented image was yet again processed with MediaPipe to extract its keypoints. At this point, there still are wrongly estimated skeletons and statistical outliers in the augmented data.

To miminze these errors, an additional statistical method for sortig out bad skeleton data was devised:

1. For each skeleton, the normalised distances between wrist and key-points was calculated and then grouped per alphabet letter, resulting in a graph as shown in figure 4.7. Wrongly estimated skeleton are made visible since they also result in wrong key-point distances.

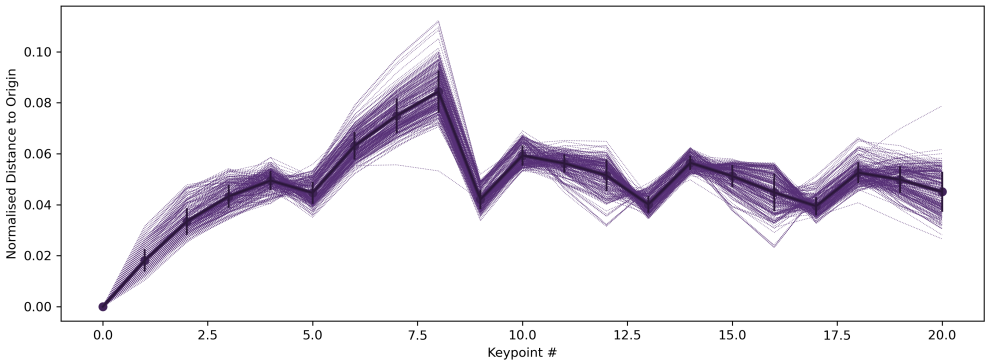


Figure 4.7: Distances to wrist for all skeletons of letter "D"

2. Out of all distances per alphabet letter,a letter specific mean vector was calculated.
3. Skeletons, whose distances do not deviate significantly from the class mean were exported in the final CSV training file, resulting in 153'240 skeletons in total.

Note:
In addition to the described training dataset, a smaller validation dataset with a total of 1063 skeletons was created using selfmade pictures unrelated to the training data.

4.3 Network Structure

The implemented network has a structure as shown in figure 5.4 (Batch normalization and dropout layers not displayed). MediaPipe Hands is used to crop the input image and extract the skeleton keypoints. The 2D-Array of the skeleton coordinates (21×3) gets flattened and then fed through the skip connections network with a total of $23.4k$ trainable parameters. The output layer consisting of $m = 24$ neurons is directly mapped to the 24 alphabet letters. The neuron with the highest value (= probability) is the predicted label.

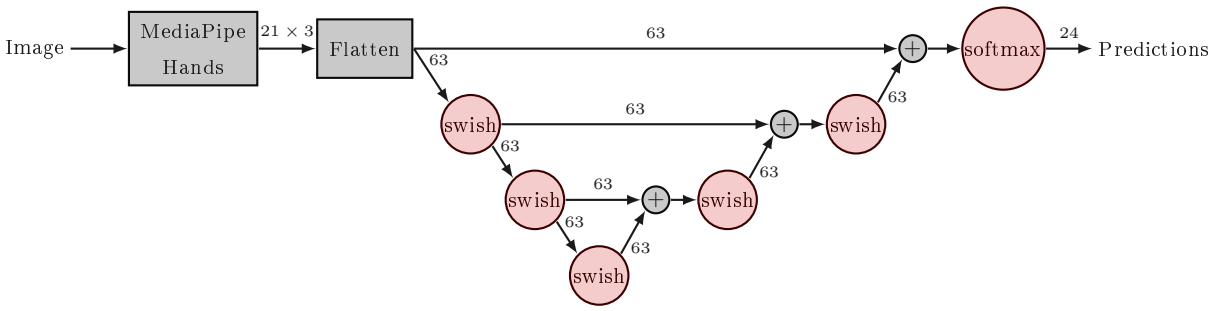


Figure 4.8: Network structure with skip connections (red nodes represent a single fully connected layer with *swish* or *softmax* activation function), full structure in appendix A.2

4.4 Accuracy, Perfomance and Limitations

As shown in figure 4.9, the maximum validation accuracy with the proposed network was 94.4% after 150 training epochs. The network was trained on an Intel Core i7-10610U CPU and for the augmented dataset of about $153k$ skeletons the total training time was approximately $15min$. It is clearly visible, that the image aumentation improved the classification significantly since the unaugmented dataset reached a validation accuracy of 83.3%. The inference time on a normal computer (running on a Intel Core i7-1065G7 CPU) lays at $120ms$, $70ms$ of which is classification inference, the rest is processing time of MediaPipe Hands.

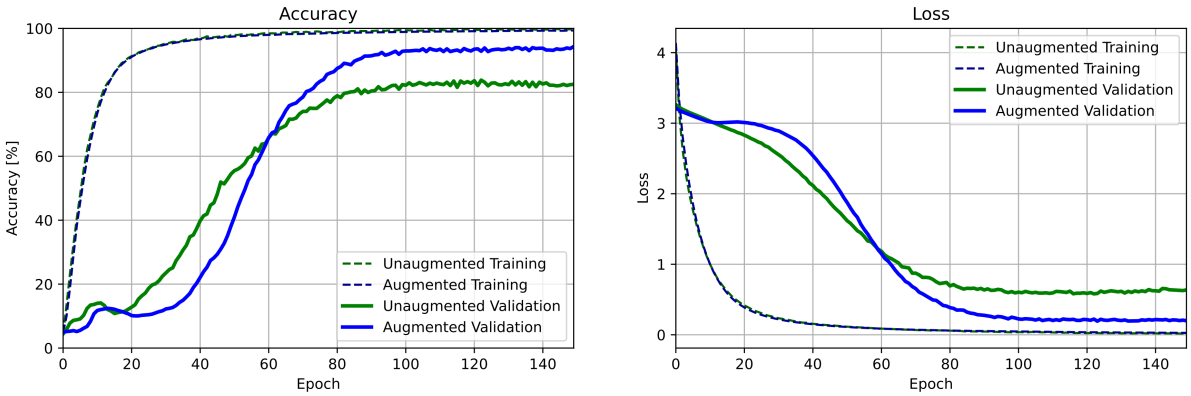


Figure 4.9: Comparison of accuracy and loss for augmented and unaugmented dataset

In the confusion matrix (figure 4.10), the weaknesses of the classification are visible. The distinction between "A", "T" and "S" is difficult since all gestures consist of a closed fist and only vary in thumb positioning. Furthermore "R" and "U" are confused relatively often. This error can arise because depending on exposure, sharpness and contrast, MediaPipe sometimes has difficulties estimating the correct keypoints if fingers are crossed or not clearly visible, as it is the case for the letter "R".

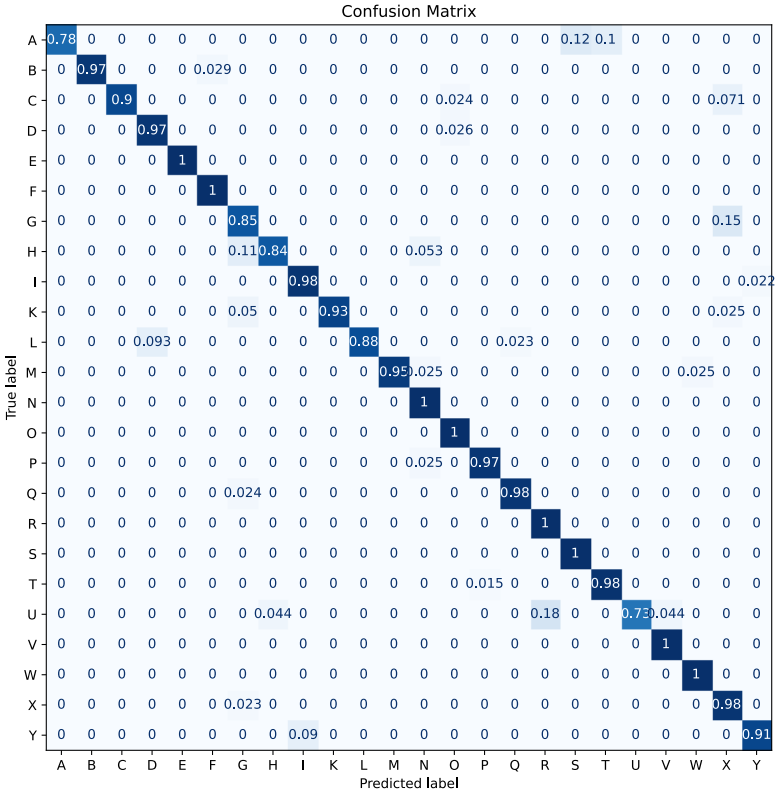


Figure 4.10: Confusion matrix of validation dataset

4.5 Functional Tests with Camera Input

The functional test using a normal pc webcam as shown in figure 4.11 confirmed the results from the validation data . The limitating factor of the recognition stream is clearly the MediaPipe Hands module and its flaws. The classification itself works very well and only breaks down when MediaPipe delivers wrong, none or inaccurate keypoints.

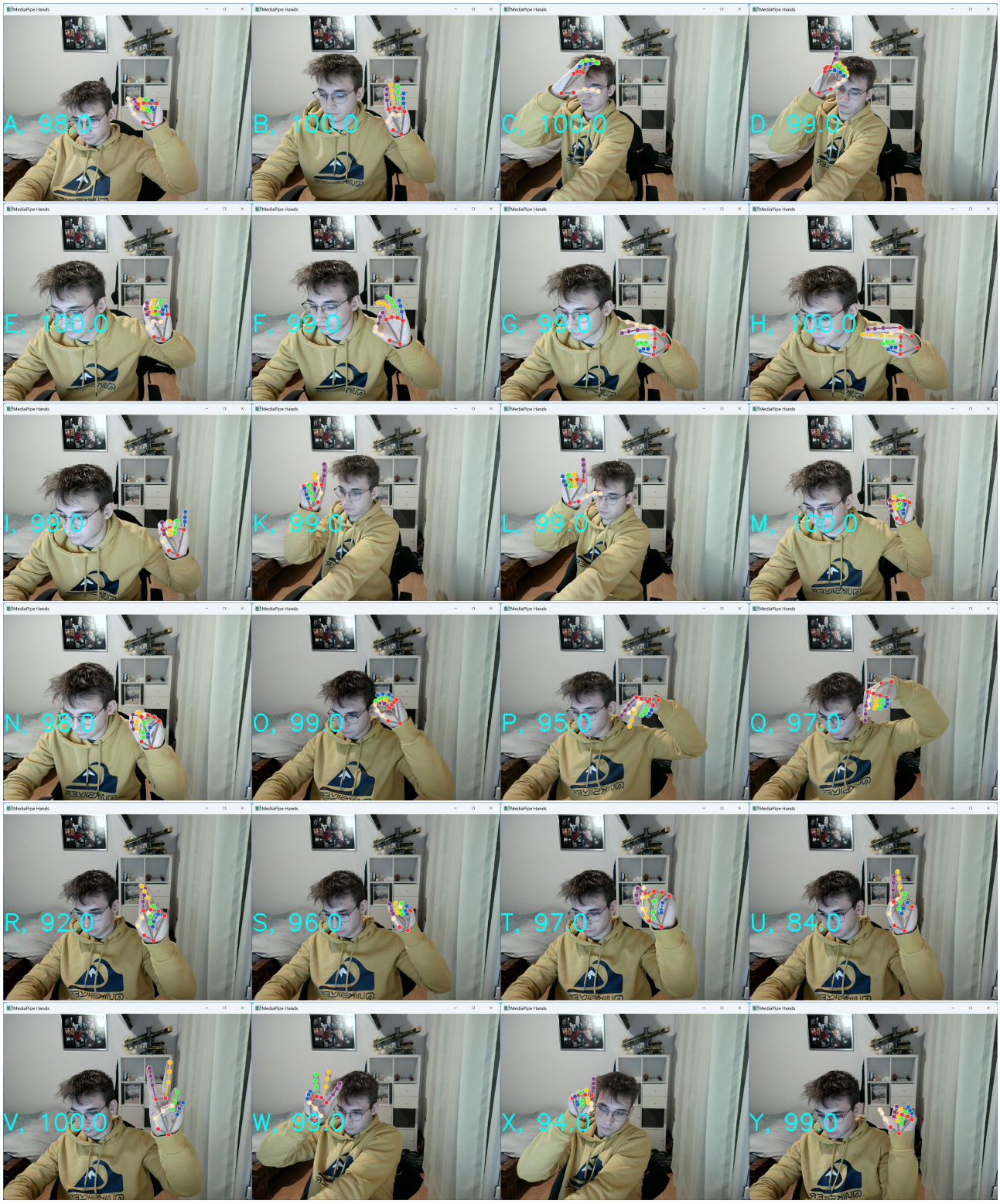


Figure 4.11: Real-time functional tests with built-in laptop webcam (720p)

4.6 Comparison to Brute-Force CNN Approach

For testing purposes, additionally a CNN was trained with the image dataset the skeleton training data was extracted from to evaluate the need of a model-based solution like MediaPipe Hands. Approaches like Convolutional Pose Machines (CPMs) [31] [32] as well as MediaPipe itself use deep CNNs to estimate keypoint coordinates. Using transfer learning, a MobileNetV2 with pretrained image-net weights was retrained to the gesture recognition application on a computanionally powerful Nvidia GeForce RTX 3090. Training with 175 Epochs in about $12h$ training time did not deliver promising results. A diverging validation loss at almost 100% training accuracy is a clear sign of overfitting to unimportant image features.

In related papers e.g. [33] or MediaPipe itself (convolutional layers for keypoint estimation) the implementation of CNNs to recognize hand poses was successful. The reason might be that these approaches only pass a cropped, isolated image of regions of interest (regions containing hands) to the CNN network. The MobileNetV2 in this work was trained with not only isolated hand images, but mostly with images were also the person performing the gesture was partly visible. A promising approach would therefore be a two-staged recognition model similar to the processing of MediaPipe Hands. The first stage would be trained for segmentation (detecting regions of interest) and only then the selected region gets passed to a second CNN for feature extraction.

Chapter 5

Recognition of Dynamic Gestures with MediaPipe Holistic and LSTM Network

5.1 Implementation of MediaPipe Holistic

For dynamic gesture recognition, it is crucial to not only evaluate the hand skeleton, but also the hand position and orientation itself. The simplest approach is to expand the segmentation and feature extraction to MediaPipe Holistic, which combines MediaPipe Hands, MediaPipe Pose and MediaPipe FaceMesh.

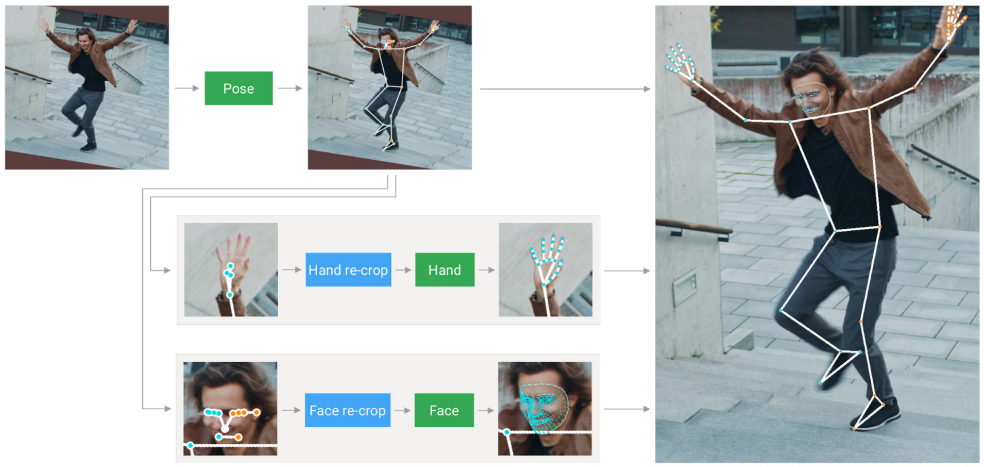


Figure 5.1: MediaPipe Holistic [17]

Instead of only classifying 21 hand keypoints, now the skeleton vector of a single image consists of 2×21 hand keypoints (for left and right hands), and 33 body keypoints. FaceMesh for now is not used, but could be implemented if needed for gestures which additionally include facial expressions.

5.2 Dataset

The dataset for dynamic gesture recognition differs to the static dataset in the following aspects:

- Most images of the static dataset only display a hand without context. For the dynamic dataset it is crucial to not only estimate the hand keypoints, but also the position of the hand in relation to the body.
- The static dataset consists only of single images, which have none or only little relations to each other. For the dynamic dataset, it is necessary to have a recorded sequence of a gesture.

Because a dynamic model now allows the recognition of all sign language gestures and is therefore no longer limited to the alphabet, the online data sets found no longer contain single letters, but more complex gestures for whole word expressions. For this reason, it was decided to include a small self-recorded dataset. Due to time constraints, the gestures were only recorded on the basis of one test person, which is why the expected results are not to be regarded as a final solution, but rather as a proof-of-principle.

For this purpose, two dataset creation tools were programmed in this project:

Video Generator

The video generator was programmed to optimize the acquisition of training videos. The letter can be selected using the keyboard. The advantage of this program is that the image files automatically get saved to the right folder.

Figure 5.2: Video sequence generator tool

For each letter, a total sequence length of about 120s of only one test person was recorded, which builds the basis for skeleton extraction in the next processing step and therefore the whole dynamic gesture prototype dataset.

Sequence Processing

All Videos were than processed with the second self-made program. The parts of the video containing specific gestures can be selected graphically or by software. The specified subsequence can then be exported as a skeleton (result of MediaPipe Holistic) by specifying the amount of augmentations (here only rotation and random shift) and the amount of sequences per augmentation. The sequences get selected randomly by creating a random start point in the subsequence and then exporting the desired amount of consecutive frames. After testing, the conclusion was drawn that for comparitavely short gestures (as it is the case for "Z" and "J") a sequence length of 25 frames is ideal.

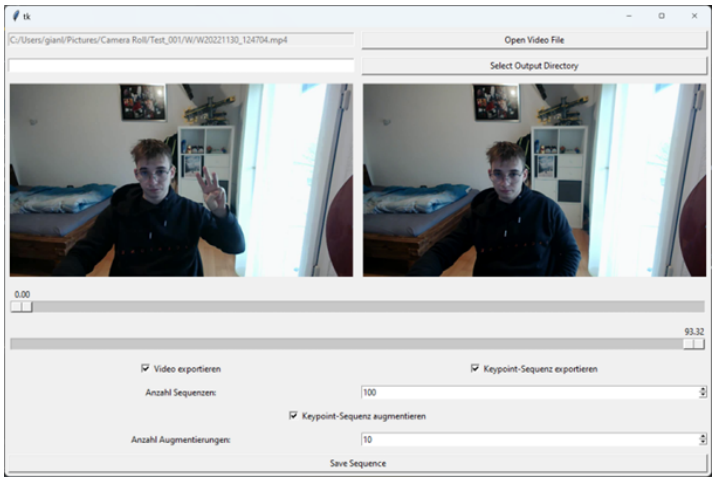


Figure 5.3: Video augmentation tool

5.3 Network Structure

The network structure was adjusted and modified continuously during testing. The best results were observed with a structure as shown in figure 5.4 (Batch normalization and dropout layers not shown). The structure is similiar to the static skip connections model with three additional LSTMs, which separately process left hand, right hand and body keypoints. The proposed structure results in a total of $62.9k$ trainable parameters.

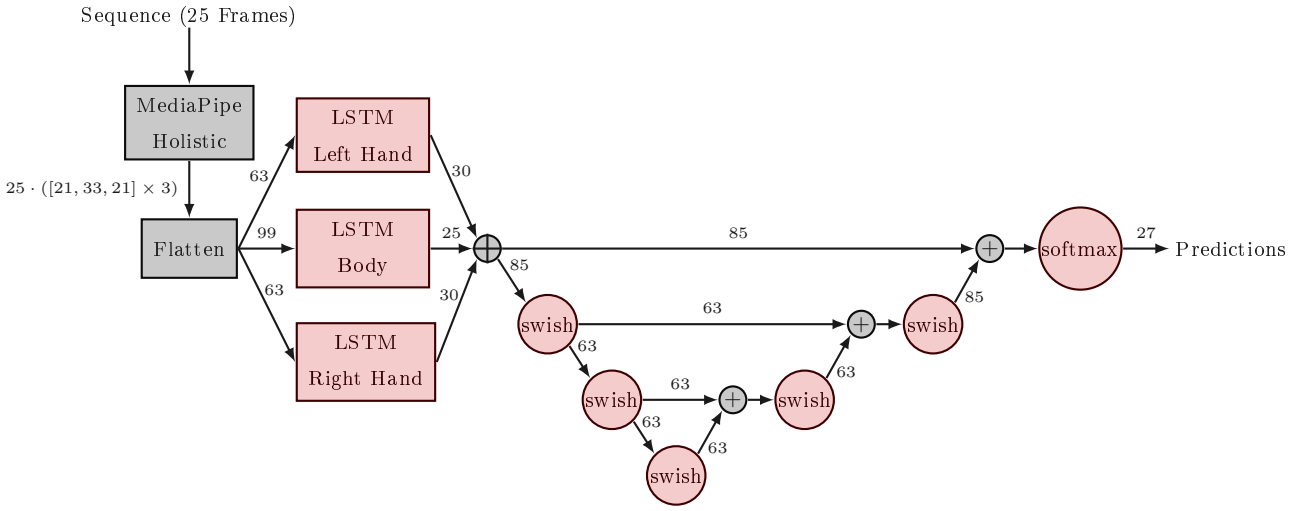


Figure 5.4: Network structure with LSTM blocks ("+" = vector addition, $\oplus$ = concatenation), full structure in appendix A.3

5.4 Accuracy, Perfomance and Limitations

As shown in figure 5.5, the maximum validation accuracy with the proposed network was 83.7% after 150 training epochs. Training on an RTX 3090 GPU took approximately $15min$ for a dataset with about $2k$ sequences per gesture. The network shows comparable performance to the unaugmented dataset of the static model, which is remarkable considering that the dataset consist of sequences recorded by only one person. Extending the training data by recording data in different environments, from different angles, and based on multiple test subjects could optimize the accuracy similar to the improvement shown in the static model (section 4.4).

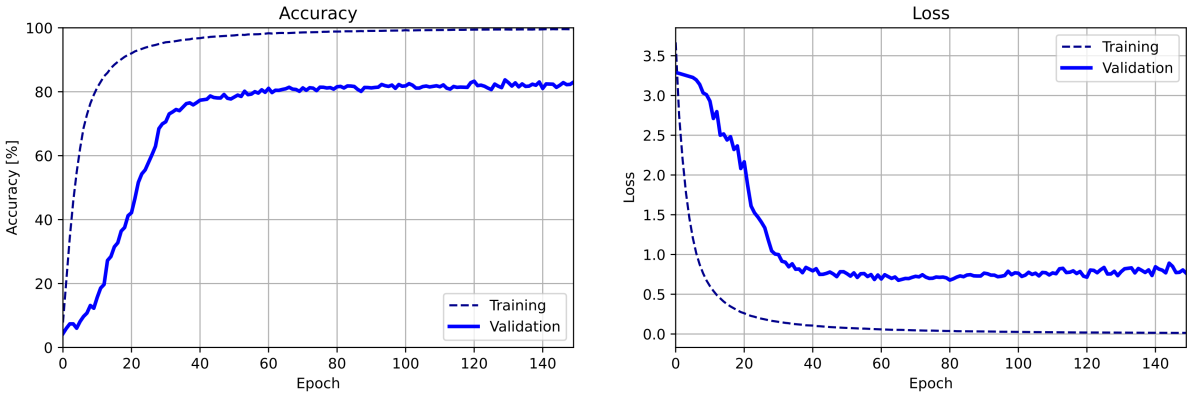


Figure 5.5: Accuracy and loss for dynamic gesture recognition model

Although the Confusion Matrix (figure 5.6) clearly shows generally worse accuracy than the static model, it is noteworthy that the gestures "Z" and "J" (dynamic letters) are clearly distinguishable from their static relatives "D" and "I", thus indicating the inclusion to the temporal information by the LSTM network. In contrast to the static model, the prediction gets stuck at the last performed gesture if no gesture is shown. To solve this problem, "no gesture" (= "0") was additionally trained as the 27th class.

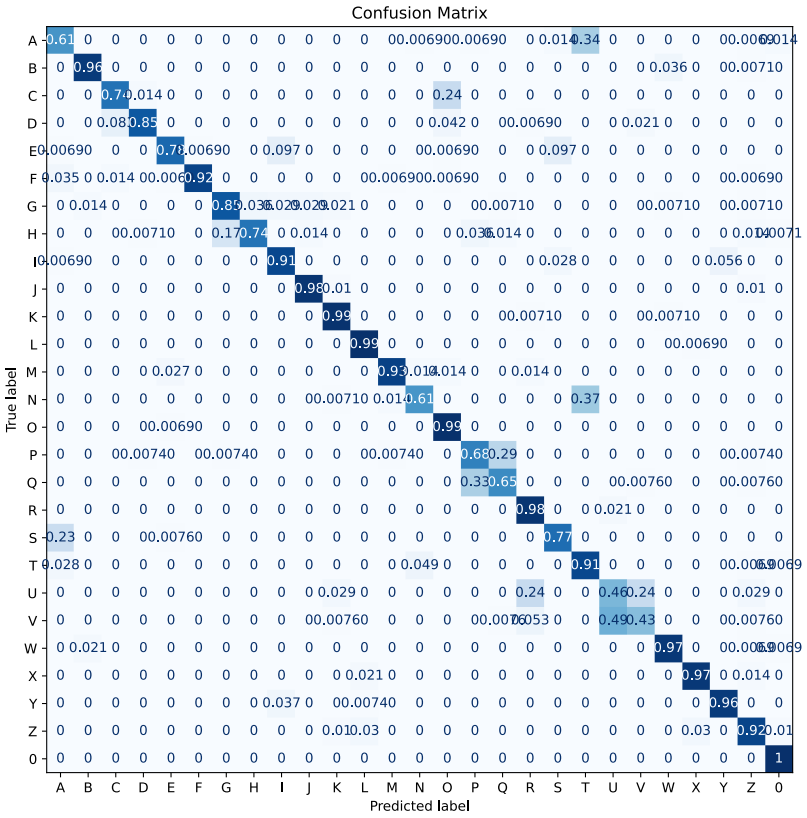


Figure 5.6: Confusion matrix of dynamic gesture dataset

Chapter 6

Edge-Device Implementation in iOS-Application using CoreML

As a continuation of the proof-of-principle, the model for the recognition of static gestures (MediaPipe Hands and the skip connection classifier) was implemented in an iOS app. The camera of the iOS device is the input for MediaPipe. The output of MediaPipe (hand landmarks) should be displayed in the camera frame on the display and serve as input for the classifier predicting the corresponding letter which then also is visualized.

The iOS devices used are an Apple iPhone X (2017) and an Apple iPhone 14 Pro (2022). Programming is done in Xcode version 14.1 on an Apple Mac mini (2014).

6.1 Implementation of MediaPipe in XCode

The first step of the integration is made up by the implementation of the MediaPipe Hands solution. To achieve this, the following two different approaches were examined.

6.1.1 First Approach: Use of the MediaPipe Example Program

In the first approach, the implementation on the iPhone was tried using the example program provided by MediaPipe. The instructions on the MediaPipe website were used for this. Due to versioning problems, however, the instructions could not be applied step by step. Intensive internet research was necessary to solve several problems that arose. In order to load the example programs onto the end devices, additionally an Apple developer account had to be created to generate necessary provisioning profiles. The example programs were programmed in Objective-C, which is why the integration of the classifier should preferably also be done in this programming language.

After intensive trial and testing, the decision was made that the implementation in this way is too cumbersome and unhandy. The example is programmed in a complicated way, there are a lot of dependencies and the project structure is too complicated because all sample programs use the same view controller. Since the integration should be done by beginners in iOS application development, an alternative was sought.

6.1.2 Second Approach: Example Program with a MediaPipe Framework

The second variant uses a framework approach to implement MediaPipe into an iOS application. This framework has a very narrow interface which makes it easy to integrate and existing example programs for framework integration simplify the task. An additional advantage is that this solution uses the more comprehensible Swift programming language. The example program is also very easy to understand, which makes it easy to implement the classifier later.

6.2 Classifier Implementation using Core ML

After successfully setting up the MediaPipe environment, the nextstep is the integration of an exported Tensorflow model (in this case the skip connection classifier) into an iOS application.

From Tensorflow Model to Core ML model

With a self-made converter script, the exported Tensorflow model (.h5 file) is converted to a Core ML (.mlmodel file) model so that it can be integrated into an Xcode project. Core ML models are extremely efficient and convince with small inference times because the model structure is optimized specifically for the CPU, GPU and neural engine of Apple's own processor series. When implemented in the project structure, a compiled version of the Core ML model (.mlmodelc file) is created as well as an automatically generated interface Swift file. The Swift file now contains all functions to integrate the classifer into the application. It should not be modified.

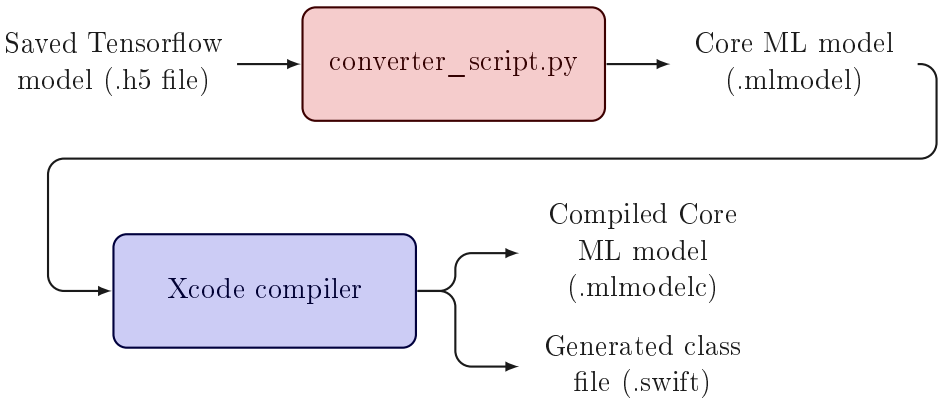


Figure 6.1: Basic steps to integrate a Tensorflow model via Core ML in an Xcode project.

Converter Script in more Detail

In the converter script, the paths to the Tensorflow model and to the destination folder of the resulting Core ML model have to be specified. The script can then be started and the following processing steps are executed sequentially:

1. The Tensorflow model is loaded from the input path.
2. The Tensorflow model is converted to a Core ML model using the Core ML library.
3. Two test predictions based on hardcoded data are compared with the target values to verify and test the functionality of the Core ML model.
4. The generated and tested Core ML model is stored at the output path.

At each of the above steps, the program will abort with an error if the respective step was not successful.

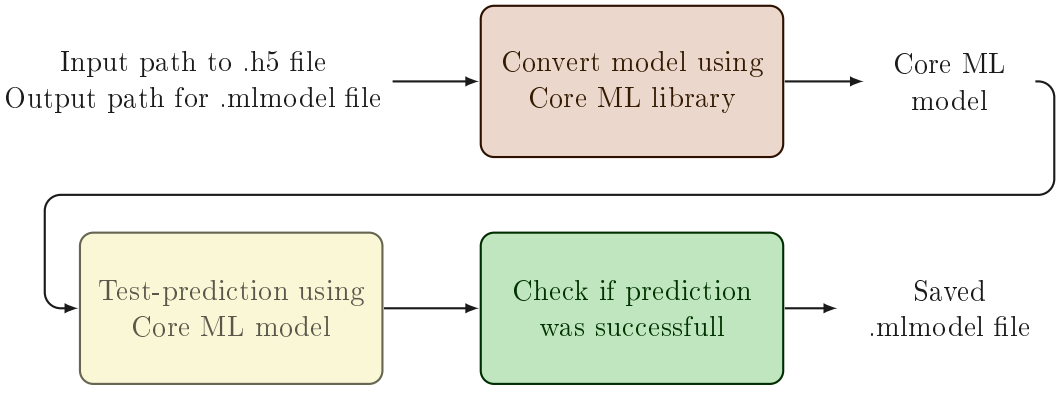


Figure 6.2: Steps of the converter script to convert a Tensorflow model to a Core ML model.

6.3 Performance compared to Desktop Version

To compare the performance of desktop version and iOS-Application, an extensive functional test was carried out. For each test device (iPhone 14 Pro, iPhone X and Surface Book 3 with an x86 processor) the average processing time calculated over 200 measurements was tested for three different gestures. The results in figure 6.3 show that the iPhone 14 Pro is about 100 times faster than the Surface Book 3 with less processor load. The older iPhone X is three times slower than the iPhone 14 Pro, mainly because the processor technology doesn't support the Core ML neural engine optimization and the clock frequency of the iPhone 14 processor is about 1.5 times higher.

Reasons for the significant acceleration compared to the desktop app could be the following:

- iPhone Processors newer than the A11 Bionic generation use the mentioned neural engine cores, whose internal structures are specifically designed and optimized for fast matrix multiplications and floating-point processing as used in neural networks.
- Core ML models are optimized for the neural engine of the iPhone.
- Python is interpreted at runtime and not compiled to native code and therefore comparably slow.

The time per frame is 33 milliseconds on average for both iPhones because the camera setting is fixed at 30 FPS in the programming of the application.

	iPhone X	iPhone 14 Pro	Surface Book 3
Production year	2017	2022	2020
Processor	A11 Bionic (ARM)	A16 Bionic (ARM)	i7-1065G7 (x86)
Classifier inference A	2.357 ms	0.780 ms	69 ms
Classifier inference O	2.277 ms	0.847 ms	72 ms
Classifier inference V	2.033 ms	0.725 ms	70 ms
Time per frame	33 ms	33 ms	120 ms
CPU utilization	10%	8%	40%

Figure 6.3: Time comparison of classifer prediction between different devices. The fastest device (iPhone 14 Pro) is approximately 100 times faster than the slowest (Surface Book 3).

6.4 Results of Functional Tests

Figure 6.4 shows the finished application running on an iPhone 14 Pro. The same three gestures are shown that were used for the measurements in section 6.3.

The MediaPipe results are visualized directly in the image and can be disabled (as shown in the screenshot on the right) with the toggle button in the lower part of the screen. The handedness (in this case the right hand) is displayed at the top of the screen. Additionally, the hand is framed with a square and the hand landmarks are drawn at the corresponding image position.

The processing time per frame as well as the inference of the classifier itself is displayed in the upper middle part of the GUI for debugging and testing purposes.

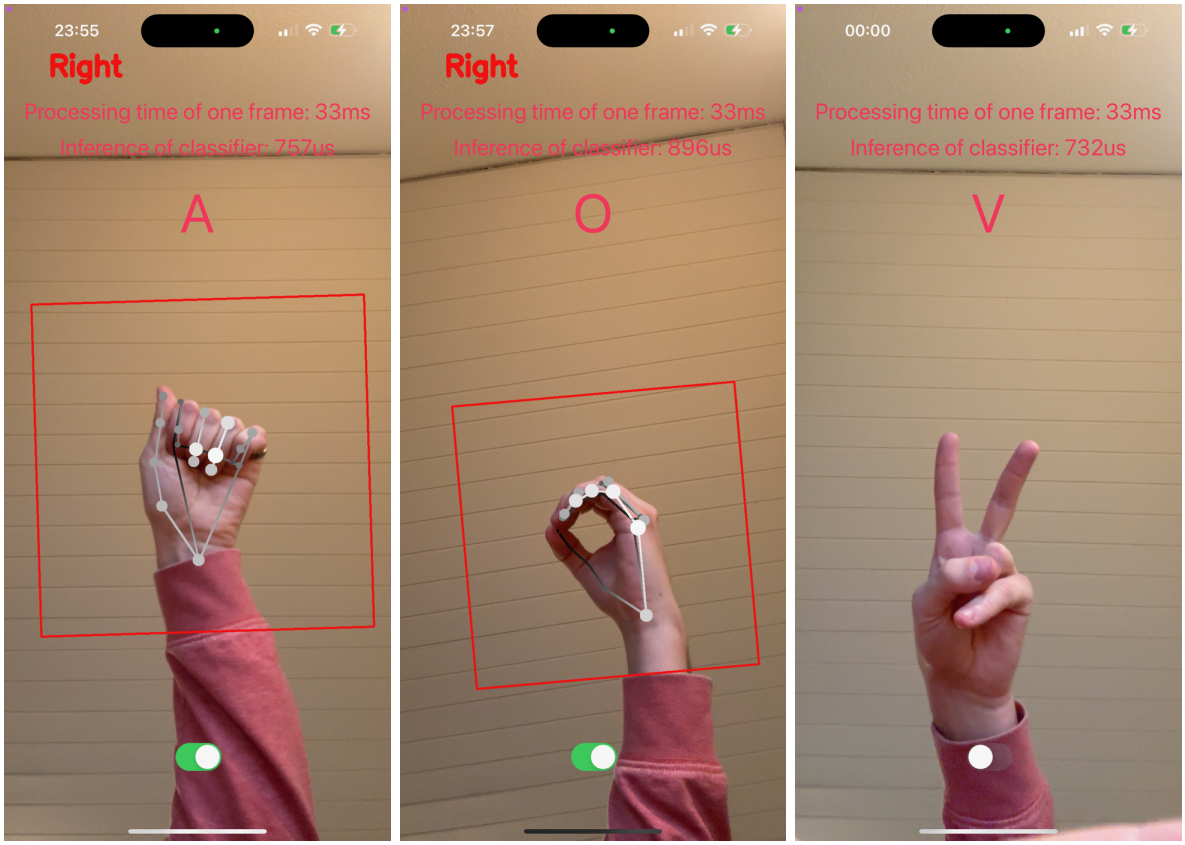


Figure 6.4: Screenshots of three different gestures captured in the iOS application

Chapter 7

Discussion and Conclusion

The three main parts of this thesis show the proof-of-principle for sign language recognition based on model-based feature extraction as available in the MediaPipe modules. With an accuracy of 94.4% percent, the static model trained with a diverse and augmented dataset achieves very good results and the implementation of the static model in a smartphone application shows the real-time potential of the approach. Although the dynamic model can still be improved due to a small dataset of only one person, the results dubiously demonstrate the potential of LSTM network structures for learning dynamic gestures. This approach is not limited by its principle, but rather by the relatively high effort required to create a meaningful and effective dataset.

7.1 Optimization approaches

The potential for extending this work is great. The dynamic model can be optimized with the acquisition of a diversified data set (several persons, several viewing angles, different light and background conditions). Also conceivable would be an implementation of gestures, where both hands simultaneously and facial expressions play a role. The model thus can be extended to detect virtually any gesture.

In a further step, the iOS app could also be converted to dynamic gesture recognition. this would require the implementation of the MediaPipe Holistic Model and the dynamic classifier with LSTM structure. In combination with sentiment analysis it would even be possible to spell out complete sentences and texts. Because the implementation of the MediaPipe modules has turned out to be demanding and cumbersome, it would additionally be worth to consider including MediaPipe as a Core ML model. These models are optimized for the neural engine processors in the newer iOS devices and therefore have extremely short inference times in relation to the Python applications.

7.2 Application areas

Even though this work only researched the proposed solutions on the use-case of ASL recognition, the approach itself can of course be adapted for other problems requiring gesture recognition.

High-performing VR applications as of today still use sensor-based approaches to detect hand and body movement. The proposed solutions could be adapted to make such unhandy and unrealistic VR controllers redundant.

Furthermore, the solution can be applied for general human-computer interaction as an alternative input device. Dynamic solutions allow the implementation of drag and drop operations. This would also be exceptionally useful in a medical, sterile environment, where gesture control would help during operations to show patient information without touching a potentially contaminated keyboard or mouse.

Bibliography

[1] D. E. Rumelhart, G. E. Hinton and R. J. Williams, „Learning representations by back-propagating errors", *Nature*, vol. 323, no. 6088, pp. 533–536, Oct 1986.

[2] Elite Data Science. (2022, July) *Overfitting in Machine Learning: What It Is and How to Prevent It.* [Online]. URL: https://elitedatascience.com/overfitting-in-machine-learning [Retrieved: Dec 19, 2022].

[3] G. Huang, Z. Liu and K. Q. Weinberger, „Densely Connected Convolutional Networks", *CoRR*, vol. abs/1608.06993, 2016.

[4] K. He, X. Zhang, S. Ren and J. Sun, „Deep Residual Learning for Image Recognition", *CoRR*, vol. abs/1512.03385, 2015.

[5] O. Ronneberger, P. Fischer and T. Brox, „U-Net: Convolutional Networks for Biomedical Image Segmentation", *CoRR*, vol. abs/1505.04597, 2015.

[6] H. Li, Z. Xu, G. Taylor and T. Goldstein, „Visualizing the Loss Landscape of Neural Nets", *CoRR*, vol. abs/1712.09913, 2017.

[7] I. Adeyanju, O. Bello and M. Adegboye, „Machine learning methods for sign language recognition: A critical review and analysis", *Intelligent Systems with Applications*, vol. 12, p. 200056, 2021.

[8] T. Chatzis, A. Stergioulas, D. Konstantinidis, K. Dimitropoulos and P. Daras, „A Comprehensive Study on Deep Learning-Based 3D Hand Pose Estimation Methods", *Applied Sciences*, vol. 10, no. 19, 2020.

[9] M. Oudah, A. Al-Naji and J. Chahl, „Hand Gesture Recognition Based on Computer Vision: A Review of Techniques", *Journal of Imaging*, vol. 6, no. 8, 2020.

[10] S. Garg, A. Saxena and R. Gupta, „Yoga pose classification: a CNN and MediaPipe inspired deep learning approach for real-world application", *Journal of Ambient Intelligence and Humanized Computing*, Jun 2022.

[11] C.-H. Yoo, S. Ji, Y.-G. Shin, S.-W. Kim and S.-J. Ko, „Fast and Accurate 3D Hand Pose Estimation via Recurrent Neural Network for Capturing Hand Articulations", *IEEE Access*, vol. 8, pp. 114 010–114 019, 2020.

[12] M. Oberweger and V. Lepetit, „DeepPrior++: Improving Fast and Accurate 3D Hand Pose Estimation", in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[13] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang and M. Grundmann, „MediaPipe Hands: On-device Real-time Hand Tracking", 2020.

[14] Z. Cao, G. Hidalgo, T. Simon, S. Wei and Y. Sheikh, „OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 01, pp. 172–186, Jan 2021.

[15] H.-S. Fang, S. Xie, Y.-W. Tai and C. Lu, „RMPE: Regional Multi-person Pose Estimation", Oct 2017, pp. 2353–2362.

[16] P. Radzki. (2022, Apr) *Detection of human body landmarks - MediaPipe and OpenPose comparison*. [Online]. URL: https://www.hearai.pl/post/14-openpose/ [Retrieved: Oct 8, 2022].

[17] Google LLC. (2022, Dec) *MediaPipe - Website*. [Online]. URL: https://google.github.io/mediapipe/ [Retrieved: Nov 25, 2022].

[18] C. Gong, Y. Zhang, Y. Wei, X. Du, L. Su and Z. Weng, „Multicow pose estimation based on keypoint extraction", *PLOS ONE*, vol. 17, no. 6, pp. 1–18, Jun 2022.

[19] S. K. Yadav, A. Singh, A. Gupta and J. L. Raheja, „Real-time Yoga recognition using deep learning", *Neural Computing and Applications*, vol. 31, no. 12, pp. 9349–9361, Dec 2019.

[20] D. Swain, S. Satapathy, P. Patro and A. K. Sahu, „Yoga Pose Monitoring System using Deep Learning", Jun 2022.

[21] N.-H. Nguyen, T.-D.-T. Phan, S.-H. Kim, H.-J. Yang and G.-S. Lee, „3D Skeletal Joints-Based Hand Gesture Spotting and Classification", *Applied Sciences*, vol. 11, no. 10, 2021.

[22] S. Agrawal, A. Chakraborty and M. Rajalakshmi, „Real-Time Hand Gesture Recognition System Using MediaPipe and LSTM", *International Journal of Research Publication and Reviews*, vol. 3, no. 4, pp. 2509–2515, Apr 2022.

[23] Akash. *ASL Alphabet*. [Online]. URL: https://www.kaggle.com/datasets/grassknoted/asl-alphabet [Retrieved: Oct 9, 2022].

[24] *ASLLRP Sign Bank*. [Online]. URL: http://dai.cs.rutgers.edu/dai/s/signbank [Retrieved: Oct 9, 2022].

[25] D. Li, C. R. Opazo, X. Yu and H. Li, „Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison", *ArXiv*, 2019.

[26] A. L. C. Barczak, N. H. Reyes, M. E. Abastillas, A. Piccio and T. Susnjak, „A New 2D Static Hand Gesture Colour Image Dataset for ASL Gestures", 2011.

[27] R. Farrapo Pinto Junior and I. Cavalvante de Paula Junior, „Static Hand Gesture ASL Dataset", 2019.

[28] N. Pugeault and R. Bowden, „Spelling It Out: Real–Time ASL Fingerspelling Recognition", in *1st IEEE Workshop on Consumer Depth Cameras for Computer Vision*, 2011.

[29] V. Athitsos, C. Neidle, S. Sclaroff, J. Nash, A. Stefan, Q. Yuan and A. Thangali, „The American Sign Language Lexicon Video Dataset", in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2008, pp. 1–8.

[30] C. Shorten and T. M. Khoshgoftaar, „A survey on Image Data Augmentation for Deep Learning", *Journal of Big Data*, vol. 6, no. 1, p. 60, Jul 2019.

[31] S.-E. Wei, V. Ramakrishna, T. Kanade and Y. Sheikh, „Convolutional Pose Machines", 2016.

[32] T. Simon, H. Joo, I. Matthews and Y. Sheikh, „Hand Keypoint Detection in Single Images Using Multiview Bootstrapping", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4645–4653, 2017.

[33] V. J. Schmalz, „Real-time Italian Sign Language Recognition with Deep Learning", 2021.

Appendix A

Appendix

A.1 Task

Projektarbeit *PA22_loma_2*

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering

BA-Studenten:	Gianluca Pargätzi (pargagian) Michael Wäspe (waespmic)	Betreuer:	Dr. Martin Loeser Tobias Welti
Industriepartner:	-		
Ausgabetermin:	Montag, 19. September 2022	Abgabetermin:	Montag, 19. Dezember 2022
Anzahl Credits:	6 (ca. 180h Arbeitsaufwand)	Präsentation:	Form und Zeitpunkt folgen später

Thema: AI-Based Translation of Sign Language

1. Hintergrund und Zielsetzung

Gebärdensprache ist ein wichtiges Kommunikations-Tool für Personen mit Hörbeeinträchtigung. Sie stützt sich sehr stark auf Mimik und Gestik und muss – wie jede Fremdsprache – über einen längeren Zeitraum erlernt werden. Ziel dieser Arbeit ist es, mit Hilfe eines Artificial Intelligence-basierten Systems einfache Gesten automatisch zu erkennen und zu übersetzen. Das zu entwickelnde System soll dabei so gestaltet sein, dass es in Echtzeit auf einem Computersystem mit beschränkten Ressourcen läuft, beispielsweise auf einem Raspberry Pi oder einem Smartphone.

2. Aufgabenstellung

Die Aufgabenstellung gliedert sich dabei in einen Pflicht-Anteil, der auf jeden Fall zu bearbeiten ist, und einen erweiterten Teil, der sich nach den Interessen der Studierenden und dem Fortschritt der Arbeit richtet.

- **Minimal-Teil**
  - In einem ersten Schritt soll ein geeigneter Satz an Trainingsdaten erstellt werden. Dieser Datensatz soll aus kurzen Video-Sequenzen bestehen, die jeweils genau eine Geste bzw. einen Ausdruck zeigen. Jedes Video trägt dabei ein eindeutiges Label, das die gezeigte Geste klassifiziert. Dabei sollten Gesten ausgewählt werden, die einzeln bzw. statisch erkennbar sind.
  - Im zweiten Schritt geht es darum, ein geeignetes Neuronales Netz zu entwickeln und mit dem zuvor generierten Datensatz zu trainieren. Hier soll Python-Code erstellt werden, der auf einem PC lauffähig ist. An dieser Stelle ist noch keine Echtzeit-Fähigkeit des Codes gefragt – es ist ausreichend, wenn auf dem PC gespeicherte Videos korrekt klassifiziert werden. Hier liegt der Fokus zunächst auf Einzelbildern bzw. statischen Frames.
  - Publikation der Arbeit mit Sourcecode auf GitHub.

- **Erweiterter Teil**
  - Der letzte Schritt besteht in der Implementierung einer kleinen Demo-Applikation, die die Echtzeit-Lauffähigkeit zeigt.
  - Mögliche Optionen sind die Nutzung einer Webcam in Kombination mit einem PC, oder die Implementierung einer entsprechenden App auf einem Smartphone (Android oder iOS).
  - Umstieg von Einzelframes auf sequentielle Daten (Video-Sequenzen).

3. Bericht/Präsentation/Randbedingungen

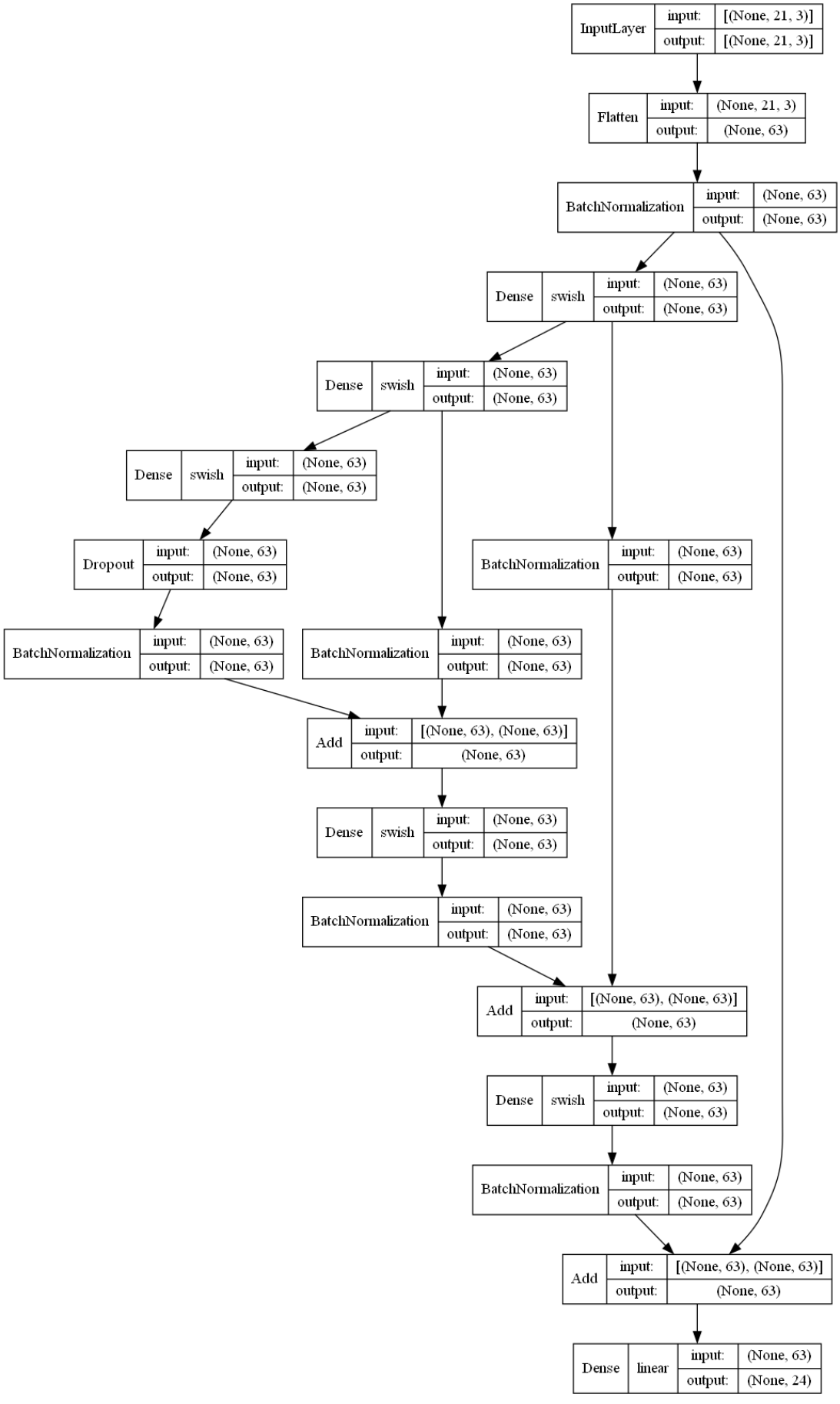
- Sie übernehmen die Leitung des Projektes. Sie organisieren, leiten und treffen alle nötigen Massnahmen, um das Projekt zu einem erfolgreichen Abschluss zu führen.
- In der Regel findet eine wöchentliche Besprechung statt. Rechtzeitig vor der Besprechung ist wöchentlich ein kurzer Report per Mail an die Betreuer zu senden, mit präzisen Angaben zu den Berichtspunkten „**diese Woche gemacht**“, „**für nächste Woche geplant**“, „**Probleme**“, etc.
- Führen Sie, in Absprache mit den Betreuern, eine Meilenstein-Besprechung mit allen Beteiligten durch. Zu der Meilensteinsitzung gehören eine formelle Einladung mit Traktandenliste und ein Protokoll, welches den Beteiligten rasch zugesendet wird und dann im Anhang des Berichtes erscheint.
- Über die Projektarbeit ist ein Bericht zu schreiben. Der Bericht ist termingerecht abzugeben. Es gelten die üblichen Dokumente zum Ablauf und zur Form (Leitfaden_PABA.pdf, Zitierleitfaden, etc.).

4. Bewertungskriterien, Gewichtung:
Projektverlauf, Leistung, Arbeitsverhalten ca. ⅓; Qualität der Ergebnisse ca. ⅓; Form und Inhalt des Berichts ca. ⅓.

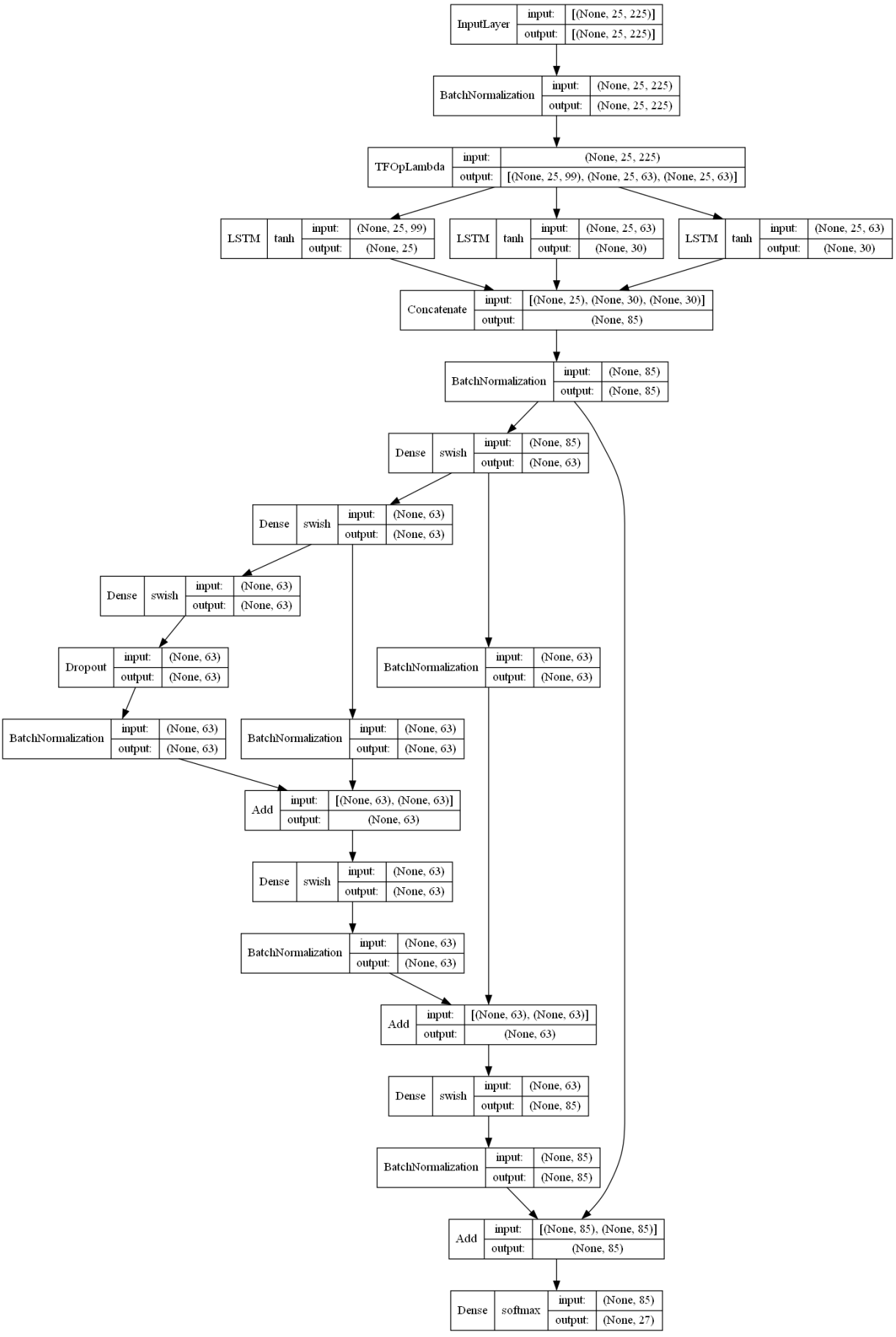
5. Literaturverzeichnis
Fachliteratur zum Thema wird von den Betreuern zur Verfügung gestellt bzw. von den Studierenden recherchiert.

2

A.2 Full network structure of static model



A.3 Full network structure of dynamic model



A.4 Code and models

All code files and the trained models can be found in the github repository "https://github.com/gianlucapargaetzi/AI-Based-Sign-Language-Recognition". The datasets are not available online because of the datasize. If necessary, the datasets can be requested via E-Mail to "gianluca.pargaetzi@parmail.ch" or "michi.waespe@bluewin.ch".

A.5 Project management

All protocols of the weekly project meetings and milestones and the timetable as PDF can be found in the attached "Appendix.zip" folder.