American Sign Langugage Recognition using DeepLearning

Felix Schlatter
CAS Advance Machine Learning
University of Bern
felix.schlatter@bluemail.ch

Asad Bin Imtiaz
CAS Advance Machine Learning
University of Bern
asad.imtiaz@unibe.ch

***Abstract* – This study presents the research conducted during the Certificate of Advanced Studies (CAS) in Advanced Machine Learning program of 2022 at the University of Bern, focusing on the development of an intricate approach for the recognition of distinct signs in American Sign Language (ASL). The work is inspired from the 2023 Kaggle Isolated Sign Language Recognition challenge by Google** [1]**. The cornerstone of this project is building a robust and deep-learning framework-agnostic setup that can be adapted to train models in either TensorFlow or PyTorch, wherein exhaustive testing with numerous models to identify one among 250 unique signs in short video sequences supplied by Kaggle. The highest-performing models yielded an overall test accuracy of approximately 80%. The design of this project framework is modular, making it a viable reference for future endeavours in diverse deep learning tasks.**

Keywords—Sign Language Recognition, American Sign Language (ASL), Kaggle Isolated Sign Language Recognition challenge, Deep learning

Keywords—Sign Language Recognition

I. INTRODUCTION

In the realm of communication, language is a key element that binds our society together. This is particularly true for the deaf and hard-of-hearing community, for whom American Sign Language (ASL) is their primary language. The American Sign Language is a natural language used primarily by deaf and hard-of-hearing individuals in the United States and parts of Canada [2]. ASL is a visual-gestural language that utilizes handshapes, facial expressions, body movements, and space to convey meaning. It has its own grammar and syntax separate from spoken languages like English. Learning the language is a challenging task and is taking up the time and resources of the learners. In the United States, up to 3 of 1000 infants are born with hearing loss, which makes this the most common birth defect [3]. This birth defect is basically not hereditary and can therefore affect all children. To be able to communicate with the child, as well as to teach the child a way of communication, the parents must acquire another language (sign language). To learn the language there are different methods and approaches. In addition to books, online media, there is also the possibility to experience the language in a playful way. PopSign is a provider, which allows users to learn sign language in a playful way by means of a game [4].

For the development of a game to facilitate learning sign language, Google has partnered with the Georgia Institute of Technology, the National Technical Institute for the Deaf at Rochester Institute of Technology, and Deaf Professional Arts Network to compile a comprehensive data set so that a model can be developed to help learn the language [5].

In response to these challenges, we developed machine learning models capable of recognizing and classifying isolated ASL signs. Unlike previous attempts [6], our current models have been developed using both TensorFlow and PyTorch frameworks, demonstrating our commitment to the utilization of state-of-the-art technologies. We have implemented a deep learning framework agnostic trainer, which allows us to train these models and document the results in a flexible, efficient manner. Our aim was to provide a more effective tool for ASL recognition and learning, creating a more inclusive environment for deaf children and their families, thereby improving communication, and reducing the potential impacts of Language Deprivation Syndrome.

II. SCOPE OF THE PROJECT

The objectives of this project were dual pronged. The first aim was to leverage the power of Deep Learning to extract valuable insights from an existing dataset. Concurrently, we strived to avoid binding our project to a single Deep Learning framework. Our approach was focused on establishing a versatile boilerplate setup conducive for the development of models using both PyTorch and TensorFlow frameworks. This strategy enabled us to delve

into and exploit the unique capabilities and subtleties of each platform, ensuring our work was framework-agnostic and compatible with the most widely used Deep Learning technologies.

By implementing the boilerplate setup, we aimed to streamline the process of deploying Deep Learning models in real-world scenarios, regardless of the framework preference of the end-users. The goal was to provide a unified and easily adaptable codebase that could be utilized by others working with Deep Learning models, simplifying the transition from development to deployment. Through this project's scope, we sought to advance our understanding of Deep Learning concepts, maximize the potential of the dataset, and contribute to the broader Deep Learning community by providing a practical solution for model deployment using PyTorch and TensorFlow.

In the spirit of creating a user-friendly and dynamic setup, we centralized the control over various inputs through the *config.py* file. This configuration file offers a one-stop solution for defining and tweaking project-specific parameters, settings, and variables. Users can readily customize the project's behavior and characteristics by simply adjusting the values in the *config.py* file, thereby eliminating the need to directly modify the code.

The project involved rigorous data preprocessing, cleaning, and augmentation to tailor the dataset for the specific use case. The setup was designed to be scalable, capable of handling larger datasets or more complex models. It was built with a view to accommodate future developments in deep learning and sign language recognition. One key aspect of the scope was to create reusable modules and functions to ensure that parts of the project can be easily used in other similar tasks, thereby reducing future development time. Thereby we entailed detailed documentation to ensure that other researchers and developers can easily understand and extend the work done.

III. DATASET

The dataset used for this project was provided by Google and hosted by Kaggle and was part of the 2023 challenge *Isolated Sign Language Recognition* [1][3]. This rich and comprehensive dataset has been a joint contribution from Google, the Georgia Institute of Technology, the National Technical Institute for the Deaf at Rochester Institute of Technology, and Deaf Professional Arts Network [5].

The data comprised a total of **54.43** GB of processed data. split into **94'479** unique files containing one of **250** distinct signs. Each sign is performed by different participants, thus capturing a wide variety of signing styles, personal quirks, and potential minor variations in sign performance. This arrangement allows for the capture of a broad spectrum of signing styles, personal idiosyncrasies, and possible minor discrepancies in sign execution. The breadth and depth of this dataset make it an invaluable resource for understanding and modeling sign language patterns.

The videos are preprocessed using the MediaPipe -Holistic pipeline [4] and are thus represented as a series of 543 distinct landmark coordinates (x,y,z). The landmarks comprise coordinates for pose, both hands, and face. This representation not only helps reduce the data complexity but also enables efficient and accurate analysis of the sign language patterns.

Figure 1: Example of landmarks gathered from MediaPipe Holistic Model. Source of the image: MediaPipe github repository[1]

A. Dataset initial collection

The dataset was recorded by American Sign Language signers using ASL as their primary language. In total, **21** different signers collected **94'479** unique files [1]. The data was recorded by the video camera of a Pixel 4a phone with a preinstalled collection app [6]. The app prompted the signers to sign a randomly selected sign out of the **250**-sign vocabulary. The video was recorded, when a button is pressed and held, and the video stopped when the button was released. Each video was afterwards buffered with 0.5 seconds, which matches the game setup.

This dataset underwent a coarse cleaning process post-collection. Thus, the data collection includes data of signers, that may contain files of signers, which made variants of the sign, based on their background and regional dialects. It also contains signs that are fingerspelled, missed completely, or performed incompletely due to early or late

[1] https://www.google.com/imgres?imgurl=https%3A%2F%2Fcamo.githubusercontent.com%2F1aa048b9b75f8c33e201f3ff62a56a4db6995549e8cf717ec807b548c8e0dc42%2F68747470733a2f2f6d6564696170706970652e6465762f696d616765732f6d6f62696c652f686f6c69737469635f73706f7274735f616e645f676573747265735f6578616d706c652e676966&tbnid=MeQsdWZdBV5_kM&vet=12ahUKEwikhOCcm7j_AhWTk6QKHb74BmIQMygAegUIARCwAQ..i&imgrefurl=https%3A%2F%2Fgithub.com%2Fgoogle%2Fmediapipe%2Fblob%2Fmaster%2Fdocs%2Fsolutions%2Fholistic.md&docid=qlg66FUx054XHM&w=712&h=400&q=mediapipe%20holistic&ved=2ahUKEwikhOCcm7j_AhWTk6QKHb74BmIQMygAegUIARCwAQ

release of the recording button. This potential for variation may result in undesired cropping of signs. Moreover, since sign language is influenced by an individual's natural hand preference, the dataset encapsulates the preference of right-handed people to sign with their right hand and vice versa. Another noteworthy aspect of the collected data is the inclusion of non-sign specific features. The video recordings might contain incidental movements such as scratches or itches, which do not belong to the sign language itself. The model, therefore, has the added challenge of distinguishing these artefacts from the actual ASL signs.

B. *Dataset description*

The dataset is relatively more or less evenly distributed. Each signer recorded between **3338** and **4968** unique files out of the 250 sign dataset.

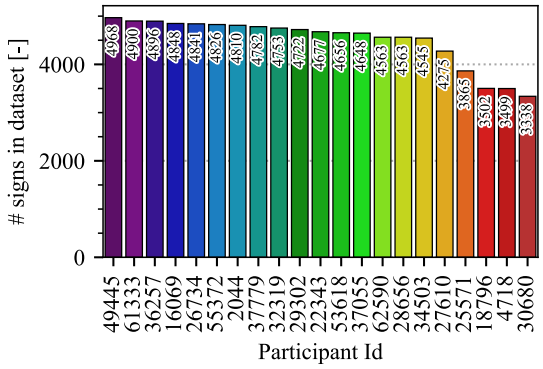

Figure 2: Number of signs per participant.

The sign *zipper* has the least number of files with 299 recordings, whereas the sign *listen* had the most number of files with 415 different recorded signs.

The dataset has already undergone preprocessing using the *MediaPipe Holistic pipeline* [7], transforming the video content into a model-ready format. As such, each file in the dataset represents a sequence of 3D landmark coordinates capturing the signer's pose, hand, and face movements. However, the sequence lengths within the dataset are not uniform. They vary due to factors such as the complexity of different signs and the signing speed of different signers.

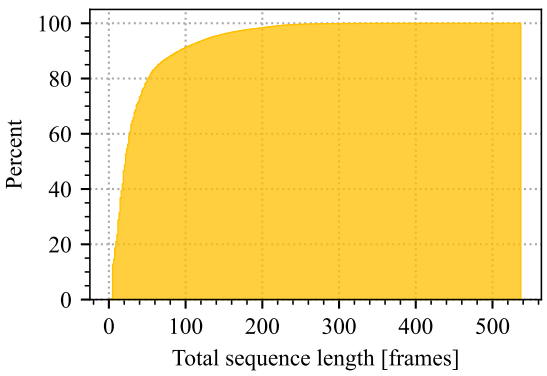

Figure 3: Sequence length in the dataset.

Table 1: Statistics over all signs in the dataset. reveals that, on average, the signs in the dataset consist of 38 frames. This indicates the typical duration of a sign gesture in terms of the number of frames captured.
Among all the signs, the sign "mitten" stands out as the one with the longest duration, averaging 59 frames. This suggests that the sign "mitten" requires a relatively longer time to complete compared to other signs in the dataset.

Table 1: Statistics over all signs in the dataset.

Metric	count	mean	std	min	25 %	50 %	75 %	max
Value	250	37.9	5.2	26.4	34.3	37.8	41.2	58.8

In contrast, the sign "like" exhibits the shortest average duration, spanning just **26** frames. This suggests that the gesture for "like" is relatively brief and requires fewer frames to fully capture its movement. This varied sequence length and the extensive range of signs underscore the richness of the dataset.

C. *Data processing*

The selection of important landmarks from the 543 different landmarks provided by the *Mediapipe Holistic pipeline* was based on the significance of gestures and mimicry in sign language. The focus was primarily on capturing the essential features related to hands, facial expressions (lips), and basic pose features (upper torso). The landmarks chosen were prioritized based on their significance in conveying sign language gestures and facial expressions, two crucial aspects of sign language communication. By considering these specific landmarks, the aim was to capture the most relevant landmarks for gestures and facial expressions that contribute significantly to conveying meaning and facilitating effective communication in sign language.

In total, the model incorporated:

- 40 face landmarks: These primarily focus on key features such as the lips and eyes, which play a significant role in sign language.
- 21 left-hand and 21 right-hand landmarks: These capture the intricate movements of the fingers and palms, which are vital for forming various signs.
- 14 pose landmarks: These help in understanding the context of the signs, especially those that involve the upper body or require certain postures.

Thus, our model focuses on a total of 96 different landmarks, ensuring a comprehensive yet efficient approach to sign language recognition.

During the data preprocessing, the decision was made to omit the z-coordinate (depth information) provided by the *Mediapipe Holistic pipeline* [7]. This decision was based on the potential uncertainty and variability associated with depth estimation. The accuracy and reliability of depth

estimation can vary depending on factors such as lighting conditions, camera angles, and distance from the camera.

By focusing on the 2D landmark information, which provides reliable data on the spatial relationships between different parts of the body, we could capture the essential elements of sign language gestures, facial expressions, and poses. This approach ensures a robust model that can accurately recognize and interpret sign language, even without the depth information.

D. Data cleansing

As mentioned earlier, the data used in the project exhibits varying degrees of quality, and to address this issue, a method for filtering out bad data has been developed. This was done in accordance with several parameters outlined in the configuration file, ensuring that the data processed into the model was of high quality and suited to our specific requirements.

One of the primary issues that needed addressing was the occurrence of missing landmarks in the dataset. These missing values could be due to prediction errors from the *MediaPipe Holistic pipeline* or simply the absence of certain body parts in the video frames. To handle missing landmarks, especially of hands which are highly crucial in sign language recognition, we set a parameter `*SKIP_CONSECUTIVE_ZEROS*` to a value of 4. This implies that any sequence with more than 4 consecutive missing hand landmarks (from either hand) would be deemed unfit and removed from the dataset.

Moreover, the sequences in our dataset had varying lengths, presenting another layer of complexity. To ensure consistency, we established rules for acceptable sequence lengths, guided by the configuration parameter `*MIN_SEQUENCES*` with a value of `*INPUT_SIZE/4*` which is equal to *8*. Specifically, we discarded sequences that were too short, ensuring the remaining sequences were sufficiently detailed for the model to analyze.

Through these stringent quality control measures, we effectively excluded around **21,721** files from our original dataset. This left us with a robust dataset consisting of **72,758** files for model training, validation, and testing.

E. Data preprocessing

As most of the machine learning models need a fixed input shape, some sort of padding/cropping of the variable length sequences had to be performed. In addition, our data was also sparse in terms of missing landmarks. Especially, the hands, which are essential for sign language, are frequently missing in the dataset. Figure 4 shows the distribution of "number of usable frames" in the dataset as a percentage. This corresponds to frames, where at least one hand has coordinates that are not missing. As it is depicted in the figure, the percentage rises very steeply in the range between 0 and 35.

The instances where data contained missing values due to technical issues or errors during the data collection process were interpolated if the gap was up to a maximum of 3 missing values in the data. This is driven by configuration parameter `*INTERPOLATE_MISSING*` The value of 3 was chosen for interpolation considering the dynamics of hand and body movements in sign language. Given the nature of physical movement, it is reasonable to assume that adjacent frames would exhibit similarity in the positions of body landmarks. Therefore, in a sequence of frames, a small gap of 1 to 3 missing values can be reliably interpolated..

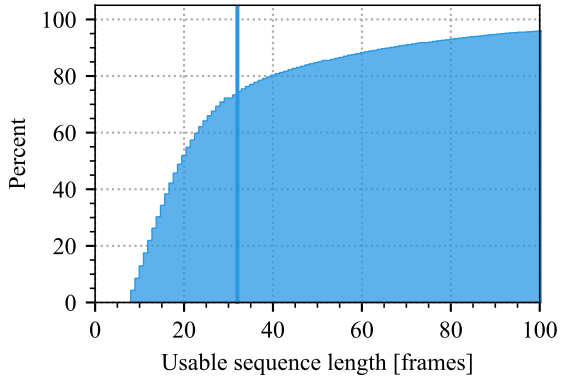

Figure 4: Distribution of usable sequence length in the dataset. Usable in this case refers to frames in the sequence, where at least one of the hands has coordinates.

F. Data augmentations

To increase the robustness of the model and to prevent overfitting, various data augmentation techniques were applied, including:

- **Frame dropout:**
  During this process, a certain number of frames are randomly selected and removed from the sequence of landmarks. Defaults to 5% but can be configured with configuration parameter `*AUG_MAX_THRESHOLD*`. The dropped frames are afterwards 0-padded (at the end of the sequence)

- **Mirror Landmarks:**
  Random mirroring involves flipping the landmarks over a central line, effectively creating a mirror image. This is particularly useful in the context of sign language, as some signs may be performed with either the left or right hand, and so the model should be capable of recognizing these mirrored gestures.

- **Normalization:**
  Normalize the frames with a given mean and standard deviation.

- **Random rotation:**
  Apply random rotation to landmark coordinates. (on **X** and **Y** only) by max angle of up to 10 degrees. It allows the model to recognize gestures regardless of their orientation.

- **Random scaling:**
  Apply random scaling to landmark coordinates between **90**% and **110**%. This can simulate scenarios where the same gesture might be performed closer to or farther from the camera, thus appearing larger or smaller, respectively.

- **Shift Landmarks:**
  This technique involves moving all landmarks by a small, randomly selected amount in any direction by up to 1% of the whole frame.

- **Standardization:**
  Standardize the frames so that they have mean 0 and standard deviation 1.
  Note: Standardization is done on all the sequences regardless.

The data augmentations are included in the dataset-class and applied during runtime in the generator. Thereby, the decision whether to apply augmentation is performed separately for each of the above listed augmentations based on random decision, controlled by the hyperparameter *augmentation_threshold*. The default decision is set to 0.35. With this default setting, only about 10% of the data loaded gets no augmentation.

G. *Data visualizations.*

The importance of effective data visualization cannot be overstated in the context of our study [8]. It plays a critical role in understanding the nuances and complexities of the sign language dataset, besides offering visual insights into the performance and behavior of our models. We have established a visualization routine that aligns a predefined number of signers side by side, converting this alignment into a video for clearer analysis.

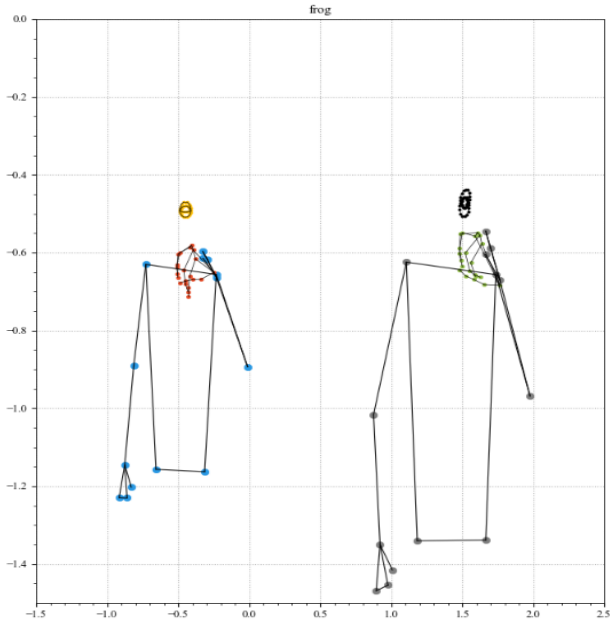

Figure 5: Visualization of a single frame for the sign frog signed by two different signers.

In Figure 5, a sample of one visualized frame is shown for the sign "*frog*" and two signers. This figure underscores the inherent challenges of dealing with the dataset. At the outset, we observe that both participants have a "missing" hand - their right hand isn't visible in the frame. This is because both signers are left-handed, but it's important to note that the same sign could be performed using the right hand or even both hands. This variation poses a significant challenge in accurately interpreting and learning from the signs. Furthermore, certain key landmarks such as fingers may not always be visible throughout the frame. They could be obscured by other body parts or may not fall within the camera's line of sight. These obscured or missing landmarks add another layer of complexity in accurately identifying the sign.

The signers' physical characteristics, including body build, height, organ ratio, and even the usage of lips, vary significantly. This inconsistency presents an additional challenge as these factors can influence the way signs are performed. Another crucial aspect is the standardization of all sequences to 32 frames. While this ensures uniformity in the length of sequences, it can sometimes result in the omission of essential frames. For instance, if there's a delay in the start of the sign or an extended slack at the end, the sign could appear to be performed very rapidly, making it harder to capture and learn from the crucial movements.

Having said that, visualizing multiple signers of the same sign side by side in this manner is an effective way to understand the diversity and variation inherent in sign language expression. Just as in spoken language, individuals express signs differently based on their unique personal style, regional differences, or other factors. Seeing these variations can help in understanding the scope of the problem and how the model should ideally learn to handle these differences.

The side-by-side comparison enables us to observe how different people sign the same concept, which can be quite different in terms of hand shape, movement, location, and non-manual signs. Furthermore, it helped in validating the preprocessing and augmenting did not introduce any distortions that could make the signs harder for the model to learn. This method of visualization allowed us to evaluate the level of complexity of the problem of sign language recognition, underlining that it's not just about learning isolated signs but about understanding a wide range of variations in the way those signs might be performed.

H. *Dataset Splitting and Preparation*

For effective training, validation, and testing of our model, we strategically divided our dataset into distinct segments using different proportions. After several experiments, we found the optimal split to be a 90%/5%/5% distribution for the training, validation, and testing sets, respectively. This distribution offers an ample quantity of data for model training (**65,482** sequences) while also providing a significant amount of unseen data (**3,637** sequences each)

for validation and testing. Stratified sampling helped to ensure that each sign is proportionally represented across the training, validation, and testing sets.

To feed our data into the model during different stages of model development, we utilized data loaders based on our generic dataset class. The choice of data loader is guided by our configuration framework, `*DL_FRAMEWORK*`, which manages the provision of appropriate train, test, and validation data loaders for the deep learning framework in use. This functionality ensures seamless integration of our data with the selected deep learning framework, simplifying the model training, validation, and testing processes.

IV. MODELING

Given that the proposed problem involves classifying sequences of data, specifically sequences of frames with coordinates, it can be considered as a multidimensional sequence classification problem [9]. In such cases, sequence modeling techniques are often well-suited for addressing these challenges effectively. Sequence modeling approaches are designed to capture and understand the dependencies and patterns present in sequential data. These techniques consider the temporal nature of the data and aim to learn representations or models that can effectively capture and exploit the sequential information.

To ensure maximum flexibility and facilitate the implementation and testing of different models, a custom *Trainer class* was developed as a bridge between the TensorFlow and PyTorch deep learning frameworks. This Trainer class serves as an interface for training, validating, and testing models, allowing seamless transitions between the two frameworks. With this Trainer class, the project benefits from a consistent and standardized approach to model training and evaluation, regardless of whether TensorFlow or PyTorch is being used. This class abstracts away the framework-specific details, providing a higher-level interface for working with models, datasets, optimization, and evaluation metrics.

Our initial hypothesis for training with processed fixed sequence data of ASL signs led us to focus on sequence-based models like LSTM and Transformers. These models have shown promising results in handling sequential data, outperforming older models like RNNs in terms of accuracy. While CNNs, such as ResNet-152 used for baseline comparison, have their strengths, they are not inherently designed for sequence data. We then moved to explore an array of model architectures, creating multiple combinations of LSTM and Transformer models and even creating ensembles to maximize accuracy.

Subsequent sections provide an in-depth exploration and discussion of the various models we experimented with, the challenges we faced, and the results we obtained.

A. Baseline Computer-Vision Model (CV)

Our preprocessing routine, detailed in chapter III, shaped the input sequences into a format akin to image data. The sequences took the shape of ***(BATCH_SIZE, 32,96,2***) *for (BATCH_SIZE, SEQ_LEN, Number of landmarks, Number of coordinates)*. This strongly resembles the shape of images. (***…, Height, Width, channels***). Thus, given this resemblance to image data, simple computer vision (CV) techniques were tested as an initial approach to the problem.

One such approach involved utilizing the `***ResNet-152***` architecture. [6]
`***ResNet-152***` is a deep convolutional neural network (CNN) architecture that has achieved remarkable performance in image classification tasks. It consists of multiple layers with residual connections, allowing for the effective training of very deep networks.

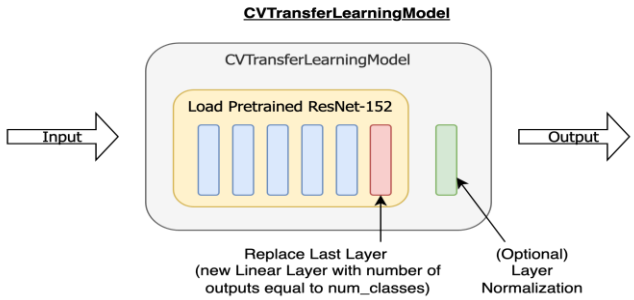

Figure 6: Baseline Computer-Vision Model architecture

By applying the ***ResNet-152*** architecture to the reshaped input sequences, the idea is to profit from the powerful feature extraction capabilities of the resnet152 architecture. By treating each sequence as a "*pseudo-image*" we aim to facilitate the learning of meaningful representations and categorize them accordingly. This approach posits that the ***ResNet-152*** architecture could extract and learn valuable features from these pseudo-images, potentially leading to effective sign classification.

In the transfer learning setting, the last fully connected layer is replaced with a new fully connected layer that matches the number of target classes.

$$L_{new} = Linear(f, c, b)$$

An exponential learning rate scheduler is also initialized, with a decay rate denoted by `gamma`.

$$Optimizer = Adam(B.parameters, \eta)$$
$$Scheduler = ExponentialLR(Optimizer, \gamma)$$

B. TransformerPredictor

The paper "*Attention is All You Need*" [5] introduced the Transformer architecture, which has revolutionized sequence modeling tasks, especially in the field of natural language processing. Leveraging the ideas from this paper,

a Transformer-based approach was developed for the `***ASL-Dataset***`. For this task, only the encoder part of the Transformer architecture was utilized. This is because the encoder-decoder structure of the Transformer is typically used for tasks involving sequence-to-sequence mappings, such as machine translation or text summarization. Since the current task focuses on sequence classification, only the encoder was necessary.

The key aspect of the implementation involved utilizing different Transformer blocks. A Transformer block consists of multiple self-attention layers and feed-forward neural networks, which collectively enable capturing complex relationships and dependencies within the input sequence. By stacking multiple Transformer blocks, the model can effectively learn representations of the input sequence.

The self-attention mechanism, a fundamental component of the Transformer, allows the model to attend to different parts of the input sequence while capturing relevant contextual information. This attention mechanism helps the model focus on important elements or landmarks within the sequence, contributing to better understanding and classification of the sign language gestures.

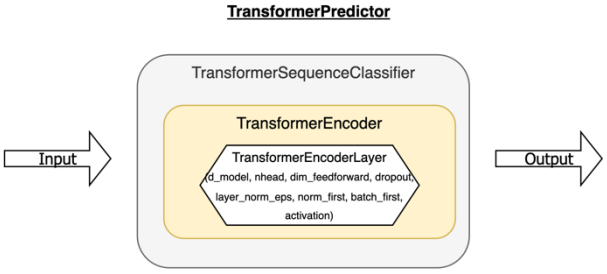

Figure 7: Transformer Predictor architecture

By leveraging the Transformer architecture and experimenting with different configurations of Transformer blocks, the implementation aimed to find an optimal model architecture for the sequence classification problem. The attention mechanism and the hierarchical nature of the Transformer architecture were expected to improve the model's ability to capture long-range dependencies and effectively model the temporal dynamics of the input sequence.

A more specialized transformer based model, TransformerSequenceClassifier, is constructed by extending this base. This classifier utilizes a transformer encoder, which provides attention mechanisms to efficiently handle sequential data. Given a batch of data (*x, y*), where *x* represents the input and *y* represents the corresponding labels, it passes x through the model to get the output ŷ. The loss is calculated using Cross-Entropy loss as follows:

$$L(\hat{y}, y) = -\sum_{i} \quad y\, i \log(\hat{y}_i)$$

C. LSTM-Model

LSTM (Long Short-Term Memory) models are a type of recurrent neural network (RNN) that are widely used for sequence modeling tasks. They are particularly effective for capturing long-term dependencies in sequential data, making them suitable for tasks involving temporal patterns, such as speech recognition, natural language processing, and, in this case, sign language recognition.

In the context of sign language recognition, LSTM models can be employed to process and classify sequences of landmark data over time. Each landmark frame is treated as an input at a specific time step, and the LSTM model learns to extract relevant features from sequences to discern patterns and dependencies across these inputs, thereby facilitating accurate recognition of sign language gestures over time.

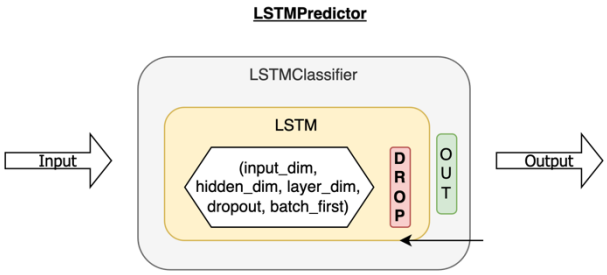

Figure 8: LSTM Predictor architecture

Our `***LSTMPredictor***` is a wrapper around the `***LSTMClassifier***`. It extends our `***BaseModel***` and uses the Adam optimizer and an exponential learning rate scheduler for training. The LSTMClassifier is a sequence classifier based on Long Short-Term Memory (LSTM), a type of recurrent neural network (RNN) that is effective in processing sequence data with long-term dependencies. This model contains three main components: an LSTM layer, a dropout layer for regularization, and a readout layer (a linear transformation).

The loss function used in this context is Cross-Entropy Loss, which can be defined as:

$$L(y, \hat{y}) = -\frac{1}{N}\sum i = 1N = y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

where:

- y_i are true labels
- $\hat{y}_i$ are predicted labels
- N is total observations

D. HybridModel

Our HybridModel class implements a classifier that combines the powers of both Long Short-Term Memory (`***LSTMPredictor***`) as discussed in section C and `***TransformerPredictor***` as described in section B. This is a relatively common practice in machine learning where

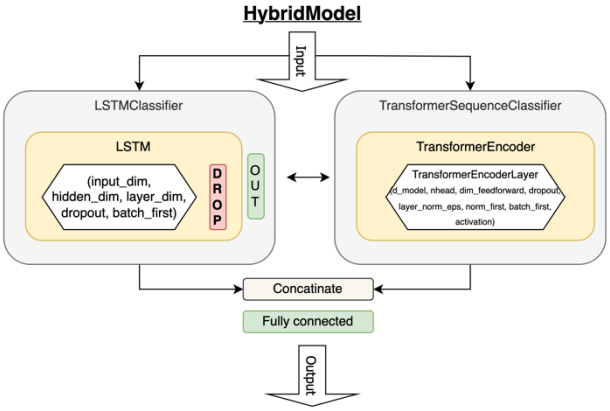

Figure 9: Hybrid Model architecture

different models are used to extract different types of features which are then concatenated and passed through a final layer (usually a fully connected one) for prediction.

The forward pass for the hybrid model is:

1. We pass the input sequence ***x*** through the LSTM model and the Transformer model to get *lstm_output* and *transformer_output* respectively.
2. Concatenate lstm_output and *transformer_output* along the feature dimension to get a combined output.
3. We pass the combined output through a fully-connected (linear) layer *self.fc* to produce the final output.

The training process of the model will involve the standard backpropagation algorithm to compute the gradients of the loss function with respect to the model parameters and then adjust the parameters in the opposite direction of the gradient:

$$\theta = \theta - \eta \forall_\theta L(y, \hat{y})$$

where:

- θ are model parameters
- η is learning rate

E. TransformerEnsemble

Our `**TransformerEnsemble**` class creates an ensemble of our `***TransformerPredictor***` models. An ensemble model combines the predictions of multiple models to improve the overall performance. By using an ensemble of Transformers, the model aims to provide more robust and accurate predictions compared to a single Transformer model.

Each Transformer model in the ensemble is designed with a different number of layers, allowing the ensemble to capture patterns at various levels of complexity. This hierarchical approach ensures that the ensemble model can handle both simple and complex sequence structures.

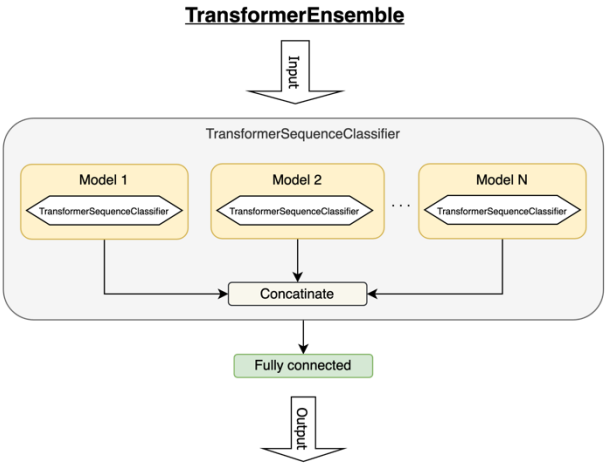

Figure 10: Transfer model architecture

The ensemble aggregates the predictions from each of its constituent models to form the final prediction. This is done by concatenating the outputs of all the Transformer models along the feature dimension and passing this through a fully-connected (linear) layer to produce the final output.

Here's the process:

- For each Transformer model in *self.models*, we pass the input sequence **x** through the model, producing a list of Transformer outputs.

$$T_i = Transformer_i(X)$$

- Concatenate these outputs along the feature dimension to get combined output.

$$T = Concatenate(T_1, T_2, \ldots, T_n)$$

- Pass the combined output through a fully-connected (linear) layer *self.fc* to produce the final output.

$$O = FC(T)$$

F. HybridEnsemble

Our `***HybridEnsembleModel***` class is an extension of the `***HybridModel***` class, combining multiple LSTMs and Transformers into an ensemble of models. An ensemble model combines the predictions of multiple models to improve the overall performance, typically reducing overfitting and improving generalization.

Here's the process:

- For each LSTM model in `*self.lstms*`, pass the input sequence x through the model, producing a list of LSTM outputs. Concatenate these outputs along the feature dimension to get `*l_combined*`.

$$L_i = LSTM_i(X)$$

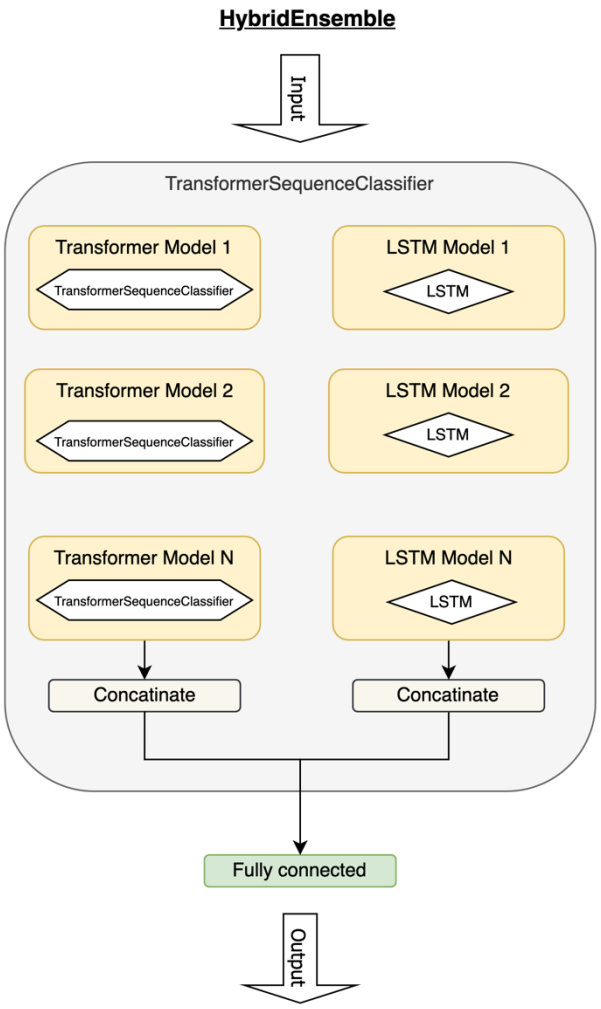

Figure 11: Hybrid Ensemble Model architecture

- For each Transformer model in `*self.models*`, pass the input sequence ***x*** through the model, producing a list of Transformer outputs. Concatenate these outputs along the feature dimension to get `*t_combined*`.

$$T = Concatenate(T_1, T_2, \dots, T_n)$$

- Concatenate `*l_combined*` and `*t_combined*` along the feature dimension to get the overall output.

$$C = Concatenate(L, T)$$

- Pass this output through a fully-connected (linear) layer `*self.fc*` to produce the final output.

$$O = FC(C)$$

G. YetAnotherTransformerClassifier

All of the models we created so far reached a max accuracy of around 70% on the test set. As a step forward, we have aimed to introduce greater flexibility in response to our data. We coded a custom `***MultiHeadSelfAttention***` component which uses casual masking, which helps our model to prevent future information from leaking into the past, ensuring that the prediction for each step is made only based on the past and present data.

Based on this we formulated a transfer block module named `**TransferBlock**` which consists of a self-attention layer and a feed-forward layer, both of which are preceded by layer normalization and followed by dropout for regularization. Finally multiple layers of the `***TransformerBlock***` classes are created for classification in a custom transformer named `***YetAnotherTransformerClassifier***`. This classifier utilizes the multi-headed self-attention mechanism and positional encodings, crucial elements of the transformer architecture, to handle sequence data effectively and produce class probabilities for the sequence classification task.

The `*MultiHeadSelfAttention*` mechanism utilizes multiple attention heads to capture various aspects of the input sequence from different representational spaces. This is done by applying attention in parallel across all heads.

$$MultiHead(Q, K, V) = concat(head_1, \dots, head_h)W^O$$

where:

$$\text{head}_\text{i} = \text{Attention}\left(\text{QW}_\text{i}^\text{Q}, \text{KW}_\text{i}^\text{K}, \text{VW}_\text{i}^\text{V}\right)$$

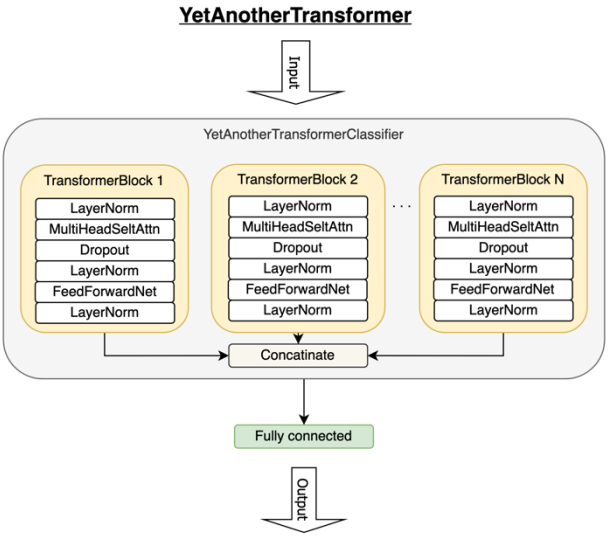

Figure 12: Yet Another Transfer architecture

`***YetAnotherTransformer***` class is a wrapper for the `***YetAnotherTransformerClassifier***` and extends from `***BaseModel***`. It defines the learning rate, optimizer, and scheduler, and places the model on the appropriate device.

H. YetAnotherTransformerEnsemble

The `***YetAnotherEnsemble***` model is an ensemble of `***YetAnotherTransformer***` models. It leverages the power of multiple `***YetAnotherTransformer***` models by training them independently and combining their predictions. This

approach increases the robustness and generalization capabilities of the model by capturing different patterns or aspects from the data.

In the `***YetAnotherEnsemble***` model, multiple layers of `***YetAnotherTransformer***` models are created and stored in a *nn.ModuleList*. Each of these transformer models is designed with different configurations (e.g., varied number of layers), which provide a rich and diverse set of representations of the input data. The output from each transformer model is then concatenated and passed through a final fully-connected layer to produce the final output. The main idea here is that different transformer models may learn and focus on different aspects or features of the data, and by combining them, the ensemble can often achieve better performance.

During the forward pass, each YetAnotherTransformer model processes the input independently and generates its own output. These outputs are then concatenated along the feature dimension, forming a long vector that captures the combined knowledge of all models.

Mathematically, for a single YetAnotherTransformer model's output o_i, the combined output can be represented as:

$$O = [o_1, o_2, \ldots, o_n]$$

where *n* is the total number of models in the ensemble.

The FC layer can be represented as:

$$FC(O) = OW_{fc} + b_{fc}$$

The output of the model is a probability distribution over classes, which is achieved by applying a softmax activation:

$$Softmax(FC(O)_i) = \frac{e^{FC(o)_i}}{\sum_j e^{FC(O)_j}}$$

The ***YetAnotherEnsemble***` is a high performing model reaching accuracy up to 78%. The differing layers in the transformer models provided varied depths of abstraction, thereby enabling the model to learn a broader range of patterns from the training data. This resulted in a model with excellent generalization capabilities, as evidenced by its decent accuracy on the test set. Additionally, the use of ensemble learning helped mitigate issues of overfitting and improved the robustness of the model, further contributing to its performance.

As a final note, the efficacy of the `***YetAnotherEnsemble***` model in our task underlines the value of ensemble techniques and diverse model architectures in tackling complex prediction tasks in machine learning.

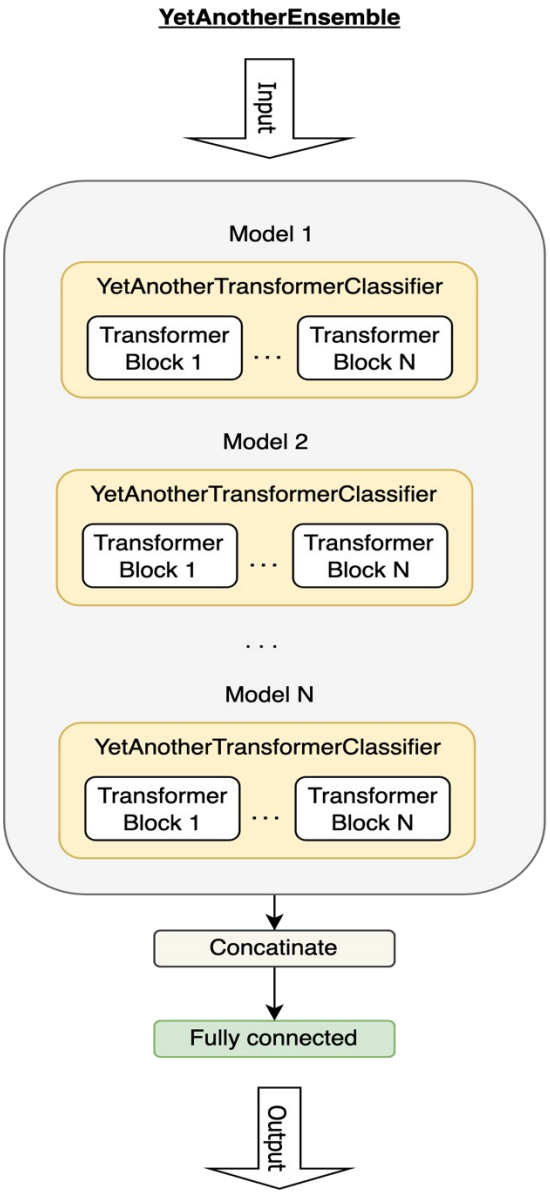

Figure 13: Yet Another Ensemble architecture

V. RESULTS

In our evaluation of the different models, we conducted a series of experiments with a limited scope of hyperparameter optimization. The models were trained using the same settings, including the implementation of EarlyStopping based on accuracy to mitigate the risk of overfitting.

To assess the performance of each model, we utilized the test set, which serves as an independent dataset for unbiased evaluation. By evaluating the models on the test set, we can obtain a reliable estimate of their generalization capabilities and assess their performance in real-world scenarios. This approach ensures that the models' performance is evaluated on unseen data, providing valuable insights into their effectiveness and robustness.

For evaluating the performance of the models, we utilized several test metrics to gain insights into their effectiveness. These metrics include:

- **Accuracy**: It measures the proportion of correctly classified instances, providing an overall assessment of the model's correctness.
- **Loss**: This metric represents the average loss value calculated during the evaluation process. It indicates the extent to which the model's predictions deviate from the actual values.
- **F1 Score**: It is a harmonic mean of precision and recall, which provides a balanced measure of the model's performance by considering both false positives and false negatives.
- **Precision**: It quantifies the proportion of correctly predicted positive instances out of the total instances predicted as positive. It indicates the model's ability to minimize false positives.
- **Recall**: It represents the proportion of correctly predicted positive instances out of the total actual positive instances. It reflects the model's ability to capture true positives effectively.

In Table 2, the results from the experiments are shown for the different models. We also added the comparison with the Kaggle public leaderboard prize winner (H.

Table 2: Classification results of our best performing model architectures. Measures F1, Precision and Recall are calculated with macro setting.

Model	Acc.	Loss	F1	Prec	Recall
LSTMPredictor C	0.50	2.05	0.48	0.46	0.47
TransformerEnsemble E	0.66	1.47	0.66	0.65	0.65
HybridEnsembleModel F	0.75	1.32	0.75	0.74	0.74
YetAnotherEnsemble H	0.78	1.38	0.78	0.77	0.77
CVTransferLearningModel A	0.78	0.88	0.79	0.77	0.77
YetAnotherTransformer G	0.78	1.0	0.78	0.77	0.77
HybridModel D	0.79	0.85	0.8	0.78	0.78
TransformerPredictor B	**0.81**	**0.86**	**0.81**	**0.8**	**0.8**
HOYSO48	*0.81**	-	-	-	-

*On public leaderboard in Kaggle on the competition. It is not clear, how these measures compare to our test data. [2]

We observed that most of our models tend to plateau in accuracy at around 75-80%. This indicates that our models achieve an accuracy of approximately 80%, meaning that 8 out of 10 predictions are correct. Considering the challenge of distinguishing among 250 different sign classes, this accuracy rate is quite impressive and significantly better than random guessing. Also, when we compare our model's performance to the Kaggle leaderboard, it indicates that our model is in proximity to the top performers. Considering that over 1000 different teams have worked on the problem, this is a promising outcome for our model. However, it is important to note that the Kaggle leaderboard evaluation is based on a different dataset, which may have variations compared to our dataset.

[2] https://www.kaggle.com/competitions/asl-signs/leaderboard?tab=public

VI. Applications

In order to utilize the model, a camera feed was developed to make predictions on signs captured by the camera. In this scenario, the camera feed captures real-time video footage, and the model analyzes each frame to predict the signs present.

VII. Discussions

While achieving an accuracy of 78% may seem relatively low compared to a perfect accuracy of 100%, it's important to consider the context and the specific task we are working on. However, the accuracy of the model is greatly influenced by the quality of the data and the preprocessing steps applied. In our case, data cleansing posed significant challenges due to the sheer volume of data and the abstract nature of the data itself. Remember, the consisted of a vast number landmark coordinates and the only control that humans can do is by visualizing the keypoints as we did.

Furthermore, the data preparation was a critical step itself. A lot of decisions and assumptions have been made that actually could influence model performance. These decision involved interpolation strategies, landmark usage, sequence length selection, padding strategies and finally also augmentation strategies. We decided for one strategy, but it is clear that other solutions might be more promising. In future work, the data preparation has to be reconsidered again, as it might pose a potential point for improvement.

Training multiple models requires a good structure in the project. This is especially true when experimenting also with two different frameworks. It basically doubles the effort, as all the functions, loaders, callbacks have to be deep learning framework independent.

VIII. Outlook

As we implemented a flexible boilerplate for testing different models in different frameworks, our project structure can be used for other tasks.

IX. Acknowledgements

We would like to express our gratitude to the following individuals and organizations for their invaluable contributions to this project:

Kaggle: We extend our appreciation to Kaggle for providing the dataset used in this project.

MediaPipe: We would like to acknowledge the MediaPipe team for developing their landmark extraction model. Their robust framework allowed us to efficiently extract and analyze key visual features from our data.

PyTorch: We are grateful to the PyTorch community for developing the deep learning framework that served as the backbone of our model. The flexibility and efficiency of

PyTorch enabled us to train and optimize our neural network effectively

I. LITERATURE

[1] Kaggle.com, „Google - Isolated Sign Language Recognition," Kaggle Inc., 2023. [Online]. Available: https://www.kaggle.com/competitions/asl-signs.

[2] languagesunlimited, „American & Canadian Sign Languages Compared," Language Unlimited LLC, 2023. [Online]. Available: https://www.languagesunlimited.com/american-canadian-sign-languages-compared/.

[3] A. Chow, G. Cameron, M. Sherwood, P. Culliton, S. Sepah, S. Dane and T. Starner, *Google - Isolated Sign Language Recognition,* 2023.

[4] K. He, X. Zhang, S. Ren und J. Sun, „Deep Residual Learning for Image Recognition," *arXiv,* Nr. arXiv:1512.03385v1, 2015.

[5] A. S. o. D. Children, „RIT/NTID joins forces with Google, Georgia Tech to create PopSign," ASDC News, 31 May 2022. [Online]. Available: https://deafchildren.org/2022/05/rit-ntid-joins-forces-with-google-georgia-tech-to-create-popsign/.

[6] inc., Kaggle, „Leaderboard: Google - Isolated Sign Language Recognition," Research Code Competition, 2023. [Online]. Available: https://www.kaggle.com/competitions/asl-signs/leaderboard.

[7] M. Solutions, „MediaPipe Holistic," Mediapipe, 2023. [Online]. Available: https://github.com/google/mediapipe/blob/master/docs/solutions/holistic.md.

[8] E. B. Kate Brush, „Data Visualization," *TechTarget,* 2023.

[9] K. G. a. C. R. Albert Gu, *Efficiently Modeling Long Sequences with Structured State Spaces,* 2022.

[10] PopSign, „PopSign," [Online]. Available: https://www.popsign.org/. [Zugriff am 2023].

[11] B. Krista, W. J. Holstrum und J. Eichwald, „Hearing Screening for Newborns: The Midwife's Role in Early Hearing Detection and Intervention," *Journal of Midwifery & Women's Health,* pp. 18-26, 2009.

[12] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-}. Chang, M. G. Yong, J. Lee, W.-}. Chang, W. Hua und M. Georg, „MediaPipe: A Framework for Building Perception Pipelines," *arXiv,* Bd. 1906.08172, 2019.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser und I. Polosukhin, „Attention is all you need," *arXiv,* Bd. arXiv:1706.03762, 2017.

[14] Inc., Kaggle, „Dataset Card for the Isolated Sign Language Recognition Corpus," Kanngle Inc, [Online]. Available: https://www.kaggle.com/competitions/asl-signs/overview/data-card.