LSTM Architectures with Parameter Fine-Tuning for Enhanced Turkish Sign Language Action Recognition

Alpay Turgan

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
January 2025
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ali Hakan Ulusoy
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Zeki Bayram
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Asst. Prof. Dr. Ahmet Ünveren
Supervisor

Examining Committee

1. Prof. Dr. Ahmet Adalıer
2. Assoc. Prof. Dr. Adnan Acan
3. Asst. Prof. Dr. Ahmet Ünveren

ABSTRACT

Sign language action recognition is very important to meet the communication needs of people with hearing impairments. This study proposes a deep learning method combined with Long Short-Term Memory (LSTM) networks and a parameter tuning technique, an advanced optimization technique, to recognize sign language actions at a realistic level. The dataset used can understand pose, hand, and face signs up to 15 different actions using the media pipe holisctic feature extraction technique. The entire data set used in the model was prepared by myself and is mentioned in detail in the study.

The proposed model architecture uses layers for media pipe holistic feature extraction. It captures frames in videos and processes them to the LSTM layer. The parameter setting used in the development of the model was performed using the random search method.

With the special data set used, the accuracy rate is 98% in validation and up to 90% in testing, and parameter tuning has played a major role in improving these rates.

Keywords: sign language action recognition, deep learning, parameter, lstm, gesture recognition

ÖZ

İşaret dili eylem tanıma, işitme engelli kişilerin iletişim ihtiyaçlarını karşılamak için çok önemlidir. Bu çalışma, işaret dili eylemlerini gerçekçi bir düzeyde tanımak için Uzun Kısa Süreli Bellek (LSTM) ağları ve gelişmiş bir optimizasyon tekniği olan parametre ayarlama tekniği ile birleştirilmiş derin öğrenme yöntemini önermektedir. Kullanılan veri seti, media pipe holistic özellik çıkarma tekniğini kullanarak 15'e kadar farklı eylemi, pozu, el ve yüz işaretlerini anlayabilir şekilde hazırlanmıştır. Modelde kullanılan tüm veri seti benim tarafımdan hazırlanmıştır ve çalışmada ayrıntılı olarak belirtilmiştir.

Önerilen model mimarisi, media pipe holistic bütünsel özellik çıkarma için katmanlar kullanır. Videolardaki kareleri yakalar ve bunları LSTM katmanına işler. Modelin geliştirilmesinde kullanılan parametre ayarı, rastgele arama yöntemi kullanılarak gerçekleştirilmiştir.

Kullanılan özel veri seti ile doğrulamada doğruluk oranı %98 ve testte %90'a kadar doğruluk oranı gözükmektedir ve parametre ayarlaması bu oranları iyileştirmede önemli bir rol oynamıştır.

Anahtar Kelimeler: işaret dili eylem tanıma, derin öğrenme, hiperparametre, lstm, jest tanıma

ACKNOWLEDGMENTS

I would like to express my gratitude to Asst. Prof. Dr. Ahmet Ünveren for his mentorship throughout this research. I find this support very valuable as he has contributed greatly to this research with his recommendations and suggestions.

In addition, I would like to thank everyone who has contributed to this study in any way.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

1D CNN	1 Dimensional Convolution
2D CNN	2 Dimensional Convolution
ASL	American Sign Language
AUTSL	A Large-Scale Multi-View Turkish Sign Language Dataset
Conv1D	1 Dimensional Convolution
CSL	Chinese Sign Language
CTC	Connectionist Temporal Classification
GCN	Graph Convolutional Network
GRU	Gated Recurrent Unit
HSV	Hue, Saturation, Value
KNN	k-Nearest Neighbors
LSTM	Long Short-Term Memory
MLP	Multi-Layer Perceptron
MS-G3D	Multi-Scale Graph-based 3D Network
RELU	Rectified Linear Unit
RGB	Red, Green, Blue
RNN	Recurrent Neural Network
S3D	Separable 3D Convolution
SLR	Sign Language Recognition
SLT	Sign Language Translation
SOTA	State of the Art
SPN	Spatial Pyramid Network
SSN	Structured Segment Network
TID	Temporal Information Density

TANH	Hyperbolic Tangent
TCN	Temporal Convolutional Network
TSL	Turkish Sign Language
WLASL	Word-Level American Sign Language Dataset

Chapter 1

INTRODUCTION

Sign language is a tool that allows people with speech impairments to communicate with others. This language combines visual expressions, such as hand movements, facial expressions, and body postures. With the latest technologies, efforts are being made to enable people with speech impairments to communicate with people without disabilities with the help of technology. The development of image processing and deep learning methods has paved the way for the development of models that work as accurately as possible for the perception and translation of sign language [14].

However, the complexity of sign language and the accurate recognition and understanding of gestures are challenging. The diversity of gesture-based datasets and the obstacles in their classification have become a significant problem for people working in this field. Turkish Sign Language is a less researched area compared to the American or British sign languages, which are widely used throughout the world. Turkish Sign Language has a unique structure with different gestures. Training and understanding these movements is a great burden for researchers. In this context, the Turkish Sign Language plays an important role in eliminating the communication problem in Turkish-speaking geographies. Modern computer vision and deep learning techniques, especially architectures such as Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM), offer a powerful solution in the analysis and classification of time series data. [14] [12].

These architectures are important for understanding complex gestures and poses. However, to operate such architectures in real life, many processes such as data processing, dataset creation, model training, and model evaluation are required. This study is an example for future studies that train some of the Turkish sign language movements. The developed system includes a deep-learning model that can recognize 15 basic movements in Turkish sign language from camera images and produce results with high accuracy rates.

1.1 Problem Definition

Currently, developed sign language recognition systems generally target the most widely spoken sign languages such as American Sign Language (ASL) or British Sign Language (BSL). However, Turkish Sign Language has both similarities and differences with these languages. Therefore, studies on local sign languages such as the Turkish Sign Language are largely lacking [12].

In cases where hand signs are similar to each other, correct classification can be difficult. Especially the coordinates of the fingers and short-term movements make the study very difficult. This situation is the reason for a stable solution for Turkish Sign Language recognition systems [12].

Additionally, detecting subtle differences between gestures is another important challenge for current deep-learning models. For example, being able to distinguish between similar hand positions or gestures such as Figure: 3.1(“eight”) and Figure: 3.2(“three”) is a critical factor that determines the success of the system [14].

Finally, while developing this model, I did it on my laptop and my main goal is for daily users to benefit and use this model. The features of my laptop are as follows. Lenovo Legion 5 15ACH6H, 16GB RAM, Nvidia RTX 3060(Laptop) GPU, AMD Ryzen 7 5800H.

1.2 Contribution

Due to the limited number of studies on local sign languages, the developed models provide a new solution that perceives Turkish Sign Language [3] [12]. The main purpose of this study is to develop a system that can vividly recognize 15 basic gestures suitable for Turkish Sign Language. The main contribution of this study is that it uses "parameter tuning" to optimize deep learning models for the dataset you create. It also contributes to the literature by using modern deep learning types such as LSTM and Conv1D-LSTM [3].

Additionally, my goal is to develop a system that daily users can benefit from. One of our main goals is to enable daily users to run this system on their laptops without any additional costs.

1.3 Methods

The model is designed to be taught and operated via a laptop webcam. The camera records the movements and positions of the user's hands for a certain period. This data collection method is the first step in the visual analysis of sign language. The reason for using the webcam is to ensure that every user of the laptop can easily benefit from this system. The data obtained from the videos are processed after making them suitable for classification. The dataset consists of 15 basic sign language gestures and each gesture is performed by the researcher. Each gesture is created from 2-second videos recorded by the camera and then the videos are divided into individual frames. A data structure is used where these gestures are represented in 1629 feature dimensions

for 30-time steps. The reason for this is to make it compatible with the models we use. During feature extraction, the basic visual and spatial features of each gesture are preserved (ex X, Y, and Z). The time series data of the gestures allows the model to understand these gestures [3]. Among the deep learning models used in the system, there are especially Long Short Term Memory (LSTM) and Conv1D.

We use 1D CNN based on the dataset we prepared, the reason for this is that since we use 30 temporal dimensions and 1629 features each time step, 1D CNN aims to extract patterns specific to time series data by applying convolutions along the axis in understanding sequential data. 2D CNN, on the other hand, is not suitable for our dataset because it deals with two different concepts such as length and width, and it is not as successful as 1D CNN in extracting meaningful temporal features. It is not an architecture suitable for sequential datasets.

In the development of the models, a different method was tried by combining and parallelizing the layers of Batch Normalization, Max Pooling, LSTM layers, Dropout, Dense Layers, and Conv1D. While the first two LSTM layers learn the time series features of the movements, the last LSTM layer transmits this information to a dense layer [5]. During the training of the models, the dataset is divided into three different parts: training, validation, and testing. Training Data is used to learn the model, Validation Data is used to prevent overfitting and evaluate the performance, and Test Data is used to evaluate the real-life validation of live video recordings. The classification accuracy of the model was optimized using the categorical cross-entropy loss function and the Adam optimization algorithm. The ReduceLROnPlateau method was used to dynamically adjust the learning rate [8]. The main goal of the developed system is to replace a real-time sign language

recognition application. Gestures are defined by processing live images taken from the camera.

1.4 Thesis Outline

In the introduction section, the basic information about the study is given. In the literature review section, the summaries of 14 different articles and the contributions of the articles to sign language are discussed. In the dataset section, how the videos in the study were prepared and made suitable for the model is explained in detail. In the Proposed Method section, as much detail as possible about the model and the summary information of the model are given. In the Experimental Results section, the value information of the model, that is, the details of the results, and then the comparison with the data in the 14 different articles mentioned before are made. Finally, in the conclusion section, a general summary is given and a note is given to thank the subsequent studies.

Chapter 2

LITERATURE REVIEW

In this section, which is examined from 14 different sources, the important points of what these sources do and what they contribute to the world of science will be emphasized. A detailed comparison of the models will be in the "Experiments and Results" section.

Nguyen Huu Phong et al. (2023), introduce a dynamic ASL dataset to enforce the limitations of static datasets and focus on temporal coordinates with the help of hand gesture lists. In the study, 150 videos consisting of 10 signals were improved to 225 videos consisting of 15 signals. Background subtraction and noise filtering techniques were used. Deep learning models, DenseNet201, InceptionV3 , and InceptionResNetV2 were used. Multilayered associative networks (MLPs) and Long Short-Term Memory (LSTM) networks were used. They stated that when DenseNet201 and LSTM models were combined, they achieved 86% accuracy for ASL movements. They mentioned that the movements in the words "mouse" and "cat" were very close to each other and caused problems in classification [1].

Songyao Jiang et al. (2021), they combine dynamic SLR with video data. While skeletal data accurately represents data such as hands and bodies, RGB videos provide information in a related way. This combination has used the advantages of both sides by preventing problems such as noise and incomplete skeletal data. They used temporal convolutional networks (TCNs) for temporal alignment. They

emphasized that this model works better compared to recurrent neural networks (RNNs). In the conclusion, they stated that more than one dataset, including the CSL (Chinese Sign Language) dataset, showed better performance in basic movements. With the combined system, we understand that the final version of the model will be sufficient for perceiving basic sign language [2].

Razieh Rastgoo et al. (2021), they added to the Songyao Jiang work [2]. They use a combination of Convolutional Neural Networks and Long Short-Term Memory Networks. Again, there is a fusion. In the Songyao Jiang work, they used similar RNN and TCN models but combined them with CNN and LSTM to achieve more advanced results [3].

Yutong Chen et al. (2022), the developed TwoStream-SLR and TwoStream-SLT frameworks bring innovations to the datasets. The TwoStream-SLR framework uses a binary framework to process videos and key points. The model includes a Signal Pyramid Network (SPN) to capture brightnesses and uses Connectionist Time Classification (CTC) losses for feature extraction [4].

Anirudh Tunga et al. (2021), spatial dependencies are captured using GCNs that encode connections between key points in each frame. Temporal dependencies are captured using the transformer-based BERT model to resolve connections between frames. A significant feature of this work lies in its analysis of the WLASL dataset, which is a major challenge for word-level sign language recognition. The GCN-BERT model includes late fusion by combining the spatial recognition power from GCN and the temporal power from BERT in the last layers [5].

Manuel Vázquez-Enríquez et al. (2021), the authors used the AUTSL dataset, which is a Turkish sign language dataset. The MS-G3D model was compared with SOTA. The main innovation of the study is to examine the improvements in the model by working with the MS-G3D and S3D models. The combination of these models gave a better result than the classical CNN+LSTM solution. However, the authors noticed that this improvement did not make much difference in the percentage increase [6].

C.K.M. Lee et al. (2021), the methodology uses the Leap Gesture Controller. Difficult feature extractions such as palm center location, sphere radius, and fingertip locations are extracted for better and more accurate model training. The LSTM-RNN model plays a better role in understanding more complex and complicated letters. With the addition of KNN to the model, complex gestures play a big role in the development of the model with better layers [7].

Roama Sri Lakshmi Murali et al. (2022), in this study hand movements were captured through webcam and these movements were grayscale, backgrounds were removed using HSV color space, segmentation was used and reduced to 64x64 pixels and processed with deep learning technique. This data was classified using binary pixel features 2D CNN [8].

Muhammad Al-Qurishi et al. (2021), the authors combined these two systems with sensor-dense Xbox Kinect and a camera to record RGB videos resulting in a hybrid system that recorded both sensors and videos. A mixture of CNN and RNN was used as a model and the advantages of both models were utilized [9].

Yutong Chen et al. (2022), the paper addresses the task of sign language translation (SLT) with a two-stage approach: Sign2Gloss for the conversion of sign language videos into semantic annotations and Gloss2Text for the translation of these annotations into a real language. The framework performs well on multilingual data by using the mBART translation network trained on visual encoders Kinetics-400 and WLASL, improving generalization accuracy. The model maps complex structures using the V-L Mapper, which combines visual and linguistic features. The two-stage pipeline uses luminance sequences as intermediate representations. Sign2Text translation has shown better end-to-end accuracy. Evaluations of the PHOENIX-2014T and CSL-Daily datasets have shown that they achieve better results than previous models [10].

Elif Taşdemir et al. (2024), the model uses classical CNN layers. To improve its efficiency, three optimization algorithms, namely Adagrad, Stochastic Gradient Descent (SGD), and Adadelta, are compared and investigated in the study. The results show that the Adagrad algorithm achieves the highest accuracy (97.01%) after 60 epochs, outperforming SGD (96.71%) and Adadelta (70.91%) [11].

Mesut Toğaçar et al. (2021), in this study TID achieved sufficient performance in number recognition using SSNs, which are deep-learning models. The SSN model used in the study consists of two similar subnetworks and analyzes image pairs as detailed as possible in understanding the data highlighted by their similarities and focusing on this analysis [12].

Ishak Pacal and Melek Alaftekin's (2023), work aims to improve the TID classification accuracy using CNN-based models and provides an important perspective on this improvement. The work uses leading CNN architectures such as ResNet, DenseNet, VGG, MobileNet, and EfficientNet, each of which is fine-tuned using transfer learning with pre-trained ImageNet weights [13].

İlyas Demir et al. (2024), the main experiments in the work compare LSTM, GRU, and RNN architectures. There are almost very similar parts to the existing work and this work. I learned a lot from this paper and improved it. This is an LSTM-based work and especially it made a significant difference of 94% in accuracy. A superior success was achieved in the perception of Turkish sign language and a very good system was developed using the AUTSL dataset [14].

Chapter 3

DATASETS AND PREPROCESSING

The datasets used for the model were created with a laptop camera. Since daily users are our target. This dataset was created by recording 15 different movements in separate videos. All videos were prepared by myself. Pre-processing is to detect hand, face, and position movements from the videos and calculate hand coordinates to ensure consistency between samples. It consists of training, testing, and validation data, and a total of 3000 videos were shot for 15 different movements.

3.1 Datasets

The dataset consists of videos capturing actions such as “hello”, “thanks”, “love you”, “why”, “yes”, “zero”, “one”, “two”, “three”, “four”, “five”, “six”, “seven”, “eight”, and “nine”. These 15 movements were recorded multiple times to provide sufficient data for training, validation, and testing, and these data were divided into 3 main headings:

- Training Set: It is the main data that will be used in training the model 70% of videos (2100).
- Validation Set: This is the part designed to prevent the model from being overly compatible. 15% of videos (450).
- Test Set: Before and after the model is created, this is the final testing phase and this is where we understand the accuracy of the data. 15% of videos (450).

For validation, evaluate the model generalization during the training process. Helps tune parameters and prevents overfitting. Different types of data decrease overfitting

and optimize models to help test sets.

The videos were prepared at 15 frame rate (FPS), which is the value that an average laptop camera can give, and in the form of 2-second videos, which is the average duration of the action to be taught. We have a total of 30 frames for each video. Each video has its subclass folder and 200 videos have been prepared for each class. For example, as you can see in the figures, there is more than one action that is similar to each other. In Turkish Sign Language, the "three" Figure: 3.2 action and the "eight" Figure: 3.1 action are very similar to each other.

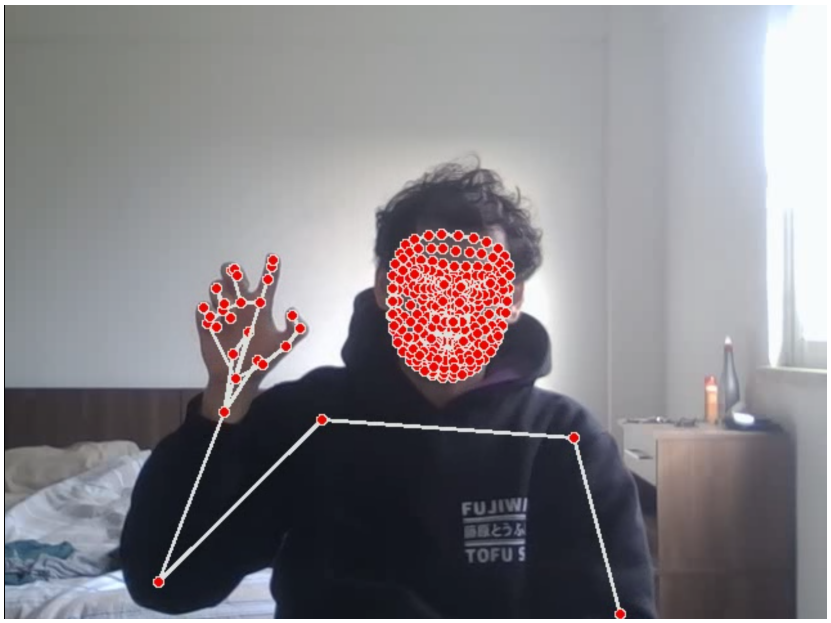

Figure 3.1: Label Eight

In the figures here:(3.3, 3.4, 3.6, 3.5, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12, 3.1, 3.2, 3.15, 3.13, 3.14, 3.17, 3.18, 3.16). As can be seen, there are two types of actions in the data set, such as many motions and movements that are fixed or have a beginning and an end.

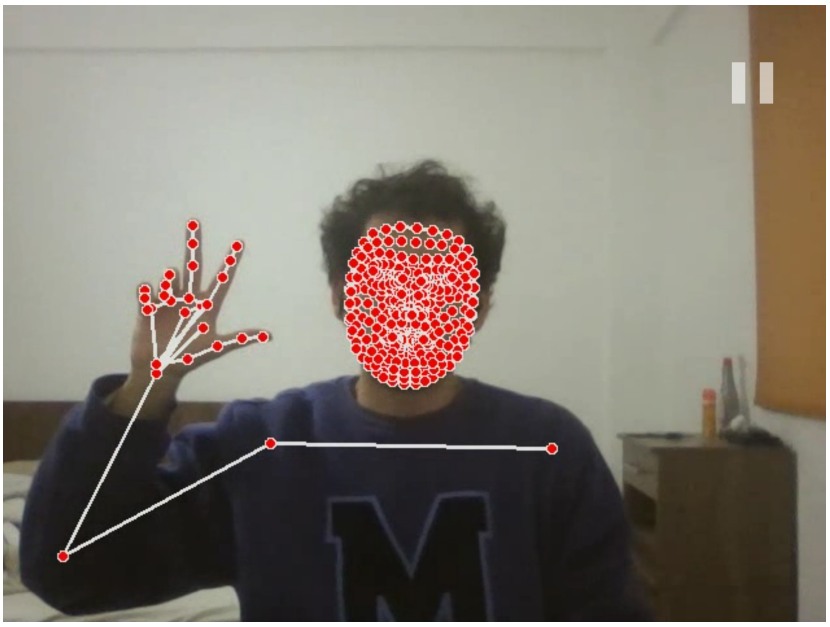

Figure 3.2: Label Three

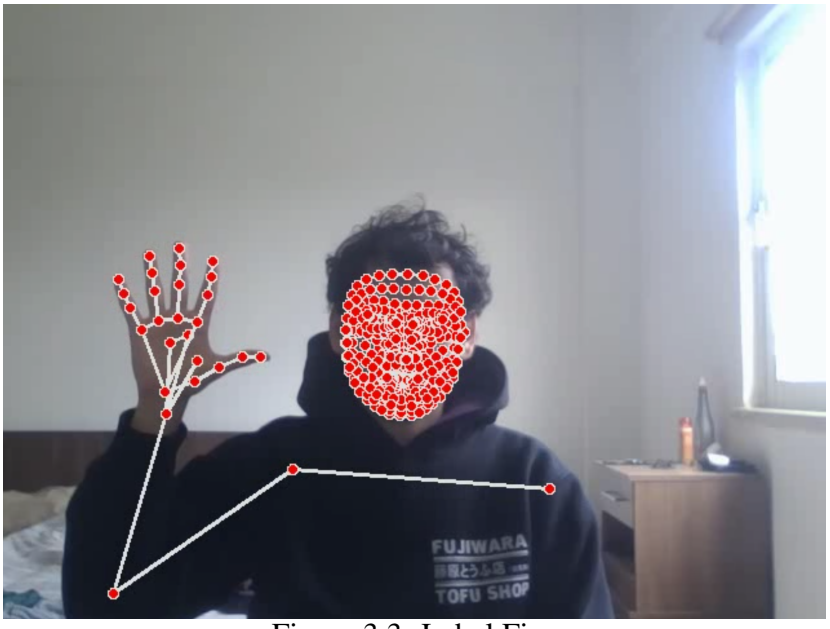

Figure 3.3: Label Five

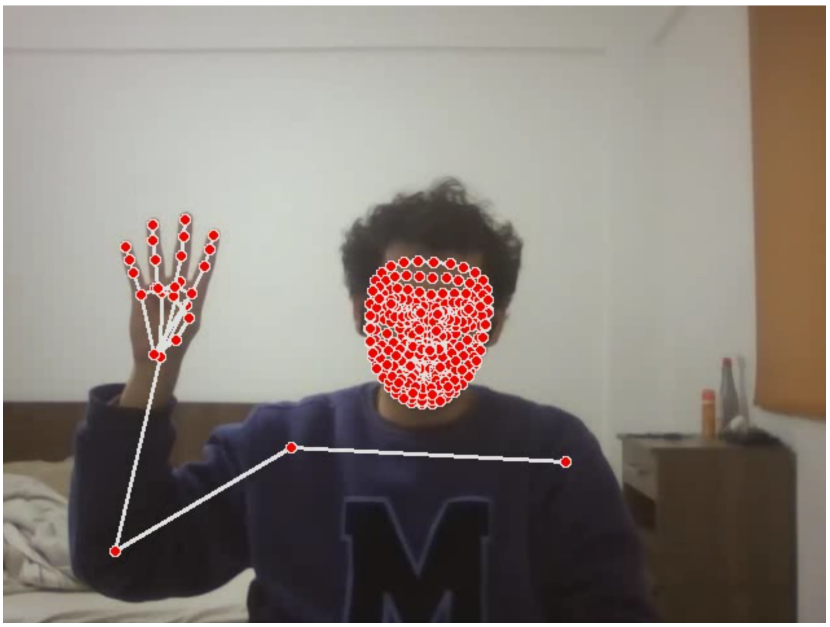

Figure 3.4: Label Four

Figure 3.5: Label Hello Start

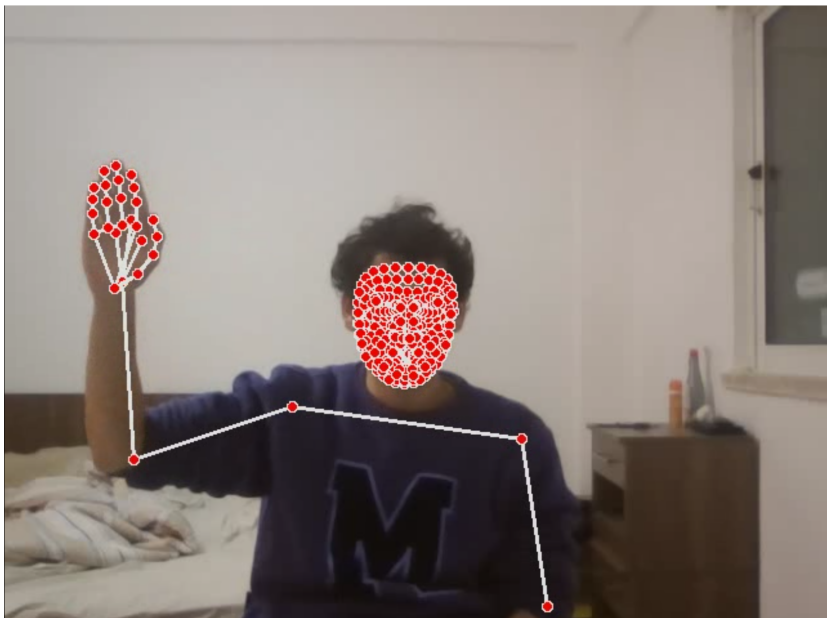

Figure 3.6: Label Hello End

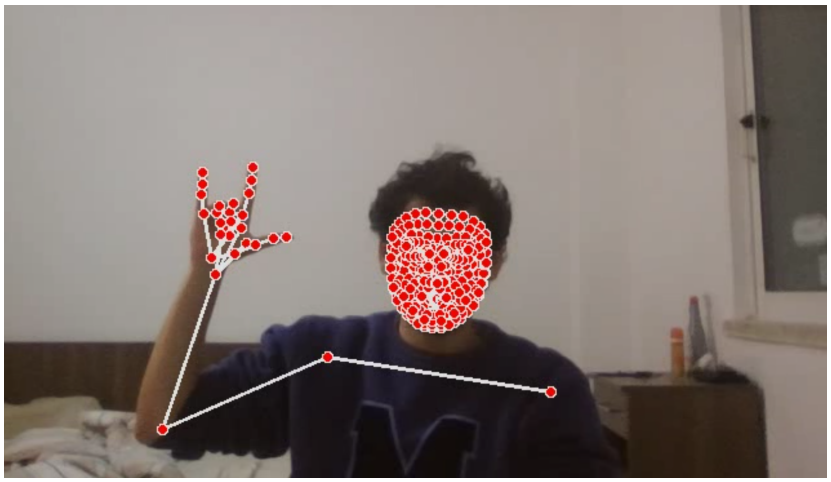

Figure 3.7: Label ILoveU

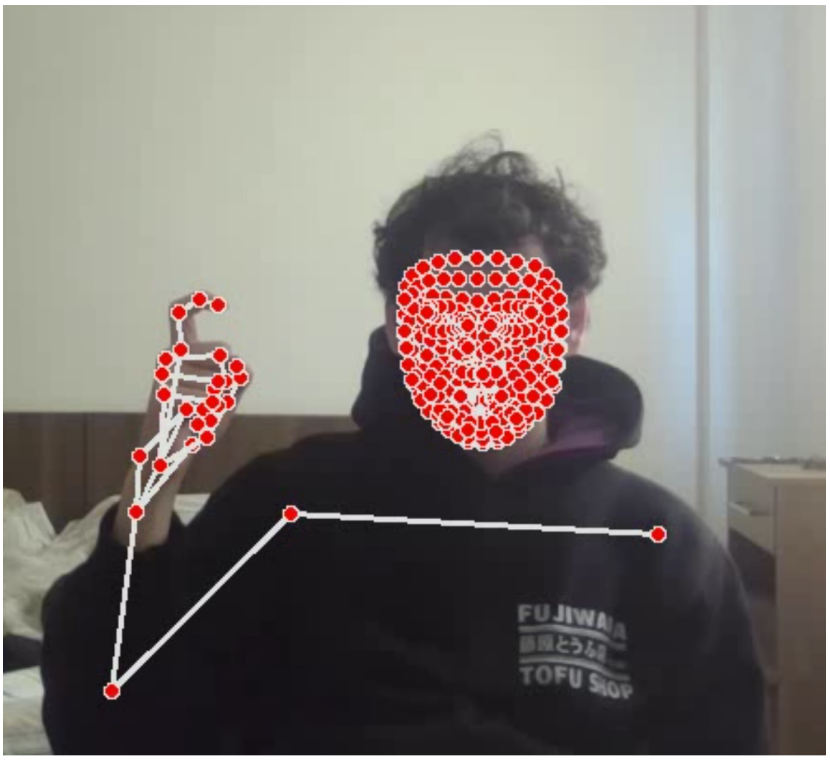

Figure 3.8: Label Nine

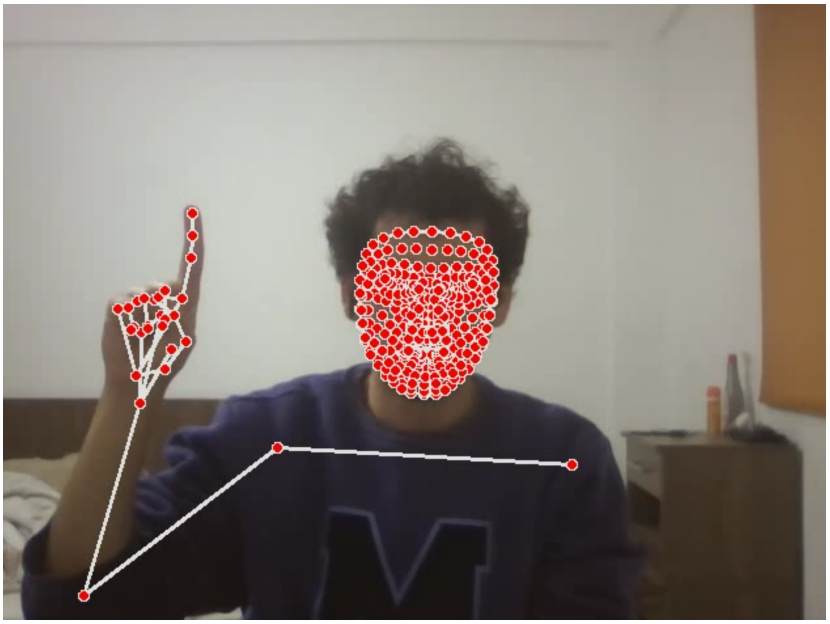
Figure 3.9: Label One

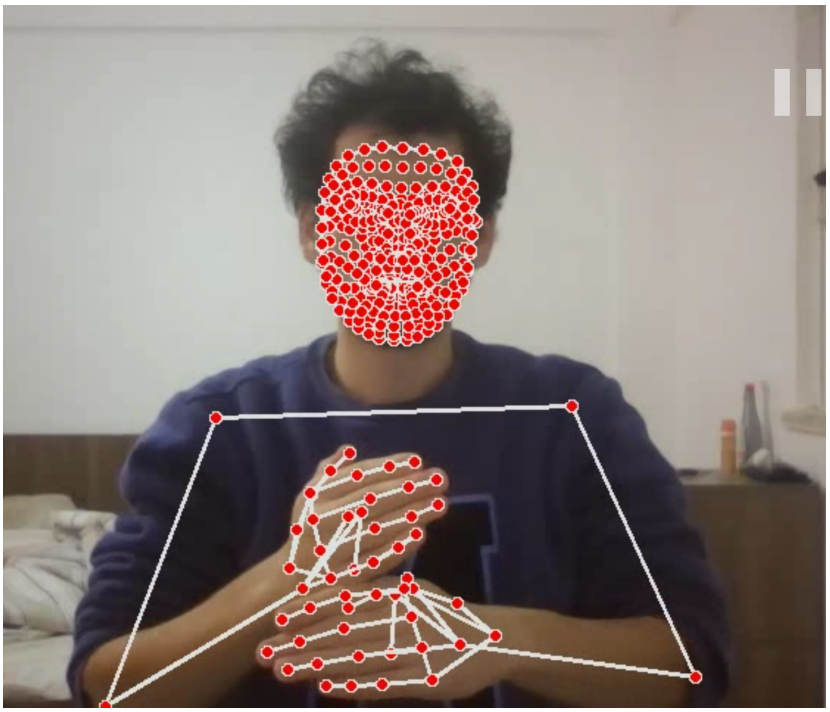

Figure 3.10: Label Reason

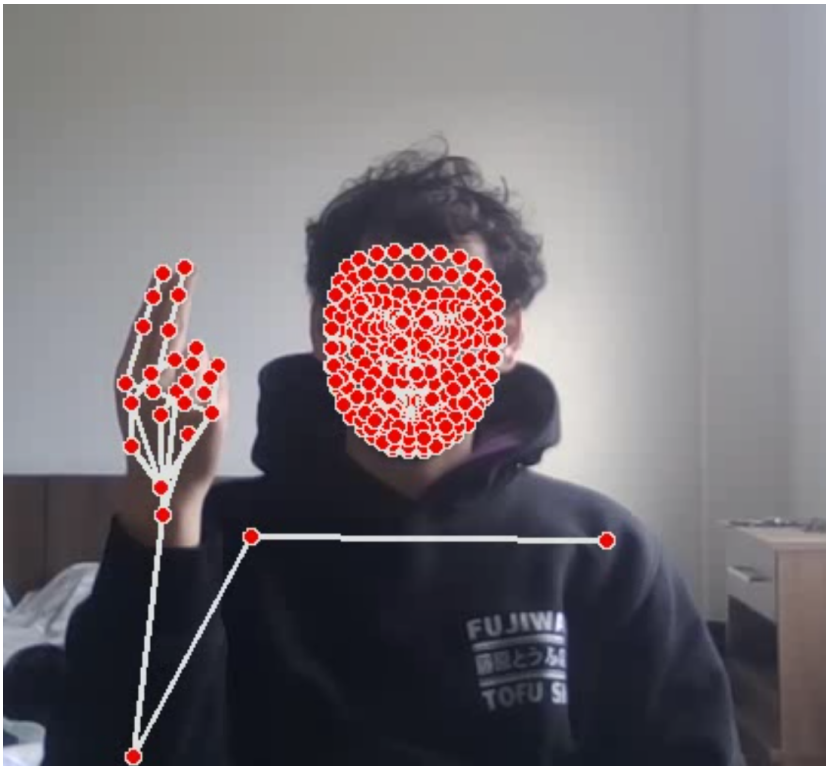

Figure 3.11: Label Seven

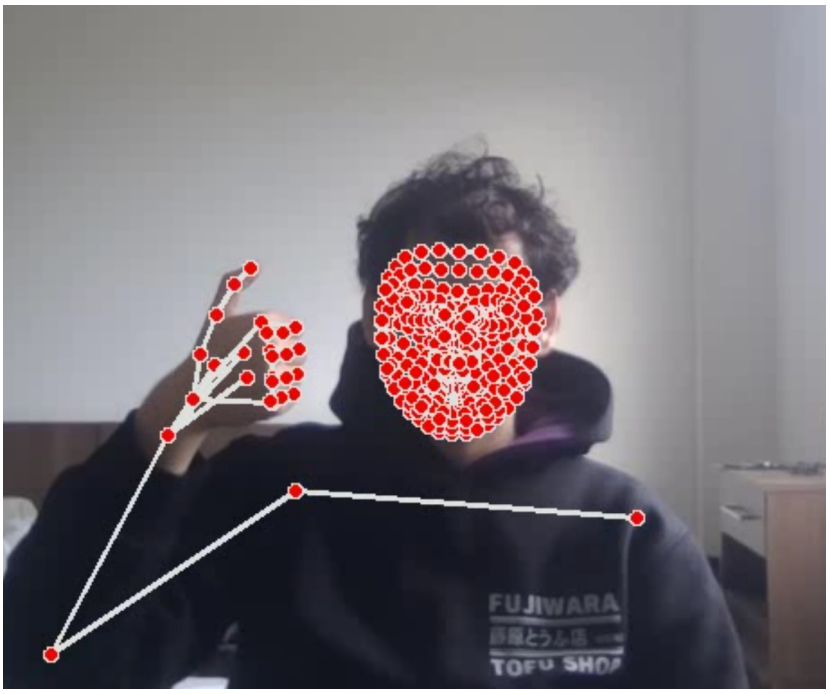

Figure 3.12: Label Six

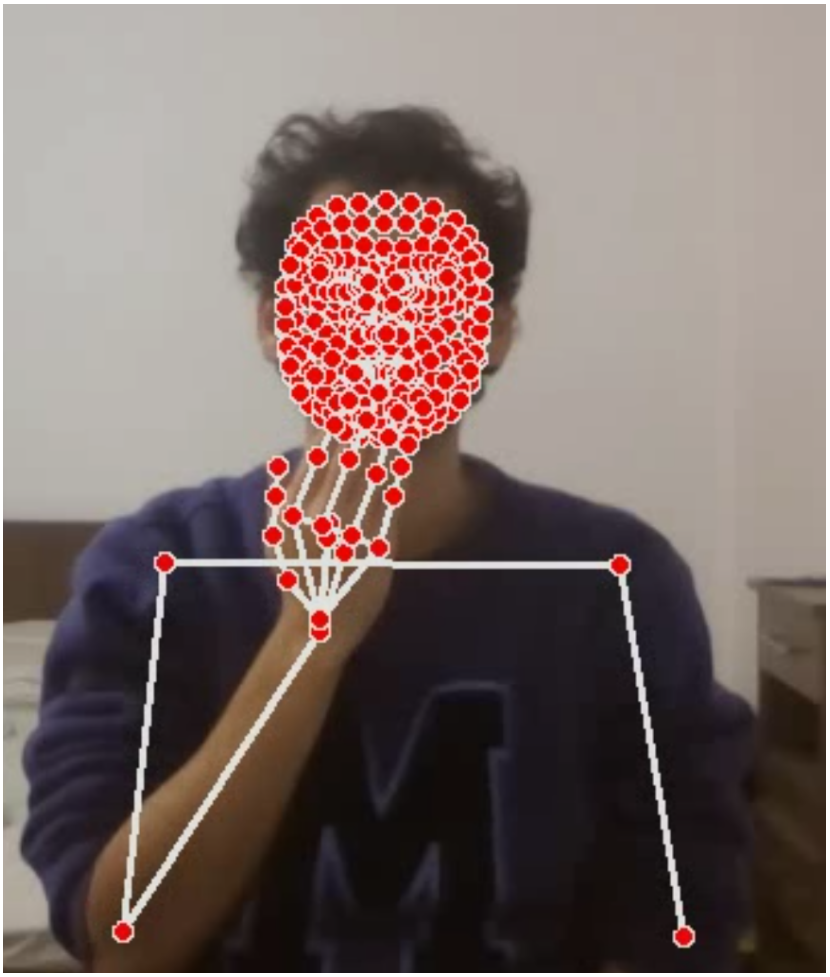
Figure 3.13: Label Thank You Start

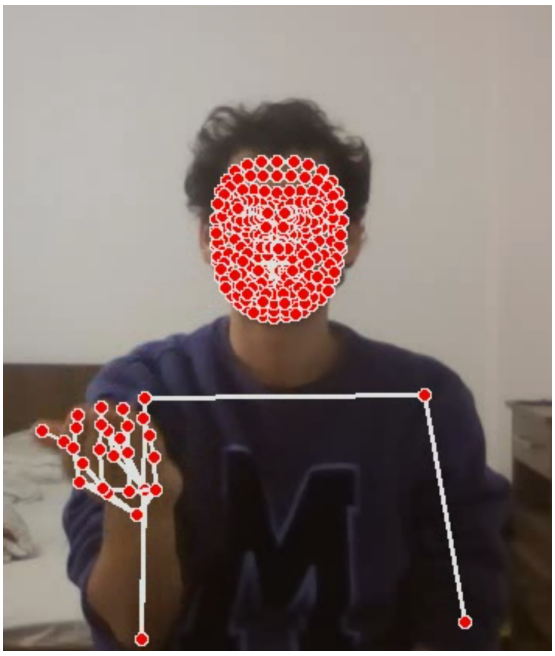

Figure 3.14: Label Thank You End

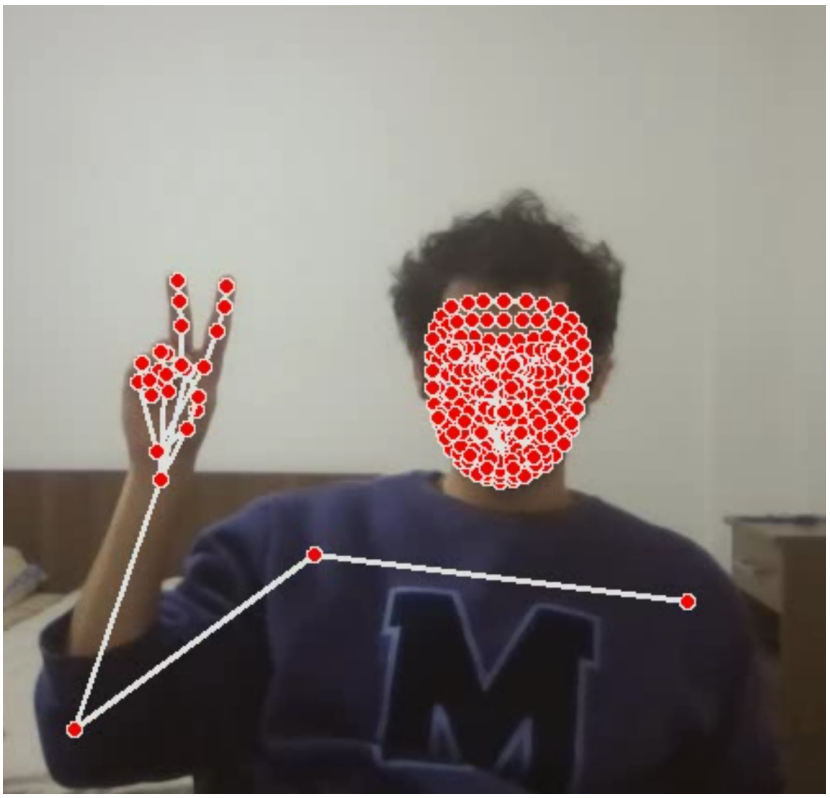

Figure 3.15: Label Two

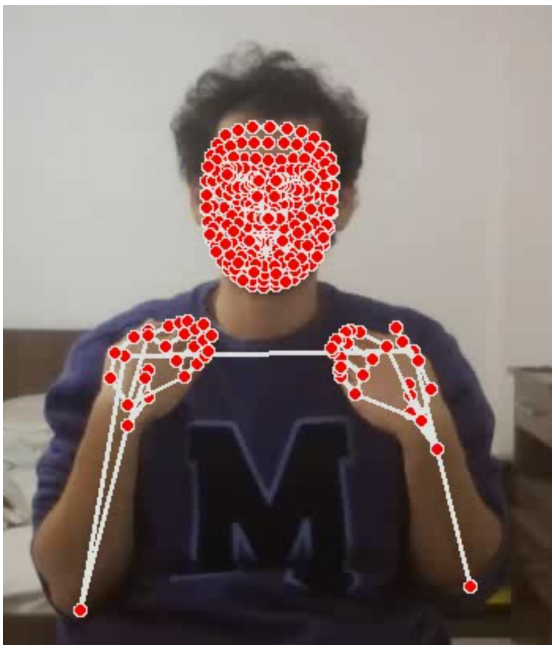

Figure 3.16: Label Yes Start

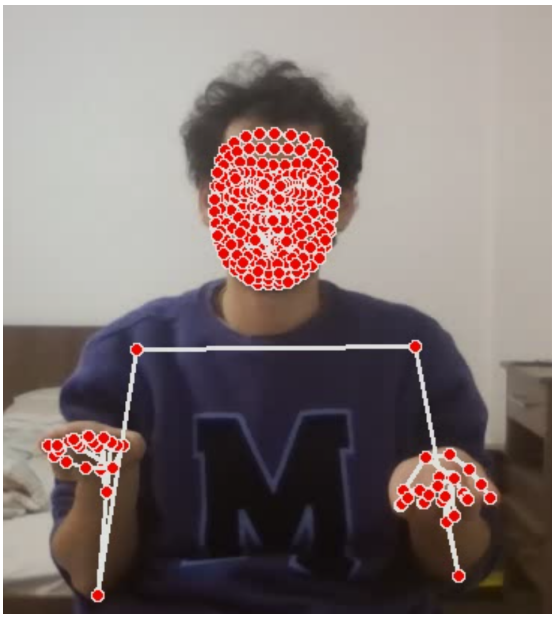

Figure 3.17: Label Yes End

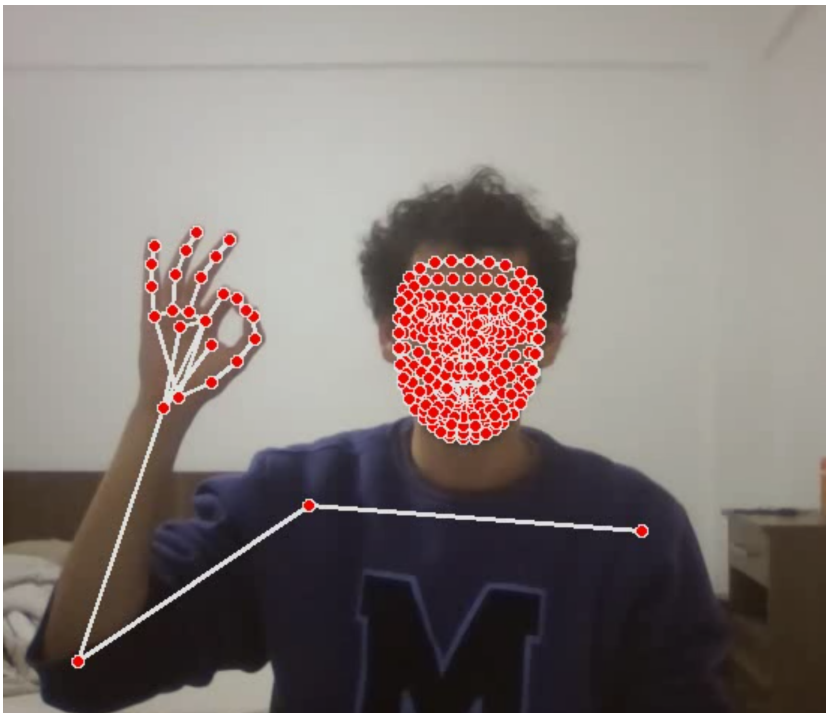

Figure 3.18: Label Zero

3.2 Pre-Processing Steps

- **Video Recording and Segmentation:** Each video was divided into individual photos with the .npy format, maintaining a fixed resolution of 640x480 pixels and 15 frames per second.
- **Feature Extraction with Mediapipe:** The Mediapipe Holistic model was used to extract body pose, face, and hand landmarks from each frame.
  - Pose Landmarks: Represented the overall body pose.
  - Facial Landmarks: Captured facial features.
  - Landmarks: Extracted left and right-hand positions.

  The extracted landmarks were stored as NumPy arrays containing x, y, and z coordinates for each keypoint.
- **Normalization:** To ensure consistency, the coordinates of the landmarks were normalized to a standard scale by removing all variations due to differences in subject height, camera angles, or distances.
- **Data Organization:** The processed data was stored hierarchically in: The Root folder named dataset. Subfolders for each action (e.g. "hello", thanks" etc.). Further divided into "training", validation, and, test folders.
- **Output Format:** Preprocessed data was stored in:
  - Landmark Sequences: Saved as .npy files for efficient storage and input to the neural network.
  - Video Files: Saved as .mp4 files for visualization.

The full data flow pipeline, including frame extraction, feature normalization, and dataset preparation, is detailed in Appenix A:Data Flow Process – From Collection To Model Training.

Chapter 4

PROPOSED METHOD

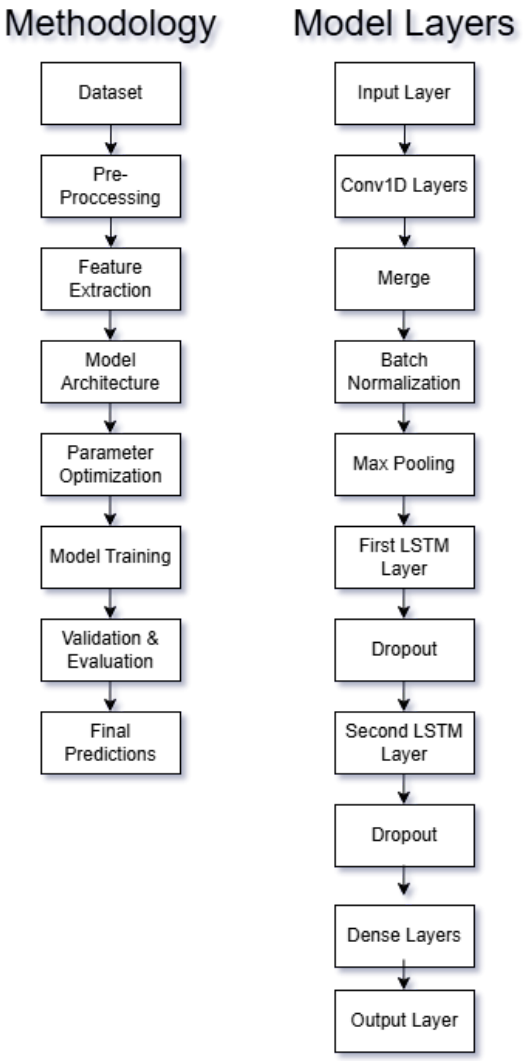

Figure 4.1: Model Map

In the proposed model, a new model emerged by combining convolutional layers, LSTM layers, and parameter optimization. You can see the progress in the methodology in the "Methodology" heading in Figure: 4.1. Now we will examine them in detail. We have explained Dataset and Preprocessing in detail, now we will

continue with our next heading, "Feature Extraction".

4.1 Feature Extraction

Feature extraction is a very important feature to extract meaning from consecutive photographs or frames. In this work, a combination of Conv1D layers and LSTM layers are used to extract both spatial and temporal features from the input data and the accuracy is greatly improved with parameters.

- **Input Layer:** The input layer is an array of 30 frames, each containing a feature vector of size 1629 representing pose, hand, and face landmarks extracted using mediapipe holistic.
- **Conv1D Layers:** Two parallel Conv1D layers with kernel sizes of 3 and 5 are used to capture local patterns in spatial features:
  - *Conv1D (Kernel=3):* Works finer.
  - *Conv1D (Kernel=5):* Works broader.

  The outputs of these layers work differently and, when combined, reduce complexity.
- **Post-Conv1D Operations:**
  - *Batch Normalization:* Improves stability during training by normalizing features.
  - *MaxPooling1D:* Reduces the dimensionality of spatial features and highlights the most important patterns.
  - *Dropout:* Prevents overfitting by randomly disabling some neurons during training.
- **Temporal Feature Extraction with LSTM:** The processed spatial features are added to LSTM layers to capture the time dependencies:
  - *First LSTM Layer:* Outputs a set of hidden states that model the temporal dependencies between consecutive frames.

- *Second LSTM Layer:* Encodes the entire sequence into a single temporal feature vector.

- **Output:** The extracted temporal feature vector is passed to classification layers for action recognition.

4.2 Model Architecture

Input layer: Defines an input data structure of size (30, 1629).

- **30:** Each input contains a sequence of 30 time steps. This means that each data is a time series of 30 frames.
- **1629:** Each frame consists of 1629 features extracted by mediapipe holistic. These features include location (x, y, z coordinates) and visibility values.
- **Input Layer:** Takes the raw data and is tasked to pass it to the next layers. Here, the raw data does not require any pre-processing before being passed to the Conv1D layer because all the data is already normalized and prepared in a fixed format. It is the most important and initial part of learning the model.

Conv1D Layers & Merge: Conv1D layers are an important layer in understanding and extracting meaning from data that is connected in a cascade. These layers use filters that move through one-dimensional data to analyze local patterns in specific windows of data and create high-level representations. Conv1D layers process the input data with a fixed window size for each step. During this process, filters (kernels) are used to learn how each window represents the data. During the learning process of the model, the weights of these filters are optimized and important features are extracted.

The details of how Conv1D layers are created for the model are as follows:

- **Processing of Input Data:** Since the input data for the model consists of 30 frames, we use 30 time steps and we need to use a tensor of size 1629 to be compatible with LSTM. To get output from these steps, we first use the Conv1D layer.
- **Use of Filters (Kernels):** For Conv1D layers, we have 3, 5, and 3, 7 kernel sizes. These sizes are used in conjunction with each other and form a parallel structure. With parallelism, it reads both a small area and a large area and takes advantage of both.
- **Activation Function:** ReLU (Rectified Linear Unit) activation function is used in Conv1D layers. This function reduces negative values to zero while leaving positive values as they are. ReLU allows the model to learn non-linear relationships and is computationally efficient.
- **Padding Strategy:** In Conv1D layers, the "same" option is used. This feature ensures that the input and output dimensions remain constant. Thus, the process is performed without losing information at the edges of the data.
- **Merging Features:** The outputs of the Conv1D layers running in parallel are merged to create a richer feature space for the model: merged = concatenate([convA, convB], axis=-1). This process allows features learned in different dimensions to be combined into a single representation.

Batch Normalization & Max Pooling: Batch normalization is an important technique for improving accuracy and speed in neural networks. It normalizes the input to a layer for each mini-batch during training, ensuring that the mean of each input is zero and the standard deviation is one.

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}} \tag{4.1}$$

Batch Normalization Computes:

- μ: Mean of the batch.
- σ^2: Variance of the batch.
- ε: Small constant to prevent division by zero.

$$y = \gamma\hat{x} + \beta \tag{4.2}$$

After normalization, the normalized value is scaled and shifted using trainable parameters γ and β: Max pooling is an important subsampling technique in neural networks. It reduces the width and height of feature maps while preserving important information. A pooling operation involves sliding a window (e.g., 2×2 or 3×3) over the feature map.

For each window, it selects the maximum value: y=max(x1,x2,..., xn). Here x1, x2,..., xn are the values in the pooling window. Max pooling is applied to reduce the size of the normalized feature maps, which helps extract the most salient features and reduce the computational load.

LSTM: LSTMs have a memory that allows them to retain information for long periods. They consist of three basic gates:

- **Forget Gate:** Determines which parts of the cell's memory should be discarded. Uses a sigmoid activation function to scale values between 0 (forget) and 1 (retain).
- **Input Gate:** Decides what new information should be added to the memory. This gate is activated by a sigmoid function and combined with a candidate memory update via a tanh function.
- **Output Gate:** Controls the output of the LSTM cell by deciding how much of the cell's memory will affect the output.

These gates ensure that important temporal features are preserved in sequential data, making LSTMs suitable for processing sequential data.

LSTM Layers: This model uses two LSTM layers:

- **First LSTM Layer:** Processes the data feature maps sequentially and preserves the temporal relationships between frames. Outputs a sequence that is passed to the second LSTM layer.
- **Second LSTM Layer:** Aggregates the output from the first LSTM layer into a single vector that captures the overall temporal dynamics of the input data.

These LSTM layers enable the model to learn both short-term and long-term temporal associations, which is critical for tasks like action recognition.

Dropout Layers: Dropout layers are added at the end of each layer to decrease overfitting.

Dense Layers: Dense (fully connected) layers are the final layers in the architecture, responsible for collecting features and performing classification:

- **First Dense Layer:** Aggregates the temporal features output by the second LSTM layer into a higher dimensional representation.
- **Second Dense Layer:** Further refines the features and reduces dimensionality in preparation for the final classification step.

As a result of the study, changes were made with many layers, and final tests were performed. At the final tests, better results were obtained when the dense layers were removed. Validation accuracy, which was 94%, increased to 98%and the test accuracy value, which was 85%, increased to 90%. Only the dense layer was added to the output layer to access softmax technique, and the results were examined in detail.

Output Layer: Uses a softmax activation function to generate class probabilities for 15 action classes. Dense layers map the learned features to a probability distribution and complete the action recognition task. Combined with dropout, these layers are robust and provide high performance on both training and test datasets.

4.3 Effect of Layers on Data Processing

First, we have 30 frames and 1629 features in our input layer, we need to make an inference from them. After the input layer, 2 combined Conv1D layers were used to perform multiscale feature extraction, the first one used a small kernel size and the second one used a large kernel size. The reason for this is that more local patterns (fingers) were obtained in the small part, and broader spatial relationships (arm position, body position) information was obtained in the large part, and then local and global spatial features were combined by combining these two layers. In the combination, our axis value was added as "-1" the timestep was kept constant, and only our filter size was doubled. After these operations, the batch normalization feature was used and the outputs in the layers were normalized ([-5, 0] => [0, 1]). After batch normalization, the maxpooling1D technique was used to reduce timesteps and provide benefits in extracting more intense features. After maxpooling1D, random neurons were deactivated together with the dropout layer to prevent over-learning. After these layers, we enter the LSTM layers, which we call our main model. The first LSTM layer learns by taking a general look at all the movements in the videos. The second LSTM layer provides a general perspective by only looking at the last frame. The first LSTM layer is more detailed and focuses on how the fingers change, but the second LSTM layer is used for a more complex understanding and understanding of large changes. Dense layers were also used, but better success was achieved without these layers, so there is no point in explaining the dense layers in great detail.

4.4 Parameter Optimization

Parameter optimization plays an important role in deep learning models developed with artificial intelligence. It searches for the best parameter set to maximize the accuracy of the model and uses that result. Parameters are configuration settings outside the model that cannot be learned during training. Examples include the number of filters in convolutional layers, learning rate, or dropout rates. In this study, a structured approach to parameter tuning was used using the Keras Tuner Random Search method.

Parameter Search Space

The following parameters were optimized:

- **Filters in Convolutional Layers:** The number of filters in Conv1D layers was set to {64, 128}. Increasing the number of filters allows the network to learn more features but also increases the computational cost.
- **Kernel Sizes:** Two kernel sizes were considered for each Conv1D layer: {3, 5} for one branch and {3, 7} for another branch. The kernel size determines the size of the receptive field and allows the network to capture features with different granularity.
- **LSTM Units:** The number of units in the two LSTM layers was optimized with possible values of {128, 64} for the first LSTM layer and {64, 32} for the second LSTM layer. More units allow LSTM to capture more temporal dependencies but may result in overfitting.
- **Dropout Rates:** Dropout rates were tuned for convolutional, LSTM, and dense layers:
  - **Conv1D Layers:** {0.2, 0.3}
  - **LSTM Layers:** {0.3, 0.4}
  - **Dense Layers:** {0.4, 0.5}

This parameter controls the proportion of neurons that randomly drop out during training to prevent overfitting.

- **Learning Rate:** Learning rates of {1e-3, 1e-4} were considered. The learning rate controls how fast the model updates its weights during training. Lower learning rates generally lead to more stable convergence but require more epochs.

Parameter Tuning Process

- **Keras Tuner Framework:** The Random Search method in Keras Tuner was used to test various configurations by randomly sampling parameter combinations from the defined search space.
- **Search Target:** The target metric was set to validate categorical accuracy to optimize the model's ability to generalize to unseen data.
- **Execution Details:** The tuner was configured to test up to 100 trials, with each trial running for a maximum of 20 epochs. Early stopping was implemented to terminate training if the validation loss did not improve for 5 consecutive epochs.
- **Result Evaluation:** After all trials, the parameter configuration with the highest validation accuracy was selected to build the final model.

Selected Parameters: The best-performing parameters determined by the tuner were:

- **Filters:** 128
- **Kernel Sizes:** 3 and 5
- **LSTM Units:** 128 and 64
- **Dropout Rates:** 0.3 (Conv1D), 0.4 (LSTM), 0.5 (dense)
- **Learning Rate:** 1e-4

These parameters achieved high performance on the validation dataset by balancing model complexity and generalization.

Importance of Parameter Tuning

Parameter tuning significantly impacted the performance of the model:

- **Improved Accuracy:** Optimal parameters led to improved validation accuracy compared to the initial baseline configuration.
- **Reduced Overfitting:** Dropout regularization enabled the model to generalize well to the test set.
- **Computational Efficiency:** Automatic tuning minimized manual trial-and-error, reducing development time and computational resources.

By utilizing an automatic and systematic approach, the parameter tuning process enabled the proposed model to achieve robust performance, making it suitable for real-world applications in sign language action recognition. It is too early to talk about it here, but all the features of the model are visible in this figure (Figure 4.2 and we will cover the details in great detail in the parameters section.

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_layer (InputLayer)       [(None, 30, 1629)]   0           []

 conv1d_6 (Conv1D)              (None, 30, 128)      625664      ['input_layer[0][0]']

 conv1d_7 (Conv1D)              (None, 30, 128)      625664      ['input_layer[0][0]']

 concatenate_2 (Concatenate)    (None, 30, 256)      0           ['conv1d_6[0][0]',
                                                                  'conv1d_7[0][0]']

 batch_normalization_4 (BatchNo  (None, 30, 256)     1024        ['concatenate_2[0][0]']
 rmalization)

 max_pooling1d_4 (MaxPooling1D)  (None, 15, 256)     0           ['batch_normalization_4[0][0]']

 dropout_10 (Dropout)           (None, 15, 256)      0           ['max_pooling1d_4[0][0]']

 conv1d_8 (Conv1D)              (None, 15, 256)      196864      ['dropout_10[0][0]']

 batch_normalization_5 (BatchNo  (None, 15, 256)     1024        ['conv1d_8[0][0]']
 rmalization)

 max_pooling1d_5 (MaxPooling1D)  (None, 7, 256)      0           ['batch_normalization_5[0][0]']

 dropout_11 (Dropout)           (None, 7, 256)       0           ['max_pooling1d_5[0][0]']

 lstm_4 (LSTM)                  (None, 7, 64)        82176       ['dropout_11[0][0]']

 dropout_12 (Dropout)           (None, 7, 64)        0           ['lstm_4[0][0]']

 lstm_5 (LSTM)                  (None, 64)           33024       ['dropout_12[0][0]']

 dropout_13 (Dropout)           (None, 64)           0           ['lstm_5[0][0]']

 dense_6 (Dense)                (None, 128)          8320        ['dropout_13[0][0]']

 dropout_14 (Dropout)           (None, 128)          0           ['dense_6[0][0]']

 dense_7 (Dense)                (None, 64)           8256        ['dropout_14[0][0]']

 dense_8 (Dense)                (None, 15)           975         ['dense_7[0][0]']

==================================================================================================
Total params: 1,582,991
Trainable params: 1,581,967
Non-trainable params: 1,024
```

Figure 4.2: Model Summary

4.5 Model Training, Validation & Classification

In this study, training, validation, and classification processes were systematically conducted to ensure the robust performance of the proposed model. Experiments were conducted on a single dataset divided into training, validation, and testing subsets. The methodology adopted multiple training and validation iterations to determine the optimal model configuration.

Training Phase: The model training process used datasets drawn from the training subset. In this phase, the architecture went through multiple training cycles (epochs) where each epoch represented a complete pass over the training data. The model learned to minimize the loss function and optimize its parameters through backpropagation. Early stopping mechanisms were incorporated to terminate training when the validation performance stopped improving, thus preventing overfitting.

Validation Phase: During the training process, the validation dataset was used to verify the performance of the model. Validation was performed after each epoch and the model with the lowest validation loss and the highest data accuracy was obtained and worked with. Validation plays an important role, especially in the parameter part. We can detail important parameters such as learning rates and dropout rates with the validation data.

Test Data Classification: After finding the best model, our validation was retested with the test dataset. The consistency between the data was compared and the classification scores were used to calculate a threshold value for binary classification.

Performance Evaluation: After the classification, outputs such as accuracy, precision, recall, F1 score, and confusion matrix were obtained and evaluated accordingly for detailed examination and evaluation of the performance. We will evaluate the detailed explanations of the results in the evaluation in the next topic.

Chapter 5

EXPERIMENTS & RESULTS

A special dataset consisting of 15 different sign language movements was prepared for this model. To test the model, data compatible with the model and similar data were used. To run the model, the installation via Python, the libraries used, how the datasets were divided to develop the model, and all the details of the results in the training of the model are explained in great detail in this section. In the detail section, there is a detailed explanation, especially about the accuracy of the data, the loss of verification, the confusion matrix of the model in both test and verification data and evaluation metrics. After all the details of the model are created and examined, we can find the most optimum parameters with the parameter system to find the best result. Finally, it was compared in detail with previous studies and evaluated with these machine learning approaches.

5.1 Experimental Setup

In the experimental setup section, there are details on which computer the developed model was made and the software used in the model.

5.1.1 Libraries & Environment

The model created was prepared with Python 3.8.20 version. Especially in the training part, powerful hardware is required, so using a powerful laptop that I already had was sufficient for my job. My computer features are as follows: RTX3060, Ryzen 7 5800H, 16 GB RAM.

Libraries Used in the Implementation: The following libraries are used in the implementation:

- **OpenCV (cv2):** Used for real-time image processing and video stream processing. It captures frames from the webcam and prepares them for gesture recognition.
- **NumPy:** Used for numerical operations and array manipulations that are critical to processing pre-processed data and feature arrays.
- **Matplotlib:** Used to visualize training progress, loss measurements, and model predictions.
- **Mediapipe:** A framework designed for real-time hand and body tracking, used to extract key points for gesture recognition in Turkish sign language.
- **TensorFlow/Keras:** A deep learning framework. Key components include:
  - tensorflow.keras.models: Used to build and compile the LSTM model for gesture classification.
  - tensorflow.keras.layers: Provides LSTM, Dense, and other layers for model architecture.
  - tensorflow.keras.utils: Facilitates the conversion of labels into a categorical format for multi-class classification.
  - tensorflow.keras.callbacks: Contains tools like `EarlyStopping` and `ReduceLROnPlateau` to optimize the training process.
  - tensorflow.keras.optimizers: Adam optimizer is used for efficient gradient descent.
- **Keras Tuner:** A library used to fine-tune the LSTM model for parameter optimization and better performance.
- **CUDA:** An artificial intelligence acceleration unit developed by Nvidia. This technology enables fast processing of large datasets.

Library Versions The library versions used in this study are as follows:

- **OpenCV:** 4.10.0.84.
- **NumPy:** 1.24.4.
- **Matplotlib:** 3.7.5.
- **Mediapipe:** 0.10.11.
- **TensorFlow:** 2.8.0.
- **Python:** 3.8.20.
- **CUDA:** 11.2.
- **cuDNN:** 8.8.

5.1.2 Splitting Datasets

There are 15 different gestures in the dataset created to understand the Turkish sign language gestures used in this model and these gestures are divided into 3 separate categories: Training, testing, and validation. These distinctions are divided to increase the performance of the developed model and to optimize the performance.

- **Training Set (70%):** Used to train the LSTM model by learning the patterns in gestures.
- **Validation Set (15%):** Used to monitor the performance of the model during training and prevent overfitting.
- **Testing Set (15%):** Reserved to evaluate the final model and measure its generalization ability.

Each gesture is represented by a sequence of 30 frames, with each frame containing 1629 features extracted using the Mediapipe framework. The dataset contains a total of 3000 samples (**15 gestures x 200 videos per gesture**), divided as follows:

- **Training Set:** 2100 videos (70% of the total dataset).
- **Validation Set:** 450 videos (15% of the total dataset).

- **Test Set:** 450 videos (15% of the total dataset).

In addition, all videos were divided frame by frame and saved in ".npy" format for training. Each video consists of 30 frames and was prepared for training by dividing it into 30 individual frames saved in ".npy" format.

5.1.3 Parameters

This model is actually a model that aims to increase the accuracy and stability of a simple LSTM model with Conv1D and advanced metric tuning.First, let us explain the layers and parameters used in detail, then let me explain in detail how the parameter I developed works.

Parameters Used In the Model:

Input_layer(shape(30, 1629)): We set the input layer to 30 and 1629 to be suitable for the data sets we prepared.

Conv1D Layers:

- **convA = Conv1D:**
  - Filters: 64 and 128.
  - Kernel Size: 3, 5.
  - Activation: relu.
  - Padding: same.
- **convB = Conv1D:**
  - Filters: 64 and 128.
  - Kernel Size: 3, 7.
  - Activation: relu.
  - Padding: same.
- **Merged** = concatenate([convA, convB], axis=”-1”):

The reason for using 64 and 128 filters is to use them to improve the model in both a small area and a large area. In the kernel, 3 learns the association well in shorter time windows. 5 and 7 learn the contexts better in longer time windows. The reason for using ReLU as activation is that it is better than tanh in calculations and prevents problems with derivatives. It converts negative values to zero. Padding, on the other hand, must be kept the same as in the input section, otherwise, the input and output dimensions will not be preserved and our operations will be in vain. Merged, on the other hand, combines the features of two different blocks and gains a wide feature. The reason for the axis being -1 is that the features of the two blocks are desired to be evaluated together.

BatchNormalization(Merged), MaxPooling1D(poolsize=2), Dropout(0.2, 0.3)

BatchNormalization is used on Merged to balance different distributions and facilitate combined learning. It is used because it normalizes the input data and makes the outputs of the layers more balanced. MaxPooling, on the other hand, reduces the data to a smaller size. It focuses on getting the maximum value in every 2-unit window. Reduce dimension + remove important parts and reduce noise. Dropout, on the other hand, disables feature maps according to the given ratio. In this way, it provides benefits such as overfitting, good generalization, and stochastic learning. The reason we keep it low in the first layers is that the first layers are the part where learning is the most.

Conv1D_1:

- **Filters:** 128, 256.
- **Kernel Size:** 3.
- **Activation:** relu.
- **Padding:** same.

BatchNormalization(Merged), MaxPooling1D(poolsize=2), Dropout(0.2, 0.3)

In addition, the kernel size is kept constant and the size of the filters is increased. The reason for this is to extract wider and deeper features and to learn more complex data. The kernel is kept small to understand 3-time series and to work more finely.

LSTM_1:

- **Filters:** 128, 64.
- **Return_sequences:** True.
- **Activation:** tanh.
- **Dropout:** (0.3, 0.4).

LSTM_2:

- **Filters:** 64, 32.
- **Return_sequences:** False.
- **Activation:** tanh.
- **Dropout:** (0.3, 0.4).

The filters at the beginning were increased and then decreased, this is because the first LSTM layer understands more complex data, and the second LSTM layer extracts features from less complex data. The Return_Sequences section is there to produce output for each time step. The second LSTM layer only produces output for the last time step. The reason for this is to stay fixed at the beginning and end of each movement and to improve learning on fixed data, and this part is quite necessary to create a fixed-size vector for input to dense layers. Activation tanh is the part that people with RTX graphics cards will use the most in model development, I can say because it speeds up your work a lot and is worse than RELU. The reason for the increase in dropout is to prevent overfitting by removing more of the random data since too much data is learned.

Dense_1:

- **Filter:** 128.
- **Activation:** relu.
- **Dropout:** (0.4, 0.5).

Dense_2:

- **Filter:** 64.
- **Activation:** relu.

Output_Layer:

- **Dense_3:**
  - **Gesture_Size:** 15.
  - **Activation:** softmax.

Learning_Rate:[1e-3, 1e-4].

Dense layers process the features created by LSTM and make them most efficient for the final classification after processing. Dropouts have increased because we need to eliminate more data as we use denser layers. Activation softmax is used in multi-class classification problems. It normalizes the probabilities of each class and provides output so that the sum is equal to 1. Learning rate allows the model to learn faster when it is first run and then learn slower. Because the data starts to overlap with each other and needs to be learned more slowly and intensively to understand the difference.

Parameter Tuner:

This is done by recording the given parameters in a function (build_model) in advance and then trying all these given parameters one by one with the parameter tuner to get the best result.

- **Filters:** 64, 128, 256.
- **Kernel Sizes:** 3, 5, and 3.7.

- **Dropout Rates:** 0.2, 0.3, 0.4, 0.5.
- **Learning Rates:** 1e-3, 1e-4.

"Keras Tuner" is used by selecting 100 different parameter combinations on these rates. This Tuner is ideal for finding random combinations using the RandomSearch algorithm. It takes a lot of time but gets the best output to find the best result. There are 256 different parameters according to the system but 100 different settings were set for this study.

While finding 100 Different Parameter Combinations:

- **Epochs:** 20.
- **Batch_size:** 32.
- **Callbacks:** tbcallback, earlystopping, reducelr.

The callback part is as follows,

- **TensorBoard Callback:** It is used to monitor and analyze data such as loss and accuracy in the training of the model and records it in a log file.
- **Early Stopping Callback:** If the validation loss does not decrease for a certain period, it is used to end the training process. It monitors the loss for 30 consecutive validators and stops if there is no improvement. When it stops, it restores the best weights.
- **Reduce Learning Rate on Plateau Callback:** If the validation loss of the training model, i.e., the improvement, stops, it is used to reduce the learning rate. It also monitors the validation loss and reduces the learning rate from 1e-3 to 2e-4, further reducing it if there is no improvement for 3 consecutive epochs.

After finding the best parameters, the best combination found is used in model training.

Training the Best Model:

- **Epochs:** 200
- **Batch_size:** 32
- **Callbacks:** tb_callback, early_stopping, reduce_lr

The number of epochs is increased and the final model is saved after 200 different scans.

5.2 Evaluation Metrics

The model stopped the model at the 48th epoch due to the early stopping feature and the values obtained and Figure: 5.1, Figure: 5.2 ended with a loss value of 0.0760, categorical_accuarcy 0.98, val_loss 0.3140, val_categorical_accuarcy 0.9644 and learning_rate 2.0480e-11. The model loss and accuracy figures show how stable and accurate it is despite the correct selection of parameters and the low learning rate. If we had continued with more epochs, there would have been overfitting, and due to our parameter setting that controls it, our accuracy rate has become more realistic by obtaining sufficient learning.

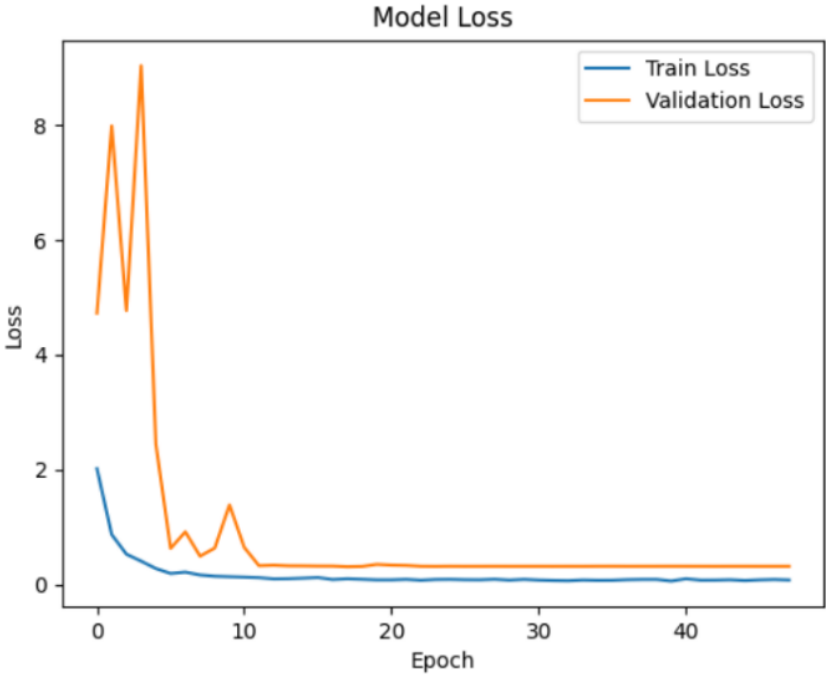

Figure 5.1: Model Loss

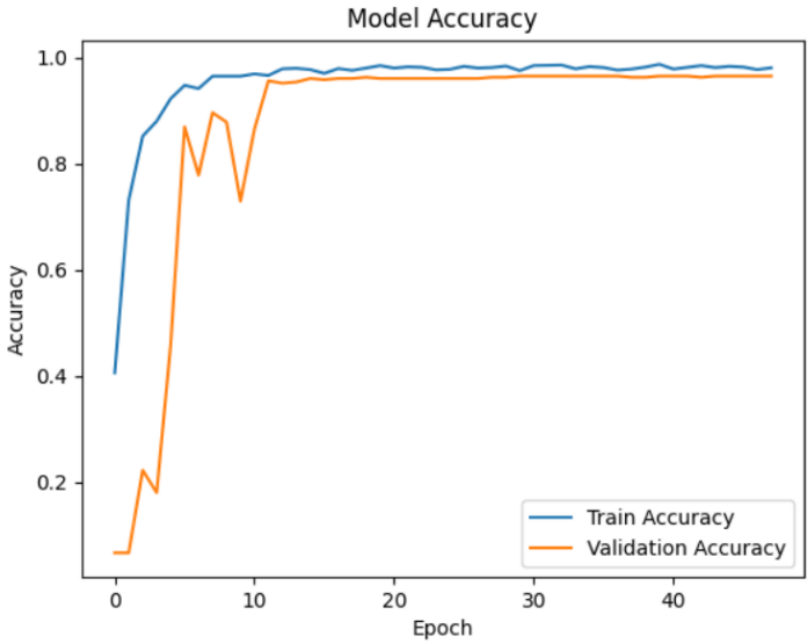

Figure 5.2: Model Accuracy

The results of the complexity matrix created from the model development are shown in Figure: 5.3. There is a problem with the “eight, four, hello, and zero” movements. We need more data to solve this problem. In other movements, the test data added in the training section works very well. It is a big fact that these problems will decrease especially if the data increases.

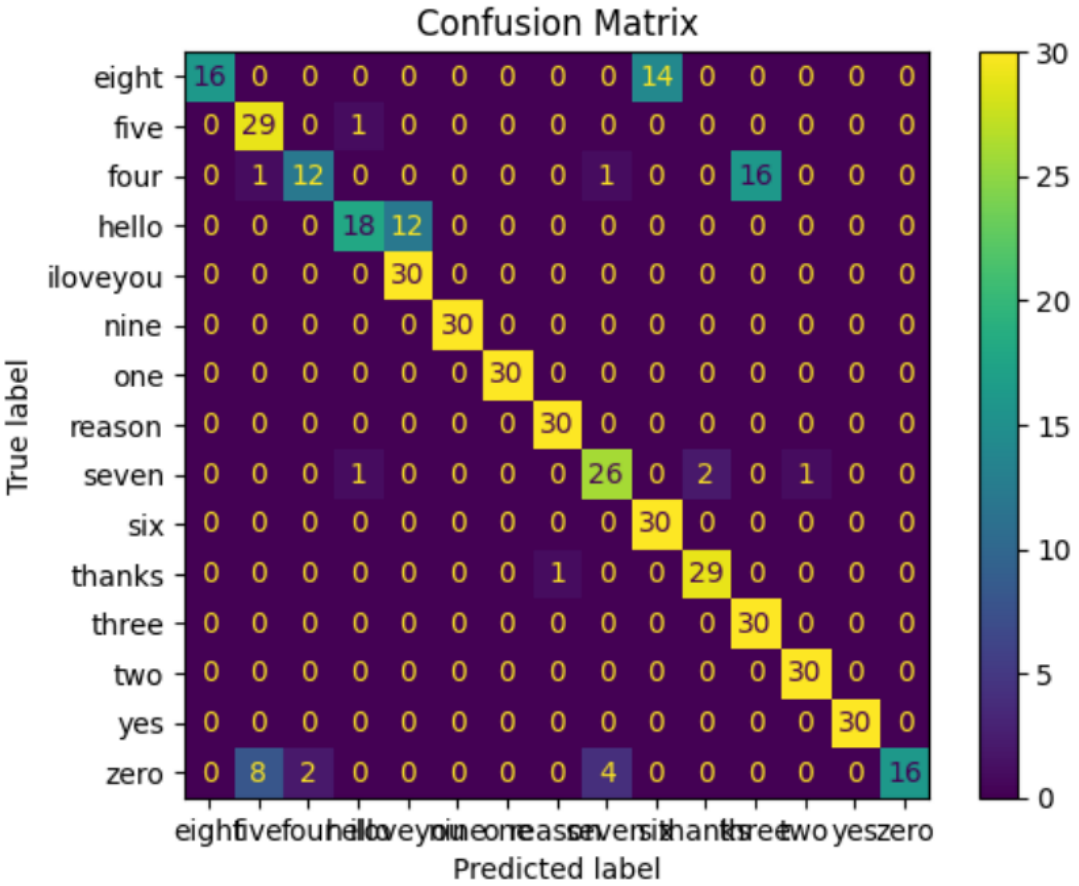

Figure 5.3: Model Confusion Matrix

The test and validation results of the model are as follows. As seen in Figure: 5.4, our accuracy is 0.8578, and as seen in Figure: 5.5, our accuracy is 0.9600.

Figure 5.4: Test Loss and Accuracy

Figure 5.5: Val Loss and Accuracy

The most important part of our validation values is the confusion matrix. In this part, we see a serious truth table except for Figure: 5.6.

As can be seen in the figure, our accuracy rate is significantly high, and great success has been achieved in the "validation" part. Of course, we are not close to 100% accuracy due to the similarity of the "five" and "hello" movements, but as mentioned at the beginning, if we want more accuracy, we need to work with larger data.

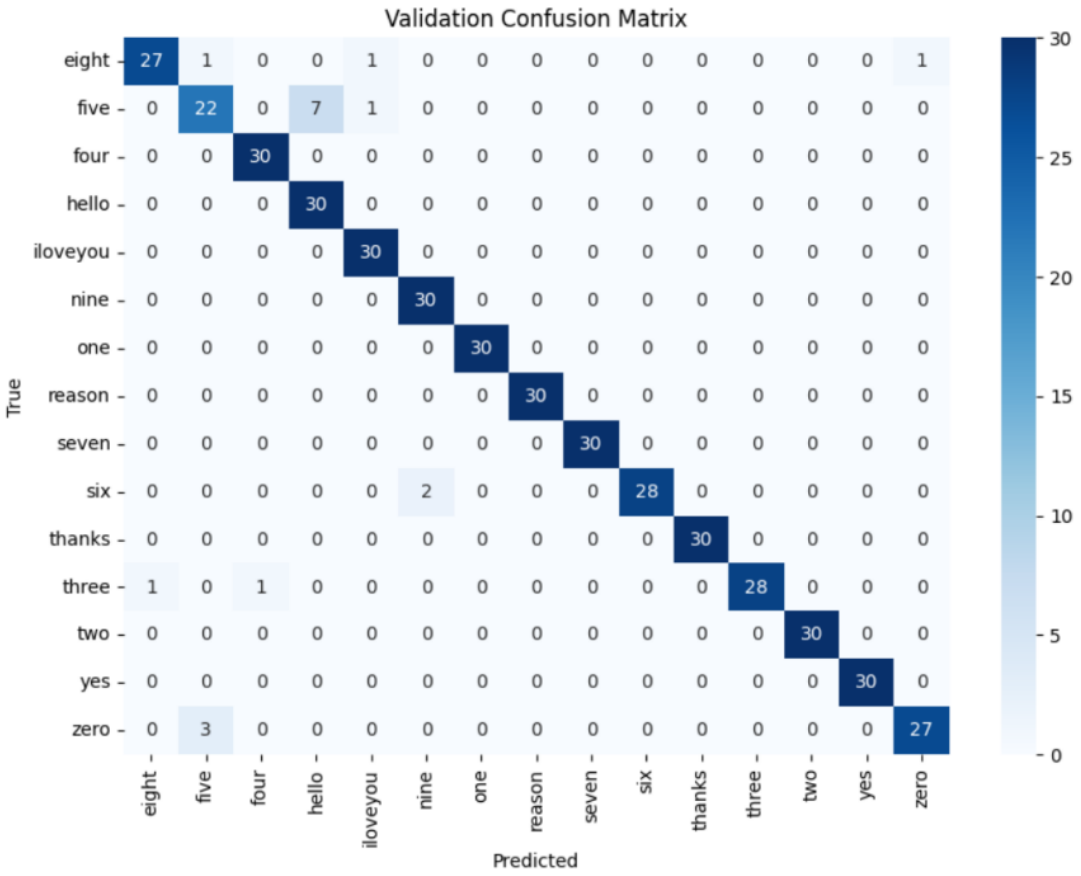

Figure 5.6: Validation Confusion Matrix

```
Validation Precision: 0.9619
Validation Recall: 0.9600
Validation F1-Score: 0.9596

Validation Classification Report:
              precision    recall  f1-score   support

       eight       0.96      0.90      0.93        30
        five       0.85      0.73      0.79        30
        four       0.97      1.00      0.98        30
       hello       0.81      1.00      0.90        30
    iloveyou       0.94      1.00      0.97        30
        nine       0.94      1.00      0.97        30
         one       1.00      1.00      1.00        30
      reason       1.00      1.00      1.00        30
       seven       1.00      1.00      1.00        30
         six       1.00      0.93      0.97        30
      thanks       1.00      1.00      1.00        30
       three       1.00      0.93      0.97        30
         two       1.00      1.00      1.00        30
         yes       1.00      1.00      1.00        30
        zero       0.96      0.90      0.93        30

    accuracy                           0.96       450
   macro avg       0.96      0.96      0.96       450
weighted avg       0.96      0.96      0.96       450

Validation ROC-AUC Score: 0.9885
```

Figure 5.7: Validation Details

Figure: 5.7 for Validation Values:

- **Precision Value:** 0.9619
- **Recall Value:** 0.9600
- **F1-Score:** 0.9596
- **ROC-AUC Value:** 0.9885

Precision: Measures how many of the values the model predicted as positive were positive. True positive / (True Positive + False Positive).

Recall: Measures how many of the true positives it correctly predicted as positive. True Positive / (True Positive + False Negative)

F1-Score: Harmonic mean that balances Precision and Recall. 2 *(Precision * Recall) / (Precision+Recall)

ROC-AUC: The area under the ROC curve measures the discrimination power of the model. It shows the relationship between the model's true positive rate and false positive rate.

With these values, we will be able to make a much more detailed examination of the model we made. With the precision value, we can see how high the precision is 96.19%. With the recall value, we can see how high the sensitivity of the model is at 96%. With the similarity of the ratios of these values, we can see how balanced the F1-Score is 95.96%. Finally, the ROC-AUC model's performance is 98.85%, which shows that the model gives a very good performance in terms of being able to test the positive and negative classes in the finest detail.

Test Values: In the test section, problems such as the model's training are more obvious. "eight, four, hello, zero". It is very obvious that we have a lack of data in these movements Figure: 5.8.

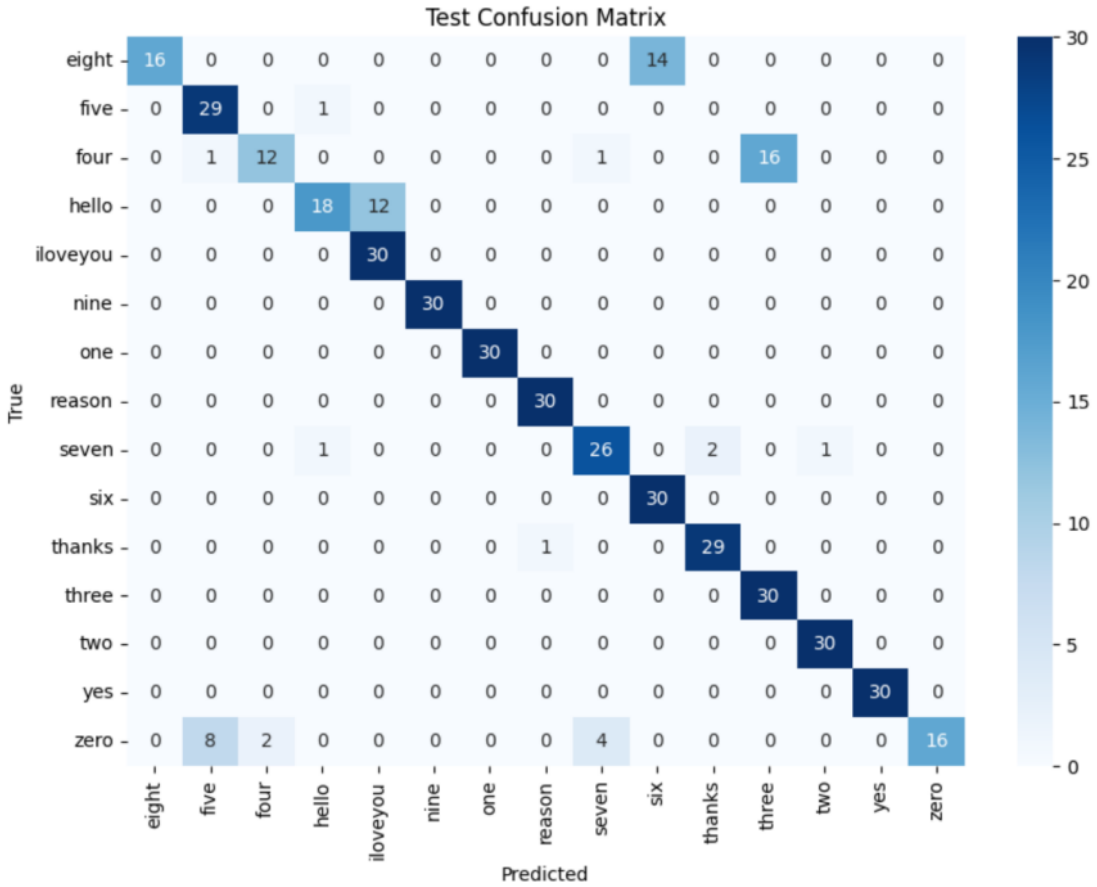

Figure 5.8: Test Confusion Matrix

```
Test Precision: 0.8852
Test Recall: 0.8578
Test F1-Score: 0.8476

Test Classification Report:
              precision    recall  f1-score   support

       eight       1.00      0.53      0.70        30
        five       0.76      0.97      0.85        30
        four       0.86      0.40      0.55        30
       hello       0.90      0.60      0.72        30
    iloveyou       0.71      1.00      0.83        30
        nine       1.00      1.00      1.00        30
         one       1.00      1.00      1.00        30
      reason       0.97      1.00      0.98        30
       seven       0.84      0.87      0.85        30
         six       0.68      1.00      0.81        30
      thanks       0.94      0.97      0.95        30
       three       0.65      1.00      0.79        30
         two       0.97      1.00      0.98        30
         yes       1.00      1.00      1.00        30
        zero       1.00      0.53      0.70        30

    accuracy                           0.86       450
   macro avg       0.89      0.86      0.85       450
weighted avg       0.89      0.86      0.85       450

Test ROC-AUC Score: 0.9915
```

Figure 5.9: Test Details

Figure: 5.9 for Test Values:

- **Precision Value:** 0.8852
- **Recall Value:** 0.8578
- **F1-Score:** 0.8476
- **ROC-AUC Value:** 0.9915

We can see that the values in the test section have decreased. The reason for this is the insufficiency of the data. The validation data was kept almost the same as the training and the test data was taken from similar but different angles. This difference is immediately visible in the test data. We can see that there are serious decreases in the precision, recall, and f1-score table, especially in movements such as "three", "four", "hello", and "zero". These rates decrease when some movements overlap other movements, for example, as we mentioned in the previous figures, "eight" and "three" are quite similar. For this reason, we see decreases in live tests and tests of other data due to insufficient data, this is a natural situation. The solution is assumed to be solved with a very large data set.

5.2.1 Modifications in The Model

A different perspective was tried by removing the Dense layers in the LSTM + Conv1D model architecture and the results were discussed. This change works positively.

The reasons for the change are as follows: Learning movements more clearly by making decisions directly with the outputs of LSTM. The model can work faster because the number of parameters decreases as it has fewer layers.

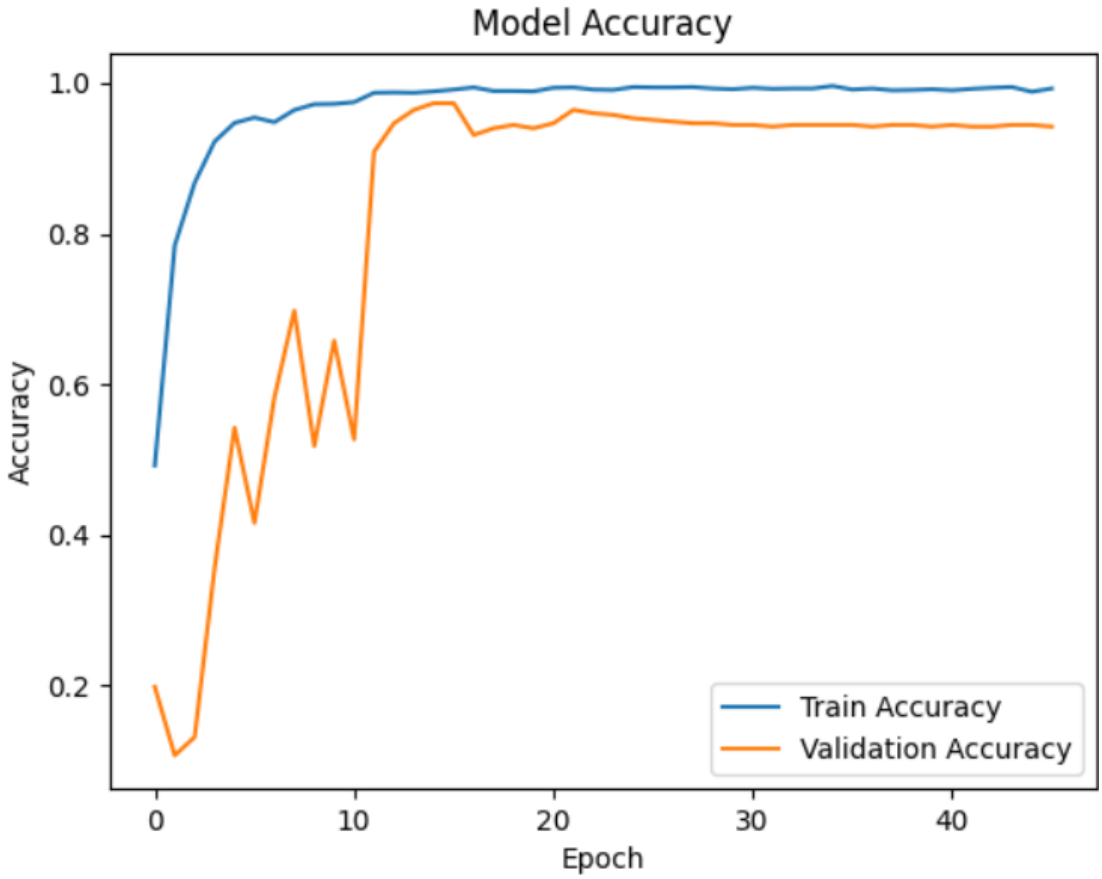

Figure 5.10: No Dense Accuracy

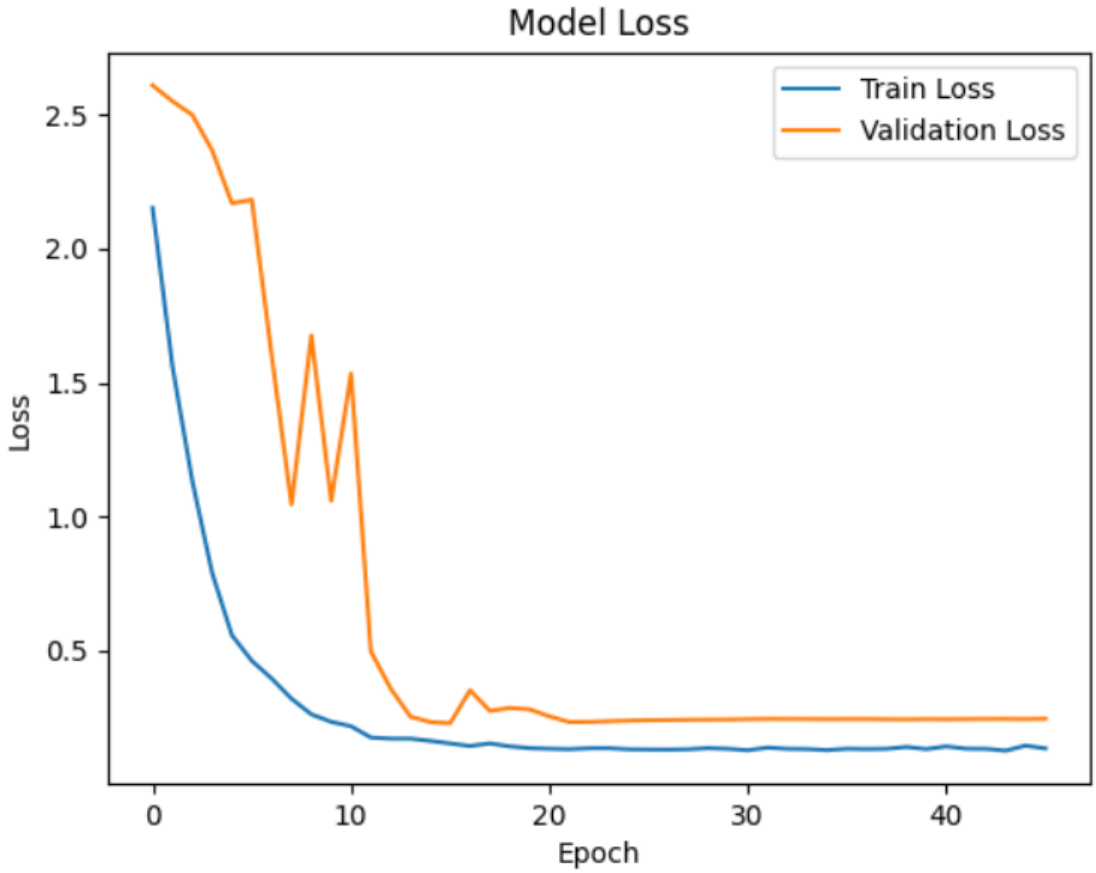

Figure 5.11: No Dense Loss

Compared to our previous model, we see low rates in the loss section, but there is a decreasing variability in accuracy. The advantage of the dense layer is that it reduces accuracy because it makes more important inferences, but although it gives a worse result in live tests, it gives a better solution in the overall result. As seen in the figures 5.10, 5.11, our results have become even better.

As can be seen in Figure:5.12, our data while training is working more optimized than the previous layers, and apart from that, using fewer layers has increased the accuracy of the data considerably. Our model stopped in the 46th epoch due to early stopping and its values are as follows:

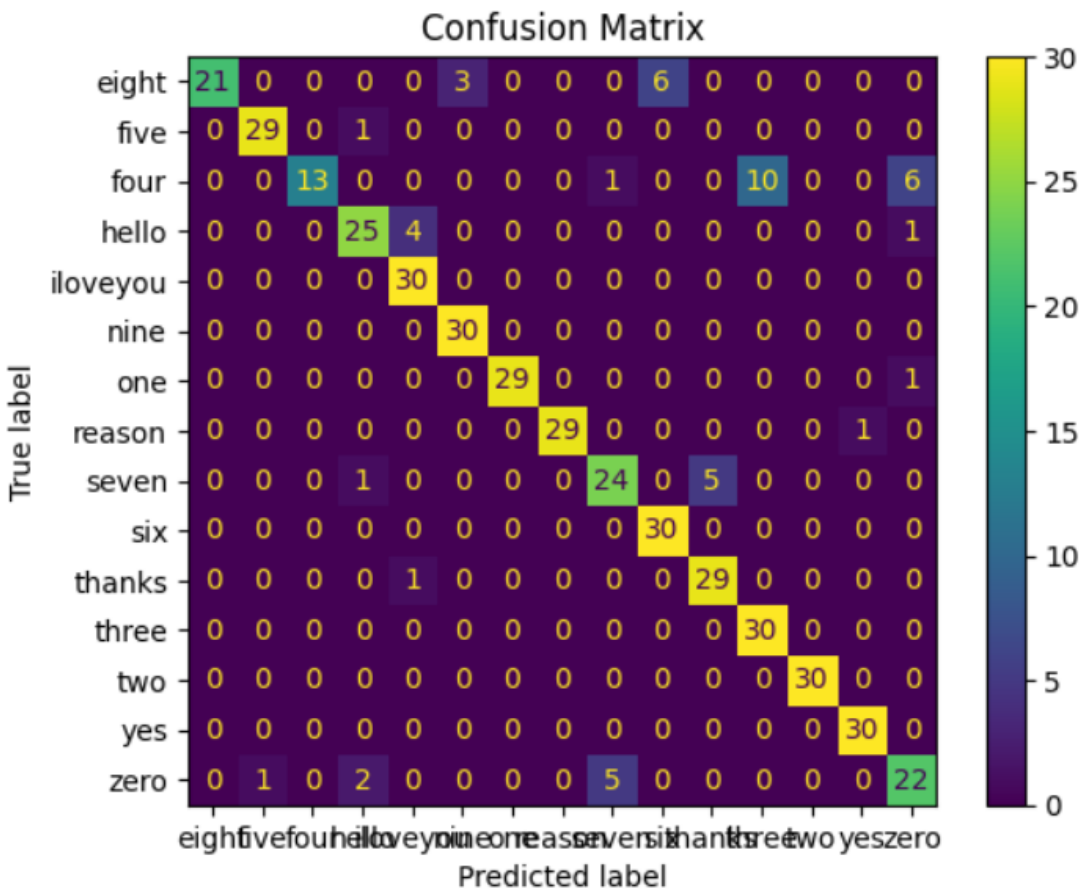

Figure 5.12: No Dense Confusion Matrix

- Loss: 13.44%
- Categorical Accuracy: 99.29%
- Validation Loss: 24.51%
- Validation Categorical Accuracy: 94.22%
- Learning Rate: 1.0240×10^{-11}

When we load additional validation and test data with these values and test again, we get data like this: In this Figure:5.13 we get a very high accuracy except for our "five" label and our values are as follows: Accuracy: 97.33% & Loss: 22.87%

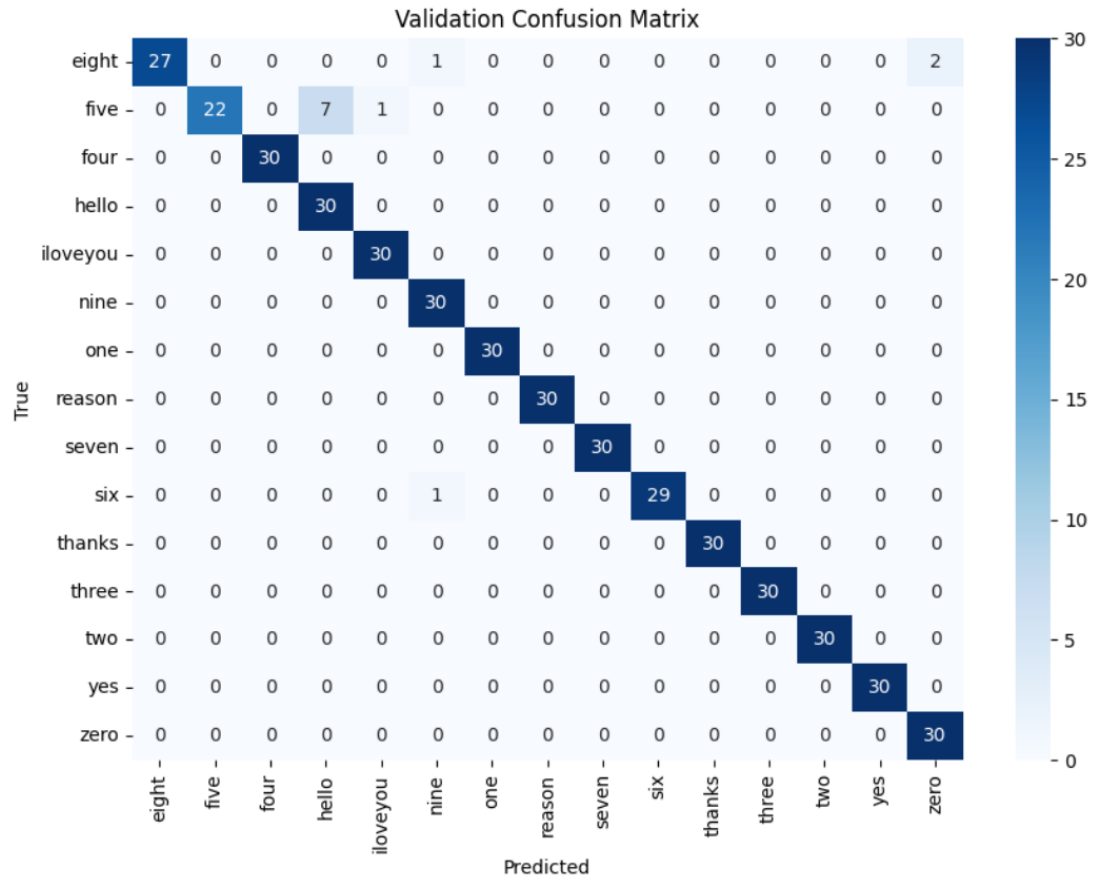

Figure 5.13: No Dense Validation Matrix

```
Validation Precision: 0.9769
Validation Recall: 0.9733
Validation F1-Score: 0.9727

Validation Classification Report:
              precision    recall  f1-score   support

       eight       1.00      0.90      0.95        30
        five       1.00      0.73      0.85        30
        four       1.00      1.00      1.00        30
       hello       0.81      1.00      0.90        30
    iloveyou       0.97      1.00      0.98        30
        nine       0.94      1.00      0.97        30
         one       1.00      1.00      1.00        30
      reason       1.00      1.00      1.00        30
       seven       1.00      1.00      1.00        30
         six       1.00      0.97      0.98        30
      thanks       1.00      1.00      1.00        30
       three       1.00      1.00      1.00        30
         two       1.00      1.00      1.00        30
         yes       1.00      1.00      1.00        30
        zero       0.94      1.00      0.97        30

    accuracy                           0.97       450
   macro avg       0.98      0.97      0.97       450
weighted avg       0.98      0.97      0.97       450

Validation ROC-AUC Score: 0.9930
```

Figure 5.14: No Dense Validation Details

In the figure here, 5.14, we see values such as precision, recall, and F1-score of the data assigned as validation, and we get a seriously positive accuracy margin.

In our challenging test data, we put our model to a serious test with different angles and different hand signals. This time, our output is shown in this figure: 5.15.

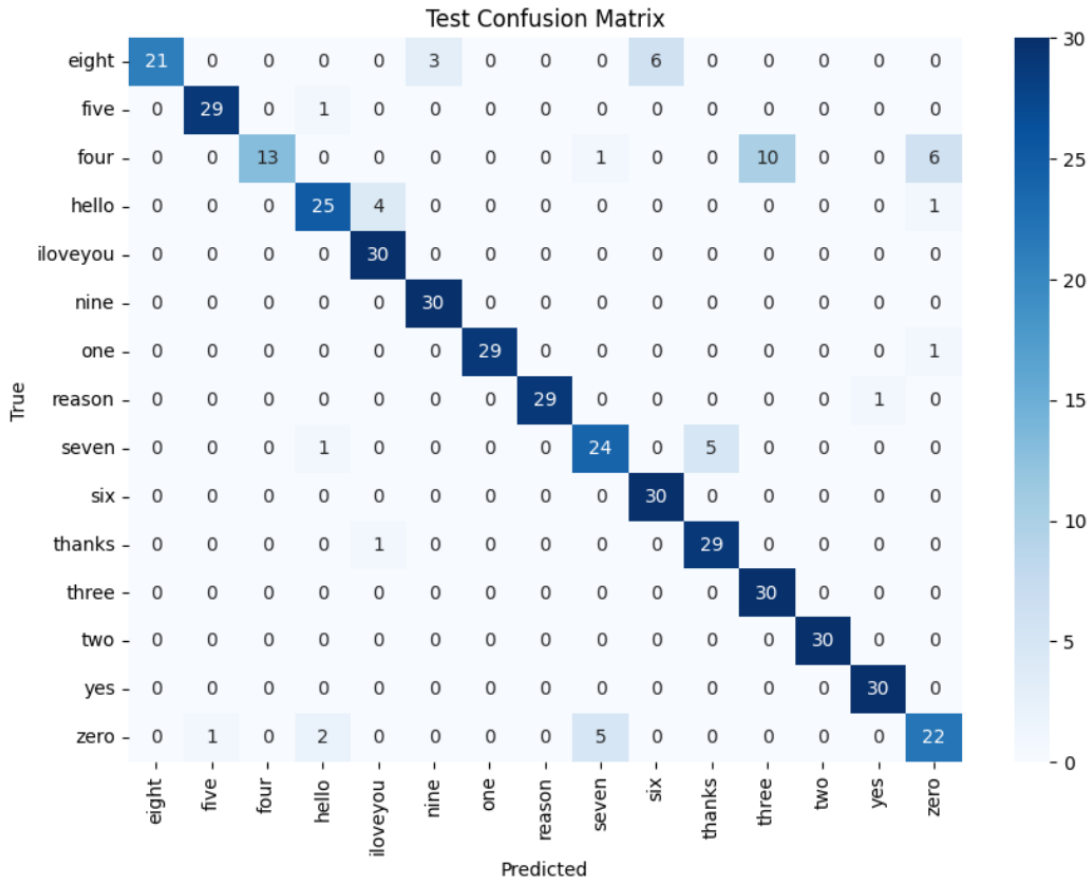

Figure 5.15: No Dense Confusion Matrix

Compared to our previous test data, there is a significant increase in success. Therefore, our data is as follows: Accuracy: 89.11% & Loss: 40.79%

It also appears to be proof that the model has achieved serious success, apart from the "four" movement, and that it works more consistently.

```
Test Precision: 0.9022
Test Recall: 0.8911
Test F1-Score: 0.8849

Test Classification Report:
              precision    recall  f1-score   support

       eight       1.00      0.70      0.82        30
        five       0.97      0.97      0.97        30
        four       1.00      0.43      0.60        30
       hello       0.86      0.83      0.85        30
    iloveyou       0.86      1.00      0.92        30
        nine       0.91      1.00      0.95        30
         one       1.00      0.97      0.98        30
      reason       1.00      0.97      0.98        30
       seven       0.80      0.80      0.80        30
         six       0.83      1.00      0.91        30
      thanks       0.85      0.97      0.91        30
       three       0.75      1.00      0.86        30
         two       1.00      1.00      1.00        30
         yes       0.97      1.00      0.98        30
        zero       0.73      0.73      0.73        30

    accuracy                           0.89       450
   macro avg       0.90      0.89      0.88       450
weighted avg       0.90      0.89      0.88       450

Test ROC-AUC Score: 0.9947
```

Figure 5.16: No Dense Test Details

Finally, in the Figure5.16: we understand more clearly that we have a seriously consistent data structure other than "four and zero" and that the dense layer takes away more than it adds to us.

5.3 Evaluation Metrics Comparison

Validation 96-98% accuracy and testing 85-90% accuracy is obtained compared to other models. This section shows the live test accuracies. Each label was tested 3 times, 0.33 accuracy means 1 of them is correct, 0.67 means 2 predictions are correct, and 1.00 means all predictions are correct. As a result of all tests, we technically get 0.311 accuracy, but the problem here is that the data is small. Otherwise, we would have to take the values in the test and validation sections.

Accuracy Results:

- **eight Accuracy:** 0.00
- **five Accuracy:** 0.33
- **four Accuracy:** 0.00
- **hello Accuracy:** 0.00
- **iloveu Accuracy:** 0.00
- **nine Accuracy:** 1.00
- **one Accuracy:** 0.00
- **Reason Accuracy:** 1.00
- **seven Accuracy:** 0.00
- **six Accuracy:** 0.67
- **thanks Accuracy:** 0.00
- **three Accuracy:** 0.00
- **two Accuracy:** 0.33
- **yes Accuracy:** 1.00
- **zero Accuracy:** 0.33

This data is live test data. Since we aim to make it more comfortable and comprehensive for daily users, it was performed via the laptop camera within these limitations. Live tests are as follows. The camera is opened via Python using CV2, and our already-trained model is integrated into this camera. Just like in the test and validation section, the camera watches you for 2 seconds and shows you a result. The accuracy is generally low because you stand still or are at a different angle while calculating these. To increase the accuracy of the live test, the entire data set needs to be created from videos and this data set needs to be created with much more content. Laptop cameras offer an average of 23 FPS. In my tests, the camera watches you for 2 seconds by fixing 15 FPS dividing it into 2 seconds, and taking the photos accordingly. Finally, it is thought that a model that can think faster can be switched to improve the results of these tests.

Table 5.1: Comparison of Papers on Sign Language Recognition

Paper	**Test Accuracy**	**Dataset Used**
Nguyen Huu Phong [1]	99.44% (ASL Alphabet), 91.82% (5-fold cross-validation)	26 ASL letters, a total of 2600 samples (100 samples per letter)
Songyao Jiang [2]	98.16% (Turkish Sign Language Numbers)	2180 images (Turkish Sign Language numbers, 10 images per student, a total of 218 students)

Paper	Test Accuracy	Dataset Used
Razieh Rastgoo [3]	94.20% (Indian Sign Language, General Sentences), 95.00% (Question Sentences)	A custom dataset, 2 types of sentences classified using an IMU device
Yutong Chen [4]	98.76% (ResNet152 model, Turkish Sign Language)	A new dataset of Turkish Sign Language numbers, details not provided
Anirudh Tunga [5]	97% (Turkish Sign Language)	Turkish Sign Language alphabet classified using Adagrad optimization
Manuel Vázquez-Enríquez [6]	98.85% (ResNet152 model)	Turkish Sign Language, static hand gestures, and a new dataset, details missing
C.K.M. Lee [7]	94% (LSTM + Attention Mechanisms, Turkish Sign Language)	Ankara University Turkish Sign Language Dataset, 36,302 video samples (226 participants)
Lakshmi Murali [8]	97.01% (CNN model, Turkish Sign Language)	Live images captured from a camera, Adagrad optimization algorithm used
Muhammad Al-Qurishi [9]	97.8% (Turkish Sign Language Numbers)	Turkish Sign Language numbers 0-9, a total of 2602 samples

Paper	Test Accuracy	Dataset Used
Yutong Chen [10]	96.67% (Turkish Sign Language, Leap Motion)	Turkish Sign Language movements captured with Leap Motion, details missing
Elif Taşdemir [11]	99% (DNN, Turkish Sign Language)	Leap Motion-based, a total of 390 features used
Mesut Toğaçar [12]	96.6% (CNN + LSTM, American Sign Language)	A custom dataset containing static and dynamic letters was used
Ishak Pacal [13]	92.88% (CNN, Indian Sign Language)	Data collected from 5 different volunteers, different backgrounds, and selfie mode
Ilyas Demir [14]	94% (Mediapipe + LSTM, Turkish Sign Language)	Ankara University Turkish Sign Language Dataset, 20 different backgrounds, and 36,302 videos
Proposed Model	94% Validation + 85% Test with LSTM parameter tuning	A custom Turkish Sign Language Dataset, 15 different gestures, and 3000 videos
Updated Model	98% Validation + 90% Test with LSTM parameter tuning without dense layers	A custom Turkish Sign Language Dataset, 15 different gestures, and 3000 videos

Chapter 6

CONCLUSION

This study successfully demonstrated the application of Long Short-Term Memory (LSTM) networks combined with parameter optimization for Turkish Sign Language action recognition. By integrating advanced feature extraction techniques with Mediapipe and using parameter tuning via random search, the proposed model achieved significant accuracy in recognizing 15 different gestures. This study not only addressed the challenge of sign language recognition in Turkish, a relatively underexplored language but also highlighted the potential of combining state-of-the-art deep learning architectures with systematic optimization to improve gesture classification performance. The findings prove the adequacy of the proposed model by showing sufficient results on both validation and test datasets. However, the dataset is specific but insufficient and shows that further improvements are needed for hand signals that are quite similar to each other. Expanding the dataset to include more participants, various environmental conditions, and higher-resolution data could improve the generalizability and accuracy of the model.

This study successfully demonstrated the application of Long Short-Term Memory (LSTM) networks combined with parameter optimization for Turkish Sign Language action recognition. By integrating advanced feature extraction techniques with Mediapipe and using parameter tuning via random search, the proposed model achieved significant accuracy in recognizing 15 different gestures. This study not only addressed the challenge of sign language recognition in Turkish, a relatively

underexplored language but also highlighted the potential of combining state-of-the-art deep learning architectures with systematic optimization to improve gesture classification performance. The findings prove the adequacy of the proposed model by showing successful results on both validation and test datasets. However, the dataset is specific but insufficient and shows that further improvements are needed for hand signals that are quite similar to each other. Especially by trying different layers, the results we got were more successful and our success values increased to 90 to 98 percent.

In addition, I believe that this study will yield better results with different models to further develop our model and enlighten future studies. I believe that studies can be conducted especially on the integration of Large Language Models (LLMs) and multimodal frameworks and that great work can be done with this dataset. Models that are powerful enough to translate gestures into words or even sentences, such as examples in multimodal contextual understanding (GPT-4 or BERT), are much more suitable for this study. And with these differences, users' communication fluidity can be improved. Especially since these types of models have their data libraries, access to more datasets will be opened and these datasets that people have previously recorded can be integrated into our model more efficiently without having problems, and such studies will yield better results.

As a result, this study has made an important discovery for more comprehensive studies to be developed especially in Turkish Sign Language. Adjusting the dimensions of the layers is a really difficult process and each attempt takes a long time if you have a lot of data. Making the setting you are looking for with the parameters given in the form of lists will save you time at first and allow you to focus more on your work later. With

the special data set used, the accuracy rate was 94-98% in validation and 86-90% in testing, and parameter tuning improved these rates. It is a clear indication that with more data sets it will be better than most improved studies.

REFERENCES

[1] Nguyen Huu Phong and Bernardete Ribeiro, *Action Recognition for American Sign Language*, arXiv preprint arXiv:2205.12261, 2022. CISUC – Department of Informatics Engineering, University of Coimbra, Portugal. [Online]. Available: https://arxiv.org/pdf/2205.12261

[2] Songyao Jiang, Bin Sun, Lichen Wang, Yue Bai, Kunpeng Li, and Yun Fu, *Skeleton Aware Multi-Modal Sign Language Recognition*, arXiv preprint arXiv:2205.12261, 2021. Northeastern University, Boston, MA, USA. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021W/ChaLearn/papers/Jiang_Skeleton_Aware_Multi-Modal_Sign_Language_Recognition_CVPRW_2021_paper.pdf

[3] Razieh Rastgoo, Kourosh Kiani, and Sergio Escalera, *Sign Language Recognition: A Deep Survey*, Expert Systems With Applications, vol. 164, p. 113794, 2021. Available online at Elsevier. [Online]. Available: https://doi.org/10.1016/j.eswa.2020.113794

[4] Yutong Chen, Ronglai Zuo, Fangyun Wei, Yu Wu, Shujie Liu, and Brian Mak, *Two-Stream Network for Sign Language Recognition and Translation*, 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Microsoft Research Asia and The Hong Kong University of Science and Technology, 2022. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/6cd3ac24cdb789beeaa9f7145670fcae-Paper-Conference.pdf

[5] Anirudh Tunga, Sai Vidyaranya Nuthalapati, and Juan Wachs, *Pose-Based Sign Language Recognition Using GCN and BERT*, arXiv preprint arXiv:2105.05073, 2021. Purdue University. [Online]. Available: https://openaccess.thecvf.com/content/WACV2021W/HBU/papers/Tunga_Pose-Based_Sign_Language_Recognition_Using_GCN_and_BERT_WACVW_2021_paper.pdf

[6] Manuel Vázquez-Enríquez, José L. Alba-Castro, Laura Docío-Fernández, and Eduardo Rodríguez-Banga, *Isolated Sign Language Recognition With Multi-Scale Spatial-Temporal Graph Convolutional Networks*, Proceedings of the CVPR 2021 ChaLearn LAP Large Scale Signer Independent Isolated SLR Challenge, 2021. atlanTTic Research Center, Universidade de Vigo, Spain. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2021W/ChaLearn/papers/Vazquez-Enriquez_Isolated_Sign_Language_Recognition_With_Multi-Scale_Spatial-Temporal_Graph_Convolutional_Networks_CVPRW_2021_paper.pdf

[7] C.K.M. Lee, Kam K.H. Ng, Chun-Hsien Chen, H.C.W. Lau, S.Y. Chung, and Tiffany Tsoi, *American Sign Language Recognition and Training Method with Recurrent Neural Network*, Expert Systems with Applications, vol. 167, article 114403, 2021. Available online at Elsevier. [Online]. Available: https://doi.org/10.1016/j.eswa.2020.114403

[8] Romala Sri Lakshmi Murali, L.D. Ramayya, and V. Anil Santosh, *Sign Language Recognition System Using Convolutional Neural Network And Computer Vision*, International Journal of Engineering Innovations in

Advanced Technology, vol. 4, issue 4, 2022. ISTS Women's College, Andhra Pradesh, India. [Online]. Available: https://ijeiat.com/images/sliders/92dc4915b9487f589cfe29a2d410cc0a.pdf

[9] Muhammad Al-Qurishi, Thariq Khalid, and Riad Souissi, *Deep Learning for Sign Language Recognition: Current Techniques, Benchmarks, and Open Issues*, IEEE Access, vol. 9, pp. 126917–126931, 2021. Elm Company, Riyadh, Saudi Arabia. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9530569&tag=1

[10] Yutong Chen, Fangyun Wei, Xiao Sun, Zhirong Wu, and Stephen Lin, *A Simple Multi-Modality Transfer Learning Baseline for Sign Language Translation*, Neural Information Processing Systems (NeurIPS) 2022. Microsoft Research Asia and Tsinghua University. [Online]. Available: https://openaccess.thecvf.com/content/CVPR2022/papers/Chen_A_Simple_Multi-Modality_Transfer_Learning_Baseline_for_Sign_Language_Translation_CVPR_2022_paper.pdf

[11] Elif Taşdemir, Uğur Talaş, and Burakhan Çubukçu, *Evrişimli Sinir Ağları kullanılarak Türk İşaret Dili Alfabesinin Tespit Edilmesi*, Rahva Teknik ve Sosyal Araştırmalar Dergisi, vol. 4, issue 2, pp. 1–10, 2024. Bilecik Şeyh Edebali Üniversitesi, Bilgisayar Mühendisliği. [Online]. Available: https://dergipark.org.tr/en/download/article-file/4242387

[12] Mesut Toğaçar, Zafer Cömert, and Burhan Ergen, *Siyam Sinir Ağlarını Kullanarak Türk İşaret Dilindeki Rakamların Tanımlanması*, DEÜ Fen ve

Mühendislik Dergisi, vol. 23, no. 68, pp. 349–356, 2021. [Online]. Available: https://dergipark.org.tr/tr/download/article-file/793273

[13] Ishak Pacal and Melek Alaftekin, *Türk İşaret Dilinin Sınıflandırılması için Derin Öğrenme Yaklaşımları*, Iğdır Üniversitesi Fen Bilimleri Enstitüsü Dergisi, vol. 13, no. 2, pp. 760–777, 2023. [Online]. Available: https://dergipark.org.tr/en/download/article-file/2849310

[14] İlyas Demir, İsa Peker, and Serkan Savaş, *İşaret Dilinin Dijital Sesi: Mediapipe ve LSTM Uygulaması*, EU 4th International Conference on Health, Engineering and Applied Sciences, April 12–14, 2024, Paris. Academy Global Publishing House, ISBN: 978-625-6283-01-5. [Online]. Available: https://www.researchgate.net/publication/380727942

APPENDICES

Appendix A: Data Flow Process – From Collection To Model Training

This appendix details the end-to-end pipeline for processing Turkish Sign Language (TSL) data, from raw video collection to model training.

First, 200 videos were shot for each label via the laptop camera. These videos were recorded at 15 frames per second with a size of 640x480 pixels. This is because it is known that a good laptop camera can go up to 30 frames per second, but an average laptop camera has 23 frames per second features. In order not to strain the system while shooting the videos and to make mathematical calculations more easily, I shot 3000 videos myself at 15 frames per second. After all the videos were shot, only the names of these videos were changed with a simple Python code. For example, 200 videos were recorded while the "eight" gesture was made and 200 videos were kept in a folder as "video1.mp4-video200.mp4". As seen in the Figure:A.1.

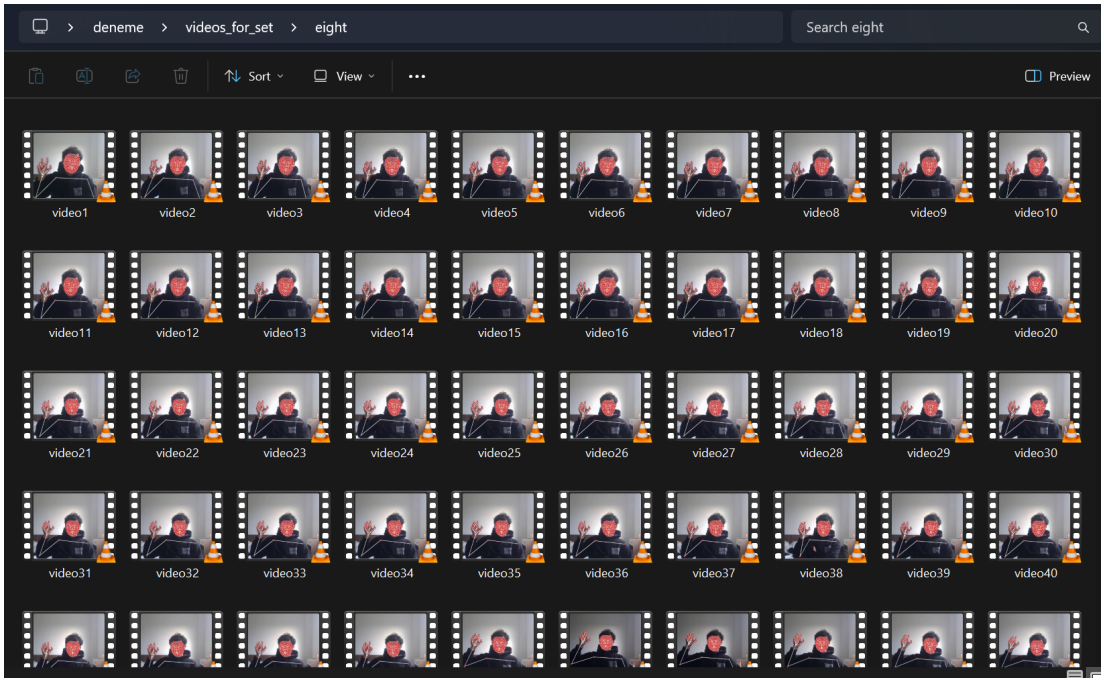

Figure A.1: Videos

After adding the videos to the folders, we need to extract features from them. The library that will do this is mediapipe holistic, which has done feature extraction. All videos, namely 3000 videos, 200 for each label, 140 of which are for training, 30 videos for testing, and the remaining 30 videos for validation. As seen in the figure here A.2. Again, the data set was separated with a simple Python command.

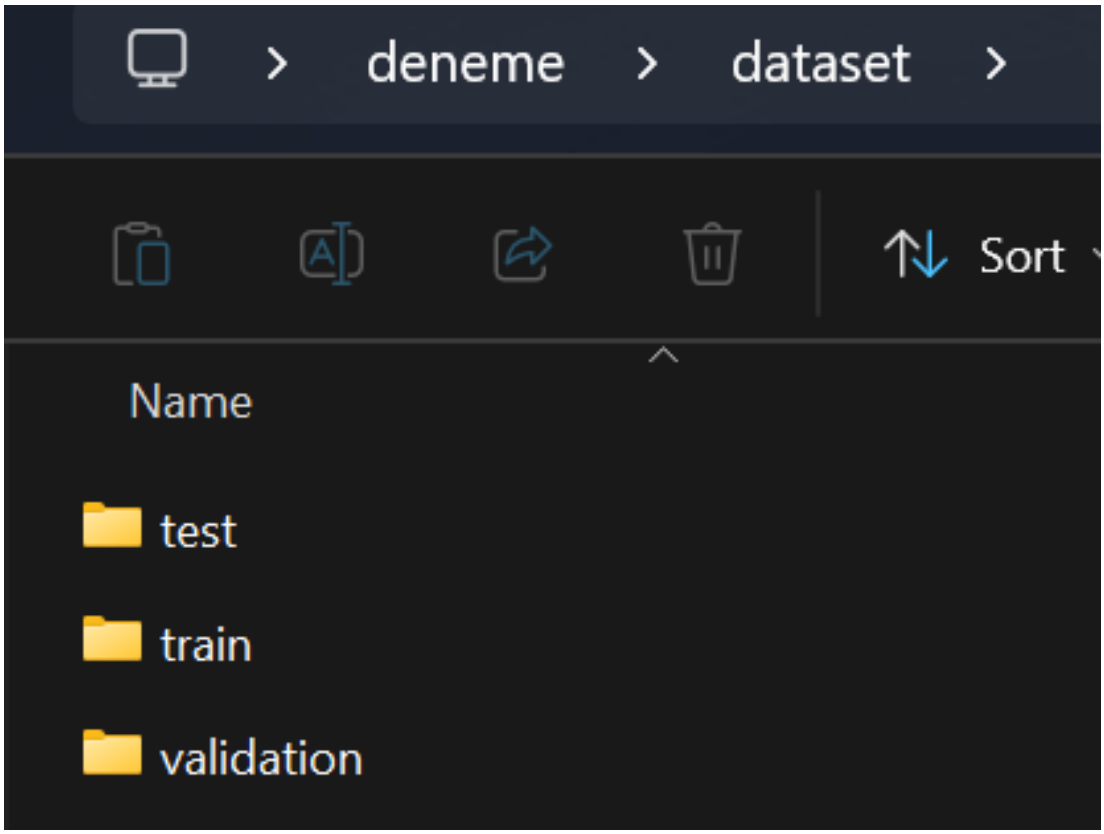

Figure A.2: Split Dataset

For example, in this figure, where we have 15 labels in the test folder, figure:A.3 are listed alphabetically. Again, simple Python commands were used for folding and naming.

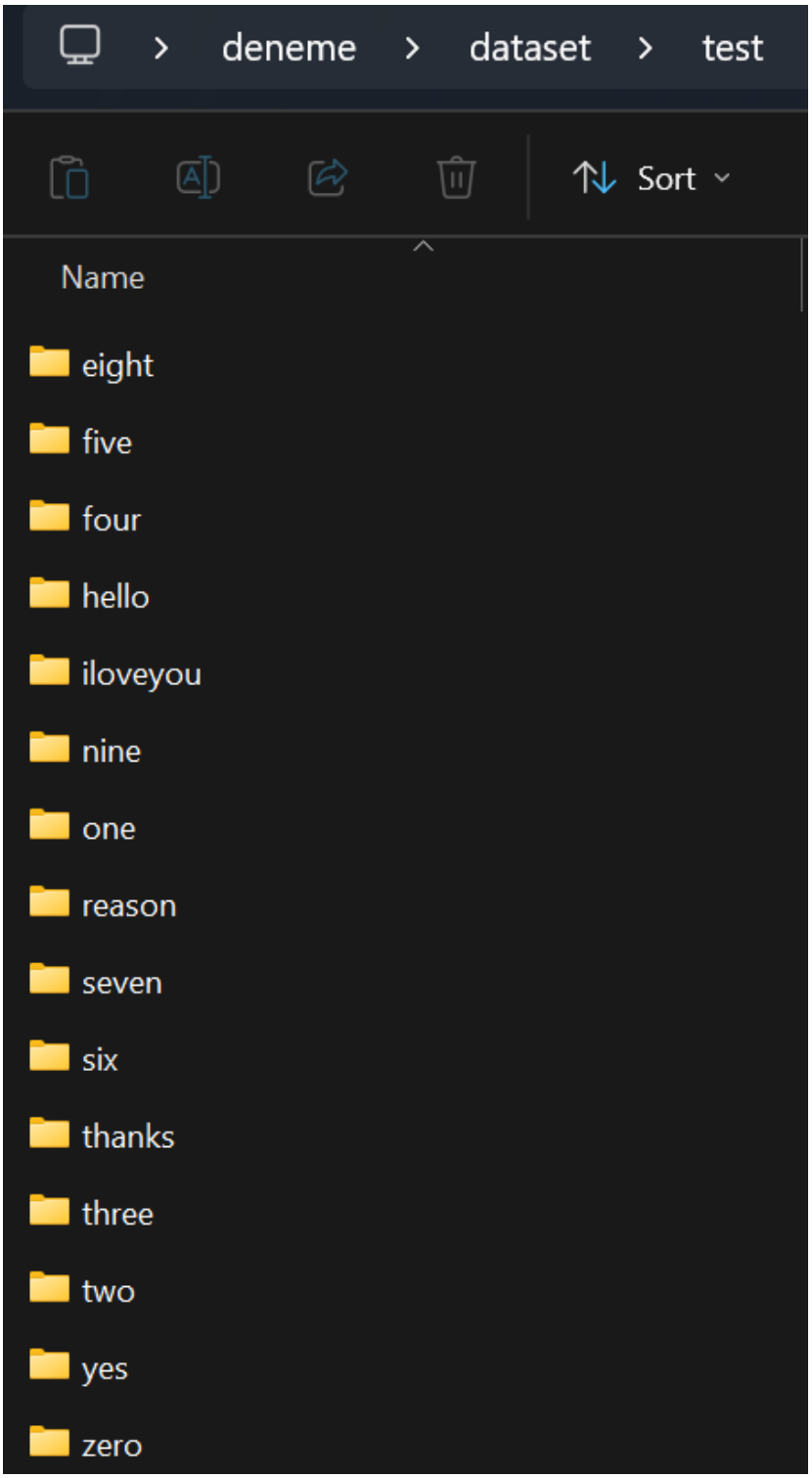

Figure A.3: Labels

For example, we see folders starting with 171 and ending with 200 inside the "eight" label. These are the last 30 videos out of the 200 created and put into folders, as seen in the figure A.4. The first 140 videos were added to the train folder, the next 30 videos to the validation folder, and the last 30 videos to the test folder.

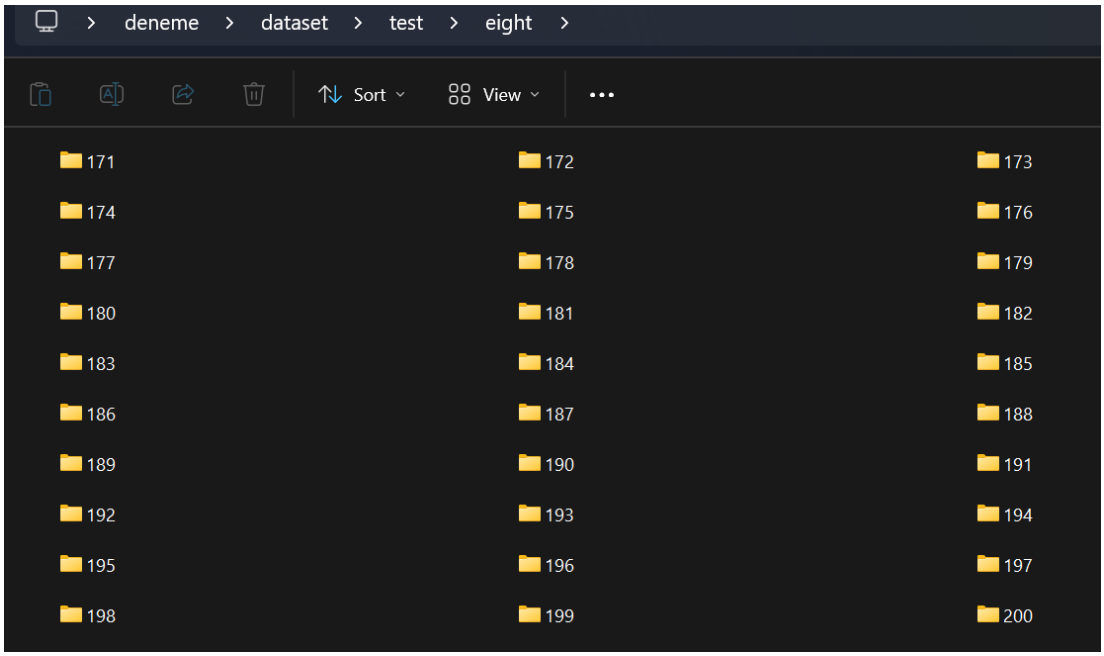

Figure A.4: Test Eight

Finally, we see that the photos in the folders here, which have been feature extracted with mediapipe holistic, are divided into 30 frames as "frame0.npy-frame29.npy". FigureA.5. This process is done with a very simple Python command.

After this part, we need to prepare this dataset to train the model in the coding part. In the figure here, A.6, we save all the frames one by one from the "dataset" folder of our data as "X-train, y-train, X-val, y-val, X-test, y-test". This code needs to be run every time the model is trained because how can we train the model where there is no dataset? After this part, we only need to enter the code that the model needs to be trained.

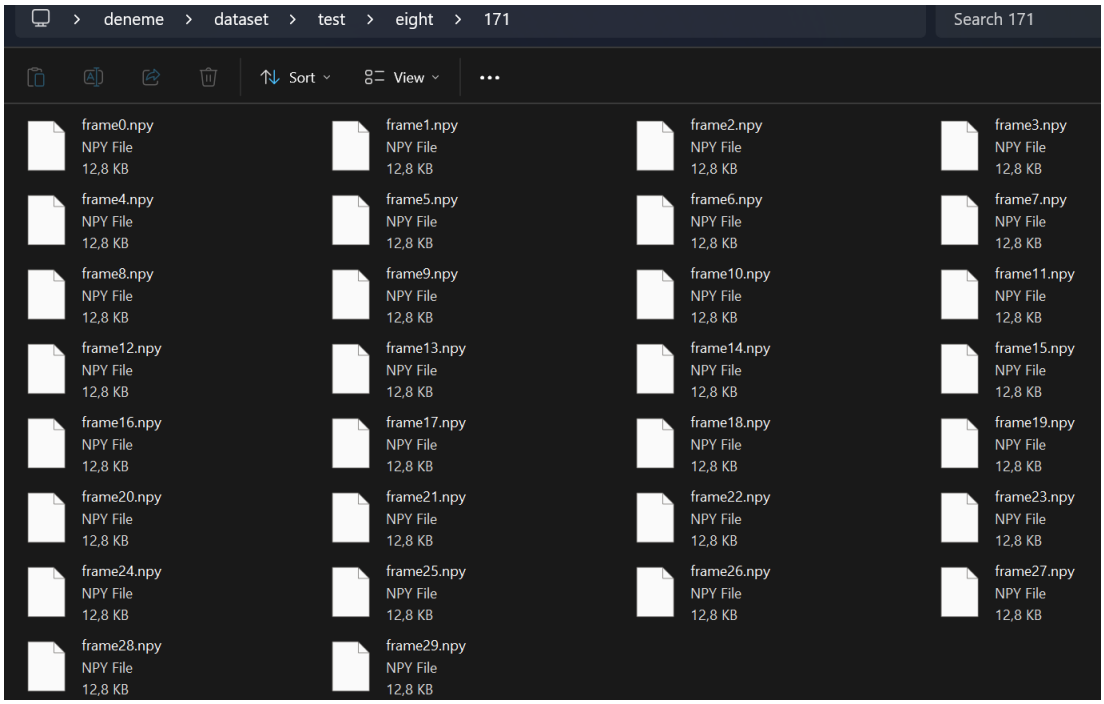

Figure A.5: Frame to Frame

```python
import os
import numpy as np
from tensorflow.keras.utils import to_categorical

DATASET_PATH = 'dataset'
sequence_length = 30

actions = sorted(os.listdir(os.path.join(DATASET_PATH, 'train')))
print("Actions:", actions)

label_map = {label: num for num, label in enumerate(actions)}
print("Label Map:", label_map)

def load_data(subset):
    sequences, labels = [], []
    subset_path = os.path.join(DATASET_PATH, subset)
    for action in actions:
        action_path = os.path.join(subset_path, action)
        for sequence in os.listdir(action_path):
            sequence_path = os.path.join(action_path, sequence)
            window = []
            for frame_num in range(sequence_length):
                npy_path = os.path.join(sequence_path, f"frame{frame_num}.npy")
                if os.path.exists(npy_path):
                    res = np.load(npy_path)
                    window.append(res)
            if len(window) == sequence_length:
                sequences.append(window)
                labels.append(label_map[action])
    return np.array(sequences), to_categorical(labels).astype(int)

X_train, y_train = load_data('train')
X_val, y_val = load_data('validation')
X_test, y_test = load_data('test')

print(f"Train shape: {X_train.shape}, Validation shape: {X_val.shape}, Test shape: {X_test.shape}")
```

Figure A.6: Load Dataset