ASL Alphabet Recognition Using CNN and MLP

Artificial Intelligence Project Report

Authors:

Benjamin Bernedo Rendel

Thomas Capelletti

Claudio Pietro Pasina

Date:

February 13, 2025

Table of Contents

Chapter 1

Abstract

The purpose of this project is to develop an application capable of recognizing the American Sign Language (ASL) alphabet using artificial intelligence techniques. ASL is a visual language that utilizes hand gestures, facial expressions, and body movements to convey meaning. It was developed by Thomas Hopkins Gallaudet and Laurent Clerc in the early 19th century and has since become the primary language of communication for the Deaf community in North America.

The main goal is to create a system that can interpret ASL letters from live webcam input, converting them into text. This would enable users to "write" using hand signs, allowing for seamless interaction between sign language users and non-signers. The application leverages deep learning techniques such as Convolutional Neural Networks (CNN) and Multi-Layer Perceptrons (MLP) to achieve accurate recognition.

Various approaches were explored to enhance recognition accuracy. First, publicly available ASL datasets, such as those found on Kaggle, were utilized to train a CNN model. To improve adaptability, custom datasets were created under different conditions: fixed backgrounds, moving backgrounds, and hand-tracking preprocessing using MediaPipe. Additionally, another model was trained using extracted node coordinates from MediaPipe, processed through an MLP network.

This technology has significant practical applications. It can be used as an assistive tool for individuals with speech impairments, facilitating communication with non-signers. Moreover, it could be integrated into educational tools to help learners acquire sign language more efficiently. With real-time webcam integration, the system aims to bridge the communication gap and increase accessibility for the Deaf community.

A possible improvement to this project would be expanding the number of recognized gestures beyond the ASL alphabet. By incorporating additional signs and words, the system could be extended to support full ASL sentence recognition, further enhancing communication capabilities and usability.

Chapter 2

Methodology: Convolutional Neural Networks and Multi-Layer Perceptron

2.1 Convolutional Neural Networks

A **Convolutional Neural Network (CNN)** is a type of deep learning model that excels at processing and classifying images due to its ability to automatically extract spatial features. CNNs are typically composed of four main components:

1. **Input Layer**: This layer receives the raw pixel values of an image, which serve as the input to the network.

2. **Convolutional Blocks**: Each convolutional block consists of three key operations:

   - **Convolutional Layers**: These layers apply a set of filters (kernels) to the input image, performing element-wise multiplications followed by summation. This operation extracts low-level features such as edges, curves, and textures.
   - **Activation Function**: The feature maps produced by the convolutional layers pass through a non-linear activation function, typically ReLU (Rectified Linear Unit), which introduces non-linearity, accelerates convergence, and improves the model's ability to capture complex patterns.
   - **Pooling Layers**: Pooling is a downsampling technique that reduces the spatial dimensions of the feature maps while preserving the most relevant information. The most common types are Max Pooling, which selects the maximum value in a given region, and Average Pooling, which computes the average value.

3. **Deepening the Network**: Additional convolutional blocks are applied sequentially, typically increasing the number of filters to detect more abstract and complex features at deeper levels. This hierarchical feature extraction enables CNNs to recognize patterns at multiple levels of abstraction.

4. **Fully Connected Layers**: After passing through the convolutional blocks, the extracted features are flattened into a one-dimensional vector. This vector is then processed by one or more fully connected layers, where each neuron is connected to all previous activations. The final layer produces the output vector, which represents the network's classification decision.

CNNs leverage these structured layers to progressively transform raw pixel data into meaningful representations, enabling robust and efficient image classification.

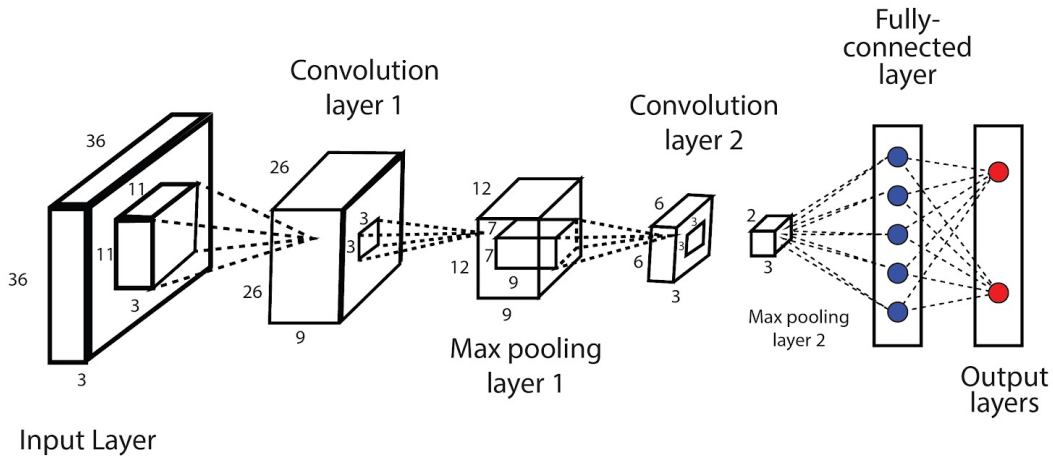


Figure 2.1: Diagram of a CNN architecture

2.2 Multi-Layer Perceptron

A **Multi-Layer Perceptron (MLP)** is one of the most basic types of neural networks. It consists of three main components:

1. **Input Layer**: This is the first layer where the data is introduced into the network. Each feature in the input corresponds to a neuron in this layer.

2. **Hidden Layers**: These layers consist of multiple neurons operating in parallel. Each neuron receives inputs from all neurons in the previous layer (whether it is another hidden layer or the input layer). The process follows these steps:

   - Each neuron receives the weighted sum of all outputs from the previous layer.
   - A bias term is added to the sum.
   - The result is passed through an activation function to introduce non-linearity.

   The output of each neuron is then propagated to all neurons in the next layer, and this process is repeated for every hidden layer.

3. **Output Layer**: The final layer of the network, which produces the model's predictions. It operates similarly to hidden layers but may use a different activation function depending on the nature of the task:

   - For classification, a softmax activation is often used to generate probability distributions over classes.
   - For binary classification, a sigmoid activation function is commonly applied.
   - For regression, a linear activation function may be used to produce continuous values.

MLPs rely on the process of backpropagation and gradient descent to adjust weights and biases in order to minimize prediction errors and improve learning over time.

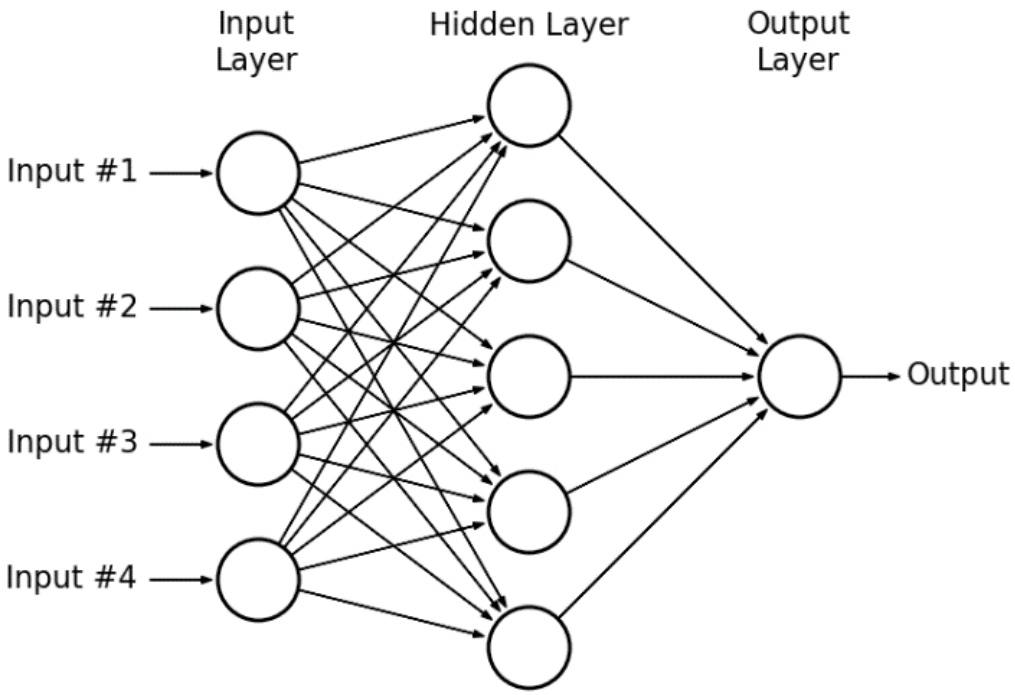


Figure 2.2: Diagram of an MLP architecture

Chapter 3

Previous Models

3.1 Model with Kaggle Dataset (CNN)

3.1.1 Dataset

The "ASL Alphabet" dataset available on Kaggle (`https://www.kaggle.com/datasets/grassknoted/asl-alphabet/data`) was used in this first model. It is a collection of around 89.000 images depicting the letters of the alphabet in American Sign Language (ASL), organized into 29 folders, each representing a different class corresponding to a specific letter of the alphabet or gesture. The images were acquired under controlled conditions, with uniform backgrounds and constant illumination, ensuring high data quality. This dataset is widely used for image classification and gesture recognition projects in the artificial intelligence community.

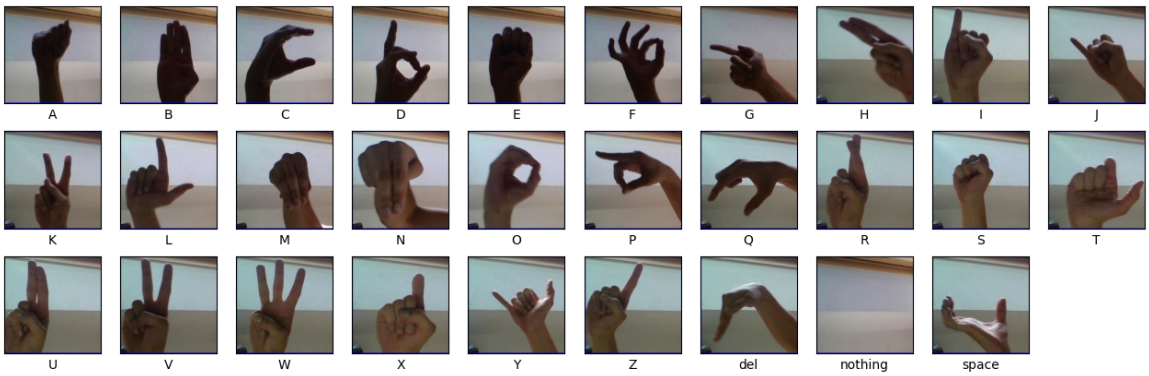


Figure 3.1: ASL labels from the Kaggle Dataset

3.1.2 Data Processing and Model Training

In the dataset, each letter has its own folder. We take the images from each of these sub folders, reshape them into a 32×32 image and transform them into a three channel ($32 \times 32 \times 3$) image, ensuring to preserve the RGB properties. The folder index is also extracted for the creation the labels for the arrays.
All this is done in the `get_data(data_dir)` function. it works by reshaping, turning into an array and assigning a label (based on the index of the folder it is), for every image inside a specific letter folder, and for every letter folder that are sorted inside the master database folder.

Data Visualization and Sample Images

In this phase, the code visualizes the images and labels to provide an overview of the dataset's quality and distribution. The number of images and labels loaded is printed to confirm successful data loading and correct structure.
The function `plot_sample_images()` is then used to display a sample of images from the 29 classes, including the ASL alphabet (A-Z) and special labels such as "del," "nothing," and "space." .
It iterates through the classes, displaying one image per class with its corresponding label below. This visualization helps verify the dataset's integrity before model training. This step is also done in future models, however, since it does not affect the performance of the model it's not mentioned in future sections.

Normalization and One-Hot Encoding

In this phase, the function `preprocess_data(X, y)` processes the dataset for training. First, the images are converted into a NumPy array and normalized to the range [0, 1] by dividing the pixel values by 255. The labels are then one-hot encoded using the `to_categorical()` function from Keras.
The dataset is split into training and testing sets using `train_test_split()`, with 90% for training and 10% for testing. The function prints the shapes of the resulting datasets to confirm the correct split and returns the processed data ready for model training.

Neural Network Architecture

The neural network used for this task is a Convolutional Neural Network (CNN), which is particularly suitable for image classification problems. Below, we describe the architecture of the network, layer by layer, and explain the role of each part of the network.
First, we define the key training parameters, such as the number of classes, batch size, epochs, and learning rate, which are essential for configuring the training process.
The classes parameter is set to 29 because our dataset contains 29 distinct classes that we want the model to classify. Each class corresponds to a unique label. This is critical for ensuring the model learns to differentiate between all the categories in the dataset.
The batch size of 32 is chosen to strike a balance between computation efficiency and the ability of the model to generalize well. A smaller batch size typically offers better generalization but comes at the cost of longer training times, while a larger batch size can speed up training but may risk overfitting.
The model will be trained for 15 epochs. This number is chosen based on experimentation and can be adjusted depending on the performance observed during training. Training for too few epochs may result in underfitting, while too many epochs can lead to overfitting.
The learning rate is set to 0.01. This value is chosen as a starting point based on common practice for training deep neural networks, particularly with the Adam optimizer, which adjusts the learning rate during training. If needed, this can be fine-tuned further.
The optimizer chosen is Adam, which is known for its efficiency in training deep neural networks. Adam adapts the learning rate throughout training, making it particularly useful for networks with large and complex architectures.
Additionally, an EarlyStopping callback is implemented to prevent overfitting. This callback will monitor the validation loss during training and stop the process if the validation loss does not improve for a specified number of epochs.
The EarlyStopping callback is a critical tool for avoiding unnecessary training once the model has stopped improving on the validation data. The patience parameter is set to 5, meaning

that if there is no improvement in the validation loss for 5 consecutive epochs, the training process will halt early. This prevents the model from overfitting by continuing to train on data that is no longer providing useful information.
Now, we will describe the architecture of the CNN in more detail, layer by layer. The model is defined using Keras' Sequential API, which allows for stacking layers in a linear fashion.

1. **Input Layer:** The first layer in the model is the input layer. This layer accepts images of size 32x32 pixels with 3 color channels (RGB):

```
model.add(Input(shape=(32, 32, 3)))
```

Listing 3.1: Input Layer

The input layer is responsible for feeding the input data into the network. We specify the input shape as (32, 32, 3) because the images are resized to 32x32 pixels with 3 color channels (RGB). This ensures that the network receives images in the correct format. The choice of 32×32 pixels is a trade-off between sufficient image detail and computational efficiency.

2. **First Convolutional Block:** The first convolutional block consists of a Conv2D layer, followed by MaxPooling2D and BatchNormalization. The Conv2D layer uses 64 filters with a kernel size of 3x3 and applies the ReLU activation function. Padding is set to 'same' to preserve the spatial dimensions of the input:

```
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
```

Listing 3.2: First Convolutional Block

This block performs the following steps:

- **Convolution**: The convolutional layer extracts features from the input image. The 64 filters learn different spatial features such as edges, textures, and simple shapes. The choice of 64 filters is a common starting point for feature extraction in image data.
- **Max Pooling**: The max pooling operation reduces the spatial size of the output feature maps by taking the maximum value from a 2x2 window. This operation helps in reducing the dimensionality of the data, retaining the most important features while improving computational efficiency.
- **Batch Normalization**: This step normalizes the activations of the previous layer to improve training stability and speed. Normalization helps in reducing internal covariate shift, enabling faster convergence during training.

3. **Second Convolutional Block:** The second block is similar to the first, but with 128 filters and a Dropout layer to mitigate overfitting. The Dropout rate is set to 20%.

```
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
```

Listing 3.3: Second Convolutional Block

The additional Dropout layer randomly sets 20% of the neurons to zero during training, preventing the model from overfitting by encouraging the network to generalize better. Dropout is particularly useful in preventing the network from becoming too reliant on specific neurons, thereby improving generalization.

4. **Third Convolutional Block:** The third block increases the number of filters to 256, continuing the process of feature extraction, followed by max pooling and batch normalization:

```
    model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(BatchNormalization())
```

Listing 3.4: Third Convolutional Block

With 256 filters, this block learns more complex patterns in the input data, and max pooling further reduces the size of the feature maps. The increasing number of filters in each successive block allows the network to learn increasingly abstract and complex features.

5. **Fully Connected Layers:** Once the convolutional layers have extracted the features, the output is flattened into a 1D vector before passing it to the fully connected layers. The first fully connected layer has 1024 neurons and uses the ReLU activation function. Additionally, a Dropout layer is applied:

```
    model.add(Flatten())
    model.add(Dropout(0.2))
    model.add(Dense(1024, activation='relu'))
```

Listing 3.5: Fully Connected Layers

The Flatten layer converts the output from the convolutional blocks (which is a 3D tensor) into a 1D vector that can be fed into the fully connected layers. The Dropout layer is included to help with regularization, and the Dense layer with 1024 neurons allows for high-level feature combinations. The choice of 1024 neurons provides the model with a sufficient capacity to learn complex combinations of the extracted features.

6. **Output Layer:** Finally, the output layer consists of 29 neurons, one for each class, and uses the softmax activation function to produce a probability distribution over the classes:

```
    model.add(Dense(classes, activation='softmax'))
```

Listing 3.6: Output Layer

The Softmax activation function ensures that the output values are between 0 and 1, representing the probability of each class. The class with the highest probability will be selected as the predicted label. The use of Softmax is standard for multi-class classification tasks as it allows the network to output a valid probability distribution.
In summary, the network consists of three convolutional blocks, each followed by a max pooling and batch normalization layer, and a series of fully connected layers at the end. The use of Dropout throughout the network helps prevent overfitting, while the Softmax activation in the output layer ensures proper classification into one of the 29 possible classes.
The optimizer used for training the model is Adam, which adapts the learning rate during training to make the optimization process more efficient. The EarlyStopping callback ensures that training will halt if the validation loss stagnates, avoiding unnecessary computations.

In this work, we have provided the code for the setup of the first model in detail, as it serves as the foundation for the others. The configurations for the subsequent models closely follow this initial setup, with only minor adjustments in parameters. Given the high degree of similarity between the models, repeating the setup code for each would be redundant and unnecessary.

3.1.3 Performance on Test Set

After training, the model's performance is evaluated on the test set (unseen data) to assess how well it generalizes to real-world examples.

Test Accuracy

The model achieved a test accuracy of 92.64%, which is very close to the training accuracy of 93.37%. This indicates that the model was able to generalize well and is not overfitting to the training data. The test accuracy being nearly equal to the training accuracy suggests that the model learned the patterns well and is able to apply that knowledge to unseen data.

Test Loss

The test loss of 0.2740 suggests relatively accurate predictions, as lower loss values indicate better performance in terms of how closely the predicted outputs align with the actual labels. Given the relatively small gap between training loss (0.2374) and test loss, we can confidently say that the model generalizes well without being overly complex or underfitting.

Confusion Matrix

The confusion matrix demonstrated that the model performed very well on the test set, classifying most images correctly. However, this high accuracy is misleading because the test images were extracted from the same Kaggle dataset used for training, meaning they share a high degree of similarity with the training images. Since these images are frames from videos, they are extremely similar to one another, making it easier for the model to classify them correctly. This led to severe overfitting, where the model essentially memorized the dataset rather than learning generalizable features. As a result, while the model appears to perform well on the given test set, it fails to recognize different data when tested in real-world scenarios, highlighting its inability to generalize beyond the training distribution.

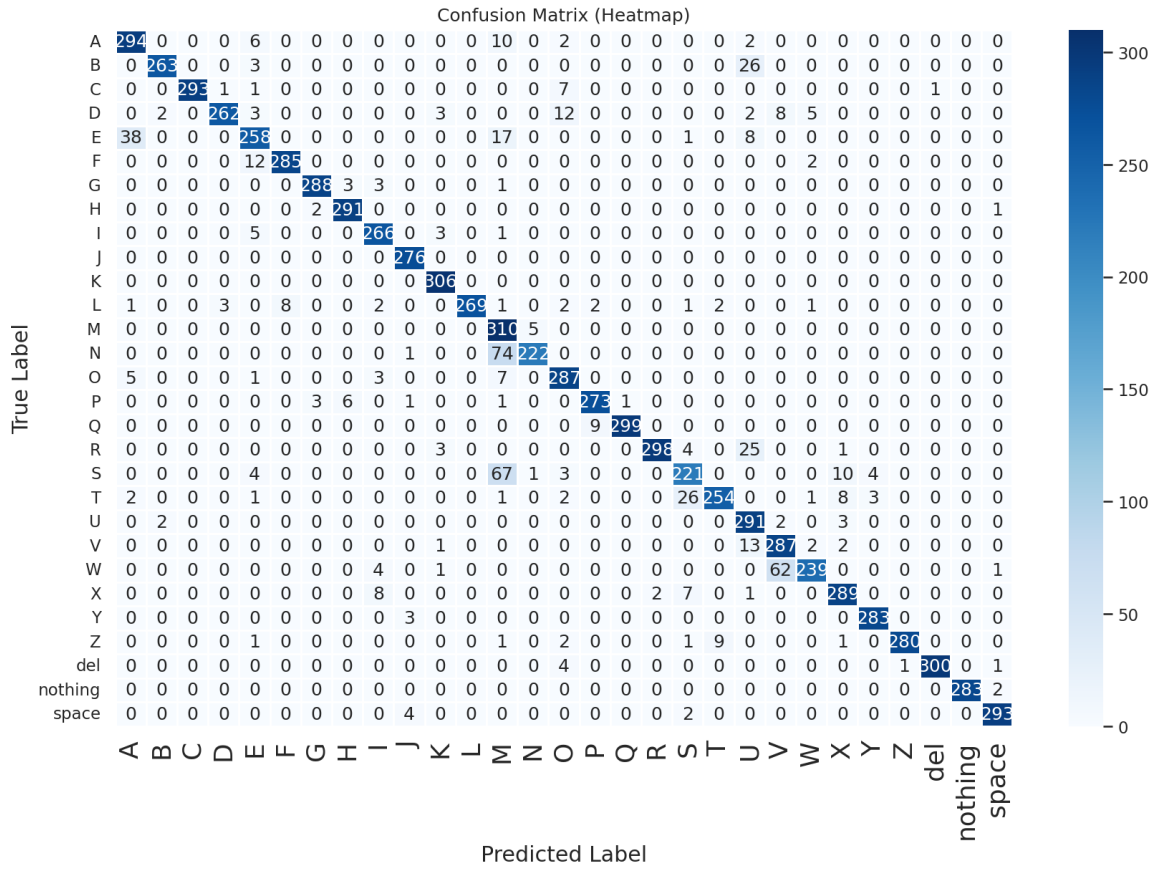


Figure 3.2: Confusion Matrix of the model with the Kaggle Dataset: good predictive performance on test set caused by the utilization of images belonging to same Dataset.

Graphical Analysis

From the Accuracy and Loss graphs we can see that both the training accuracy and loss behave how one would expect. Having big changes in the early stages and then having small but persistent performance improvement as its trained. However the validation graphs shows us that there were some instances of overfitting on epochs 6 and 10. Another situation that could cause this oscillations in the values would be having a learning rate that is too big, so the model "jumps around", but this can be discarded because that kind of issue would also affect the train accuracy and loss. Since those behaved correctly the oscillations can be attributed to overfitting instances.

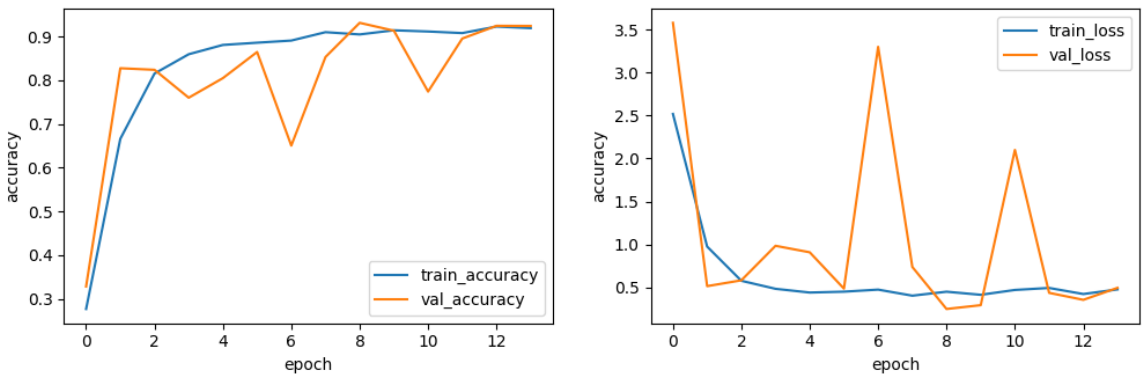


Figure 3.3: Model accuracy using the Kaggle Dataset

Result Impact

The small difference in test and training loss suggests that the model is robust, capable of maintaining accuracy and minimal loss even when evaluated on unseen data.
However good the numbers were, we soon realized that both the training set and the test set were very similar, the images all looked very similar between sets, so in reality the whole model had learned from data in which overfitting was almost inevitable and then tested on data that was very similar. It all became clear when we tried classifying some out-of-set images taken by us and the accuracy dropped significantly (less than 30%), and we realized that we had an overfitting problem, caused by the similarity between the training and test sets. To try to avoid this, we decided to create our own dataset, as presented in the next section.

3.1.4 SHAP Interpretability Analysis

The SHAP interpretability analysis reveals that the CNN primarily focuses on the contours of the hands for classification. However, certain letters exhibit significant contributions from the background, indicating potential overfitting or reliance on contextual information rather than the actual hand shape. Additionally, for gestures like "space" and "del," the model assigns importance to seemingly random areas, suggesting it has not fully learned the distinguishing features. Class separability issues are also evident, particularly with letters like M, N, and U, where dispersed SHAP contributions imply confusion due to their similar hand poses, leading to ambiguity in training.

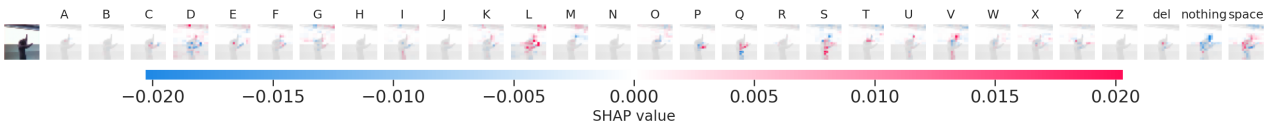


Figure 3.4: Example of SHAP analysis for a label in the Kaggle Dataset.

3.2 Model with Handmade Data - Fixed Background (CNN)

3.2.1 Dataset

For this model we decided to try with a dataset created by ourselves. Since a video is just a rapid succession of pictures, creating a dataset that contains a couple thousand photos of a specific hand position does not take that long. Given the issue that we had previously with the background of the images being very irregular but constant over the dataset, we decided to create these images with a background as plain as possible. Since this was still in a testing period, we only created the dataset for the letters: D, H, K, S, and X. And took around 2000 images of each of them.

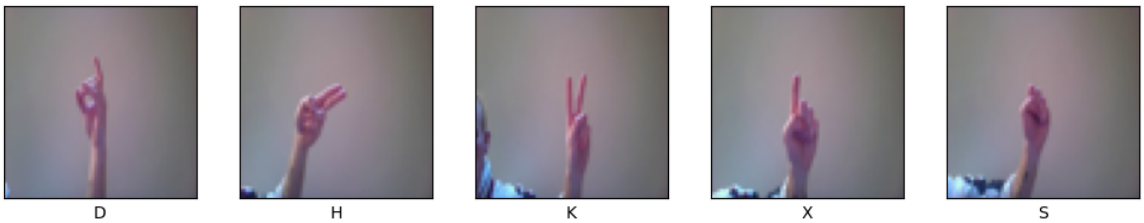


Figure 3.5: ASL labels used in the fixed background model

3.2.2 Data Processing and Model Training

Data Preprocessing

As for data preprocessing in this model the same as the previously presented model is done. The images are transformed into numpy arrays of dimension $32 \times 32 \times 3$, the values are normalized so that all move between $[0, 1]$ and finally the labels are encoded in binary form for the model to work with them.
Data augmentation techniques where used for the the data to be richer. The changes made by the augmentation algorithm were: rotations of the image up to 10°, vertical and horizontal size changes up to 20% separately, zooming up to 50% and horizontal flips of the image. The rotation parameter was the most influential one out of them all values were chosen by a trial and error process, trying to obtain the greatest amount of change in the images without worsening the performance of the model. One parameter that was set out not to change was the brightness, since any variation of this would make the model misclassify letters.

Model Training

The architecture of model used in this opportunity is almost the same as the one used in the previous data with the only difference that the dropout rate is increased from 0.2 to 0.3 to try to prevent some of the over fitting problems we had in the previous set. The model is now trained in 30 epochs instead of 15, with no early stopping and batches of size 32.
The training finished with an accuracy of: 98.64% and a loss of 0.0418. Something to note is that the validation accuracy during the training oscillates a lot, which could indicate over fitting in the training set or a very high learning rate, however as the training set is constantly improving that indicates that there is no major issue with the learning rate.

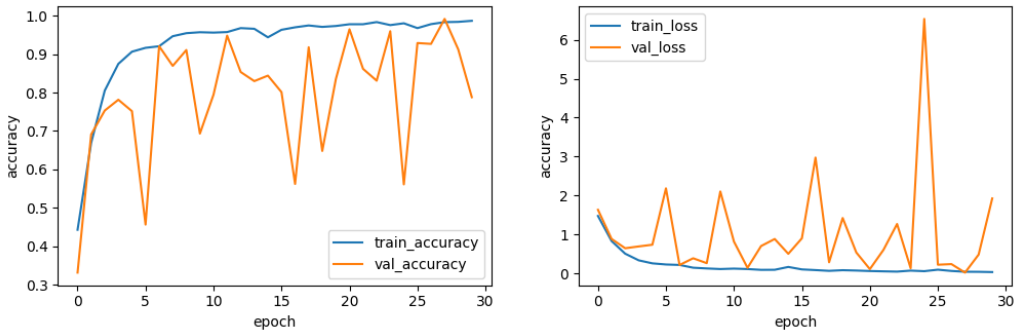


Figure 3.6: Accuracy and Loss curves for the moving background model

3.2.3 Performance on Test Set

On the Test set we have very similar result with an accuracy of 97.82% and a slightly higher loss value of 0.14. And we can see this in the confusion matrix down below, almost everything falls in the diagonal, showing that the model is predicting correctly on the test set.
If we only take into account this values, it would seem that the model was performing very well. The main issue was that when an image from outside the original dataset was introduced, the values previously seen became much worse. This comes from the same issue as in the first model, the data is too similar between themselves, the fact that the background was now uniform did not alter the fact that the images were all very similar (as they were taken at a rate of 30 per second) the model still had a very difficult time classifying images from outside

the original dataset. When tested against 15000 images that came from another dataset the model was only able to correctly classify 5097 out of 15000 (34%). With this information we decided to do some changes to the way we created the images for the dataset and arrived to the model that is explained in the following section.

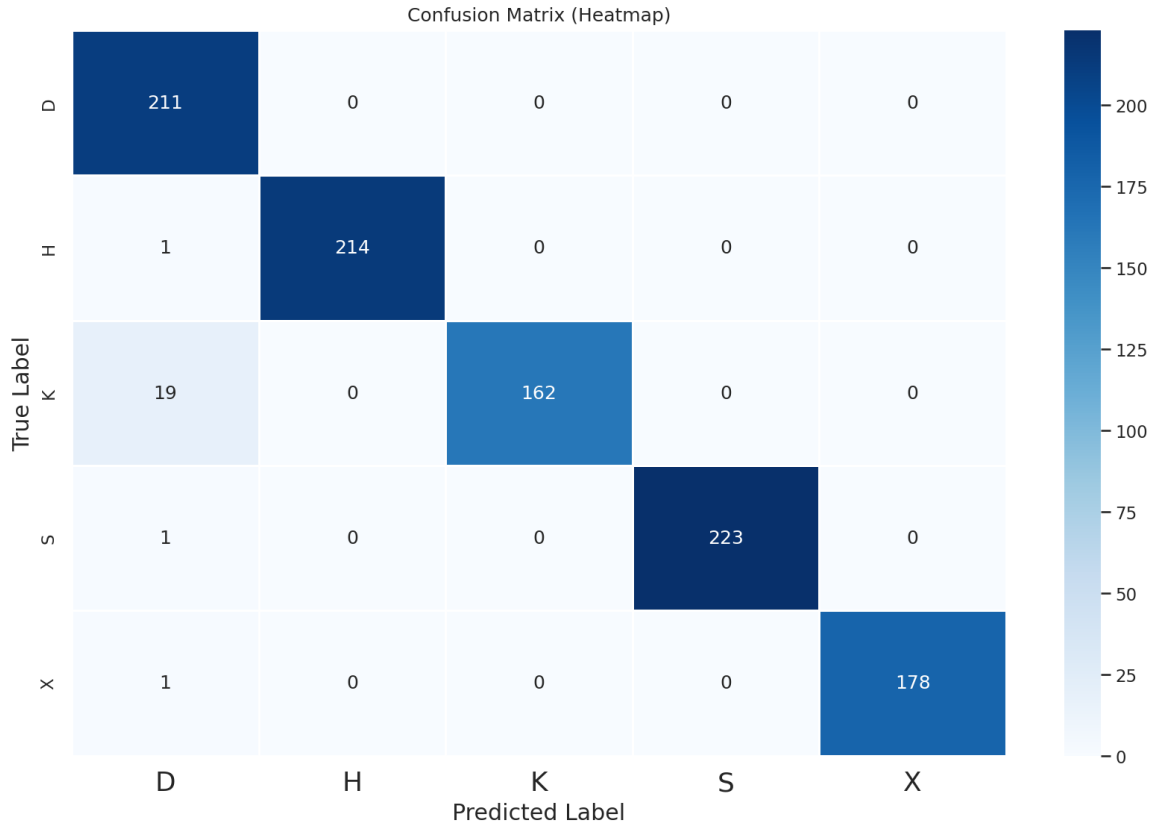


Figure 3.7: Confusion Matrix in the fixed background model: good predictive performance on test set caused by the utilization of images belonging to same Dataset.

3.2.4 SHAP Interpretability Analysis

From the SHAP analysis we can see that the model, actually looks mostly at the hands when looking for features to recognize. However there are some circumstances in which the background is still targeted. Specially when there are prominent features showing.

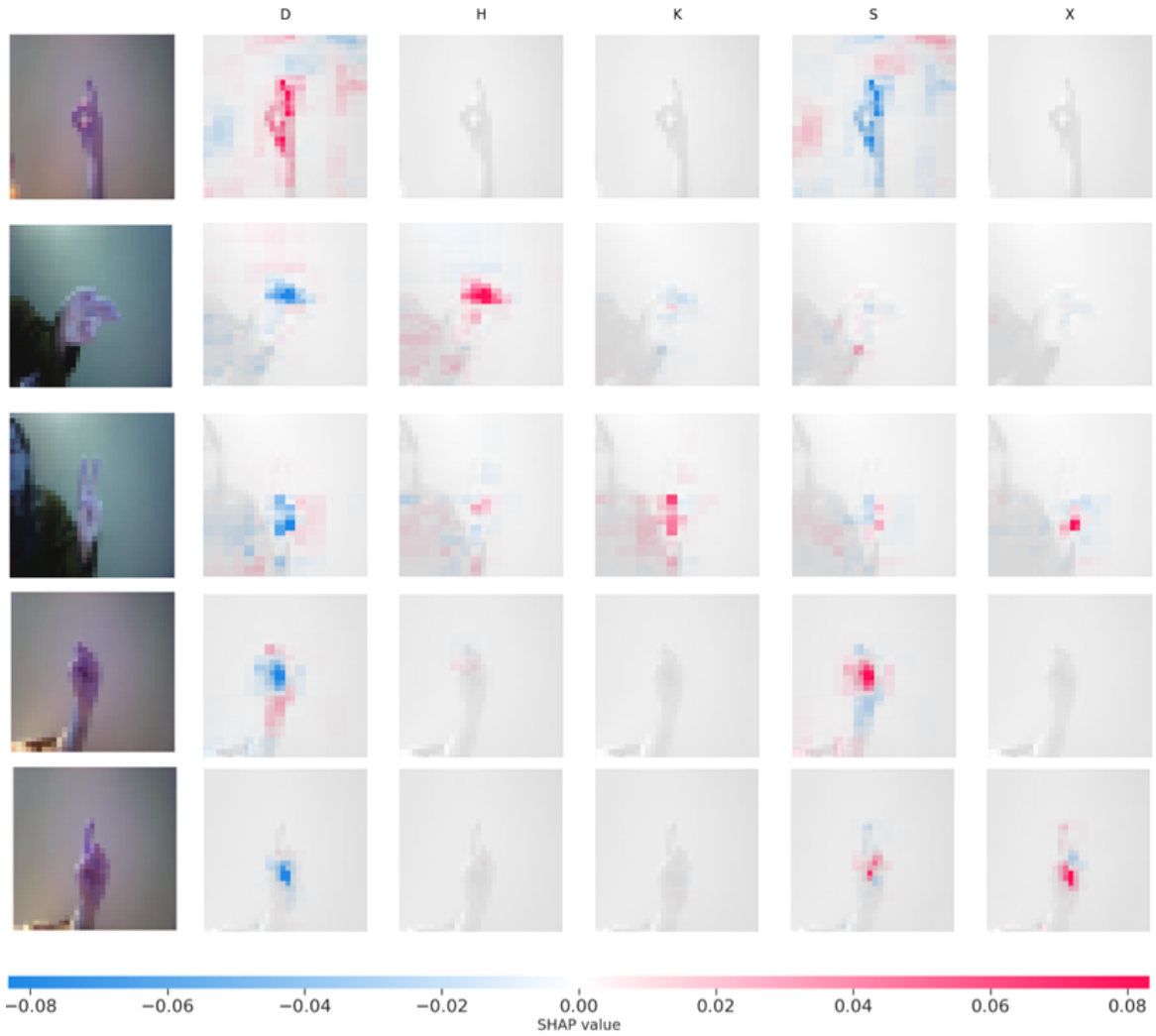
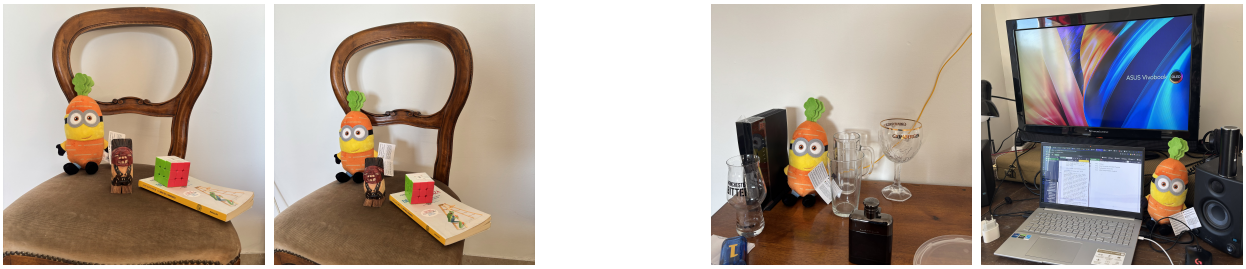


Figure 3.8: SHAP Analysis for all labels in the fixed background model

3.3 Model with Handmade Data - Moving Background (CNN)

For this next model we decide to put a changing background behind the handsigns, hoping to improve the generalizing capabilities of the model. To understand this we have a small experiment. Here are 4 different photos. The first two are to train an imaginary CNN to detect a specific object which we will call Object 1. These images are a exaggeration of what the previous datasets were doing, keeping the same background and moving the whole image a little bit. The two images in the bottom are also to train a CNN looking to detect Object 2, however in these ones the background of the object changes. As a reader, can you tell which is Object 1 and what is object 2? What we have here is that in the first case it is easy to say that both are images containing Object 1, because that is how they are labeled, however we cannot point out what in the image is Object 1. In the case of the second pair of images we can easily tell that Object 2 is the minion plushie, since its the only thing repeating between the two images. This example should illustrate the logic we applied to chose to go for a moving background, hoping it would allow the model to isolate better the shape we were looking for.



Object 1 Object 2

Figure 3.9: Comparison of Object 1 and Object 2

3.3.1 Dataset

The dataset in this case consists on images of the same letters as before: D, H, K, S, and X. With around 840 instances of each letter, but with the biggest difference being that the videos used to get the pictures were now recorded while moving around inside the house. As the videos are still recorded at 30 fps there are still a lot of frames that look very similar to each other.

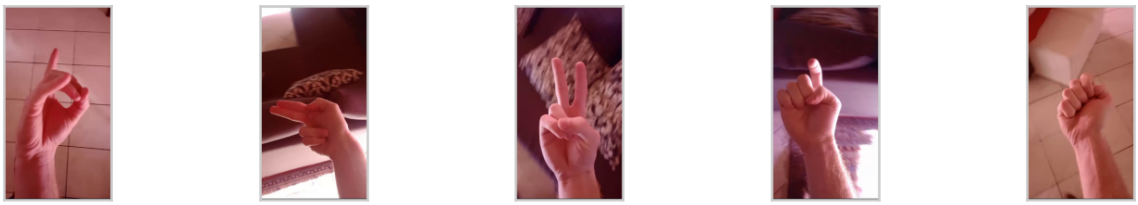


Figure 3.10: ASL labels in the moving background model

3.3.2 Data Processing and Model Training

Data Preprocessing

The data pre-processing stays almost the same but now the resolution of the images are $64 \times 64 \times 3$ and once again they are normalized to obtain a $[0, 1]$. The five labels are encoded in binary form.

Model Training

The model's architecture is once again almost the same as before (adapted to the new size of the images) but now the dropout rate is once again increased, now to 0.4 trying to steer away from the over fitting that we had seen on the previous models. Data augmentation and early stopping are both used in this case. Similarly to before we do 30 epochs with batches of size 32. The architecture of model used in this opportunity is almost the same as the one used in the previous data with the only difference that the dropout rate is increased from 0.2 to 0.3 to try to prevent some of the over fitting problems we had in the previous set. We also applied some data augmentation techniques to make the training data more variable and rich. The model is now trained in 30 epochs instead of 15, with no early stopping. The learning rate was reduced by a factor of 10 to arrive to a value of 0.0001.
The training finished on epoch 23/30 with an accuracy of: 99.37% and a loss of 0.0271. We do not see anymore the oscillation that the validation accuracy had during the training, indicating that the model likely had less over fitting in training.

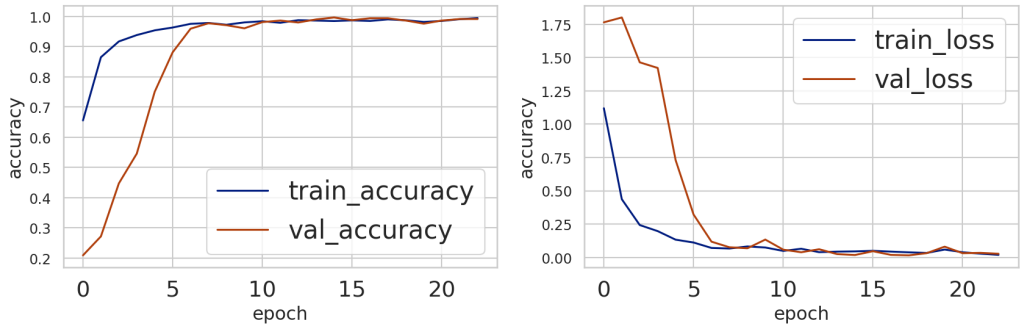


Figure 3.11: Accuracy and Loss curves for the moving background model

3.3.3 Performance on Test Set

On the test set the accuracy was od 99.31% and a loss of 0.0366 which, again, are incredibly good numbers, but are the result of dataset over fitting as in the previous cases. Even though we tried to prevent this by the moving background the fact that the pictures are taking in such rapid succession still generates a lot of frames that look similar. Given this, when an out of set image was tested the accuracy dropped significantly, performing correct predictions in only 2.444 out of the 15.000 sampled images tested (16.3%). When we look at the confusion matrix of the model we see that most of the classifications fall on the X category.
As a last attempt to fix our everlasting overfitting problem we tried some extra data preprocessing steps using mediapipe, that will be explained below.

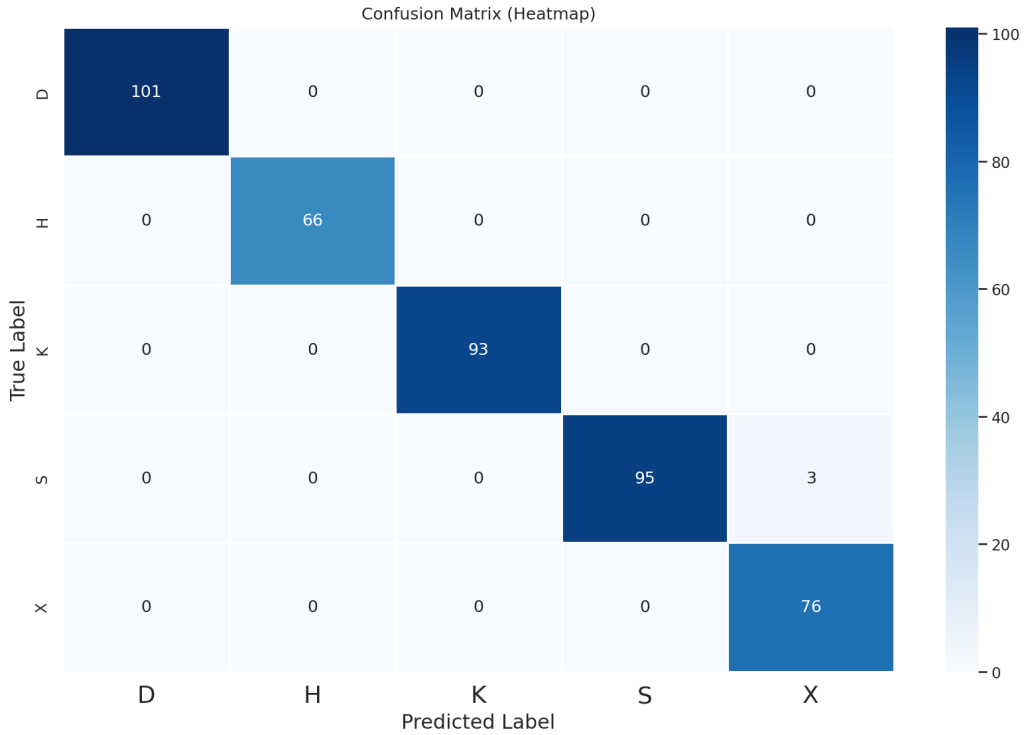


Figure 3.12: Confusion Matrix using the moving background model : good predictive performance on test set caused by the utilization of images belonging to same Dataset.

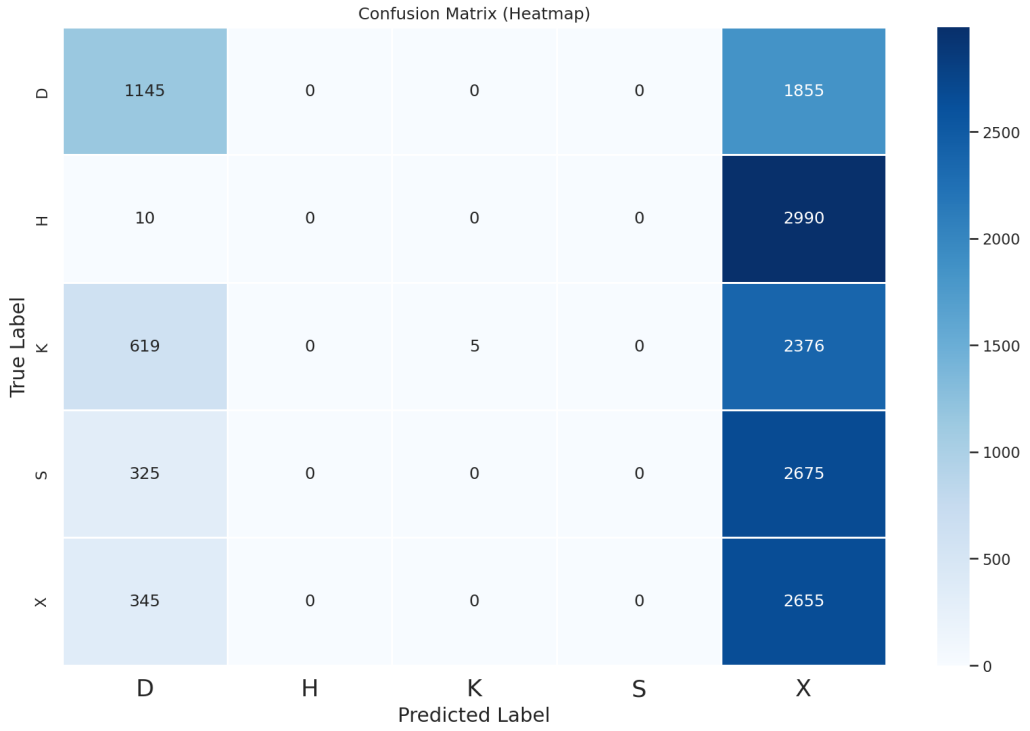


Figure 3.13: Confusion Matrix of out of set images using the moving background model

3.3.4 SHAP Interpretability Analysis

The SHAP analysis in this case gave us a very good insight about what was happening in the model. Now that the background was moving there were moments where there was something very characteristic in the background and since the pictures were taken from a video, that particular thing could have stayed in frame for a couple seconds and that would mean that it now shows up in a couple hundred pictures. What makes it worse is that it is likely that it only shows up in one specific hand gesture, so the when we looked at the SHAP graph we could see that there are cases in which the classification comes from an element in the background that only appears in that specific hand gesture, which is obviously not a good thing. In the image, even tho not particularly strong, we can see that the model is putting some importance on the background to classify the gesture.

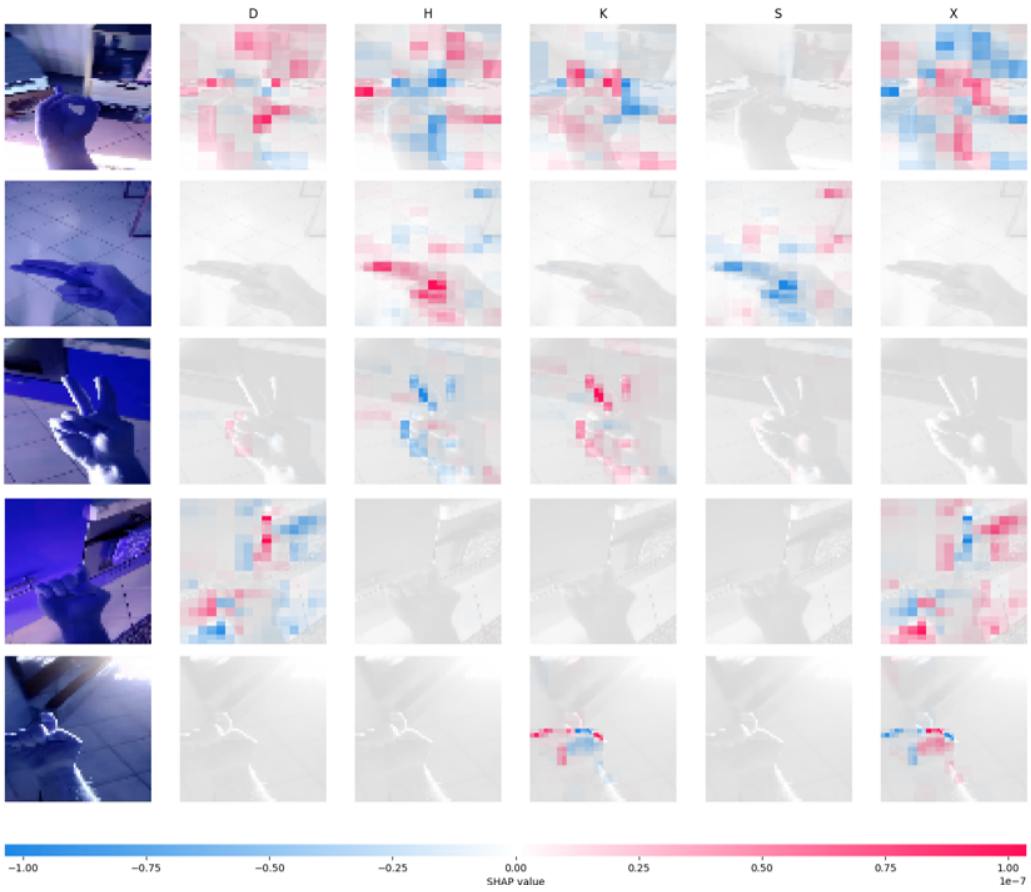


Figure 3.14: SHAP analysis on the moving background model

3.4 Model with Handmade Data - MediaPipe Preprocessing (CNN)

The model itself used in this iteration is the same as the one that was showed in the previous section, the only change that is made is an extra step of preprocessing using MediaPipe.

3.4.1 Mediapipe Hand Tracking

MediaPipe is a free, open-source tool developed by Google for real-time machine learning applications. It helps process video, images, and sensor data efficiently on various devices, including phones, computers, and web browsers.
One of its popular features is MediaPipe Hand Tracking, a pre trained model that uses a CNN to detect and track 21 key points on a hand in real time. This makes it useful for gesture recognition, sign language interpretation, gaming, and virtual interactions. Since it's open-source under the Apache 2.0 license, it's completely free to use for personal and commercial projects.

3.4.2 Preprocessing with Mediapipe

For this final model, we incorporated MediaPipe as an additional preprocessing step to generate hand landmark maps for all images in the dataset. Contrary to our initial expectations, MediaPipe successfully detected a hand in every image, so no images were removed during preprocessing. This confirmed that all images contained identifiable hands, making them valid for training the CNN.

Since the dataset remained unchanged, we trained the model using the same architecture and parameters as the previous version. The results were very similar to the previous model, indicating that while MediaPipe effectively mapped hand landmarks, its preprocessing step did not alter the input and, for this reason, the performance.

Given that this model is almost identical to the previous one, except for minor differences in the training process, we will not explore it further in this article. Analyzing this version in detail would be redundant, as it relies on the same dataset and architecture. Instead, we shifted our focus to a different approach—using only the extracted MediaPipe hand landmarks as input data for training, rather than the original images.

Chapter 4

Model with Handmade Data - MediaPipe Node Coordinates (MLP)

This section presents an in-depth analysis of a gesture recognition model based solely on X and Y coordinate inputs from hand landmarks. The model aims to classify hand gestures using a fully connected neural network (FNN). A Multilayer Perceptron (MLP) is a type of artificial neural network composed of multiple layers of nodes, where each node in one layer is connected to every node in the next layer. MLPs are known for their ability to model complex, non-linear relationships through the use of activation functions in each layer, typically employing the ReLU or Sigmoid functions. The structure of an MLP typically includes an input layer, one or more hidden layers, and an output layer. Each connection between nodes is assigned a weight, which is adjusted during training using backpropagation. A Fully Connected Neural Network (FNN) is a general term that refers to any neural network where each node in one layer is connected to every node in the subsequent layer. Thus, MLP is a specific type of FNN, and both terms are often used interchangeably in practice. While the term FNN refers broadly to this type of architecture, MLP specifically refers to networks that include multiple layers, making them capable of capturing hierarchical representations of data.

In this work, the model implemented is based on an MLP architecture, where the input features are the X and Y coordinates of hand landmarks obtained through MediaPipe. The MLP-based model is trained to classify various hand gestures, leveraging its ability to learn complex mappings from the raw input data. Below, we detail the dataset, preprocessing steps, model architecture, training process, evaluation metrics, and interpretability analysis using SHAP.

4.1 Dataset

The dataset consists of hand gesture images converted into structured numerical representations through X and Y coordinates of 21 hand landmarks, resulting in a total of 42 input features. The gestures included in this dataset are representative of different sign language handshapes.

The data is collected from video recordings, ensuring consistency in hand placement and movement. Each sample consists of 42 numerical values (21 landmarks 2D coordinates).

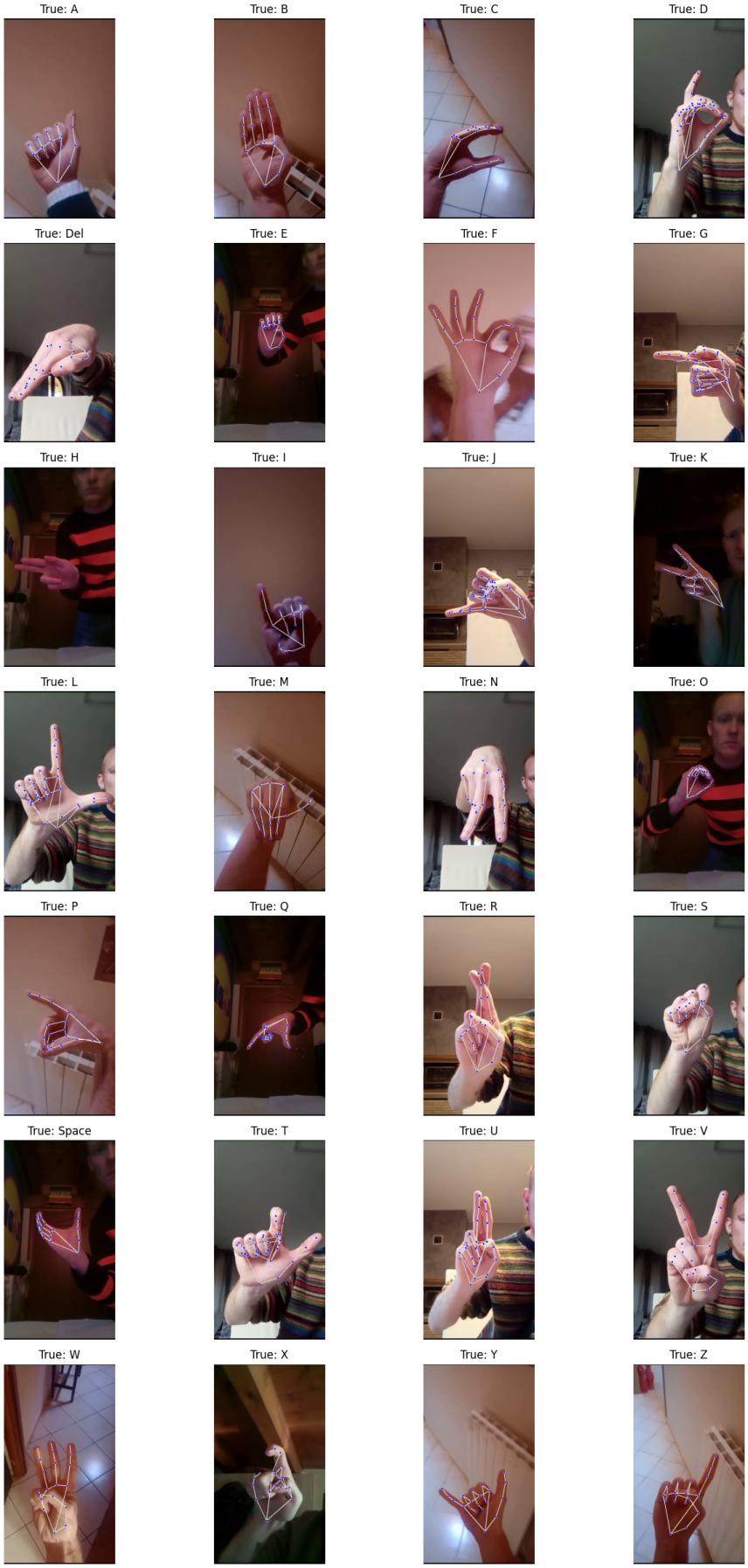


Figure 4.1: ASL Alphabet with mediapipe tracking displayed for each label

4.2 Data preprocessing and model training

Preprocessing steps include:

Normalization: Values are rescaled between for numerical stability.

Encoding: Gesture labels are one-hot encoded for categorical classification.

Splitting: The dataset is split into training and testing subsets using an 80-20 ratio.

The process begins by loading images of hand gestures from a dataset, where hand landmarks are extracted using a pre-trained model. To ensure data quality, the landmarks are filtered to remove outliers based on their Z-scores. After outlier removal, the dataset is normalized, and labels are one-hot encoded. The data is then split into training and test sets, with an 80/20 ratio.

The dataset is examined before and after the outlier removal to confirm the number of images per class, revealing only minor reductions in some classes. Despite these small drops, the dataset retains a substantial amount of data, with 18,100 samples for training and 4,526 for testing.

A subsequent balancing step ensures that each class contains exactly 543 samples, corresponding to the class with the fewest samples, leading to a uniform distribution across all 28 classes. This balanced dataset, now consisting of 12,163 training samples and 3,041 test samples, eliminates any bias from overrepresented classes, contributing to more robust model training and evaluation.

The entire process ensures a fair representation of each class, enhancing the model's ability to generalize and perform effectively on unseen data.

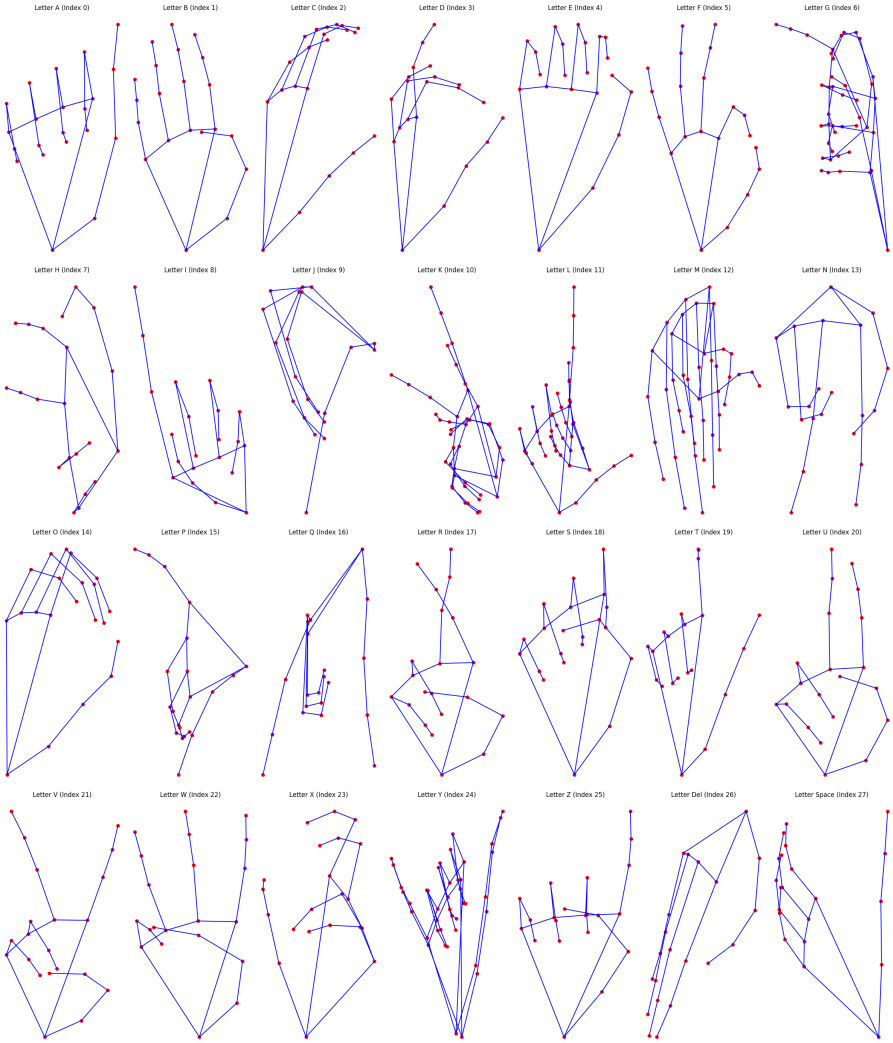


Figure 4.2: Mediapipe tracking displayed for each label

The settings for the model are then outlined. The model follows an MLP architecture optimized for multi-class classification. **Layer Configuration:**

Input Layer: 42 neurons (one for each coordinate value)

Hidden Layers:

Dense (512 neurons, ReLU activation) + Dropout (0.7)

Dense (512 neurons, ReLU activation) + Dropout (0.7)

Dense (256 neurons, ReLU activation) + Dropout (0.7)

Output Layer: Softmax activation with neurons equal to the number of gesture classes

Model parameters: after many trials with different settings, the chosen ones that granted a good performance and allowed a balanced and gradual learning process are reported. Dropout was set at 0.7 for each hidden layer to mitigate overfitting. The model is trained for a maximum of 30 epochs with an early stopping criterion: training halts if no improvement in validation loss is observed after 3 consecutive epochs. The Batch size is set to 32, while the

learning rate is set to 0.0001. Training accuracy reached 0.9518 with a final loss of 0.1629. Validation accuracy showed minimal oscillation, indicating improved generalization compared to previous models.

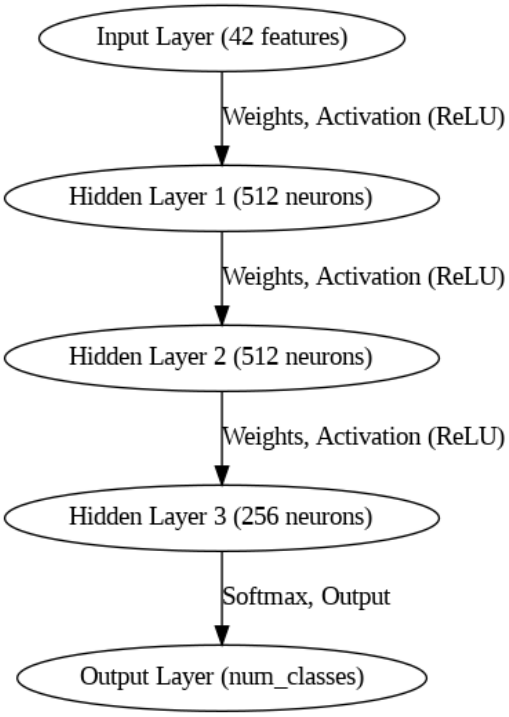


Figure 4.3: Structure of the Neural Network

4.3 Performance on Test Set

The model achieved a very high accuracy with 3041 total predictions. Out of these, 3033 predictions were correct, resulting in an accuracy of 99.74%. This suggests that the model is highly effective in classifying the hand gestures based on the input features. Only 8 predictions were incorrect, which is a minimal error rate of 0.26%. These results indicate the strong performance of the model and its ability to generalize well to new data, despite the small percentage of incorrect predictions.

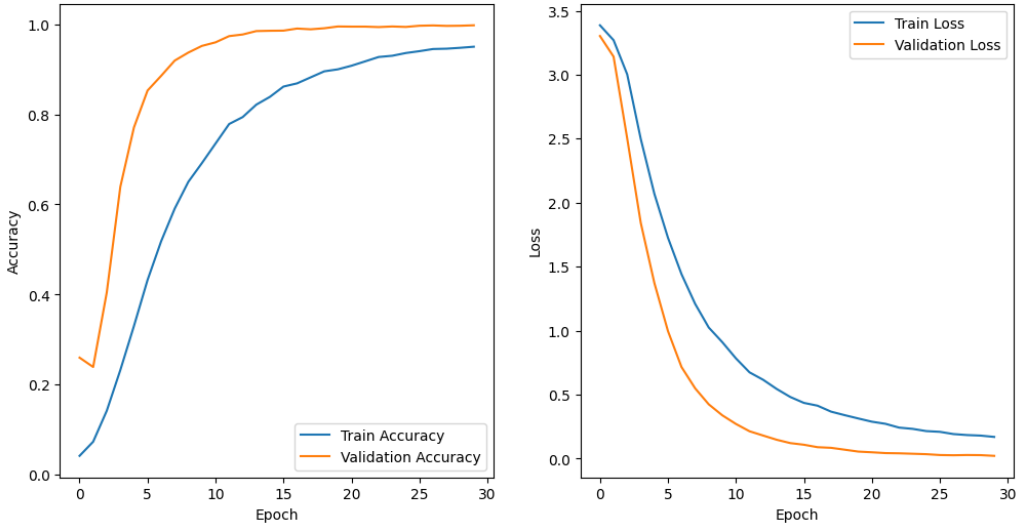


Figure 4.4: accuracy and Loss curves for the MLP model

The accuracy and loss graphs clearly illustrate the model's progress: the accuracy plot shows the training and validation curves converge, indicating good generalization, while the loss plot shows a consistent reduction in both training and validation loss. This confirms that the model has learned to recognize hand gestures effectively and can generalize well to unseen data, making it highly suitable for gesture recognition tasks.

The confusion matrix further proves the high accuracy of the model, demonstrating that the classification performance is practically perfect. It reveals minimal misclassifications, with the vast majority of predictions correctly assigned to their respective classes. This near-perfect classification pattern underscores the model's ability to accurately differentiate between the various hand gestures, solidifying its exceptional accuracy and reliability in recognizing the intended gestures with remarkable precision. The results from the confusion matrix strongly confirm the model's effectiveness in delivering highly accurate predictions.

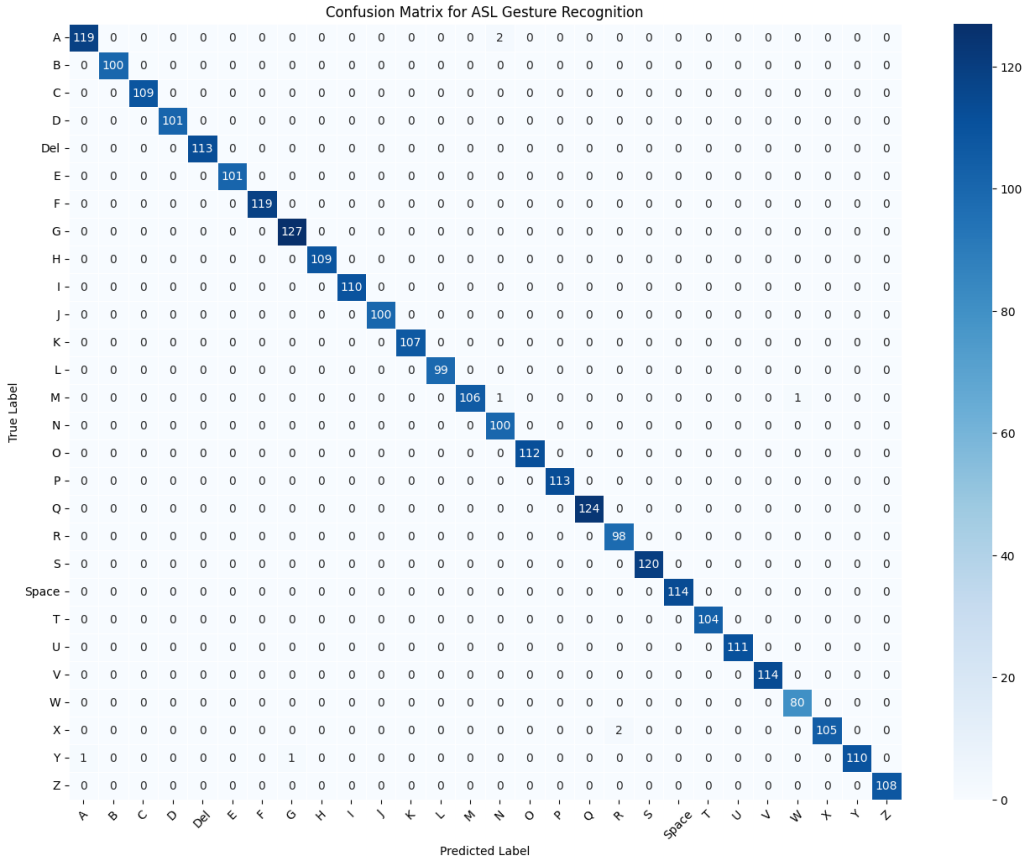


Figure 4.5: Confusion Matrix for the MLP Model

4.4 SHAP Interpretability Analysis

The SHAP (SHapley Additive exPlanations) method is applied to gain a deeper understanding of the trained model's decision-making process and identify which features have the most influence on its predictions. In this analysis, a subset of the test dataset is selected to speed up computations, ensuring that the model's interpretability analysis remains efficient. For reference during the SHAP calculations, the background dataset is set to the median values of the test dataset, offering a stable baseline that does not overly skew the results. The SHAP values are computed for this subset of data, ensuring that the shape of the SHAP values corresponds to the selected test data, which allows for accurate interpretation of the model's

behavior. The SHAP values are then visualized using a "beeswarm" plot, a powerful tool that displays the relative importance of each feature. In this case, the features represent the 42 hand landmarks, where each landmark is considered an individual feature based on its X and Y coordinates. This setup allows the analysis to pinpoint which of the coordinates are more influential in the model's decision-making process.

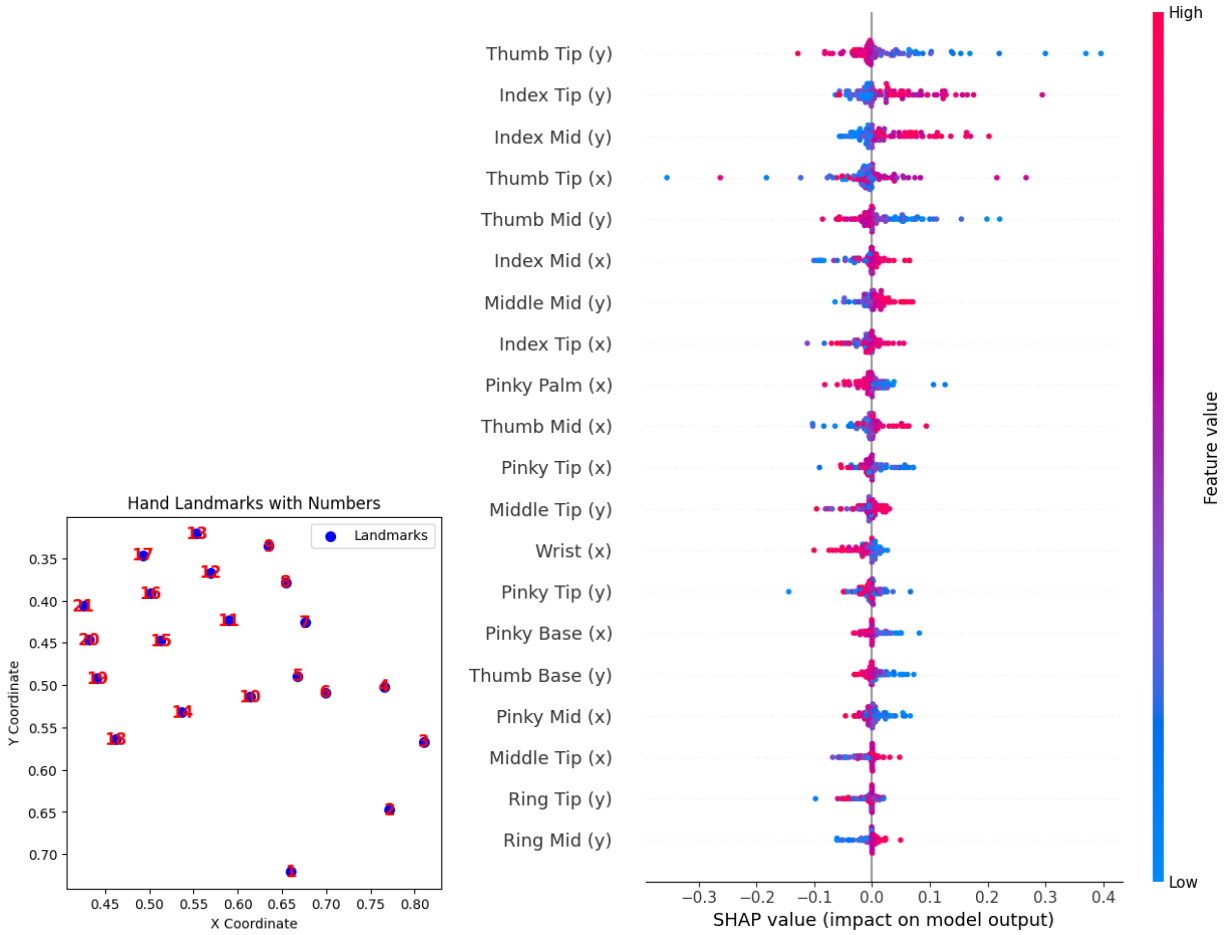


Figure 4.6: SHAP Analysis without distinguishing for label

The results of the analysis reveal that certain vertical coordinates, such as the Y-coordinates for key landmarks like the Thumb Tip (y) and Index Tip (y), are among the most influential features in the model's decision-making. These features play a crucial role in distinguishing between different hand gestures. On the other hand, some features, such as the Y-coordinate of the Ring Mid landmark, showed minimal impact on the model's performance. This suggests that the Ring Mid (y) feature could be considered for removal, as its contribution to the model's prediction is negligible, and eliminating it could streamline the model without sacrificing its predictive power. By removing such less impactful features, the model could become more efficient and possibly even improve in performance by focusing on the more relevant data.

In a further step to enhance the interpretability of the model, a more granular, class-specific SHAP analysis was conducted. Rather than evaluating feature importance across the entire dataset, this updated approach focuses on analyzing the impact of features for each individual class, such as specific hand gestures corresponding to a letter or symbol in

a gesture recognition system. For each class, two separate outputs are produced: the first output is an image showing the hand landmarks with each of the 21 key points numbered, providing a clear visual representation of the hand's structure. The second output is a SHAP Summary Plot, which visualizes the specific contribution of each feature (landmark) to the model's prediction for that particular class. This dual output approach allows for a more targeted analysis, helping to identify which specific hand landmarks or coordinate values are most important for classifying each gesture. This class-specific analysis improves upon the earlier global SHAP analysis by offering a more detailed, class-by-class breakdown of feature importance. Whereas the previous analysis provided a single, unified SHAP plot summarizing global feature importance, this refined approach generates separate plots for each class, allowing for a deeper understanding of how different features contribute to different predictions.

By visualizing how the features—such as the positions of the finger tips or joints—affect the model's predictions for each class, it becomes easier to recognize patterns and understand why certain features are more influential for particular gestures. For example, hand gestures involving finger tips are likely to show a higher feature importance for certain letters or symbols, while gestures that involve finger joints may have a different set of influential landmarks.

Furthermore, this approach allows for more nuanced insights into the model's behavior. By analyzing each class individually, it is easier to identify any biases or inconsistencies in the model's performance. If certain classes are heavily influenced by irrelevant features, it may signal a need for model optimization. Similarly, by pinpointing which landmarks are essential for each class, it becomes possible to verify the relevance of features and eliminate those that do not contribute meaningfully to the model's performance. This kind of targeted feature analysis can lead to better model optimization and refinement, ensuring that only the most relevant data is used for predictions.

In comparison to the previous method, which provided a broad overview of feature importance across the entire dataset, this new, class-specific SHAP analysis offers a more comprehensive and focused understanding of how features affect model predictions on a per-class basis. It also allows for a more flexible, dynamic analysis by iterating through the various classes and adjusting the dataset accordingly, in contrast to the fixed subset used in the earlier method. By doing so, this approach offers deeper insights into class-specific model behavior, enabling better decision-making when refining the model. Overall, this more detailed and targeted SHAP analysis not only aids in enhancing the interpretability of the model but also supports optimization efforts, improving the model's generalization and performance in classifying hand gestures.

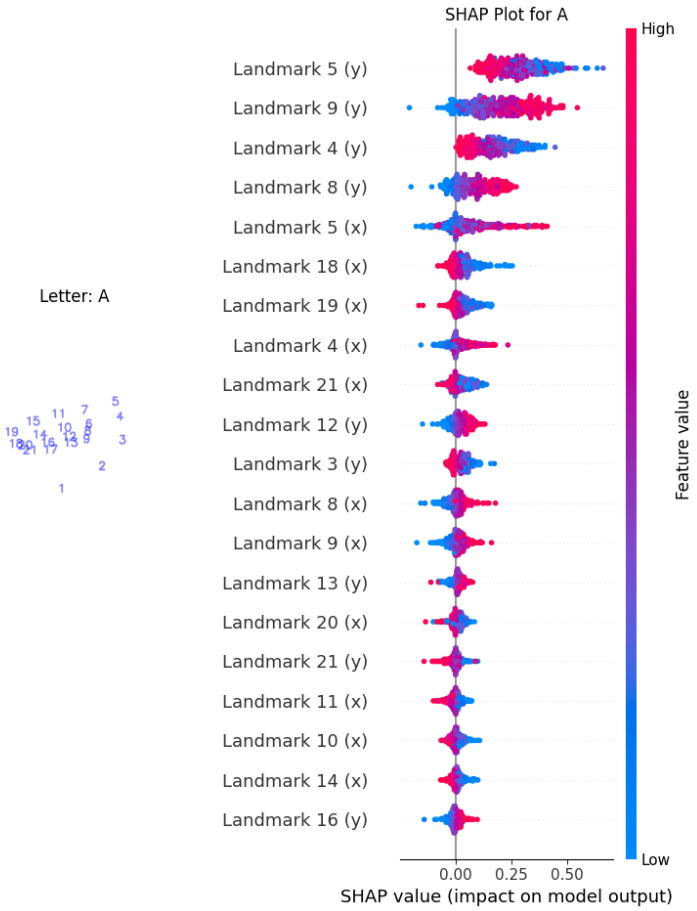


Figure 4.7: SHAP Analysis for letter A

In the displayed example, the letter "A" is analyzed, and it is observed that the most important and influential predictive feature is the tip of the thumb. This is because the thumb tip is the only part of the hand that undergoes a significant change when distinguishing the letter "A" from other letters, such as "S," where the fist is fully closed.

The dramatic movement of the thumb in the letter "A" allows it to serve as a key distinguishing feature. In contrast, when analyzing the letter "S", the results show that the tip of the thumb is again the most relevant feature but with magnitude in opposite direction. This demonstrates that each letter correctly assigns weight to the features that most effectively differentiate that specific gesture from others. Each hand gesture, represented by a letter, prioritizes the landmarks that best distinguish it from other possible gestures, ensuring that the model accurately captures the unique characteristics of each letter.

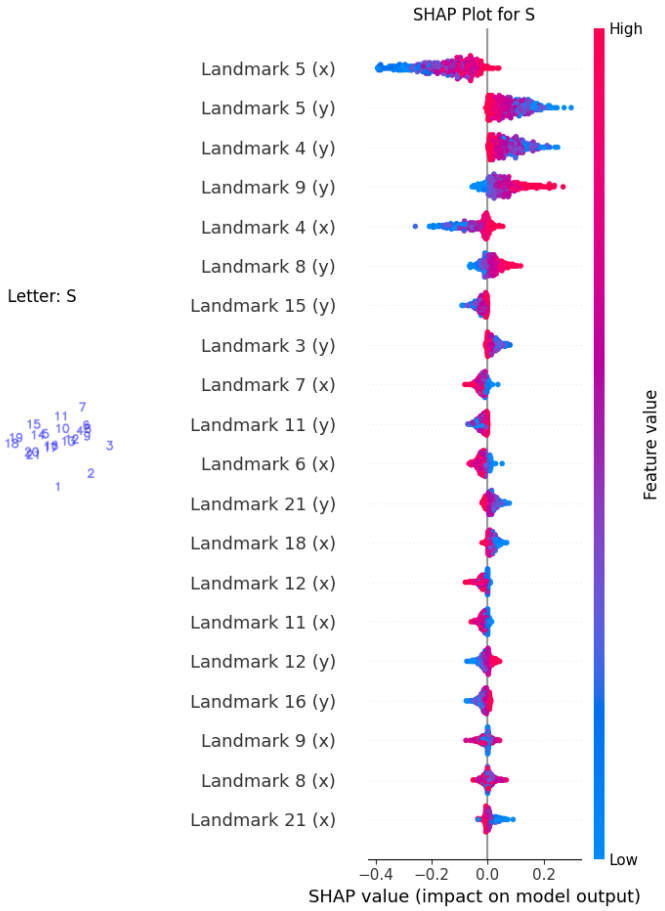


Figure 4.8: SHAP Analysis for letter S

We have only reported two examples, the letters "A" and "S", to illustrate the model's interpretability, as there are a total of 28 classes in the dataset. Including examples for all the classes would have been too extensive and would have dominated the report, potentially overwhelming the reader. By focusing on just a few representative examples, we aim to highlight key insights without overloading the analysis.

4.5 Conclusions

Although the MLP-based model is the simplest architecture used in this study, it proved to be the most effective one. Despite the more complex convolutional neural network (CNN) approach and an earlier discarded version of this model that incorporated Z coordinates in addition to X and Y, this MLP consistently outperformed both. The simplicity of the MLP model allowed it to work more efficiently, particularly in tracking, as it focused solely on the X and Y coordinates of hand landmarks. This reduced the model's complexity, making it easier for the network to learn and generalize, ultimately leading to better performance. In contrast to more sophisticated models, the straightforwardness of the MLP made it not only faster and more robust but also simpler for the machine to understand and optimize, resulting in superior accuracy for gesture recognition tasks.

Chapter 5

Application: Real-Time Webcam Interface

This chapter describes the interface of the application developed in this project. The application, currently implemented as a Python script combined with a Keras model, represents the final version of a system aimed at recognizing hand gestures and translating them into text. Previous iterations of this project explored simpler versions of hand gesture recognition, which included approaches utilizing tracking methods and Convolutional Neural Networks (CNNs). In these earlier versions, the system took screenshots of the user's screen and made predictions based on these screenshots, but these methods showed limitations in terms of real-time interaction and accuracy.

Through several refinements and iterations, the final application employs a more robust and efficient method for tracking and predicting hand gestures, which is facilitated by the real-time webcam feed. As the code runs, the user is presented with an interface where they can see a live feed of themselves in real-time through the webcam of the PC. Below the video feed, a predicted letter is displayed, which corresponds to the hand gesture made by the user. The model continuously provides feedback, displaying the predicted letter in real-time as the user moves their hand, allowing for smooth and responsive interaction.

The main interface includes several interactive elements designed to make the application easy to use. When the user is satisfied with the hand gesture they are forming, they can press the "Write" button, which will register the current letter and add it to the text string. If the symbol shown in the interface is "Space," clicking the "Write" button will insert a blank space in the text string. If the symbol displayed is "Del," the most recently written letter will be deleted. Holding the hand in the "Del" gesture and clicking "Write" multiple times will continue to remove letters from the string, allowing the user to backtrack and delete as necessary.

Additionally, the application includes a "Clear" command, which enables the user to erase the entire text string with a single click. This command is useful for resetting the input, providing a fresh start if the user wishes to correct any mistakes or begin a new input. These commands—Write, Space, Del, and Clear—combine to form an intuitive and responsive interface for users to interact with the system effectively.

One important aspect of this application is that the hand gesture recognition model was trained exclusively with data from the left hand. This limitation means that the system is designed to recognize gestures made only with the left hand. Consequently, the tracking relies

on the Mediapipe framework, which provides real-time hand tracking. While Mediapipe is a free and accessible tool, it does have certain limitations. One such limitation is its sensitivity to environmental factors, such as lighting. In particular, backlighting can significantly hinder the ability of Mediapipe to detect the hand properly, leading to reduced accuracy in gesture recognition. Therefore, it is crucial to ensure that there is minimal light coming from behind the user, as this could interfere with the tracking performance.

The interface and underlying model are designed to work seamlessly together, providing an interactive experience where the user can write text using only hand gestures. Although the current version is based on a Python script and a Keras model, the application has the potential to be expanded and optimized further. In future developments, the system could be transformed into a standalone application, a web application, or a mobile app. Optimizing the model could allow it to recognize gestures made with both hands, making the system more versatile. The application could also be improved by using a tracking that is more robust under different lighting conditions.

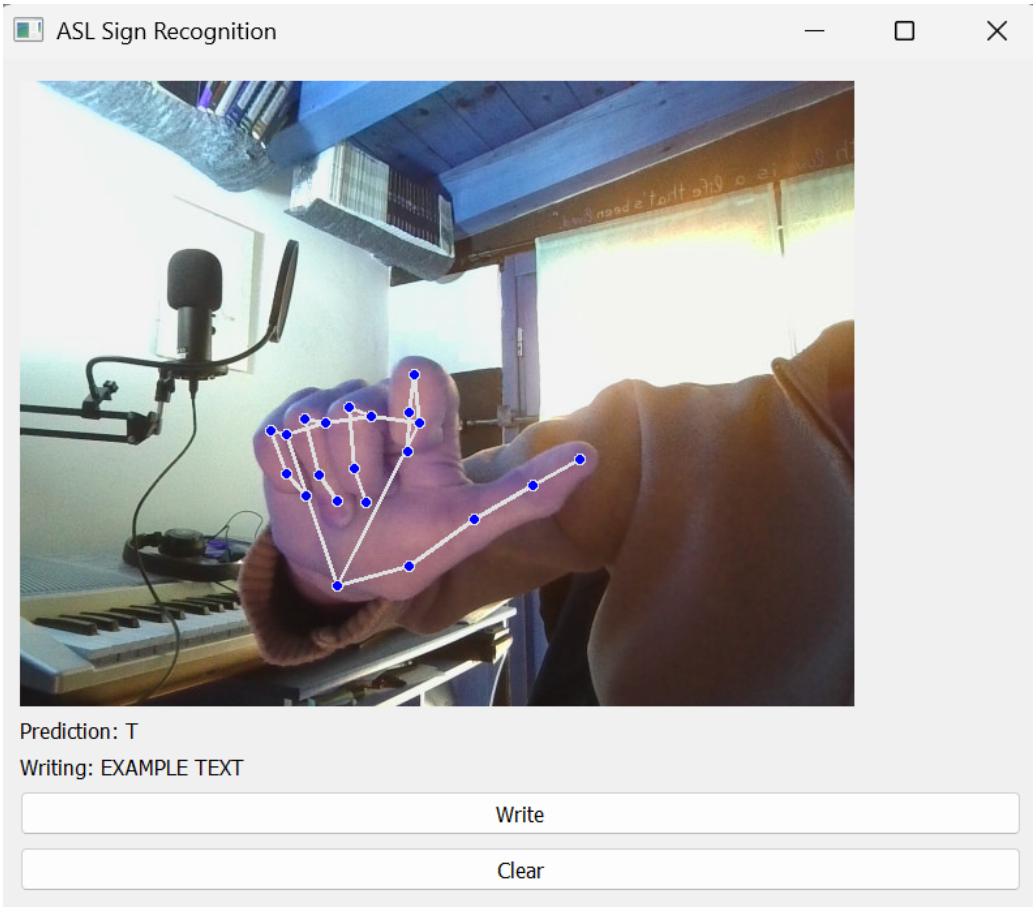


Figure 5.1: Screenshot of the ASL Handwriting Application Interface

This application represents a significant step toward more accessible communication tools, especially for individuals who are proficient in American Sign Language (ASL). By allowing users to input text using only hand gestures, it has the potential to bridge communication gaps and support sign language users in digital environments. Although there are limitations, such as the requirement for the user to be in a well-lit environment and the model's current restriction to left-hand gestures, the system shows great promise as a tool for gesture-based text input.

Chapter 6

Explainable AI: facilitating the comprehension of our model

In the realm of artificial intelligence, particularly in systems that influence decision-making, explainability is a crucial aspect of fostering trust and accountability. Our project, focused on recognizing American Sign Language (ASL) using Convolutional Neural Networks (CNN) and Multi-Layer Perceptrons (MLP), has profound implications for a variety of stakeholders. These stakeholders have different concerns and reasons for wanting to understand how the system works, and it is essential to address these concerns through appropriate explainability methods.

6.1 Stakeholders of the ASL Recognition System

The primary stakeholders for our application include the following:

End Users (Deaf and Hard-of-Hearing Individuals): The end users of this technology are primarily members of the Deaf and hard-of-hearing communities who rely on ASL for communication. They would benefit from an application that enables them to translate ASL into text, facilitating communication with those who do not know sign language. For these users, understanding how the system works is essential for ensuring its reliability and consistency in interpreting their gestures correctly. They may also be concerned about potential errors in recognition or biases that could impact the system's accuracy, so transparency regarding how the model makes its decisions is important. Users should also have access to feedback channels where they can report issues or suggest improvements.

Developers and Data Scientists: The developers and data scientists who design, train, and maintain the system need to ensure that the model operates effectively and ethically. They are also responsible for addressing any biases that may exist in the data or model. For them, explainability methods such as SHAP (SHapley Additive exPlanations) and visual representations of the model's architecture are valuable tools for understanding and improving model performance. They can use these techniques to identify which features of the ASL gestures are most influential in the decision-making process.

Installers and System Administrators: Installers or system administrators responsible for deploying the application need to ensure the software runs smoothly across different environments, such as various operating systems or devices. While they may not be directly concerned with the intricate workings of the AI model, they will benefit from an understanding of the model's requirements, such as computational power, memory, and the necessary dataset

for training the system. This understanding can be conveyed through clear documentation and a troubleshooting guide.

Ethical Oversight Committees and Regulatory Bodies: Organizations and regulatory bodies that oversee the ethical use of technology, particularly in sensitive areas like accessibility, will require transparency regarding how the model is trained and its decision-making process. These stakeholders are primarily concerned with the fairness, transparency, and accountability of the AI system. They will be particularly interested in knowing how the system deals with issues such as bias and the treatment of different dialects of ASL.

General Public and Researchers: The general public and academic researchers may also have an interest in understanding how this technology works, especially in the context of its application to other sign languages or related AI systems. Researchers might want to understand the underlying algorithms and how they can be improved or adapted to other fields, while the public might seek to understand the societal implications of AI in accessibility technology.

Each of these stakeholders has distinct reasons for wanting to understand how the AI system operates. To address these needs, we must utilize effective post-hoc explainability methods that communicate the workings of the model in an accessible and understandable way.

6.2 Post-Hoc Explainability Methods

Post-hoc explainability refers to techniques used to explain the decisions made by a machine learning model after it has been trained. This is crucial when the model is complex, as is the case with deep learning models like CNNs and MLPs. There are several ways to achieve post-hoc explainability, ranging from human facilitators to visual tools and documentation.

6.2.1 Human Facilitators: Direct Communication and Support Channels

One of the most straightforward and effective ways to facilitate understanding is through human facilitators. These facilitators could be individuals involved in the installation, deployment, or support of the system. For example:

Installer and Support Staff: When users install the software, the process could include an explanation of how the model works. The installer could walk the user through the key features of the system and explain how it processes ASL gestures. In the case of more complex systems, having knowledgeable support staff available via phone or email can help clarify doubts and ensure users understand the system. This would be particularly valuable for individuals who may not be comfortable with technical language or who need additional assistance.

Customer Support Line: On the deployment website or within the app, a support number could be provided for users to call if they need further explanation or encounter issues with the system. Having a direct line to a knowledgeable person can significantly enhance user trust, particularly if the user is unsure how the system works or has encountered errors that need clarification.

User Guide and Documentation: Another key aspect of post-hoc explainability is the creation of a user guide that explains the workings of the system in layman's terms. This document could explain the key concepts behind ASL recognition and how the machine learning models contribute to the recognition process. Including diagrams, screenshots, or even videos could greatly enhance comprehension.

6.2.2 Textual and Visual Explanations: Simplification and Clarification

To further facilitate understanding, the documentation can be augmented with textual and visual explanations. A key component of this approach is simplification:

Textual Explanations: Providing a clear, accessible explanation of the underlying technologies used, such as CNNs and MLPs, and how they are applied to ASL recognition, is essential. This report itself can serve as a useful starting point for explaining the methodology. We aim to explain the technical details in a manner that is comprehensible to users without a technical background. Additionally, this information could be expanded in an online user guide or FAQ section, where common queries are answered.

Example-Based Explanation: The explanation can also be based on examples. For instance, showing how a particular gesture is processed from the input (a webcam image) through to its classification as an ASL letter. The guide can explain the flow of the data through the model, demonstrating each stage of the recognition process. By providing such examples, users can better understand how their input is interpreted.

6.2.3 Model Visualization: Understanding the Architecture

Visual tools can also be valuable in making the inner workings of the model more transparent. This could include:

Model Architecture Diagrams: A diagram showing how the CNN and MLP models are structured and how they interact with the input data can help users visualize the flow of information. The inclusion of such a diagram on the website, or within the app's documentation, could be an effective way to communicate the model's decision-making process.

Activation Maps and Filters: Visualization techniques such as saliency maps or activation maps could be used to highlight the regions of the input image that the model deems most important when making a decision. For example, showing which parts of a hand gesture the model is focusing on can give users an insight into how the model is recognizing ASL letters.

6.2.4 Feedback Mechanism: User-Driven Improvement

Finally, it is important to provide a means for users to evaluate the explanations they receive and offer feedback. This could be implemented in the following ways:

Feedback System: On the website or within the software, there could be a feedback system where users can rate the clarity of the explanations on a scale from 1 to 10. This will not only provide valuable insights into the effectiveness of the explanations but also allow for continuous improvement of the documentation and user experience.

Continuous Updates: Incorporating feedback from users will allow for the refinement of both the explanation materials and the system itself. For example, if users consistently struggle to understand certain aspects of the model's decision-making process, this feedback can be used to adjust the explanations or even improve the model's clarity.

6.3 Conclusion

Explainability is a vital component of any AI system, particularly those impacting accessibility and communication, like our ASL recognition application. By engaging stakeholders through various post-hoc explainability methods—including human facilitators, textual and visual explanations, and feedback systems—we can ensure that users feel confident in the system's performance and understand how it works. Clear communication and transparency will help build trust in the technology and foster a sense of empowerment for users, particularly within the Deaf and hard-of-hearing communities. Through ongoing engagement and feedback, we can continue to improve the model and the ways in which it is explained to users.

Chapter 7

Ethical Considerations

The development of an ASL recognition application presents several ethical considerations that must be addressed to ensure its responsible deployment and impact.

One of the key ethical concerns is accessibility. While the application is designed to facilitate communication for individuals using ASL, it is essential to ensure that the technology remains inclusive. This means accounting for variations in signing styles, hand shapes, and even regional differences in ASL. If the model is not trained on a diverse dataset, it may introduce biases that exclude certain users from effectively utilizing the system.

Solution: To address this, datasets should be carefully curated to include diverse signers from different backgrounds, age groups, and regional dialects of ASL. Regular updates and community involvement can help improve inclusivity.

Another significant ethical consideration is privacy. The system relies on real-time video input from users, raising concerns about data security and consent. Any application utilizing webcam-based recognition should prioritize user privacy by implementing measures such as local processing (to avoid data storage on external servers), encrypted data handling, and transparent policies informing users about how their data is processed.

Solution: Implementing strict data policies, allowing users to process data locally without storing images, and ensuring compliance with privacy regulations such as GDPR can mitigate these concerns.

Beyond individual users, the widespread adoption of such a technology could have societal implications. For instance, there is a potential risk that automated sign language recognition could be used to replace human interpreters in critical settings, such as hospitals or legal institutions, where human intuition and contextual understanding are essential. It is important to frame the system as an assistive tool rather than a replacement for human expertise.

Solution: Clear documentation and ethical guidelines should emphasize that this technology is meant to supplement human interpreters rather than replace them. Stakeholder consultations with interpreters and the Deaf community can provide insights into responsible deployment.

On a positive note, the application has immense potential to improve accessibility and inclusion for the Deaf and hard-of-hearing communities. By enabling real-time conversion of

ASL into text, it could help bridge communication gaps in everyday scenarios, such as ordering at a restaurant, asking for directions, or engaging in public services. Additionally, it could serve as an educational tool, assisting new learners in understanding and practicing ASL more effectively.

Solution: Developing user-friendly interfaces, multilingual support, and educational tutorials can enhance the application's accessibility and usability.

Future advancements could expand the system's capabilities to recognize more complex gestures, allowing full sentence recognition rather than just individual letters. Integrating natural language processing (NLP) techniques could help interpret ASL grammar and structure, making interactions even more seamless.

Solution: Continuous research, open-source collaboration, and AI model refinement should be encouraged to enhance recognition capabilities over time.

Ultimately, ethical AI development requires continuous evaluation, feedback from the Deaf community, and transparent decision-making in model training and deployment. Addressing these concerns ensures that technology serves as an enabler of communication rather than a barrier or a source of unintended harm.

Chapter 8

Conclusion

Throughout this project, we explored various approaches to recognizing American Sign Language (ASL) letters using artificial intelligence. Initially, we experimented with Convolutional Neural Networks (CNNs) trained on publicly available datasets, but we encountered significant limitations due to dataset biases and overfitting. To address these issues, we created our own datasets under different conditions—fixed backgrounds, moving backgrounds, and hand-tracking preprocessing using MediaPipe.

Our results demonstrated that CNNs could achieve high accuracy when trained on large, well-balanced datasets with diverse samples. However, due to our dataset limitations, the CNN models struggled to generalize well to real-world images. If we had access to a dataset with thousands of hand gestures made by different people in various environments, CNNs could have been a viable approach. This suggests that CNN-based ASL recognition remains a promising avenue for future work.

To overcome our dataset constraints, we shifted our approach by leveraging hand-tracking data from MediaPipe. Instead of raw images, we extracted numerical coordinates representing key hand landmarks and trained a Multi-Layer Perceptron (MLP) model. This approach proved to be highly effective, achieving near-perfect accuracy while significantly reducing computational complexity. By focusing on essential hand features rather than processing entire images, we made the model more lightweight and adaptable for real-time applications.

From an ethical standpoint, we ensured user privacy by processing hand gestures locally rather than storing image data, while providing possible solutions for ethical aspects that should be considered before the deployment. Additionally, we incorporated Explainable AI (XAI) techniques, such as SHAP analysis, to provide transparency into the model's decision-making process, ensuring that stakeholders could understand and trust its predictions.

Ultimately, our project successfully developed an efficient and practical ASL recognition system. While CNNs remain a viable option given a sufficiently large and diverse dataset, our MediaPipe-MLP approach provided a robust alternative suited to our constraints. This research lays the foundation for future improvements, including expanding the system to recognize full ASL sentences and enhancing real-world usability.

Bibliography

[1] Keras Applications API. Keras applications api. Accessed: 2025-02-12.

[2] Alfredo Baldominos, Yago Sáez, and Pedro Isasi. A survey of handwritten character recognition with mnist and emnist. *Applied Sciences*, 2019.

[3] Mayank Bali. Sign language detection using deep learning, mediapipe opencv. Accessed: 2025-02-12.

[4] Anwar Botalb, Moiz Moinuddin, U.M. Al-Saggaf, and S.S.A. Ali. Contrasting convolutional neural network (cnn) with multi-layer perceptron (mlp) for big data analysis. In *Proceedings of the International Conference on Intelligent and Advanced Systems*, 2018.

[5] Kaggle – ASL Classification Using CNN. Kaggle – asl classification using cnn. Accessed: 2025-02-12.

[6] Gabriel Cohen, Shabnam Afshar, John Tapson, and Arend van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[7] DataCamp. Introduction to shap values and model interpretability, 2021. Accessed: 2025-02-12.

[8] Kaggle – ASL Alphabet Dataset. Kaggle – asl alphabet dataset. Accessed: 2025-02-12.

[9] Kaggle – Sign Language Detection. Kaggle – sign language detection. Accessed: 2025-02-12.

[10] Mediapipe Hand Landmark Detection. Mediapipe hand landmark detection. Accessed: 2025-02-12.

[11] SHAP Docs. Mnist example. Accessed: 2025-02-12.

[12] MediaPipe Documentation. Mediapipe documentation. Accessed: 2025-02-12.

[13] SHAP Documentation. Shap documentation. Accessed: 2025-02-12.

[14] TensorFlow.js Documentation. Tensorflow.js documentation. Accessed: 2025-02-12.

[15] TensorFlow/Keras Documentation. Tensorflow/keras documentation. Accessed: 2025-02-12.

[16] Luciano Floridi. *Artificial Intelligence, Ethics, and Society.* 2018.

[17] GitHub – Deep Learning for Computer Vision. Github – deep learning for computer vision. Accessed: 2025-02-12.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[19] Jing Huang, Wei Zhou, and Hengrong Li. Sign language recognition using deep neural networks. In *Proceedings of the 23rd ACM International Conference on Multimedia*, pages 97–101, 2015.

[20] GitHub – Sign Language Detector in Python. Github – sign language detector in python. Accessed: 2025-02-12.

[21] Kirenz. Explainable ai with tensorflow, keras, and shap, 2022. Accessed: 2025-02-12.

[22] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[23] Machine Learning Mastery. Why one-hot encode data in machine learning, 2021. Accessed: 2025-02-12.

[24] Pavlo Molchanov, Shun Yang, Xianzhi Gupta, Kyung Kim, Steve Tyree, and Jan Kautz. Hand gesture recognition with 3d convolutional neural networks. In *Proceedings of the IEEE CVPR Workshops*, 2015.

[25] PyImageSearch. Convolutional neural networks (cnns) and layer types, 2021. Accessed: 2025-02-12.

[26] SHAP GitHub Repository. Shap github repository. Accessed: 2025-02-12.

[27] Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[28] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2016.

[29] scikit-learn Documentation. scikit-learn documentation. Accessed: 2025-02-12.

[30] TechTarget. Definition of convolutional neural networks (cnn). Accessed: 2025-02-12.

[31] Max Tegmark. *Life 3.0: Being Human in the Age of Artificial Intelligence*. 2017.

[32] GitHub – ASL Recognition with MediaPipe ML. Github – asl recognition with mediapipe ml. Accessed: 2025-02-12.

[33] GitHub – ASL Detection with OpenCV. Github – asl detection with opencv. Accessed: 2025-02-12.

[34] Ryo Yamashita, Masahiro Nishio, R. K. G. Do, and Kei Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9:611–629, 2018.