Gesture recognition for video game controllers

Jonatan Ebenholm[1]

Abstract
This project explores the transition of gesture recognition from specialized hardware, such as the Microsoft Kinect, to software-defined systems utilizing standard RGB webcams. Two primary methodologies were evaluated: a custom-built Convolutional Neural Network (CNN) trained on the HaGRID and self-made datasets, and a Feed Forward Neural Network (FFNN) utilizing 21 3D hand landmarks extracted via the MediaPipe framework. Results indicate that while the CNN model achieved functional real-time performance, it struggled with hand rotations and required significantly longer training times. Conversely, the MediaPipe + FFNN hybrid approach achieved a 100% validation success rate with near-instantaneous training times of less than one second per epoch. This study concludes that landmark-based neural networks provide a superior, high-accuracy, and real-time alternative for interactive media control without the need for dedicated infrared sensors.
Source code: https://github.com/Sahriz/TNM114

Authors
[1] *Media Technology Student at Linköping University, joneb274@student.liu.se*

Keywords: Neural Networks — Computer Vision — Data Sets

Contents

1. Introduction

For most of the digital world's history, a physical controller of some sort has been used to steer or control what is seen on screen. There are, however, exceptions to this standard way of interacting with computers. One prominent case is the Microsoft Kinect [1]. Released to the public in 2010, the Kinect used infrared projectors and detectors to map body gestures and perform skeletal detection. This approach was highly effective for its time, but it required very specific hardware to run.

Since then, the fields of neural networks and gesture recognition have evolved significantly. Modern advancements mean that a standard Python program can now achieve what the Kinect was designed to do, but by using only the RGB components of a basic webcam. In this project, I will implement a system where a simple webcam can recognize gestures based on hand movements. This is achieved through a neural network trained on data specifically curated for this purpose, moving the complexity from specialized hardware into the software itself.

2. Theory

This section outlines the theoretical framework required to understand the methodology of this project. The primary objective was to develop and train a neural network capable of classifying distinct hand gestures through computer vision. The performance of such a network is heavily reliant on the quality and structure of the training data, which must be tailored to the specific architecture of the chosen model.

2.1 Neural Networks

Neural networks are computational models inspired by the biological processes of the human brain. These models consist of interconnected layers of nodes, or neurons, which process data through mathematical transformations. In a supervised learning context, a neural network is trained to map specific inputs to corresponding labels by adjusting the internal weights of these neurons. This process allows the model to identify complex patterns and separate different classes of data within a high-dimensional space.

Feed forward Neural Network A Feed Forward Neural Network is the most fundamental type of artificial neural network. In this architecture, information moves in only one direction, starting from the input layer, passing through hidden layers, and reaching the output layer. There are no cycles or loops in the network. For this project, a feed forward structure is used to take the processed hand coordinates and calculate the probability of those coordinates representing a specific gesture. Because the input data is already structured as a set of points, a feed forward network is highly efficient for this type of classification task.

Convolutional Neural Network Convolutional Neural Networks (CNNs) are specialized architectures designed primarily for processing structured arrays of data, such as images [2]. They utilize a mathematical operation called convolution to automatically detect spatial hierarchies and features within a frame. While a feed forward network works well with coordinate data, a CNN is typically required if the system must analyze raw pixel data directly. Understanding CNNs is important for this project because many modern gesture recognition systems, including parts of the MediaPipe framework, rely on convolutional layers to locate the hand within a video feed.

Mediapipe MediaPipe is an open-source framework developed by Google that provides cross-platform solutions for live and streaming media [3]. In the context of hand tracking, MediaPipe utilizes a pre-trained model to detect 21 distinct 3D landmarks on a human hand[3]. These landmarks represent key points such as fingertips, knuckles, and the wrist. By using MediaPipe, the system can bypass the need to process raw images through a custom CNN, instead receiving a clean set of coordinates that describe the hand's posture with high precision.

Putting it together The integration of these technologies creates a streamlined pipeline for real-time control. First, OpenCV captures the raw video frames from the webcam. These frames are passed to MediaPipe, which performs the heavy lifting of computer vision by identifying the hand and extracting 21 specific landmark coordinates in 3D space.

Instead of feeding thousands of raw pixels into the neural network, the system only sends these 21 coordinates. This significantly reduces the computational load, allowing the Feed Forward Neural Network to classify the gesture almost instantaneously. The resulting classification is then mapped to a specific game command, such as a key press or mouse click, completing the loop from physical movement to digital action.

2.2 Computer vision & OpenCV

Computer vision is the field of artificial intelligence that enables computers to derive meaningful information from digital images or videos. OpenCV (Open Source Computer Vision Library) is the primary tool used in this project to handle image processing. It provides the necessary functions to read the webcam buffer, resize images, and convert color spaces. These preprocessing steps ensure that the visual data is in the correct format for MediaPipe and the neural network to analyze.

2.3 Datasets

To train and use a Neural Network, tailored data needs to be used. This section will give background as to what that data might look like for different systems and what they have in common.

Preprocessing One of the most important aspects of a classifier is to preprocess the data as to not input irrelevant information that might prolong or ruin the training. This might mean removing entire datapoints, or scaling it to fit within a ceratin space or removing dimensions entirely.

Overfitting Overfitting in ML is when a model is not generalised to the problem and fits to the training data. This will cause it to fail when being tested on new data which was not in the training phase which is where the model is meant to perform.

Hagrid dataset The HAnd Gesture Recognition Image Dataset (HaGRID) is a large-scale collection of images designed for training neural networks to recognize gestures in diverse, real-world environments [4]. This dataset is particularly useful for ensuring model robustness against varying backgrounds and lighting conditions.

Landmark based dataset Mediapipe produces 21 separate coordinates, or landmarks, based on key sections of the hand, of which can be placed in 3D-space. A Neural Network can then be trained on the coordinates to learn the pattern in different gestures.

3. Method

This section discusses how the theory was applied in the project to gather the results in the next section. Every file can be found on the Github linked with this project.

3.1 Data collection

To train the networks there was a need for varied data. Three different methods for this was tried

CNN dataset Two variants of datasets were tested for the CNN-model, one of which was self made. Both datasets were preprocessed to increase both the reliability and the success rate of the results.

- **Self made dataset -** The selfmade dataset was created by using OpenCV and the python program *get_image_data.py* for data collection and Mediapipe and the program *pre_process_image.py* to preprocess the images. Each gesture were put into seperate folders with correct labels which could then be fetched when training the models. The preprocessing for the image data was to normalize the colour space and change the size of the image to

128x128. To make sure the right section of the image was saved, Mediapipe was used to first find where in the image the hand was located, then the image was cut and scaled to 128x128. The gestures that were made was as follows:

1. up
2. up-left
3. left
4. down-left
5. down
6. down-right
7. right
8. up-right
9. scatter
10. cluster
11. attack

The model was trained with a variant of self made datasets, first with around 100 images per gesture, and later there was either fewer or greater amounts, but a lot of it was trial and error. Example images with their assigned gestures are seen in

- **HaGRID dataset -** As with the self made dataset, the HaGRID dataset had to be preprocessed in the very same way. The big advantage with this dataset is that it is highly curated and has a lot of varied backgrounds and different hands which allows the model not to overfit as hard. The dataset has preset gestures, so there were no additional gestures other than those. The preset gestures were the ones seen in

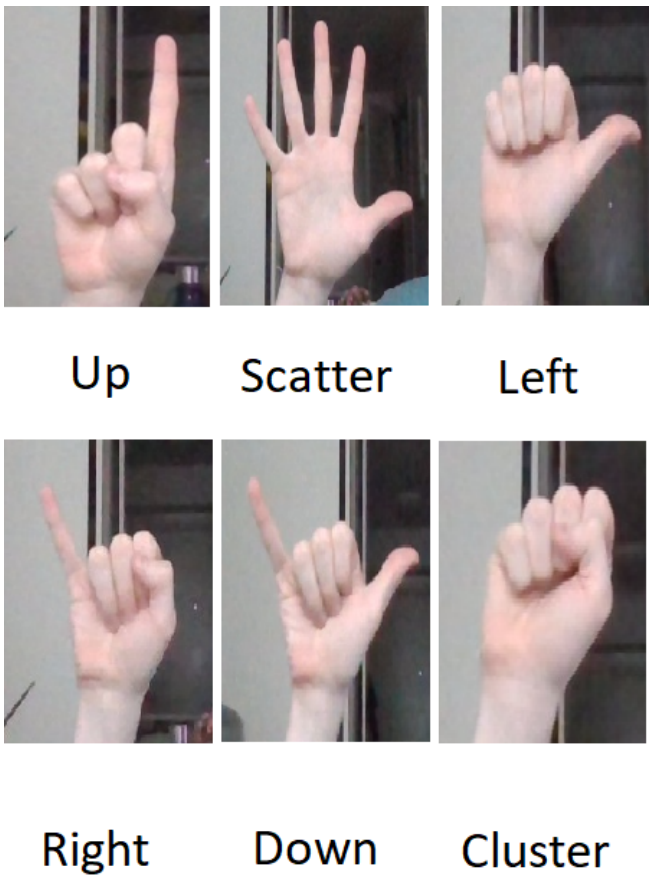

Figure 1. Example images of self made dataset. The images are not preprocessed.

Mediapipe dataset The mediapipe dataset dataset was gathered with a python program called *get_mediapipe_data.py* and preprocessed with the program *landmark-processing.py* and saved into *gesture_landmarks.csv* in a text format.

3.2 Network structure

The implementation involved two distinct architectural approaches to gesture recognition: a custom-built Convolutional Neural Network (CNN) for raw image processing and a Feed Forward Neural Network (FFNN) designed to work with landmark coordinates.

Feed forward The Feed Forward model was implemented to handle the 21 3D-coordinates provided by the MediaPipe framework.

- **Input Layer:** The input layer consists of 63 nodes, representing the x, y, z coordinates for each of the 21 landmarks.
- **Hidden Layers:** The architecture utilizes several dense (fully connected) layers to identify geometric patterns in the hand's posture.

Figure 2. Example images of all the gestures from the HaGRID dataset [4].

- **Data Format:** The model was trained using the data curated via *landmark-processing.py* and stored in *gesture_landmarks.csv*.

CNN The CNN model was designed to process the 128x128 preprocessed images from the self-made and HaGRID datasets.

- **Feature Extraction:** The network utilize convolutional layers to automatically detect spatial hierarchies and features, such as finger orientations and hand shapes.
- **Processing:** Images were converted to a normalized color space and resized to 128x128 pixels to ensure consistency during the training phase.
- **Regularization:** To prevent overfitting, the model was tested with both the self-made dataset and the highly curated HaGRID dataset to ensure robustness against varied backgrounds.

Mediapipe implementation The most successful model utilized a hybrid pipeline that combined MediaPipe's robust detection with the efficiency of a Feed Forward Neural Network.

1. **Frame Capture:** OpenCV captures raw video frames from the webcam in real-time.
2. **Landmark Extraction:** MediaPipe identifies the hand and returns 21 specific landmark coordinates in 3D space, bypassing the need for a custom CNN to process raw pixels.
3. **Coordinate Normalization:** Before classification, coordinates are processed via *landmark-processing.py* to ensure the model is invariant to the hand's distance or position in the frame.
4. **Classification and Mapping:** The FFNN classifies the gesture almost instantaneously, and the result is mapped to a specific game command (e.g., "attack" or "scatter") using a controller script.

4. Result

Single CNN The performance of the CNN model can be seen in both figure 3 and figure 4, while the time each epoch took to train can be seen in figure 5. The validation accuracy was 88%.

Mediapipe + Feed Forward Neural Network The performance of the Mediapipe model can be seen in both figure 6 and figure 7, while the time each epoch took to train can be seen in figure 8. The validation accuracy was 100%. '

Real-time use While it is difficult to show the real-time functionality, a simple summary of the issues found and the successes can be explained.

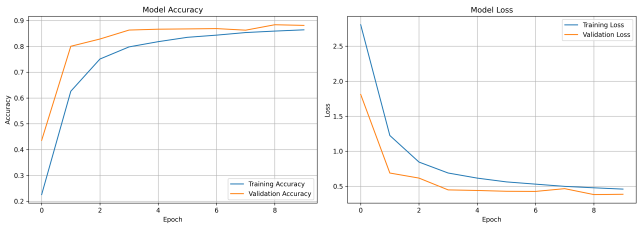

Figure 3. Training history of the CNN model with 10 epochs and a batch size of 64

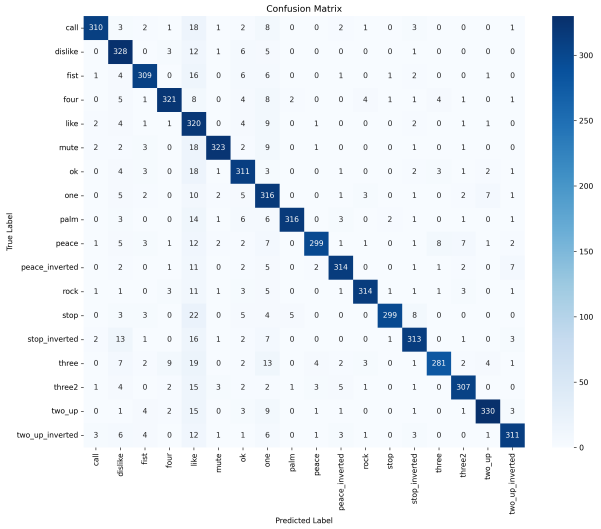

Figure 4. Confusion matrix of the CNN model with 10 epochs and a batch size of 64

```
Epoch 2/10
398/398 ━━━━━━━━━━━━━━━━━━━━ 72s 181ms/step - accuracy: 0.6261 - loss: 1.2243 - val_accuracy: 0.8803 - val_loss: 0.6884 - l
earning_rate: 0.0010
```

Figure 5. Training information of the CNN model from its second epoch.

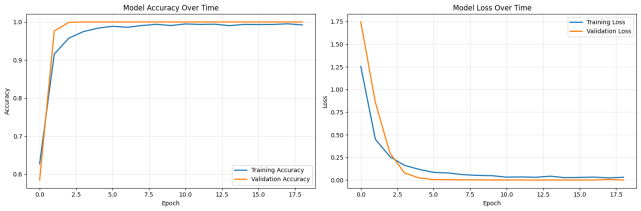

Figure 6. Training history of the Mediapipe model with 100 epochs and a batch size of 32

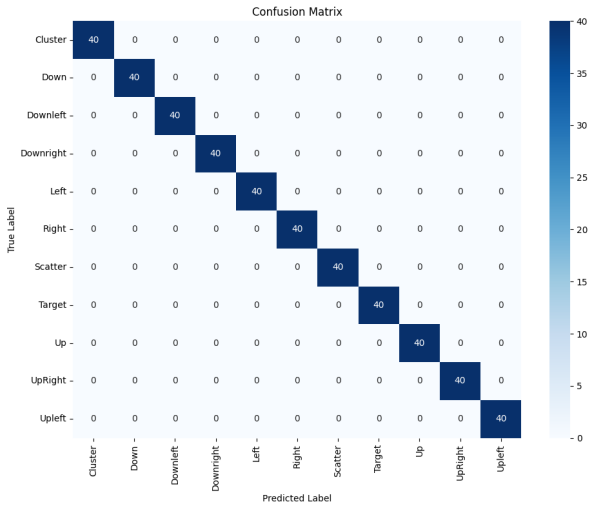

Figure 7. Confusion matrix of the Mediapipe model with 100 epochs and a batch size of 32, stopping early at 19 epochs due to the model validating at 100%.

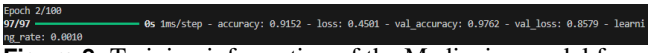

Figure 8. Training information of the Mediapipe model from its second epoch.

The CNN model was rather successful, working in real time and not giving much issues or miss labelling. It does however fail when turning the hand a bit.

The Mediapipe model however gave near perfect results, being real time and working in nearly any lighting condition and with any rotation given.

5. Discussion

Comparing the results of the different models from figure 4 and figure 7, CNN HaGRID vs FF Mediapipe, it is clear that the Mediapipe version is superior since it hits 100% success rate on validation. This is to be expected since Mediapipe is a model created by Google which gives accurate data specifically for hand related problems, and while the HaGRID dataset allowed for a functional single CNN that gives positive results, it is only really good for frontal views of gestures and fails when rotating the hands even slightly while being less accurate even in its best case use. Another big issue with the CNN model is that the time to train is much longer than that of the simple FF structure of the Mediapipe model. This is seen in figure 5 and figure 8, where each epoch takes a millisecond for the FF model and more than a minute for the CNN model.

Although the self made datasets do not have any results listed in the result section, it should be mentioned that they were tested and validated, but the scores they produces were no better than guessing. This is because the data was both lacking in quantity and variation (backgrounds, lighting, skin colour, etc.). An educated guess would be that the main reason for failure would be quantity data. If the self made one had vaired backgrounds, more people and differnent lighting, as well as at least ten times more data per gesture, it might perform better than what was tested in this project. This would allow for additional gestures to be added outside of the HaGRID dataset.

In the pursuit of making a game controller based on webcams RGB-outputs the project is a success. With a simple python implementation and free packages like OpenCV and Mediapipe, anyone can create their own simple dataset and train their own model to use in videogames with near 100% accuracy and real-time response time.

In the final tests which are were performed with real-time data, the CNN model performed fine, but failed with rotating gestures, which was not an issue for the Mediapipe model. If there were any other gestures that you would want to add to each model, a clear issue would be that the CNN model works with a pre made dataset which would need thousands of images added to it to retrain the model on, while also taking a lot of time to train once the data is there. The Mediapipe model would not run into such an issue, since adding another gesture is low effort and would likely yield great results and works great with any pair of hands regardless of background, and then it will train in less than a minute and be ready to use again.

6. Conclusion

In conclusion, it is difficult to beat the performance of a Neural Network made by Google to create specialized data for hands. The free Mediapipe model makes the training stage and user stage near perfect, and it allow for anyone to create a dataset that fits a simple FFNN and gives real-time, near 100% accurate computor vision controllers from any RGB-webcam on the market, removing the need for tailor-made hardware like the Kinect for gesture recognition, and possibly whole body controllers as well using Googles other machine vision tools.

References

[1] Wikipedia contributors. Kinect. 2025.

[2] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4):611–629, 2018.

[3] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *CoRR*, abs/2006.10214, 2020.

[4] Kapitanov Alexander, Kvanchiani Karina, Nagaev Alexander, Kraynov Roman, and Makhliarchuk Andrei. Hagrid – hand gesture recognition image dataset. In *2024*

IEEE/CVF Winter Conference on Applications of Computer Vision (WACV). IEEE, January 2024.

1. Large Figures

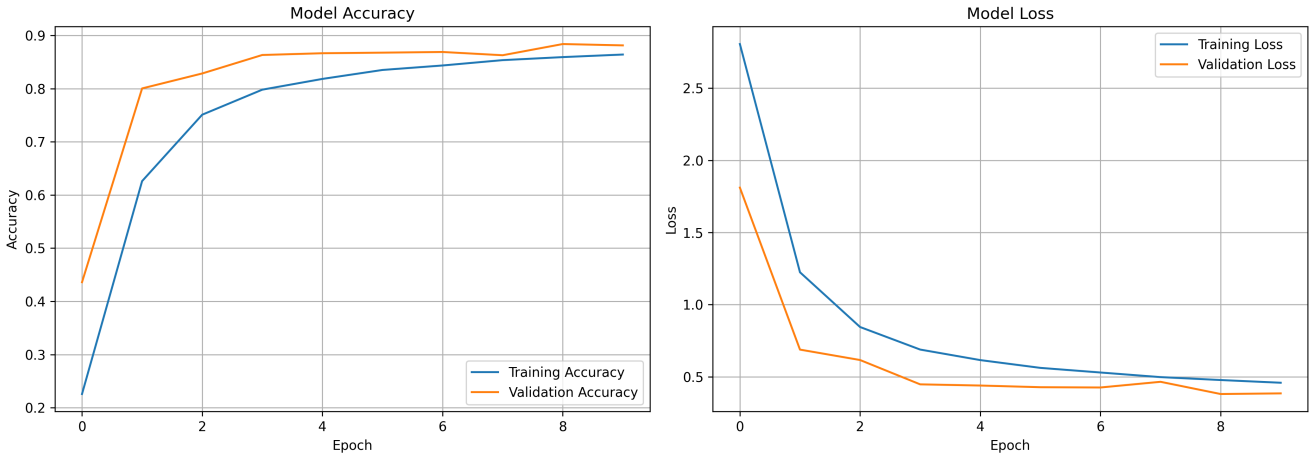

Figure 9. Training history of the CNN model with 10 epochs and a batch size of 64

,

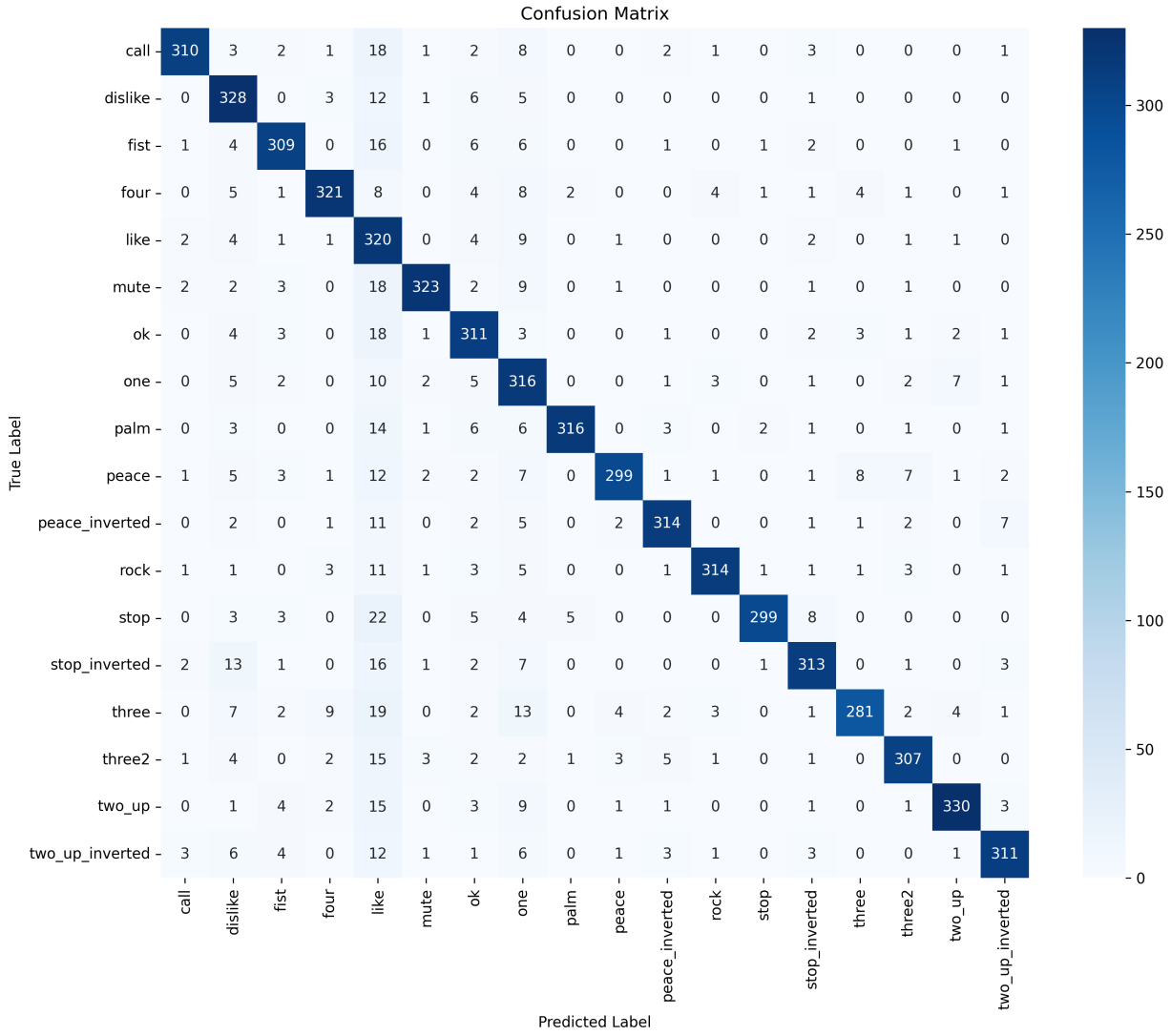

Figure 10. Confusion matrix of the CNN model with 10 epochs and a batch size of 64

```
Epoch 2/10
398/398 ━━━━━━━━━━━━━━━━━━━━ 72s 181ms/step - accuracy: 0.6261 - loss: 1.2243 - val_accuracy: 0.8003 - val_loss: 0.6884 - l
earning_rate: 0.0010
```

Figure 11. Training information of the CNN model from its second epoch.

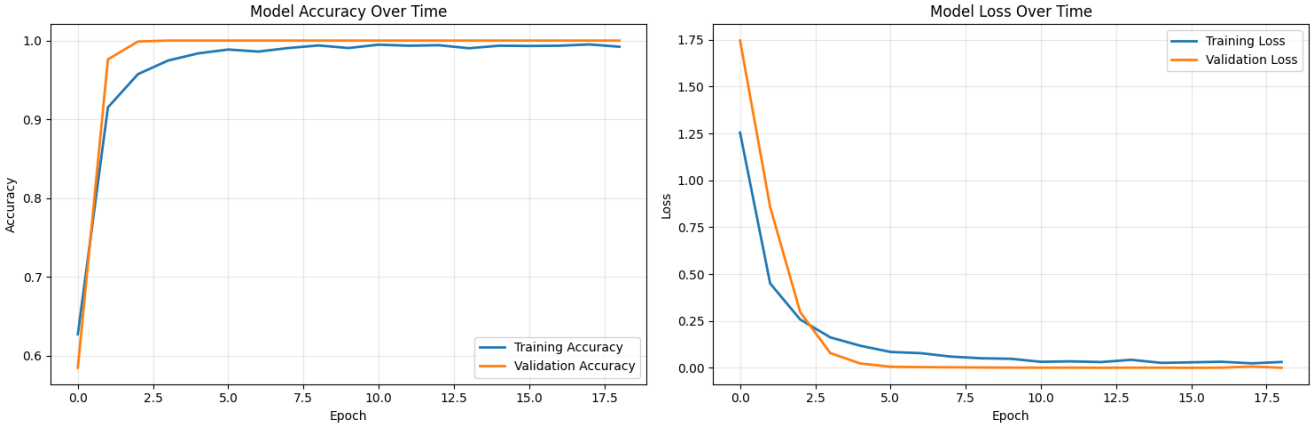

Figure 12. Training history of the Mediapipe model with 100 epochs and a batch size of 32

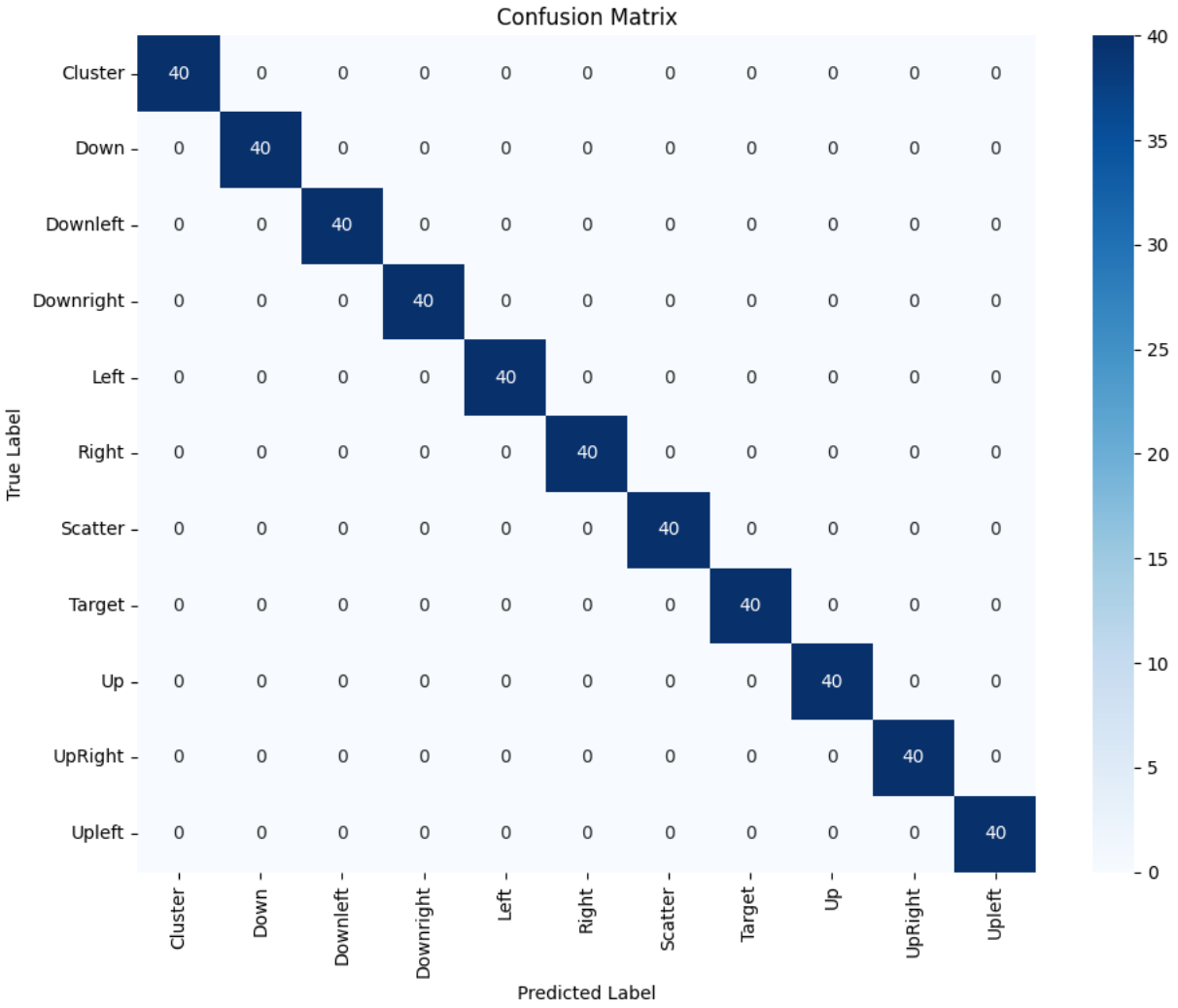

Figure 13. Confusion matrix of the Mediapipe model with 100 epochs and a batch size of 32, stopping early at 19 epochs due to the model validating at 100%.

```
Epoch 2/100
97/97 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9152 - loss: 0.4501 - val_accuracy: 0.9762 - val_loss: 0.8579 - learni
ng_rate: 0.0010
```

Figure 14. Training information of the Mediapipe model from its second epoch.