Hand Gesture Recognition with MediaPipe and TensorFlow

Bernadette Ackerl
Data Science and Engineering
University of Applied Sciences Upper Austria
Hagenberg, Austria

Fabian Altendorfer
Data Science and Engineering
University of Applied Sciences Upper Austria
Hagenberg, Austria

***Abstract*—Hand gesture recognition plays a vital role in human-computer interaction, with applications ranging from assistive technologies to virtual reality. This study explores the use of MediaPipe, OpenCV, and TensorFlow for developing machine learning models trained on a dataset of 9.600 samples across six distinct gestures: "Telephone," "Fist," "Thumb-Up," "Thumb-Down," "Open-Hand," and "Peace." Three deep learning models—two convolutional neural networks (CNNs) and one Recurrent Neural Network (RNN) were trained using MediaPipe's landmark detection.**

While the models achieved high accuracies of over 80% in controlled environments, additional testing with previously uninvolved individuals revealed limitations in MediaPipe's landmark detection, particularly in multi-hand scenarios and complex gesture recognition. The study highlights the critical dependence of gesture classification performance on the accuracy of underlying landmark detection algorithms.

To improve robustness, future work should focus on more inclusive testing with diverse demographics, enhancing environmental adaptability, and leveraging multi-frame analysis strategies. These insights provide a foundation for advancing adaptive and reliable hand gesture recognition systems.

I. Introduction

The field of human-computer interaction is witnessing more and more a paradigm shift towards natural and intuitive interfaces, with hand gesture recognition emerging as a pivotal technology. [2]This research paper is driven by the need to enhance interaction in various applications, from entertainment to healthcare, by overcoming the limitations of current gesture recognition systems, particularly in real-world scenarios, including varying lighting, different user positions, and multi-hand scenarios. This study addresses these challenges by integrating MediaPipe and TensorFlow to develop machine learning models capable of accurate and real-time gesture recognition. Specifically, it explores how MediaPipe's landmark detection framework can be leveraged while identifying its limitations in recognizing complex gestures. The research evaluates three deep learning models—a Convolutional Neural Network (CNN), an optimized CNN, and a Long Short-Term Memory (LSTM) RNN—trained on six distinct hand gestures. Through this study, the authors explore the motivations behind such advancements, articulate specific challenges in gesture recognition, review current technologies in the field, and present this system's design and implementation using Python, MediaPipe, TensorFlow, and OpenCV. By building upon MediaPipe's pre-trained framework for hand-landmarks-detection, the authors aim to demonstrate how its strengths can be harnessed while addressing its limitations in challenging environments.

A. Motivation

The motivation behind this research paper stems from the growing demand for intuitive, non-contact interfaces in human-computer interaction. Hand gesture recognition offers a natural method for interaction, enhancing user experience in applications ranging from gaming to assistive technologies. By focusing on gesture recognition through computer vision, the authors aim to contribute to the development of more accessible and interactive systems, particularly in environments where touch-based interaction is impractical or undesirable.

B. Problem Statement

The primary challenge the authors had to address was the accurate and real-time recognition of hand gestures under varying lighting conditions and distances from the camera. The goal was to develop a robust system that can reliably interpret a set of common hand gestures irrespective of ambient lighting, gender of the user, and spatial positioning relative to the capture device. The set of common hand gestures involves six types: Fist, Thumb-Up, Thumb-Down, Open-Hand, Peace and Telephone.

C. State of the Art Hand Gesture Recognition

During the search for a State of the Art approach for hand gesture recognition in the internet and the literature, the authors found the public repository of MediaPipe on the platform Github[1]. MediaPipe is an Open-Source platform developed by Google which released the first version of MediaPipe alongside a paper in 2019, with the goal to make the combined use of Machine-Learning and Computer Vision in Real-Time-Applications more accessible and easier implementable. MediaPipe offers a broad range of predefined solutions for tasks like hand gesture recognition, face detection, body analysis and object detection in videos, which enables applications to capture the motion of users, for example, in games [5]. The repository of MediaPipe has, at the time of writing, 28.400 Stars and was forked 5.200 times, which indicates

[1]url: https://github.com/google-ai-edge/mediapipe

a very strong relevance for the Open-Source Community. But also for researchers in the space, MediaPipe seems to be highly relevant with around 2.770 articles since 2024 in Google Scholar at the time of writing. This is not surprising, as MediaPipe supports many different types of hardware. With Mediapipe, so called ”perception pipelines” can be built, which consist of computation nodes - transparent boxes and inputs/outputs which together form a graph. An example for such a graph can be seen in Figure 1

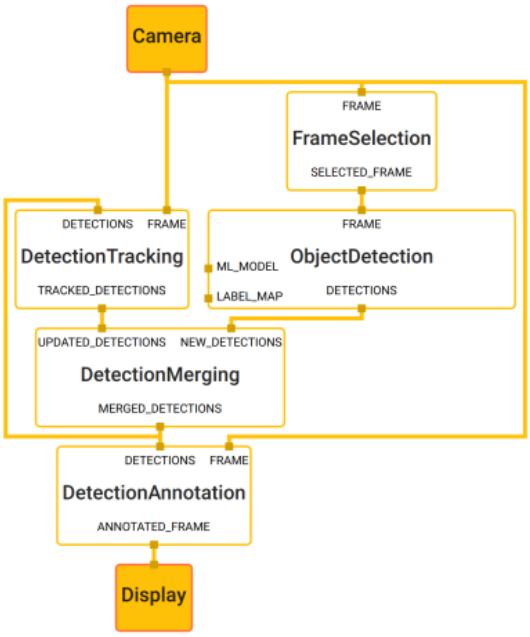

Fig. 1. Example for a MediaPipe Perception Pipeline consisting of different calculators

The computation nodes, which are also called calculators, are connected via data streams which themselfs consist of data packets. This connection to a graph is standard in modern neural network engines like TensorFlow or PyTorch, however the directed-graph used in MediaPipe operates at significantly higher semantics than is possible with neural networks, making it more performant in the audio/video domain. The higher semantics are achieved in MediaPipe by using a GraphConfig specification which allows the definition of subgraphs and their use as calculators in the graph. This architecture allows MediaPipe to be hardware agnostic, as the developer can decide how deep and ressource intensive a specific subgraph needs or is allowed to be. Calculators and therefor also subgraphs can receive multiple input streams, which is part of the reason MediaPipe is better suited for specific domains compared to the frameworks TensorFlow and PyTorch. Calculators support computation by CPU aswell as GPU. In this paper, the implementation of a solution for hand-gesture recognition is proposed, including a live-demonstration. For this usecase, the authors found that MediaPipes capabilities are ideal. For optimal performance of the live-demonstration, the landmark detection and gesture segmentation should run in parallel and not be blocked in some way, which is easily realizable with a perception pipeline in MediaPipe [3].

The term landmark in context with MediaPipe is used to refer to precise keypoint coordinates of detected hand regions. MediaPipe is able to detect those landmarks based on vast amounts of ground truth data, which consists of 30.000 real world images with 21 3D coordinates manually annotated by Google Research. As discovered in the GitHub repository of MediaPipe, a TensorFlow Lite model is utilized for the inference of the hand-landmarks, unfortunately, as of the time of writing, the trained model is not included in the publicly available repository[1].

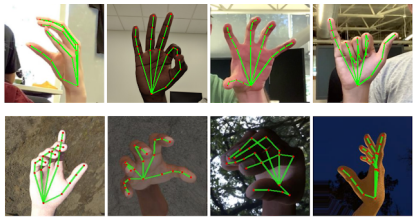

Fig. 2. Examples of Hand Landmarks detected by MediaPipe

Landmarks can be used as a basis for hand-gesture recognition. On top of the initially predicted hand skeleton, MediaPipe applies a simple algorithm to derive the gestures, which identifies the state of each finger by the angles of its landmark-joints and maps the states to pre-defined gestures. Although this can be achieved relying solely on MediaPipe, the authors extend this idea by using TensorFlow-Models outside of MediaPipe, for the prediction of the gesture.

```
node {
  calculator: "SwitchContainer"
  input_side_packet: "SELECT:model_complexity"
  output_side_packet: "PACKET:model_path"
  options: {
    [mediapipe.SwitchContainerOptions.ext] {
      select: 1
      contained_node: {
        calculator: "ConstantSidePacketCalculator"
        options: {
          [mediapipe.ConstantSidePacketCalculatorOptions.ext]: {
            packet {
              string_value: "mediapipe/modules/hand_landmark/hand_landmark_lite.tflite"
            }
          }
        }
      }
      contained_node: {
        calculator: "ConstantSidePacketCalculator"
        options: {
          [mediapipe.ConstantSidePacketCalculatorOptions.ext]: {
            packet {
              string_value: "mediapipe/modules/hand_landmark/hand_landmark_full.tflite"
            }
          }
        }
      }
    }
  }
}
```

Fig. 3. Example for a calculator/node in MediaPipe

While MediaPipe’s hand tracking capabilities are revolutionary, its utility extends far beyond this single application.

For example, MediaPipe Holistic can be used to track face, hands and posture simultaneously, even on mobile devices. MediaPipe is also able to detect Iris in Videos, Instant-Motions, and faces [1, 4].

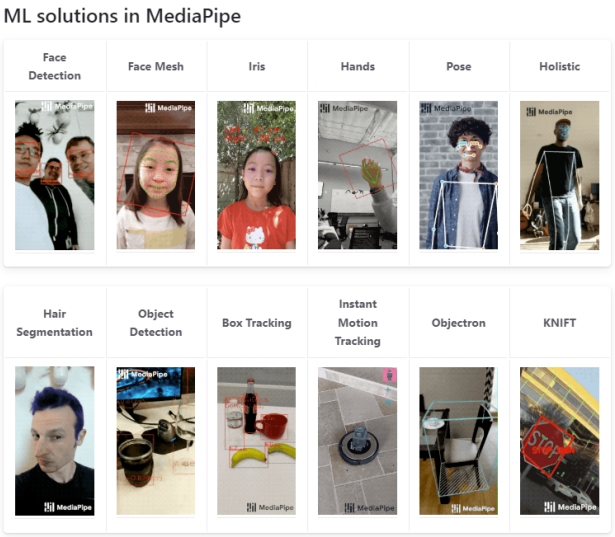

Fig. 4. Capabilities of the MediaPipe Framework

D. TensorFlow

TensorFlow is an end-to-end, open-source platform for machine learning, with a particular focus on deep learning. It supports the implementation of various neural network architectures, such as CNNs for image recognition tasks, which involves understanding convolutional layers, pooling, back-propagation, and gradient descent optimization techniques. TensorFlow operates on a computational graph model where operations are nodes and data flows as tensors along edges, optimizing computations for both CPU and GPU, which is crucial for training and inference in gesture recognition. Additionally, TensorFlow facilitates transfer learning, allowing pre-trained models to be fine-tuned on new tasks with limited data, which is particularly useful in gesture recognition where domain-specific datasets might be smaller compared to general image datasets. Although MediaPipe already uses TensorFlow behind the scenes, the authors included it nonetheless to demonstrate the versatile approach of MediaPipe. Of course, one could use PyTorch or any other framework instead of TensorFlow.

II. Material

For capturing the images and developing the code, the authors utilized their local notebooks. The training of the machine learning models was conducted using cloud resources on GoogleColab.

A. Test Data Utilized

Photographs of the authors were taken for the initial training and preliminary testing of the machine learning models, capturing various conditions. More details on this in section III-A.

For the final testing, in addition to testing-evaluation while implementing each model, photographs of selected participants were taken.

B. Hardware

The acquisition of images and the identification of gesture landmarks were performed using the integrated webcams of the notebooks. The hardware involved included a Lenovo ThinkPad T14 Gen 2 and a MacBook Pro 2018 equipped with an Intel Core i7 processor. Once the gesture landmarks were captured, all data files were systematically uploaded on a Google Drive. This data was then leveraged for the training of a Convolutional Neural Network (CNN) within a Google Colab environment, utilizing Python. The training of the machine learning model was executed on an NVIDIA Tesla T4 GPU, ensuring efficient processing and analysis of the dataset.

III. Methodology

An experiment was conducted by constructing a custom pipeline using MediaPipe, OpenCV, and TensorFlow. Three machine learning models were trained and tested as part of this experiment.

The methodology for this study encompasses the data collection process, model design, and evaluation strategy, providing a comprehensive framework for the development and assessment of hand gesture recognition models. The initial training and preliminary testing of the machine learning models used photos of the authors themselves, focusing on various conditions such as left, right, close, far, and under low and normal lighting to evaluate MediaPipe's performance. MediaPipe provides the landmarks necessary for these predictions even under those varied circumstances.

In the second phase, the models were tested in real-life scenarios with selected participants to assess their practical applicability.

A. Data Collection

The dataset was constructed to include six distinct gestures: "Telephone," "Fist," "Thumb-Up," "Thumb-Down," "Open-Hand," and "Peace." The data collection process involved two individuals (the authors), contributing 800 samples per gesture each, resulting in a total of 1600 samples per gesture. To ensure robustness and diversity, the gestures were performed under varying conditions, including changes in lighting, background, hand positions (e.g., rotating of hand, sitting close to a laptop or standing farther away), and using both the left and right hands. Some examples are shown in figure 5 and 6.

The dataset was generated using a Python script that leveraged the MediaPipe framework to extract up to 63 normalized landmark coordinates (x, y, z) for each sample. These landmarks represent key points on the hand, enabling the models to effectively distinguish between gestures (Please see Figure 2 for an example).

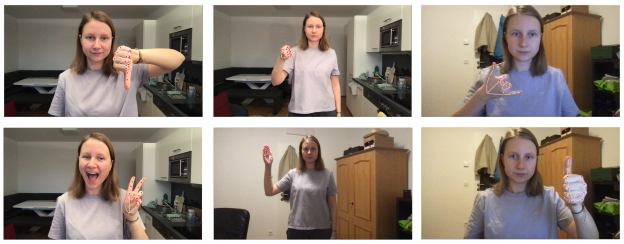

Fig. 5. Example of Data Collection Process 1

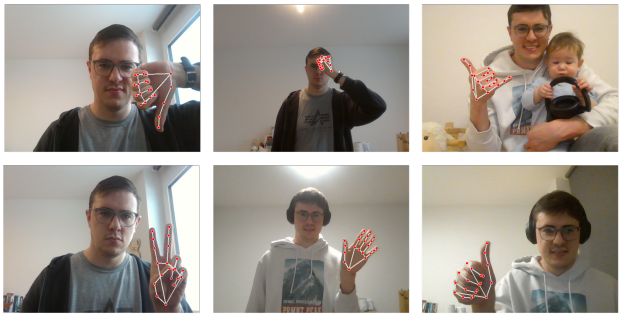

Fig. 6. Example of Data Collection Process 2

The captured landmarks were saved as individual files, ensuring compatibility for subsequent model training and evaluation.

B. Model Design

Three distinct deep learning models were implemented to explore different architectural approaches:

- A convolutional neural network (CNN) with three convolutional layers for feature extraction.
- An optimized CNN with adjusted kernel sizes and filter configurations.
- A recurrent neural network (RNN) using Long Short-Term Memory (LSTM) layers to capture temporal dependencies in the landmark data.

The specifics of each model are detailed in Section *Models*, where their architectures, configurations, and training processes are discussed in depth.

C. Evaluation Strategy

The evaluation of the models was conducted using a train-test split on the collected dataset. The training and validation results for each model are reported in Section *Results*. To further assess the generalizability of the models, additional hand gesture samples were collected from two individuals who were not part of the initial data collection process. These additional samples were used exclusively for testing purposes, offering insights into the performance of the models on previously unseen data. The testing results can also be seen in Section *Results*.

IV. Models

This section presents three deep learning models developed for hand gesture recognition based on MediaPipe landmarks. Each model leverages a specific architecture designed to effectively process and classify the input data. The architecture and training setup for each model are described below.

A. Input Data

The input data consists of hand gesture landmarks captured using MediaPipe. Six distinct gestures were selected: ”Telephon,” ”Fist,” ”Thumb-Up,” ”Thumb-Down,” ”Open-Hand,” and ”Peace.” For each gesture, 1600 samples were collected. The dataset was generated under varying conditions, including different lighting, positions (sitting in front of the laptop and standing at the back), and for both left and right hands, as already explained in section III-A. This diversity enhances the robustness of the models. Each sample contains 63 normalized landmark coordinates (x, y, z) derived from the MediaPipe framework. Consequently, the input shape of each of the following model is set to (63, 1), while the final dense layer comprises 6 units, corresponding to the number of distinct gestures.

B. Model 1: Convolutional Neural Network (CNN)

Model 1 utilizes a convolutional neural network (CNN) to extract features from the input data. The architecture consists of three convolutional layers with ReLU activation, followed by max-pooling layers. After flattening the features, two dense layers are applied, with a dropout layer for regularization. The output layer employs a softmax activation to classify six hand gestures. The detailed architecture is as follows:

- Input shape: (63, 1)
- Conv1D (64 filters, kernel size 3, ReLU activation)
- MaxPooling1D (pool size 2)
- Conv1D (128 filters, kernel size 3, ReLU activation)
- MaxPooling1D (pool size 2)
- Conv1D (64 filters, kernel size 3, ReLU activation)
- Flatten layer
- Dense (64 units, ReLU activation)
- Dropout (rate 0.5)
- Dense (6 units, softmax activation)

```
model = Sequential([
    Conv1D(64, 3, activation='relu', input_shape=(63, 1)),
    MaxPooling1D(2),
    Conv1D(128, 3, activation='relu'),
    MaxPooling1D(2),
    Conv1D(64, 3, activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(6, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.3)
```

Fig. 7. Model1

The model was trained using the Adam optimizer and categorical cross-entropy loss, with a batch size of 32 and 50 epochs, as shown in figure 7.

C. Model 2: Optimized Convolutional Neural Network (CNN)

Model 2 refines the CNN approach by using different configurations for kernel sizes, number of filters, and dropout rates. The architecture is optimized for better generalization and performance. Key features include:

- Input shape: (63, 1)
- Conv1D (32 filters, kernel size 5, ReLU activation)
- MaxPooling1D (pool size 2)
- Conv1D (64 filters, kernel size 5, ReLU activation)
- MaxPooling1D (pool size 2)
- Flatten layer
- Dense (128 units, ReLU activation)
- Dropout (rate 0.3)
- Dense (64 units, ReLU activation)
- Dense (6 units, softmax activation)

```
model = Sequential([
    Conv1D(32, 5, activation='relu', input_shape=(63, 1)),
    MaxPooling1D(2),
    Conv1D(64, 5, activation='relu'),
    MaxPooling1D(2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dense(6, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.3)
```

Fig. 8. Model2

Similar to Model 1, this model was trained with the Adam optimizer and categorical cross-entropy loss, but with 100 epochs for improved performance. Figure 8. represents it.

D. Model 3: Long Short-Term Memory Network (LSTM)

Model 3 employs a recurrent neural network (RNN) with Long Short-Term Memory (LSTM) layers to capture temporal dependencies in the input sequence data. The architecture consists of two LSTM layers, followed by dense layers for classification. Key components are:

- Input shape: (63, 1)
- LSTM (64 units, return sequences: True)
- Dropout (rate 0.2)
- LSTM (32 units, return sequences: False)
- Dropout (rate 0.2)
- Dense (32 units, ReLU activation)
- Dense (6 units, softmax activation)

This model was also trained with the Adam optimizer and categorical cross-entropy loss, using a batch size of 32 and 100 epochs, as shown in the above figure 9.
These three models provide diverse approaches to hand gesture recognition, offering a comparative perspective on convolutional and recurrent neural network architectures.

V. Results

The following table shows the training output values after the model has completed the training process. All epochs

```
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(63, 1)),
    Dropout(0.2),
    LSTM(32, return_sequences=False),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(6, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.3)
```

Fig. 9. Model3

have been processed, and the metrics reflect the performance of each model on both the training and validation datasets. Accuracy represents the percentage of correctly classified samples, while loss quantifies how far the model's predictions deviate from the actual labels, with lower loss indicating better performance.

TABLE I
Training Output Values

Model	Accuracy	Loss
Model1	0.9761	0.0638
Model2	0.9955	0.0141
Model3	0.9803	0.0631

The training output values indicate that all models performed well during training, with accuracies above 97%. Model2 achieved the highest accuracy of 99.55%, suggesting it learned the training data exceptionally well, but it also has a relatively low loss of 0.0141, indicating minimal error in predictions. Model1 and Model3 also show strong performance, with accuracies of 97.61% and 98.03%, respectively, and their validation accuracies are similarly high, indicating good generalization to unseen data.

The next table summarizes the test accuracy for each model, providing insights into their performance on unseen data.

TABLE II
Test Accuracy Overview

Model	Accuracy
Model 1	0.9854
Model 2	0.9937
Model 3	0.9743

The test accuracy overview shows that all models maintained high performance on unseen data, with Model2 again leading with a test accuracy of 99.37%. This indicates that it not only excelled during training but also generalized well to new examples. Model1 followed closely with an accuracy of 98.54%, demonstrating robust performance as well, while Model3 had a slightly lower test accuracy of 97.43%. Overall, these results suggest that all models are effective for the given task, with Model2 being the most reliable choice based on its superior metrics.

A. Test with Not-Involved Individuals

As already stated in section Methodology, the authors evaluated the generalizability of their models beyond the initial dataset and conducted an additional test phase involving two individuals who did not participate in the primary data collection. This approach aimed to assess how well the trained models perform on completely new, unseen hand gesture samples. These participants were selected to ensure a diverse testing set, introducing variability that was not present in the training data. The hand gesture samples collected from these individuals were used exclusively for testing purposes, providing a robust measure of the models' performance in real-world scenarios where the data might differ from the training set.

Fig. 10. Testsamples individual 1

Fig. 11. Testsamples individual 2

TABLE III
TEST ACCURACY OVERVIEW FOR 12 VALIDATION IMAGES

Model	Accuracy
Model 1	0.8333
Model 2	0.7500
Model 3	0.8333

The results for each model in detail are presented in the figures 12, 13, 14.

As this paper was created during the class "Computer Vision", the authors decided to take a closer look at the test result, especially the MediaPipe-Element in the described pipeline. The closer inspection revealed that MediaPipe has problems with identifying the correct hand if two hands of an individual

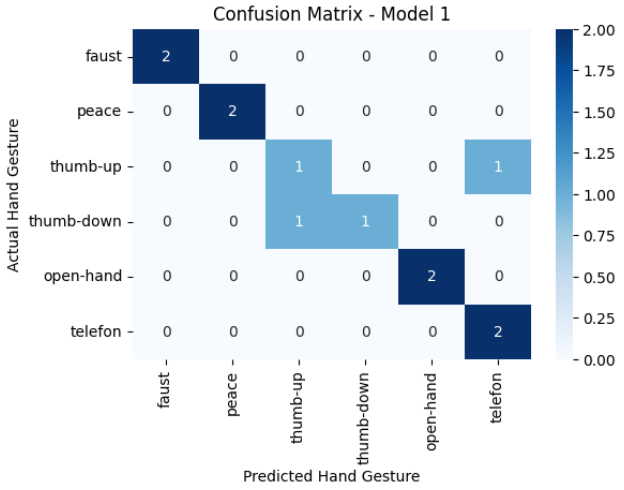

Fig. 12. Results Model 1

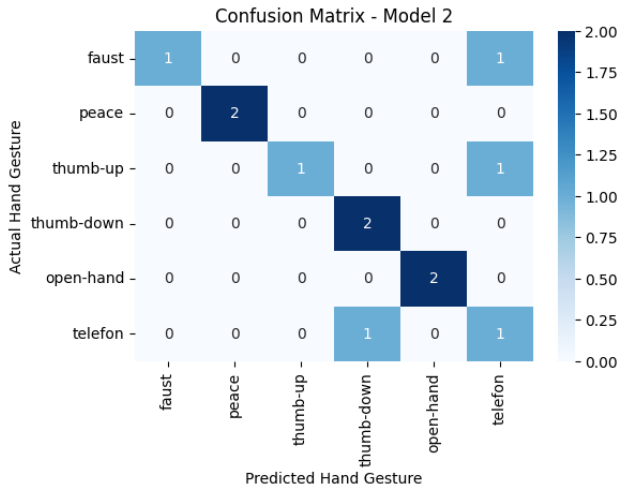

Fig. 13. Results Model 2

are present in the picture. Additionally, in one case, MediaPipe falsely detected a fist instead of the intended "thumbs-down" gesture. The examples for this can be seen in Figure 15. These false detections explain partly the misclassifications seen in the Test-Results. However, the described problems appear to be gender-neutral.

VI. DISCUSSION

The focus of this study was to evaluate the performance of MediaPipe within the context of hand gesture recognition in a computer vision pipeline. The analysis of the test results revealed several critical insights about MediaPipe's functionality and its impact on those custom gesture recognition models.

Limitations:

- **Diversity of test subjects:** The dataset used for testing (authors as well as volunteers) did not include individuals with darker skin tones, children without adults in the picture, or multiple individuals in a single frame. This lack of diversity might skew the reliability of MediaPipe under

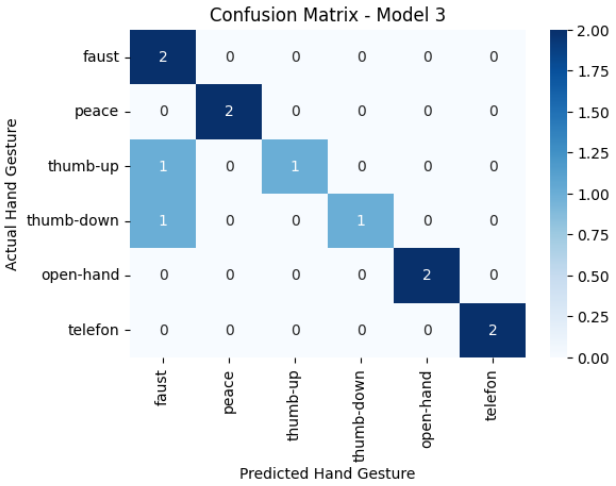

Fig. 14. Results Model 3

Fig. 15. Examples where the landmark detection failed

varied conditions, as skin color and age can significantly affect detection accuracy.

- **Environmental conditions:** These experiments were conducted without accounting for additional environmental variables such as direct sunlight or different angles of capture. These conditions could potentially alter the performance of MediaPipe in real-world applications.
- **Gesture complexity:** MediaPipe struggled with identifying the correct hand when both hands were present in the image, as well as misclassifying specific gestures like the "thumbs-down" gesture as a fist. These issues are depicted in Figure 15, highlighting the limitations in handling more complex or nuanced hand gestures.
- **Hand coverage:** There was no exploration into how MediaPipe reacts to hands covered with gloves, especially colored ones, or when part of the hand is obscured, such as by clothing like a T-shirt sleeve extending over the hand.
- **Gesture training and dependency on MediaPipe:** The models were trained exclusively on datasets derived from MediaPipe's hand landmark detection. As a result, their performance was inherently tied to how accurately MediaPipe estimated these landmarks. During testing, it became evident that misclassifications often occurred when MediaPipe failed to correctly detect or localize hand landmarks. For example, incorrect landmark placement due to occlusions or poor lighting directly impacted the model's ability to classify gestures accurately. This dependency highlights a cascading effect where errors in MediaPipe's implementation propagate through the pipeline, ultimately affecting model predictions. Additionally, there was no training or testing on scenarios involving less conventional hand positions, like using three fingers due to injury or disability.
- **Usage of other technologies:** In the presented paper, only MediaPipe was used to detect landmarks in frames of videos. Future work could broaden the scope in terms of including different technologies.

Analysis of Results:
Despite these limitations, the results for the tested scenarios were generally positive. With two models achieving accuracies above 80%. However, this success must be contextualized within the narrow scope of the test conditions. The observed misclassifications were largely attributable to challenges faced by MediaPipe in multi-hand detection and gesture misinterpretation, as demonstrated in the examples. Additionally, since the models relied solely on MediaPipe's landmark data for training, their accuracy was constrained by how well MediaPipe performed under specific conditions.

VII. Conclusion and Outlook

In conclusion, while MediaPipe shows promise in straightforward hand gesture recognition tasks, it's deployment in more dynamic or varied real-world scenarios requires further refinement and testing to address the identified limitations. Ideas for this could be:

- **Inclusive testing:** Future studies should aim to include a broader demographic, including different skin tones, ages, and conditions that affect hand visibility. Again, for example, injured hands (only 4 fingers left).
- **Environmental testing:** Conduct experiments under various lighting conditions and from multiple angles to better understand MediaPipe's robustness.
- **Alternative methods:** Consider integrating or comparing with alternative hand detection and gesture recognition systems to benchmark MediaPipe's performance.
- **Use of nultiple pictures/video:** As MediaPipe showed weeknesses when two hands are present in the picture, a more reliable approach could be built around using multiple pictures of the same individual to take multiple landmark-records into account for the prediction by the machine-learning model.

References

[1] Valentin Bazarevsky and Fan Zhang. *On-Device, Real-Time Hand Tracking with MediaPipe*. Google AI Blog. Accessed: 2025-01-23. Aug. 2019. URL: https://research.google/blog/on-device-real-time-hand-tracking-with-mediapipe/.

[2] Victor Chang et al. “An Exploration into Human–Computer Interaction: Hand Gesture Recognition Management in a Challenging Environment”. In: *SN Computer Science* 4 (2023), p. 441. DOI: 10.1007/s42979-023-01751-y. URL: https://doi.org/10.1007/s42979-023-01751-y.

[3] Camillo Lugaresi et al. “MediaPipe: A Framework for Building Perception Pipelines”. In: *arXiv preprint arXiv:1906.08172* (2019). arXiv: 1906.08172 [cs.DC].

[4] Kanstantsin Sokal. *MediaPipe 3D Face Transform*. Google Developers Blog. Accessed: 2025-01-24. Sept. 2020. URL: https://developers.googleblog.com/en/mediapipe-3d-face-transform/.

[5] Qiang Zhou and Byung Gook Lee. “Optical motion capture application based on Unity3D and MediaPipe”. In: *2024 3rd International Conference on Cloud Computing, Big Data Application and Software Engineering (CBASE)*. IEEE. Feb. 2024, pp. 618–623. ISBN: 979-8-3315-0658-2. DOI: 10.1109/CBASE64041.2024.10824452.