Illinois Institute of Technology

Department of Computer Science

CS-512

Computer Vision

Final Project - Report

Real-time Hand Gesture Recognition

Authors:
Bastien LEDUC
Paul LEGOUT

Student Numbers:
A20520860
A20522029

April 18, 2023

Team contribution

Bastien: Implementation of the entire hand tracking module using MediaPipe (hand detection, bounding boxes and landmarks retrieval), as well as the incorporation of the predictive models into the final application. Generation of two separate scripts for the different modes of the live application. Recording of live application demo. Participation in retrieving the entire dataset due to its size. Redaction of the report and presentation support.

Paul: Implementation, training, and optimization of the various networks presented, including image preprocessing. Participation in including the models in the final application, as well as retrieving the dataset. Redaction of the report and presentation support.

Table of contents

1 Problem statement

The problem statement for this computer vision project is to develop an efficient and accurate real-time on-device hand gesture recognition system that can detect and classify hand poses from video input, without relying on movement information.

Real-time on-device hand gesture recognition is a critical task in computer vision that involves identifying and classifying hand poses in real time. This is one of the most common computer vision tasks with numerous applications such as virtual reality, gaming, sign language recognition, human-robot interaction, and many others.

The main challenge in this task is to accurately detect and classify various hand poses in real time while running the algorithm on limited computational resources. This implies that the model used for the recognition must be lightweight and fast enough to run on a mobile device, with low latency and high accuracy.

Another significant constraint is the size of the data required to train such models. Hand gesture recognition models require large amounts of annotated data to be trained effectively.

In this project, we aim to develop a robust real-time on-device hand gesture recognition system that accurately detects and classifies various hand poses with high accuracy while minimizing computational resources, model complexity, and training data requirements. We will explore different deep learning and CNN architectures and evaluate their performance under different constraints, including computation resources, model complexity, and training data requirements.

2 Proposed solution

To address the problem of real-time on-device hand gesture recognition, we propose to use the MediaPipe framework, which is an open-source cross-platform solution for building pipeline processing models for different computer vision tasks. MediaPipe provides various pre-built computer vision modules, including a hand detector module that provides hand landmarks and bounding box coordinates.

We will use MediaPipe's hand detector module to detect the hand's bounding box and landmarks, which will be used as input for our hand gesture classification model. We will use a shallow 1D CNN architecture to classify different hand gestures based on the detected landmarks (see Fig. 1). This will serve as our baseline for our other models.

Additionally, we will explore the possibility of implementing our own MediaPipe framework for hand landmark detection and bounding box detection using pre-trained networks, such as VGG16 and MobileNet. We will train these CNNs on the provided dataset mentioned in the next part, and we will evaluate their performance in terms of accuracy, precision, recall, F1-Score, and speed.

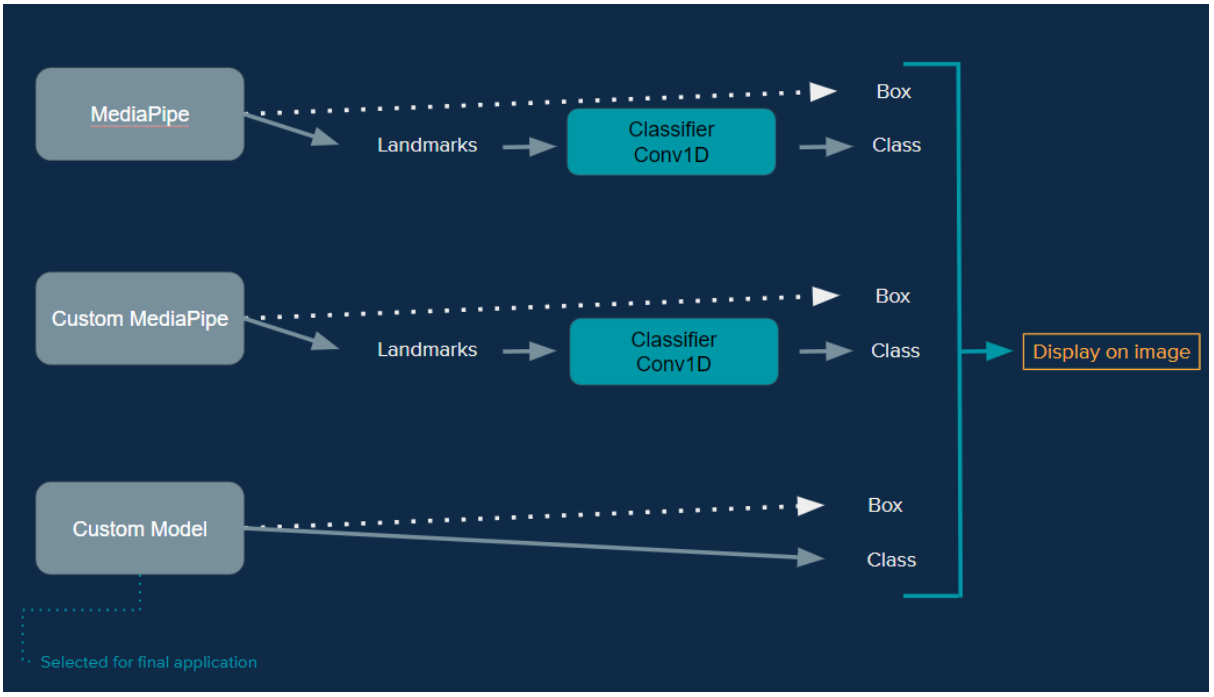


Figure 1: Pipeline of our different models.

Note 2.1: *You can find 2 demonstration videos of our models. This* ***first one*** *shows the top model of the pipeline (MediaPipe framework). This* ***second one*** *shows the bottom model of the pipeline (Custom model).*

Overall, our proposed solution leverages the power of the MediaPipe framework and pre-trained networks to develop a real-time on-device hand gesture recognition system with high accuracy, low computational complexity, and minimal training data requirements.

3 Implementation details

3.1 Chosen dataset

Dataset The data that we decided to use comes from the HaGRID - HAnd Gesture Recognition Image Dataset. HaGRID is a large and diverse dataset for Hand Gesture Recognition (HGR) tasks. The dataset is primarily aimed at building HGR systems for various applications, including video conferencing services, home automation systems, the automotive sector, and services for people with speech and hearing impairments. The dataset also focuses on interaction with devices to manage them.

The HaGRID dataset contains 552,992 mostly HD/FullHD RGB images divided into 18 classes of gestures (see Figure 2), along with an additional "no_gesture" class for cases where there is a second free hand in the frame. The dataset size is 716GB, making it one of the largest HGR datasets available. The images were collected from a diverse group of subjects in various conditions (lighting variations, blur from movement, distance from the camera, over/under exposure, etc.), making the dataset one of the most diverse in terms of subjects.

HaGRID has potential applications beyond HGR, as it can be a useful dataset for many computer vision tasks such as pose estimation and face detection. Overall, HaGRID is a valuable resource for researchers and developers interested in HGR and related computer vision tasks.

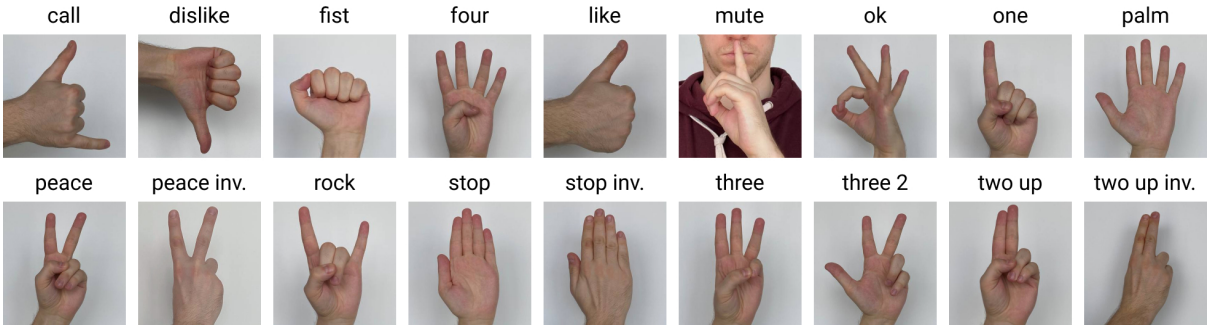


Figure 2: All 18 gestures from HaGRID.

Additionally, the dataset contains JSON annotation files for all 18 classes with the following elements. For each `image_id`, we have:

- `bboxes`: list of bounding boxes coordinates in COCO format [top left X position, top left Y position, width, height].
- `landmarks`: list of hand landmarks coordinates of shape [nb_hands, nb_landmarks=21, 3], where the last dimension corresponds to the landmark format [lm_id$\in [0, 20]$, x, y] in relative image coordinates.
- `labels`: list of gesture labels.
- `leading_hand`: markup to identify the gesture hand (`left` or `right`).
- `leading_conf`: confidence for `leading_hand` annotation.

Data preparation For obvious reasons of storage capacity, training, and computational resources, we decided to only keep subsamples of size 2000-3000 for each of the 18 classes, which of course had a negative influence on the models' performances during training, to which we can add the image downsampling to fit the input size of our pre-trained models.

Data split 70% training, 20% testing, and 10% validation for every proposed model.

3.2 Models creation

General information

- Except for contradictory information, the activation function for every layer is ReLu.
- A block consists of one type of layer, batch normalization and an activation function
- An early stopper is used, with patience = 5, monitoring the validation loss
- A learning rate reducer has been set, monitoring the validation loss. The initial value of the learning rate is 0.001.
- The Adam optimizer is used for each model

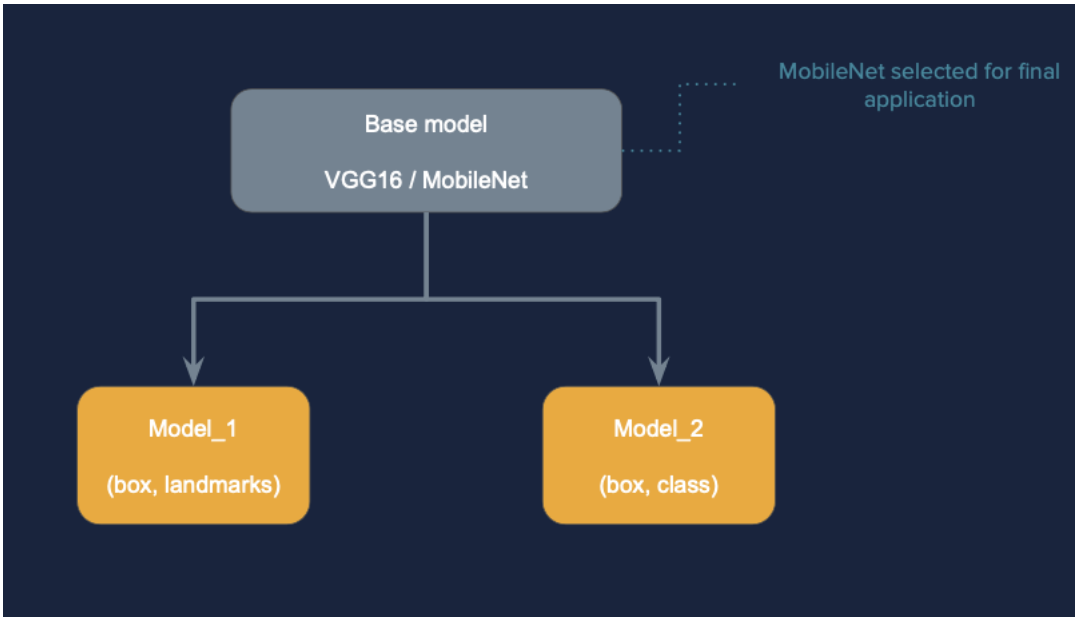


Figure 3: Models

In addition to the classifier trained on landmarks, we created four additional models, that predict directly from an image. We selected two pre-trained networks that were trained on ImageNet as our base models. Subsequently, we developed two models to integrate with the chosen base models, one that predicts the box and landmark coordinates, and another that predicts the box coordinates and class.

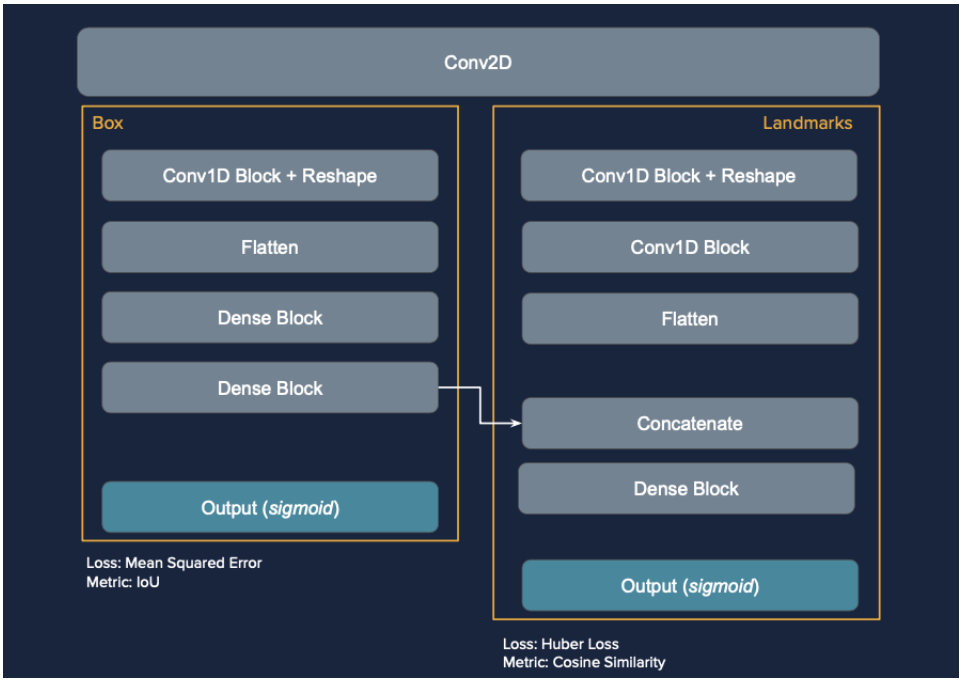


Figure 4: Model 1: box + landmarks

The first model is composed of a shared 2D convolutional layer, followed by a split into two sections: one for box prediction and the other for landmarks prediction. In 4, we observe a concatenation from the box branch to the landmark branch, which we found to enhance landmark prediction.

This model employs two loss functions: Mean Squared Error for box prediction and Huber loss for landmark prediction. Additionally, two metrics are utilized: Intersection over Union (IoU) for box prediction and Cosine Similarity for landmark prediction.

We applied the Sigmoid activation function for both outputs, as the coordinates being predicted were normalized.

One of the challenges in developing a neural network with multiple outputs is to ensure that the complexity of the individual branches matches the complexity of their respective tasks. If the complexity of the branches is not well balanced, the model may end up overfitting on one task while performing poorly on the other.

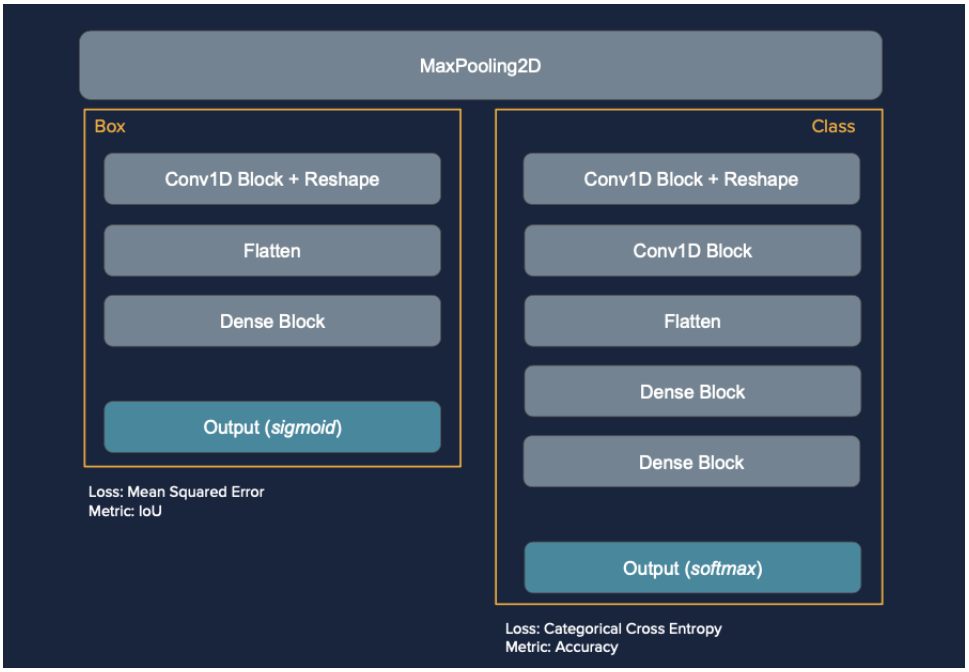


Figure 5: Model 2: box + class

The second model is composed of a shared 2D max pooling layer, followed by a split into two sections: one for box prediction and the other for class prediction.

This model also employs two loss functions: Mean Squared Error for box prediction and Categorical Cross Entropy for class prediction. Additionally, two metrics are utilized: Intersection over Union (IoU) for box prediction and Accuracy for class prediction.

We applied the Softmax activation function for the class output, as the class are encoded using the One Hot Encoding technique.

4 Results and discussions

Initially, we developed a classifier that utilized landmarks as input and made predictions about the class. We also created a customized MediaPipe implementation, which allowed us to utilize the same classifier for class predictions based on landmarks.

However, the landmarks predicted by this approach were not precise enough to yield favorable results with the classifier. Therefore, we decided to use an "all-in-one" custom model that could predict both the box and the class directly from an image without relying on the intermediate step of predicting landmarks.

This approach offered two advantages. Firstly, it improved the overall performance of the model. Secondly, it made the overall application more lightweight compared to using two sequential models, resulting in a satisfactory frame rate.

To predict from a live video stream at a decent rate per second, we chose MobileNet as the base model due to its lightweight nature.

The combination of MobileNet and the "all-in-one" model produces high accuracy as well as a good frame rate. The frame rate is comparable to MediaPipe + classifier. Nevertheless, the overall application, using our custom model, has a lower accuracy and precision of the predicted box, which is not surprising since MediaPipe is well known and used, therefore it has been optimized and trained on an important volume of data.

Metrics To evaluate the quality of our predictions, we used additional metrics to the simple accuracy.

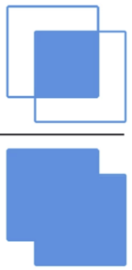
$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{4}$$

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{5}$$



4.1 Classifier (Conv1D from landmarks)

The classifier has been trained on more than 490,000 landmarks.

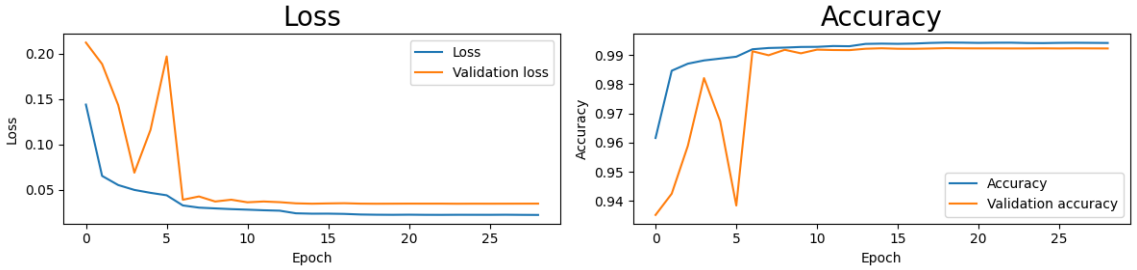


Figure 6: Classifier training

Loss	Accuracy	Precision	Recall	F1-Score	Min	Mean	Max
0.0367853	99.179%	99.177%	99.187%	99.182%	0.0	0.05263158	1.0

Table 1: Classifier metrics on test data.

The results from 1 demonstrate exceptional performance. The combination of MediaPipe and the classifier yielded outstanding outcomes.

4.2 Custom MediaPipe Model (box + landmarks)

This time, we tried to implement our own version of MediaPipe using a pretrained MobileNet model that predicts the bbox and landmark coordinates (on a single hand).

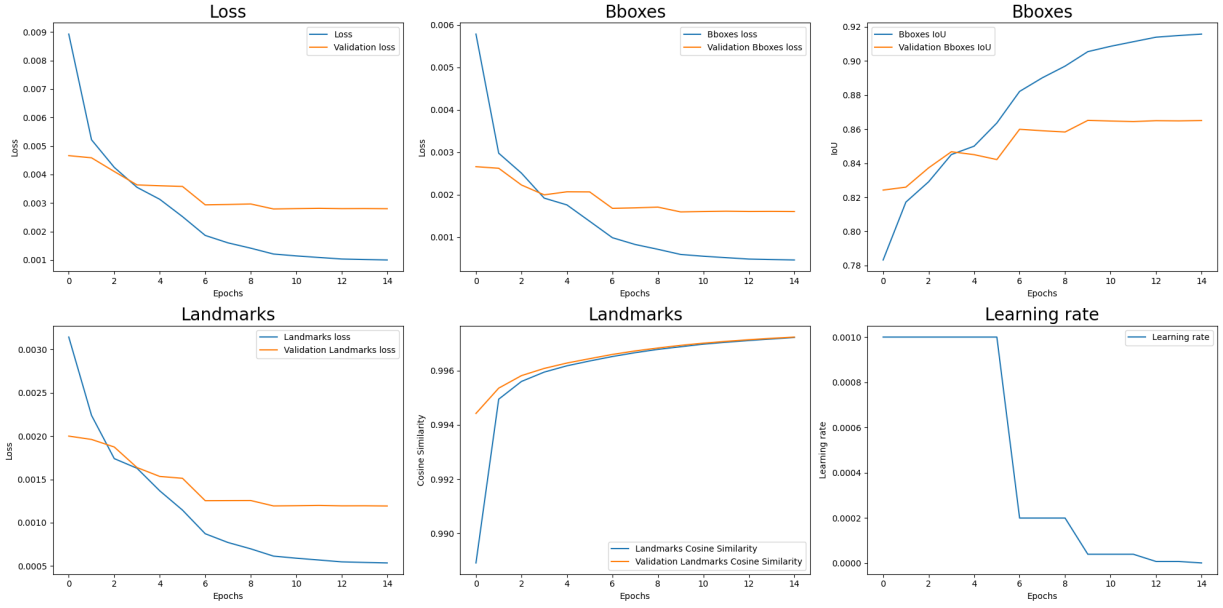


Figure 7: Custom MediaPipe training

Seeing this result 7, it appears to perform quite well. However, by adding the classifier, we obtain very poor results (see table 2 below):

Accuracy	Precision	Recall	F1-Score
4%	5.704%	1.722%	2.645%

Table 2: Custom MediaPipe Model metrics on test data.

To understand the contradiction, we plotted the predicted landmarks on several images (see Fig. 8).

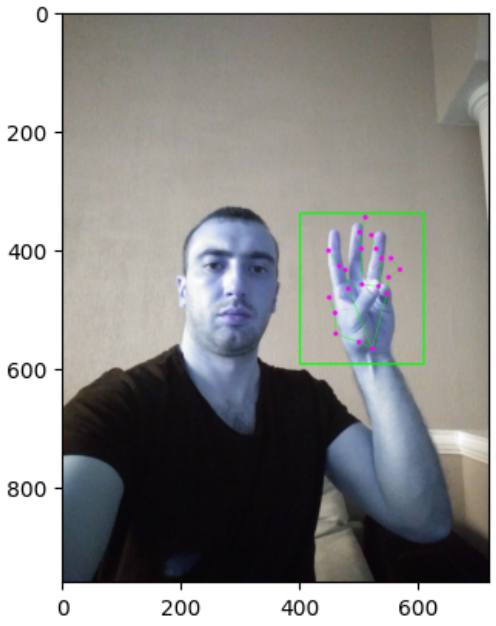


Figure 8: Plotting landmarks example

Based on the observations in 8, it can be seen that the predicted landmarks are not accurate.

It is worth noticing that the classifier has been trained on "perfect" landmarks, therefore when taking the predicted landmarks (imperfect) from the model, the classification is almost random. The prediction of the box, however, is respectable.

We could achieve slightly better performances by fine-tuning the classifier with this model. Nonetheless, given the predicted landmarks, the overall accuracy would not be great. This is the reason why we implemented the "all-in-one" model afterward.

4.3 Custom all-in-one model (Box and Class)

The custom "all-in-one" model underwent training with a dataset of over 32,000 images. This model utilizes MobileNet pre-trained network as its base model, combined with the model described 5.

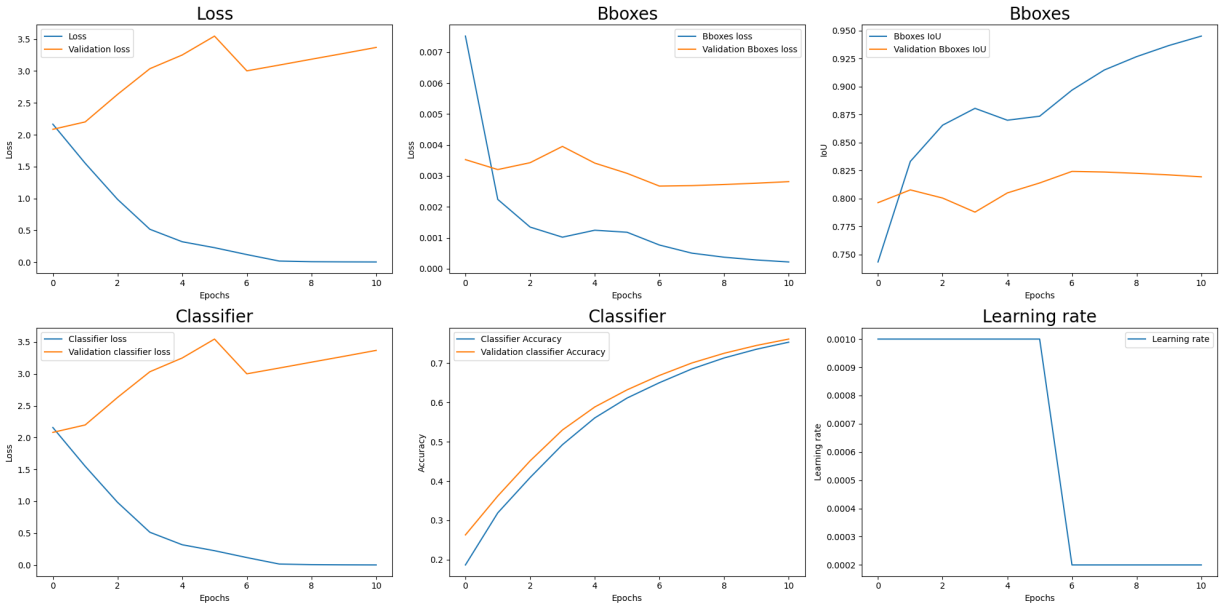


Figure 9: Model training

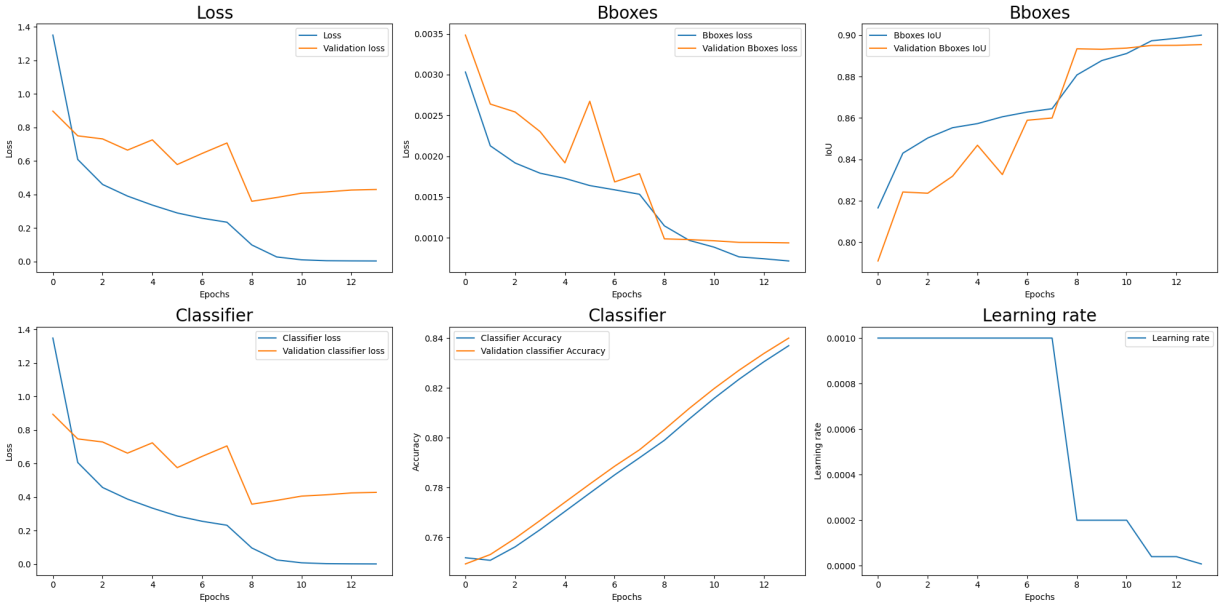


Figure 10: Model fine-tuning

Accuracy	Precision	Recall	F1-Score
86.21%	87.368%	84.984%	86.16%

Table 3: Model metrics on test data

As indicated in the table above 3, the performance metrics are satisfactory. Nonetheless, a comparison between the outcomes obtained with the "all-in-one" model 3 and the classifier

1 reveals that it is comparatively simpler to learn efficiently from landmark coordinates as opposed to images.

5 Problems encountered and choices of conception

5.1 Model architecture and number of parameters

Our aim was to generate predictions in real-time from a live video stream, thus it was essential to employ models that could operate at a speed of 20-30 frames per second. For this reason, we excluded VGG16 as a base model and instead opted for MobileNet.

5.2 Dataset size

The straightforward classifier was trained utilizing landmarks, which are comprised of 21 (x, y) coordinates. This enabled us to utilize the complete dataset annotations, encompassing more than 490,000 annotations, for training purposes. In contrast, the "all-in-one" custom model had to be trained on images, making it unfeasible to utilize the entire dataset due to the substantial computing power requirements. To address this predicament, we resorted to sampling the data and utilized roughly 40,000 images for training the model.

This choice of conception surely affects the overall performance of the model.

5.3 MobileNet input shape

The input dimensions of the pre-trained MobileNet network are $(224, 224, 3)$. However, since the dataset images vary in size, ranging from greater than 1920×1080 to less than 480×270, some preprocessing was necessary before training and making predictions. To tackle this challenge, we utilized a downsampling and resizing approach that corresponded to the original resolution of the images, as illustrated below 11.

Figure 11: Image preprocessing based on original resolution.

References

[1] G. Sung, K. Sokal, E. Uboweja, V. Bazarevsky, J. Baccash, E.G. Bazavan, C.-L. Chang, M. Grundmann, "On-device Real-time Hand Gesture Recognition", 2021, 2111.00038, arXiv, cs.CV, doi: 10.48550/arXiv.2111.00038.

[2] Y. Farhan and A. Ait Madi, "Real-time Dynamic Sign Recognition using MediaPipe", 2022 *IEEE 3rd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*, Fez, Morocco, 2022, pp. 1-7, doi: 10.1109/ICECOCS55148.2022.9982822.

[3] A. Kapitanov, A. Makhlyarchuk, K. Kvanchiani, "HaGRID - HAnd Gesture Recognition Image Dataset", 2022, 2206.08219, arXiv, cs.CV, doi: 10.48550/arXiv.2206.08219.

[4] Murtaza's Workshop - Robotics and AI, "Hand Tracking 30 FPS using CPU — OpenCV Python (2021) — Computer Vision", 2021, YouTube video, Link here.