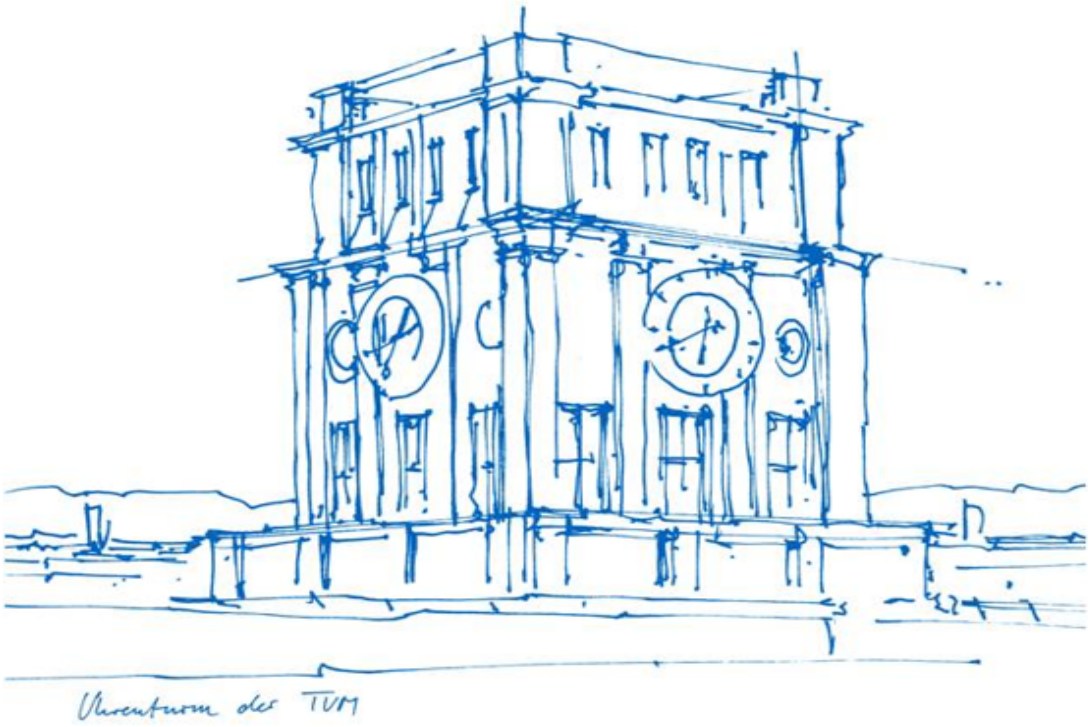
Advanced Topics in Communication Electronics

Final Project Report

Face Mask Detector



Approved by:

Abhishek Sengupta, 03736060, ge79car.sengupta@tum.de
Priyadharshini Ponraj, 03741605, priya.ponraj@tum.de
Smriti Nandy, 03737798, ge59wof@mytum.de
Vinay Shankar Kulkarni, 03738026, vinay.kulkarni@tum.de

on 04.02.2022

Abhishek Sengupta

Priyadharshini Ponraj

Smriti Nandy

Vinay Shankar Kulkarni

Abstract

COVID-19 is causing a health crisis worldwide. In this current pandemic situation, wearing a face mask is very important as it is one of the necessary safety measures to prevent infections. To ensure the use of face masks by the people in crowded places or larger gatherings, a face mask detection system can be designed. This would help reduce the possibility of spreading the virus. With the use of deep learning detection algorithms and embedded systems, the algorithm can be incorporated in the security systems of a room or any enclosed space. This system would open the gates only if masks are worn, to ensure the safety of all. Using an embedded system would reduce the cost of manufacturing face mask detection systems. A use case for the system would be a classroom, where the doors will unlock only if the incoming person wears a mask, otherwise it will notify the person to wear a mask.

Table of Contents

1. Objectives and Expected Significance

1.1 Objective of the project

We aim to design a deep learning model that will help us detect if a person is wearing a mask or not. The model is implemented on an edge device, Raspberry pi 4, which is integrated with an USB Camera and the buzzer. The model detects if a person in front of the camera is wearing a mask or not. In case the person is not wearing a mask, the buzzer will start ringing. The model concentrates on TinyML, a type of machine learning that shrinks deep learning networks to fit on tiny hardware with resource constraints. Edge devices have small CPUs, limited to a few hundred kilobytes of low-power memory (SRAM) and a few megabytes of storage. Considering these constraints, the objective is to run the model and optimize memory bottlenecks in convolutional neural networks. The project uses a popular TinyML model, MobileNetV2 architecture, in which the early layer blocks have a memory peak that reaches around 1.4 megabytes, while the later layers have a very small memory footprint. Hence, this implementation can be integrated into a bigger project like a security system of a room or a closed space.

1.2 Expected Significance of the project

The developed deep learning model can be used in security systems in closed spaces like a classroom or an auditorium. While we continue to deal with a never ending pandemic and new variants of the virus, wearing a facemask is the need of the hour to protect oneself and others. This project will help in detecting if a person is wearing a facemask or not and in case there is no facemask, the buzzer starts to ring, alarming everyone around him or her. We have implemented the project already on a Raspberry pi 4 to deal with some of the constraints which it might face while being deployed into real world applications. We sincerely hope this project helps to curb the spread of the virus and motivates everyone to wear a mask whenever they travel out of their homes.

2. Rationale and Related Background

2.1 Introduction

In recent times, there has been a tremendous rise of analyzing, processing and managing data for prediction-based models. Deep learning is a way to learn from the large amount of unstructured and unlabeled data, to make classifications and accurate predictions. A very commonly used deep learning architecture is the Convolutional Neural Network (CNN). They show high representation power and superior performance in both image and speech recognition tasks. CNNs have emerged as the dominant model across various Artificial Intelligence (AI) applications. In the era of IoT and mobile systems, the efficient deployment of CNNs on embedded systems is vital to enable the development of intelligent applications. Traditionally, CNNs use full precision weights and floating-point value operations for better accuracy. Over the past years, CNNs have become more feasible for usage in embedded and mobile devices. Tools like Tensorflow Lite have further accelerated opportunities for the application of deep learning in embedded devices.

2.2 Related work

Recent research on deep neural networks has focused primarily on improving accuracy. For a given accuracy level, it is typically possible to identify multiple Deep Neural Networks (DNN) architectures that achieve that accuracy level. With equivalent accuracy, smaller DNN architectures offer at least three advantages: (1) Smaller DNNs require less communication across servers during distributed training. (2) Smaller DNNs require less bandwidth to export a new model from the cloud to an autonomous car. (3) Smaller DNNs are more feasible to deploy on FPGAs and other hardware with limited memory.

MobileNets is one of the first initiatives to build CNN architecture that can easily be deployed in mobile applications. One of the main innovations is depthwise separable convolutions. A separable convolution separates a normal convolution kernel into two kernels. This separation reduces the number of operations needed to perform the convolution and is therefore much more efficient. However, it is not always possible to separate on the spatial dimension, so it is more common to separate on the depth (channel) dimension. This depthwise separable convolution is used in MobileNet. The model became an easy solution for mobile object detection, face detection, and image classification applications.

The MobileNetV2 introduced inverted residuals and linear bottlenecks to improve the performance of MobileNets. Inverted residuals allow the network to calculate (ReLU) activations more efficiently and preserve more information after activation. To preserve this information, it becomes important that the last activation in the bottleneck has a linear activation. Furthermore, the ReLU6 activation is introduced here to speed up the activations for low-precision calculations and to make them more suitable for quantization. The ReLU6 activation is defined as: $ReLU6(x)=min(max(x,0),6)$.

3. Approach

3.1 Components

- **Raspberry Pi 4 Model B (4GB RAM)**
  The Raspberry Pi 4 Model B [7] with 4GB RAM is being used with a key specification of 1.5GHz quad-core Broadcom BCM2711 CPU. The model is implemented on 4GB Raspberry Pi 4.

- **Raspberry Pi OS (Debian-based)**
  The latest Debian version 11 (Bullseye) is installed for the Raspberry Pi 4 hardware.

- **USB Camera**
  The specification of the USB camera is as follows: 720P HD camera with a 120° wide-angle for real-time image and video transmission.

- **Buzzer**
  KY-012 active piezo buzzer alarm sensor module is used. The buzzer works at a supply voltage of 5V and the buzzer rings with a frequency of 2.5 kHz.

3.2 Model

Face mask detection implies a mechanism in which it identifies if a person is wearing the mask or not. A deep learning binary classification model is implemented using the **MobileNetV2** [1] network. The overall model is divided into two parts- the base Model and the head Model as shown in Figure 1. The base Model consists of MobileNet V2 CNN architecture and takes raw data as input, in the form of a numpy array. It delivers high accuracy results while keeping the parameters and mathematical operations as low as possible.. The output of baseModel is fed to the headModel. Both the base model and the head model will be described in detail in the subsequent sections.

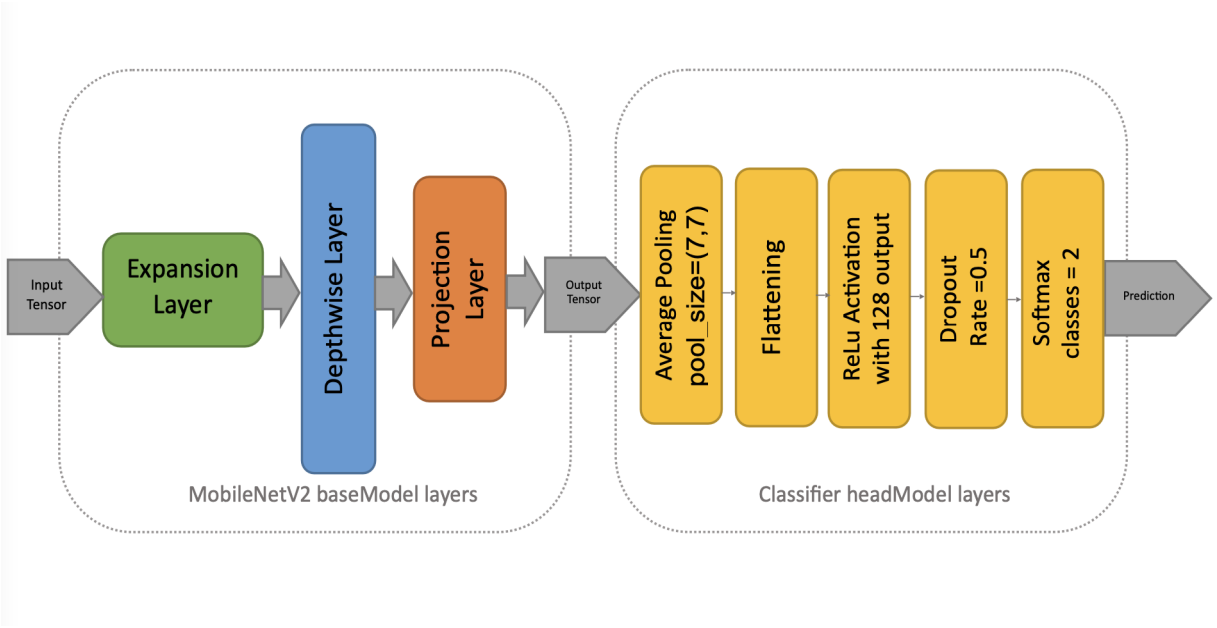


Figure 1. Block diagram of the layers of the Model

Model description:

The model used in this project contains the MobileNetV2 base model which is appended with Classifier layers to perform the classification task of “Mask” or “No Mask”.

The MobileNetV2 base model consists of 17 blocks which consists of three layers each which are described below, except the first block which consists of a 3×3 convolution with 32 channels instead of the expansion layer.

Each block of MobileNetV2 base model consists of the following layers [9]:

- **Expansion Layer**: This is the first layer in the MobileNetV2 base model. This is also a 1×1 convolution layer. It expands the number of dimensions (channels) in the data before it goes into the depthwise convolutional layer. Hence, this expansion layer always has more output dimensions (channels) than input dimensions (channels).

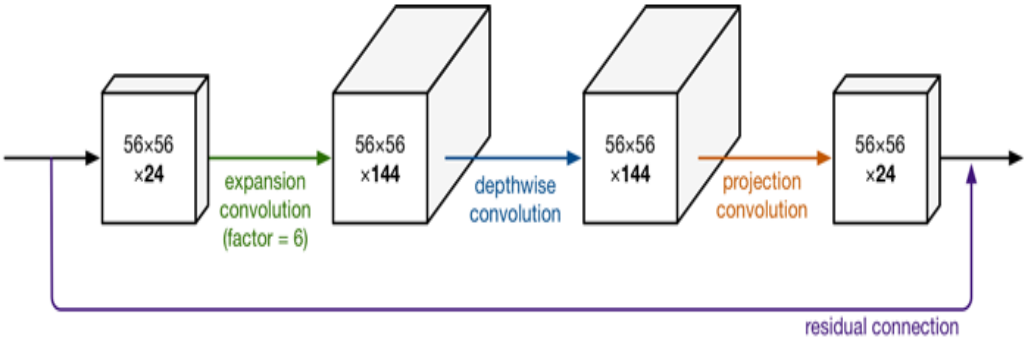


Figure 2. Block diagram of the layers of the MobileNetV2 model [9]

The data gets expanded or uncompressed by the expansion factor. The default expansion factor is 6. For instance, if the tensor going into the MobileNetV2 block (as shown in Figure.2) has 24 dimensions (channels), the expansion layer converts the input tensor into a new tensor with 24 * 6 = 144 dimensions (channels).

- **Depthwise Layer**
  The depthwise layer performs the depthwise convolution and the dimensions of the input tensor into this layer and the output tensor from this layer remains the same. It is a 3x3 convolution layer. As shown in Figure.2, if the tensor going into the depthwise convolution layer has 144 dimensions (channels), the output tensor from the depthwise convolution layer has 144 dimensions (channels).

- **Projection Layer**
  The projection layer is 1x1 convolution layer. It projects the data with a high number of dimensions (channels) into a tensor with a much lower number of dimensions (channels). The depthwise convolution layer performs convolution on a tensor with 144 channels and then the output tensor from the depthwise layer is shrinked to a tensor having 24 channels by the projection layer. The projection layer can also be termed as a bottleneck layer because it reduces the amount of data that flows through the network.

  Each of these layers in a block are followed by batch normalization and activation function. The activation function used by MobileNet is ReLU6, which prevents activations from becoming too big. ReLU6 is more robust than regular ReLU when using low-precision computation.

MobileNetV2 is used as the base model in this project because:

- The tensor is decompressed at the input of the depthwise convolution layer so that the model can learn the features well.
- The Projection layer compresses the tensor and reduces the dimensions of the tensor which will be the input to the Classifier layers, hence the number of future computations is reduced.
- MobilenetV2 gives better prediction accuracy and lower training time compared to classical Machine Learning models.

The output of baseModel is fed to the Classifier headModel consisting of following layers:

- **Average Pooling layer with pool size 7x7**
  Average pooling calculates the average for each patch of the feature map, to reduce the dimension size. This means that each 7×7 square of the feature map is down sampled to the average value in the square.

- **Flattening Layer**
  The Flattening Layer converts the data into a 1-dimensional array for inputting it to the next layer. The output of the convolutional layers are flattened to create a single feature vector.

- **ReLu activation layer with 128 dimensions of output space**
  The activation function is a simple calculation that returns the value provided as input directly, or the value 0 if input is 0 or less. Due to the computational simplicity, representational sparsity and linear behavior, it is used with most convolutional neural networks [2].

- **Dropout layer with rate=0.5**
  Dropout will set input units to 0 with a frequency of 0.5 for each step during the train time, which helps prevent overfitting.

- **Softmax layer with 2 classes**
  Softmax layer with 2 classes are used since we have only two labels, “with_mask” and “without_mask”: Using the softmax activation function, the output probabilities are interrelated and the probability sum is always one [3].

Fine-tuning process is applied to the best trained Classifier headModel. The fine-tuning process significantly decreases the time required for programming and processing a new deep learning algorithm as it already contains vital information from a pre-existing deep learning algorithm.

The output of the above is further compiled using ADAM optimizer and Binary Cross Entropy loss function for a batch size 80 and 10 epochs. Fine-tuning of the headModel with 2 epochs with a fine-tuning percentage of 65 with a learning rate of 0.000045.

Parameters:

The parameters used for training the model are described below:

- **Dataset**
  The dataset considered for the training is from the open source links [4]. The dataset consists of images of classes “with_mask” and “without_mask”. This dataset is used to train the model for the face detection system. The dataset is split with a ratio 4:1 for the training and validation data.

- **Label**
  One-hot encoding is the process of converting categorical data variables into a new categorical column and assigning a binary value of 1 or 0 to these columns. For deep learning algorithms, this improves the prediction accuracy and classification accuracy of the model. In this model, there are two labels: “with_mask” and “without_mask”.

- **Optimizer**
  During the back propagation, stochastic gradient descent (SGD) is a way to minimize the objective function $J(\theta)$, and updates the weights and determines the loss or error. The learning rate η determines the size of the steps we take to reach a minimum. Since convergence and learning rate parameters are a challenge in the conventional gradient descent method, we use adaptive moment estimation (ADAM) that computes adaptive learning rates for each parameter.

- **Loss function**
  In calculating the error of the model during the optimization process, a loss function is used. A loss function evaluates the objective function (i.e., set of weights). In a neural network, the error function is the minimization of the objective function. Here “Binary Crossentropy” is used. [5].

- Parameters for the different layers of the training model shown in Figure 1 :
  - Learning rate = 0.00045
  - No. of epochs= 10
  - Batch size= 80
  - For MobileNetV2 : apha = 0.35
  - Weights: Default MobileNetV2 weights
  - Fine-tuning process: 2 epoch, 65% fine-tuning

Libraries used to deploy the model:

- opencv-python
- numpy
- keras
- math
- tensorflow/tensorflowLite
- edgeimpulserunner

3.3 Methods

Method 1: Using Tensorflow

The model is not being trained on the Raspberry Pi, but instead we used the PC trained model file for the testing phase. Initially, the testing script used on the PC was based on TensorFlow. Therefore, the first step was to install the TensorFlow package on the Raspberry Pi. During the installation of the package, there was a version mismatch of the dependency libraries and installation was quite tedious and unsuccessful.

Method 2: Using TensorflowLite

In order to make installation of the package easier, we have converted the Tensorflow functions to TensorFlow Lite functions for the testing script used on the Raspberry Pi. While exploring other alternatives for the model file using Tensorflow package, we have come across EdgeImpulse.

Method 3: Using EdgeImpulse

Edge impulse[8] was used to design the MobileNetV2 architecture, shown in Figure.1. The model file generated after training is of the format .eim and is completely self-contained with the libraries and dependencies like tensorflow and hence importing them can be avoided. Currently, the unoptimized float 32 model version is deployed on the Pi to achieve better accuracy. For the future work, a quantized int8 version of the model can be implemented that gives low accuracy and higher system performance.

3.4 Model Training

The training sequence can be specified as follows:

- MobileNetV2 is chosen as the base model with alpha= 0.35. Alpha Controls the width of the network. This is known as the width multiplier in the MobileNet paper.
    - If alpha < 1, proportionally decreases the number of filters in each layer.
    - If alpha > 1, proportionally increases the number of filters in each layer.
    - If alpha = 1, the default number of filters from the paper are used at each layer.

  Default is set to 1.0. [10]

```
WEIGHTS_PATH='./transfer-learning-weights/keras/
mobilenet_v2_weights_tf_dim_ordering_tf_kernels_0.35_96.h5'
base_model = tf.keras.applications.MobileNetV2(
    input_shape = INPUT_SHAPE, alpha=0.35,
    weights = WEIGHTS_PATH
)
```

- The layers of the classifier head model as specified in Figure 1 are added to the output of the base model.

```
model.add(AveragePooling2D(pool_size=(7,7)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(classes, activation='softmax'))
```

- The model is compiled with the parameters as specified in Section 3.2.

```
INIT_LR = 0.00045
EPOCHS = 10
BATCH_SIZE = 80

opt = tf.keras.optimizers.Adam(learning_rate=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(optimizer=opt,
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Once the model is trained, it is tested on the validation dataset which outputs the following plot:

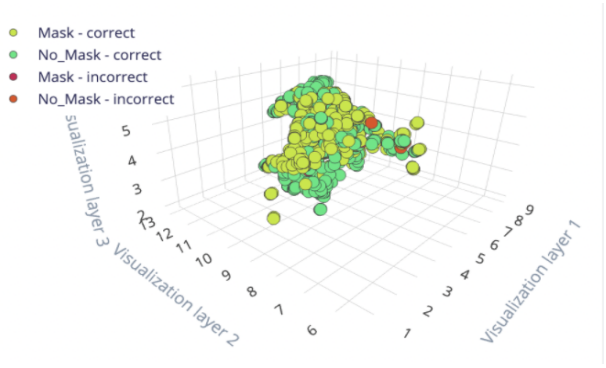


Figure 3. Classification by the model

Figure 3 shows all the data in the training set classified by the model. Items in green are classified correctly, whereas the items in red are misclassified.

3.5 Model Testing

The model has been connected as shown in the Figure.4 for testing the pre-trained model file on the RaspberryPi 4. The Raspberry Pi 4 board has a webcam, connected via the USB port. The piezo buzzer is connected to the GPIO BCM pin 17. The power supply of the Raspberry Pi 4 is given via the laptop.

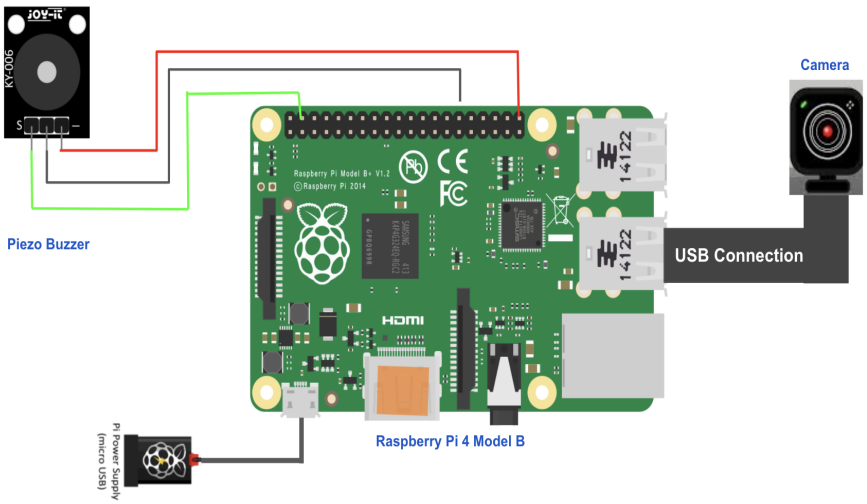


Figure 4. Hardware block

The flow of the testing script is briefly highlighted in Figure.5 flowchart. The camera and the buzzer is initialized, followed by the import of the pre-trained model file. Further, for every frame of the video, the status of the camera is being checked. If the status of the camera is active and the frame label is classified as “Mask_detected”, then the label is being printed. On the other hand, if the frame label is classified as “Mask_not_detected” and the probability of the frame prediction is greater than 0.85, then the buzzer is activated or turned ON.

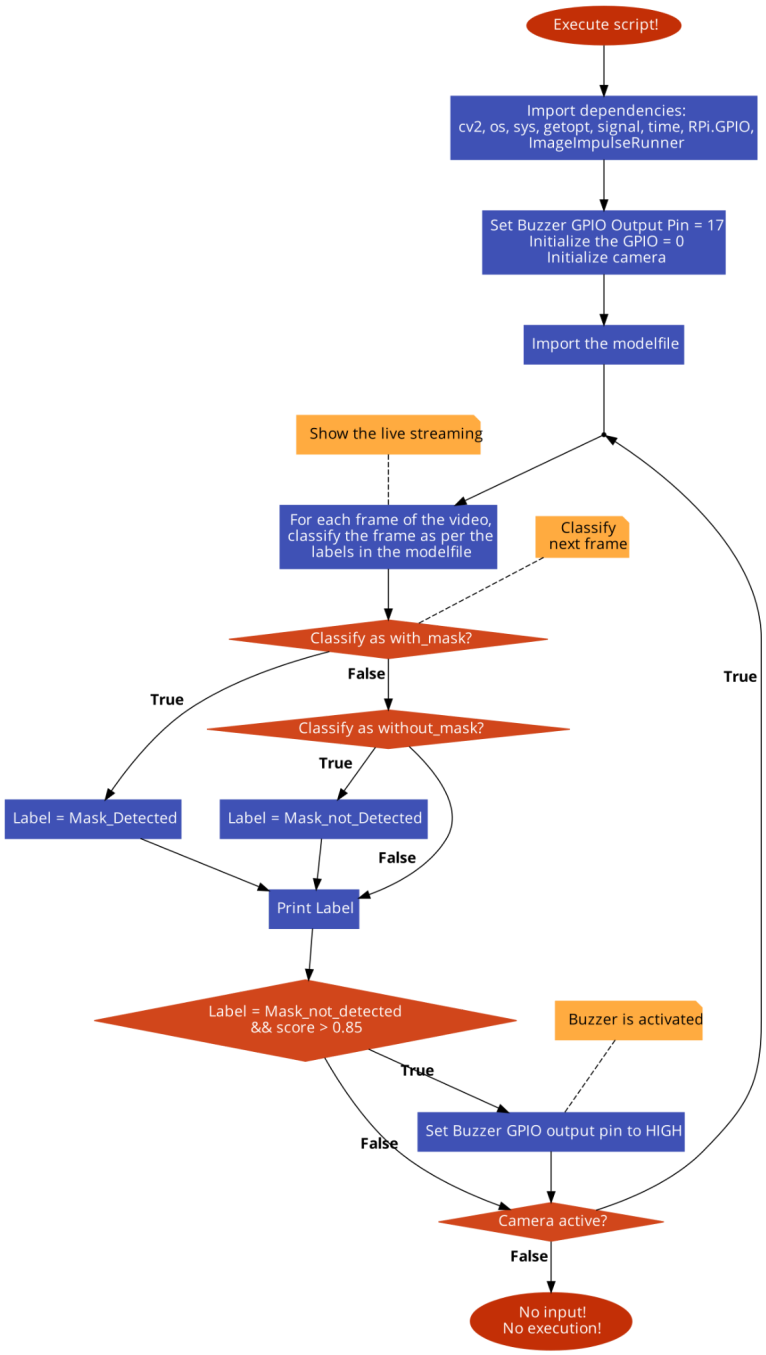


Figure 5. Flowchart of the testing script for the model

4. Project Outcome

4.1 Working of the implemented model

This section will explain about the hardware architecture of our system. The hardware setup can be seen in Figure. 6. It consists of a USB webcam, raspberry pi, and a buzzer. For the real-time application, the system detects a face without the face mask and triggers the buzzer to alert him/her to wear the face mask at a time interval of 0.5 secs. This assists in detecting the presence of a person without a facemask in a closed space like a classroom or an auditorium.

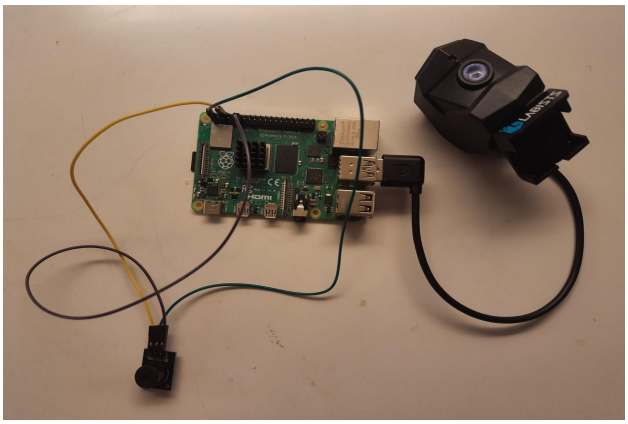


Figure 6: Hardware Setup

As depicted in Figure 7, The algorithm detects the face mask and classifies the label of the frame in the command prompt (through its identifiers) and a buzzer sound is triggered when the concerned person is not wearing a mask, until he/she wears one.

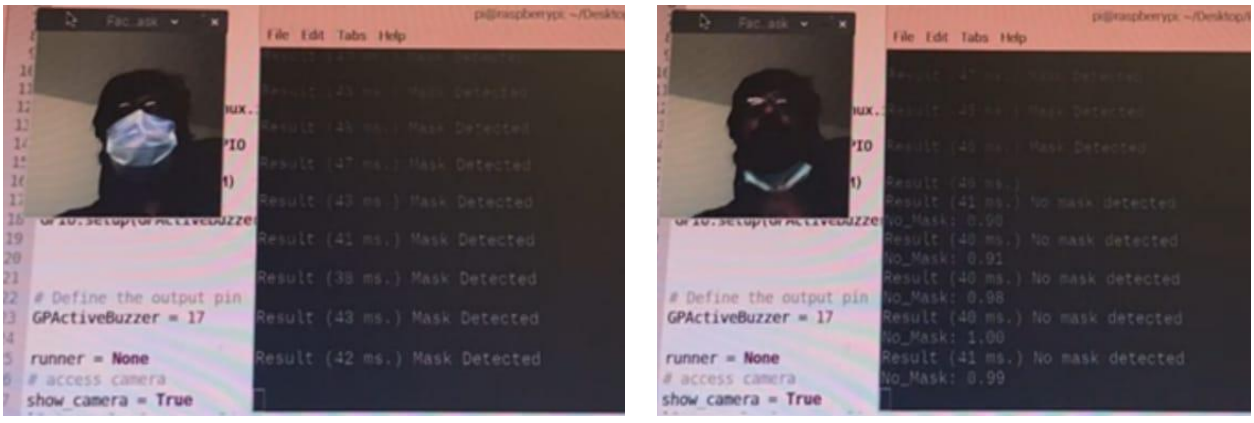


Figure7: Identification of mask

4.2 Performance Evaluation

- Table.1 shows the predictions of the model for the “Mask” and “No_Mask” labels and their corresponding F1 scores.
- We observe a high prediction accuracy of 98% with “Mask” dataset and 94.9% with “No_Mask” dataset with the unoptimized float32 model.
- **Precision** quantifies the number of positive class predictions that actually belong to the positive class.
- **Recall** quantifies the number of positive class predictions made out of all positive examples in the dataset.
- The **F1-score** combines the precision and recall of a classifier into a single metric by taking their harmonic mean.
- We observe high F1 scores of 0.96 and 0.97 with “Mask” and “No_Mask” respectively.
- The accuracy observed is 96.3% with 0.12 loss factor on the validation set.

Data Classification	**Mask (Predicted label)**	**No_Mask (Predicted label)**
Mask (Actual label)	98%	2%
No_Mask(Actual label)	5.1%	94.9%
F1 score	0.96	0.97

Table.1 : Accuracy of the float32 model

4.3 Memory Consumption

- Resource utilization of the Raspberry Pi is given in Figure 8. 26 MB of memory is used by the model file and the entire python script consumes 56 MB of memory.
- The total memory consumed by Raspberry Pi at run-time is approximately 317 MB.

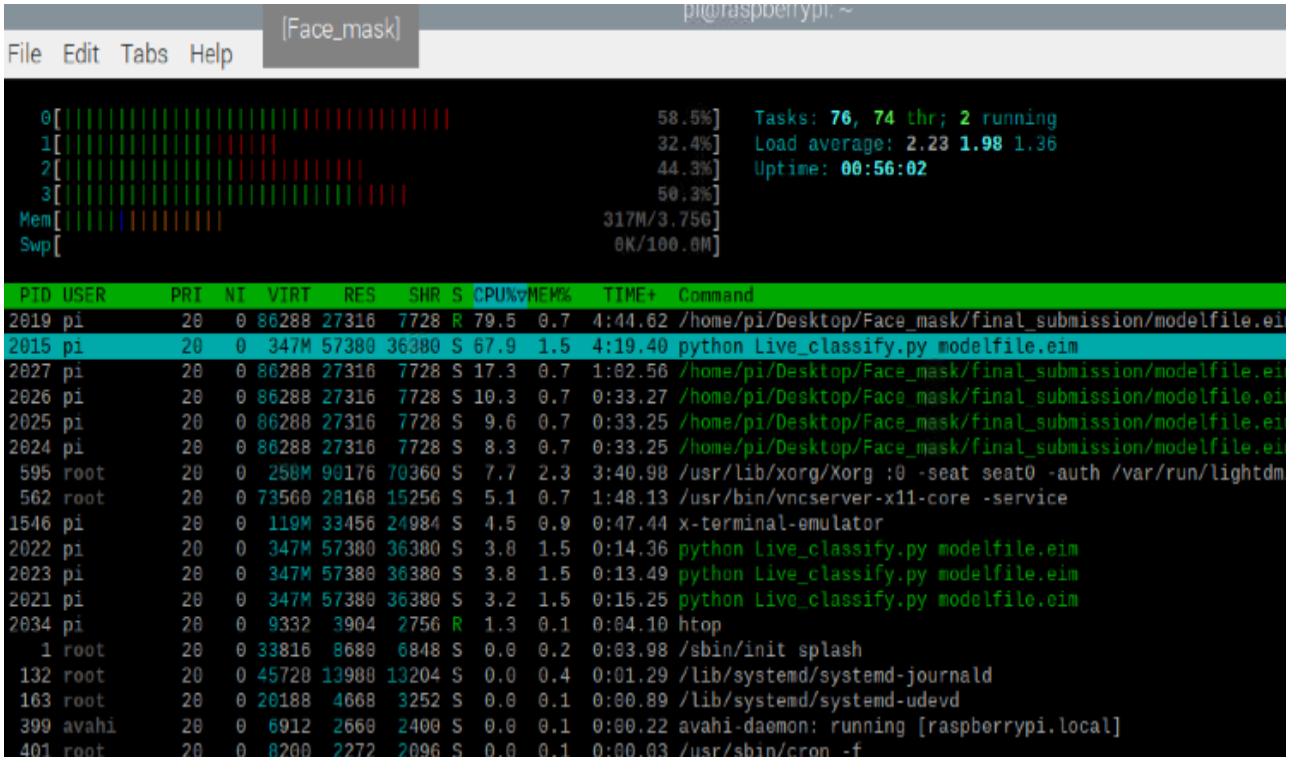


Figure 8: Resource utilization of PI

4.4 Conclusion

- COVID-19 Face Detection Mask model is achieved using the MobileNetV2 CNN architecture.
- Model is trained with ~4000 images of classes with_mask and without_mask.
- The face mask detector model is accurate, and since we used the MobileNetV2 architecture, it's also computationally efficient, making it easier to deploy the model to embedded systems.
- The implemented unoptimized float32 model has an overall accuracy of 96.3% and 56MB memory consumption for the testing script.
- For future work, the prediction accuracy has to be improved with the quantized model. The face mask detection corner use cases require a better dataset. Finally the model can be implemented using a screen for real-time application.

Bibliography

[1] MobileNetV2 : https://towardsdatascience.com/
[2] ReLu activation: https://machinelearningmastery.com/
[3] Softmax activation: https://medium.com/
[4] Dataset: https://www.kaggle.com/
[5] Binary Crossentropy:
https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy
[6] Source Code: www.github.com/
[7] RaspberryPi 4: https://www.raspberrypi.com/raspberry-pi-4-model-b/
[8] EdgeImpulse: https://www.edgeimpulse.com/
[9] MobileNetV2 : https://machinethink.net/blog/mobilenet-v2/
[10] MobileNetV2 parameters: https://keras.io/api/applications/mobilenet/

Appendix

Link for Implemented Model and Testing script on Gitlab:
https://gitlab.lrz.de/FaceMaskDetection