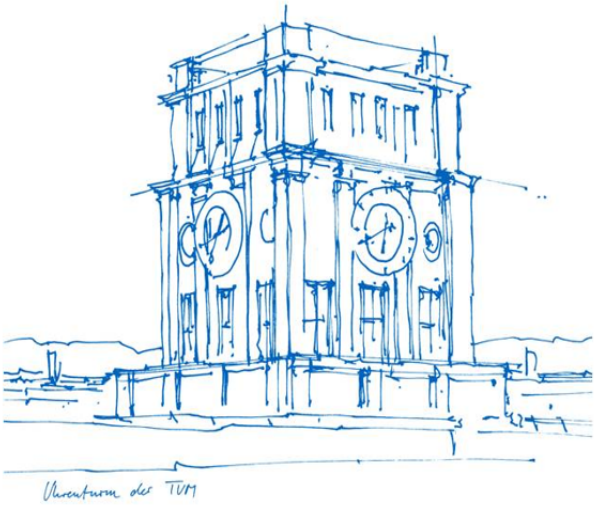
Face Mask Detector

Advanced Topics in Communication Electronics



Prepared 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

DATE: 17.12.2021

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

Model

Face mask detection implies a mechanism in which it identifies if a person is wearing the mask or not. This is done in two stages.

1. Detection of face
2. Identification of mask

A deep learning binary classification model is implemented using the **MobileNetV2**[1] network. The overall model is divided into two: The baseModel and The headModel. The baseModel 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. MobileNetV2 uses depth wise separable convolutions to build lightweight deep neural networks that can have low latency. The output of baseModel is fed to the headModel consisting of following layers:

- Average Pooling layer: Average pooling retains a lot of features from the image and gives a smooth image when compared to Max pooling.
- Flatten Layer: By flattening, the data is converted into a 1D array for the next layer.
- 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 0.5 rate: Dropout will set input units to 0 with a frequency of 0.5 rate for each step during the train time, which helps prevent overfitting.
- Softmax layer with 2 dimensions as 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].

The output of the above is further compiled using ADAM optimizer and Binary Cross Entropy loss function for a batch size of 45 and 30 epochs. Figure 1 gives the overall neural network structure of the Model being implemented.

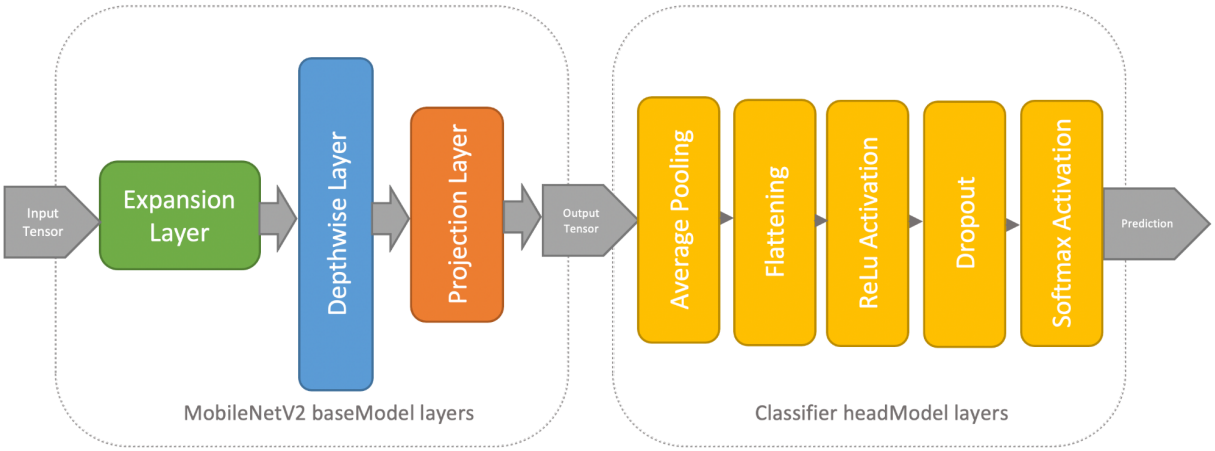


Figure 1. Block diagram of the layers of the Model

Parameters

The parameters that have been used in the script is described below:

- Dataset
  The dataset considered for the training is from the open source links available online[4]. The dataset consists of images classes with_mask and images 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 assign a binary value of 1 or 0 to these columns. For deep learning algorithms this improves the prediction 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 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. Binary cross entropy is the loss function used for binary classification models[5].

Result

- To check the performance of the implemented model, the accuracy and loss have been plotted for the training and validation phase as shown in Figure 2.
- An increase in the training accuracy and decrease in the training loss is observed after the second epoch.
- The accuracy line is stable after epoch 28, therefore the iterations to improve accuracy can be stopped at epoch 30.
- We observe that there is ~98% of training accuracy, and very small training loss.
- The validation accuracy is high than the training accuracy and the validation loss is lower than the training loss

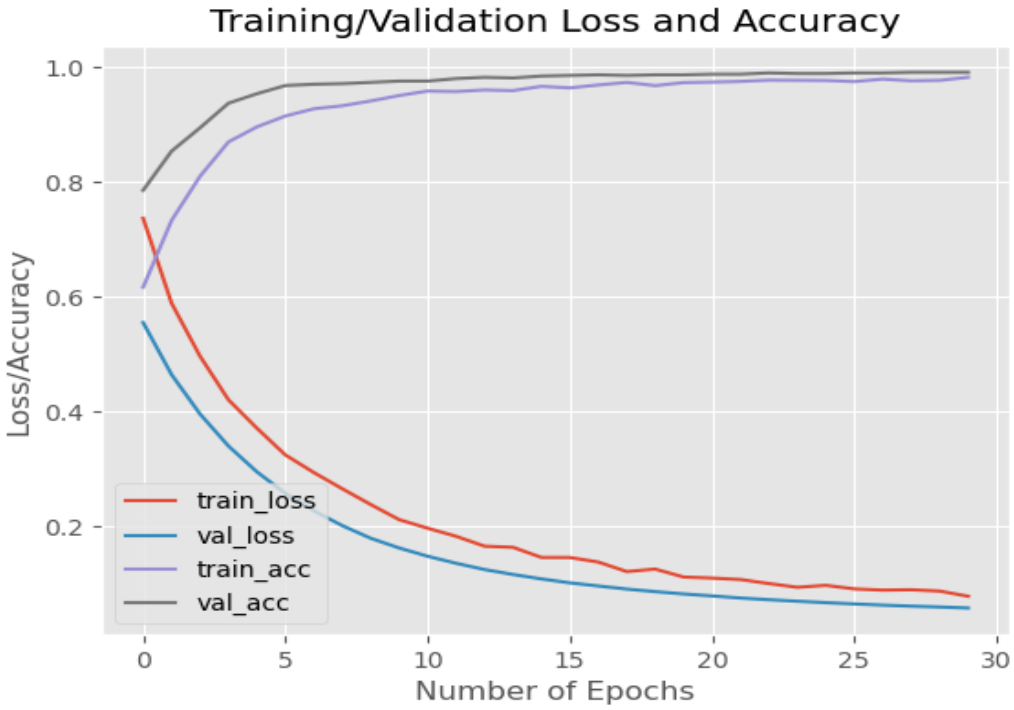


Figure 2. Training and Validation Loss/Accuracy Plot

Summary

- Model has been trained using the MobileNetV2 architecture on the laptop/PC, using a dataset of ~4000 images of classes with_mask and without_mask.
- We observe the plot of Figure 2, the Training accuracy is ~98% and the loss is very low. The training time for the model is approximately 45mins.
- The model has been tested using static images and live stream using the webcam on the laptop/PC.

Future work

- Compress the designed model to run on the embedded system.
- Implement the model on the RaspberryPi 4 with an integrated camera module for live stream inference.
- For the classroom use case, Integrate a buzzer or LED circuit with the Raspberry Pi4. This will be used to notify if the mask is worn or not.

Detection for sample images

To apply the model to images, we have to load the images from the disk and detect the face in the images using the RaspberryPi. Then apply the face mask detector model to classify the images as with_mask or without_mask. Figure 3 shows the expected raw data (left image) and the classified data (right image) for two sample static images.

Figure 3. Raw data (Left images) and classified data (Right images) of sample images for labels

Detection for live video

To apply the model to the live video, the webcam of the RaspberryPi has to be accessed and the face will be detected from the frames captured from the live stream. The face mask detector model will be applied on the frames captured from the live stream. Figure 4 shows the expected frame captured from the live stream of classes without_mask and with_mask using the camera of the laptop.

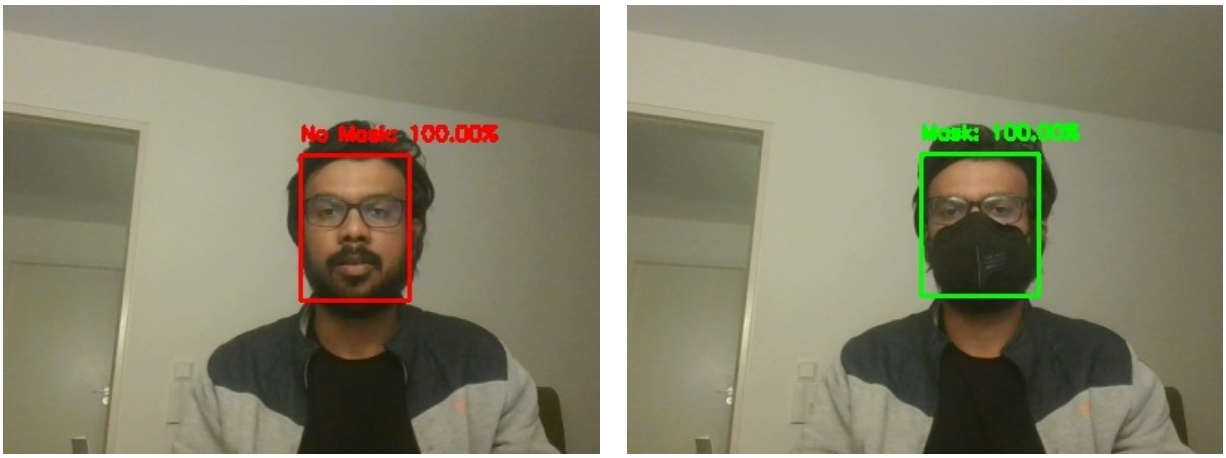


Figure 4. Classified data of without_mask (Left screenshot) and with_mask (Right screenshot) of live stream using laptop webcam

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 Cross Entropy: https://www.analyticsvidhya.com/
[6] Source Code: www.github.com/

Appendix

Link for Implemented Model on Gitlab: https://gitlab.lrz.de/