Abhishek Shah

Biometrics

Dr. Yong Zhang

Project Report

4/26/2021

This project simply detects if a face has a mask on in real-time and displays with an accuracy rate using TensorFlow with Keras library, OpenCV, and MobileNet on python. The face mask detector model trains from a dataset in the two folders i.e., “with_mask” with 1915 pictures and “without_mask” with 1918 pictures, downloaded from open source platforms and Kaggle, implementing with Keras and OpenCV.

73 weeks and 2 days ago from the due date of this project, the first case of an infectious disease called CORONA, or COVID-19 was found and which in days turned into a global pandemic. A pandemic that is so bad that it has killed 3.01 million people worldwide and still counting with 141 million ongoing cases. Not too long after WHO suggested implementing immediate lockdown worldwide. This, as result, hampered everyone from a single-family to an entire country financially and economically. The countries could not regulate the lockdown for an entire year. Soon the lockdown was lifted with so many restrictions. However, to be able to regulate the restrictions could be tough when people do not comply with them. Doctors and front-line workers were and are giving their best to keep the situation in control. It is the responsibility of every citizen to help control the situation in whatever ways they can. However, how could a computer programmer contribute to being a responsible citizen sitting in a room?

Just like the demand and supply rule, this project can be embedded with Raspberry Pi or can be integrated with CCTV cameras with few improvements as it uses MobileNetV2 architecture, it

makes it computationally efficient and easier to implement in such systems. It can be used in public places such as airports, malls, parks, and more where a human cannot monitor people wearing a mask or not.

To run this project, we need to install some packages first, that will assist us throughout the project from image processing and training to implementing it through a webcam in real-time. The packages required are:

TensorFlow: TensorFlow is a machine learning software library that is free and open-source. It can be used for a variety of tasks, but it focuses on deep neural network training and inference.

Keras: Keras is an open-source software library for artificial neural networks that offers a Python interface. Keras functions as a user interface for TensorFlow.

Imutils: Imutils is a collection of convenience functions for OpenCV and Python 2.7 and Python 3 that make simple image processing functions like translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier.

NumPy: NumPy is a python library that adds support for huge, multi-dimensional arrays and matrices, as well as a large number of high-level mathematical functions to work on these arrays.

OpenCV: OpenCV is a cross-platform library that can be used to build real-time computer vision apps. It focuses primarily on image processing, video recording, and analysis, with features such as face and object detection.

Matplotlib: Matplotlib is a data visualization and graphical plotting library for Python and its numerical extension NumPy that runs on all platforms.

SciPy: SciPy is a python library for solving science and mathematical problems that is open-source. It's based on the NumPy extension and lets you manipulate and visualize data using a variety of high-level commands.

Deep Learning models work best with arrays. One of our initial steps would be to convert all our datasets into arrays. Using the two categories in our datasets, with mask and, without mask, we now convert all our images into arrays. And with the help of those arrays, we will train our deep learning model. We create two empty lists, the data list that contains the images and the labels list that categorizes them as with mask or without mask. We append all our image arrays with the help of Keras image preprocessing and looping it through our categories. This helps us get the numerical values for the data array but our label array still contains alphabets, i.e., with mask and without mask. To convert with mask and without mask into categorical variables we use the `LabelBinarizer` method from the `sklearn` module and once that is done, it is converted into `numpy` arrays.

Training, validation, and test sets splitting the dataset is essential for a fair assessment of prediction accuracy. Therefore, our next step is to evaluate the performance of our machine learning algorithm. For that, we use `train_test_split` from the `sklearn` model which helps us take a dataset and splitting it into two subsets. It can be used for any supervised learning algorithm and can be used for classification or regression problems. However, in the project, we perform a random split into 42 subsets and setting our test size at 20%. We also use `ImageDataGenerator` from Keras, because we do not have a big enough dataset so it will let us augment our images in real-time while our model is still training.

In this project, we use MobileNet architecture as it most importantly lets us easily use it with embedded systems and is faster than compared to others. MobileNetV2's architecture includes a completely convolutional layer with 32 filters, followed by 19 residual bottleneck layers. MobileNet is built on a simplified architecture that builds lightweight deep neural networks using depth-wise separable convolutions. We use the convolutional neural network and replace it with

MobileNetV2 in the second step as shown in figure 1.1. Once the input image is processed as an array it is sent to MobileNet and then we perform max-pooling. Max- pooling down-samples an image, lowering its dimensionality and making assumptions about features contained in the binned sub-regions.

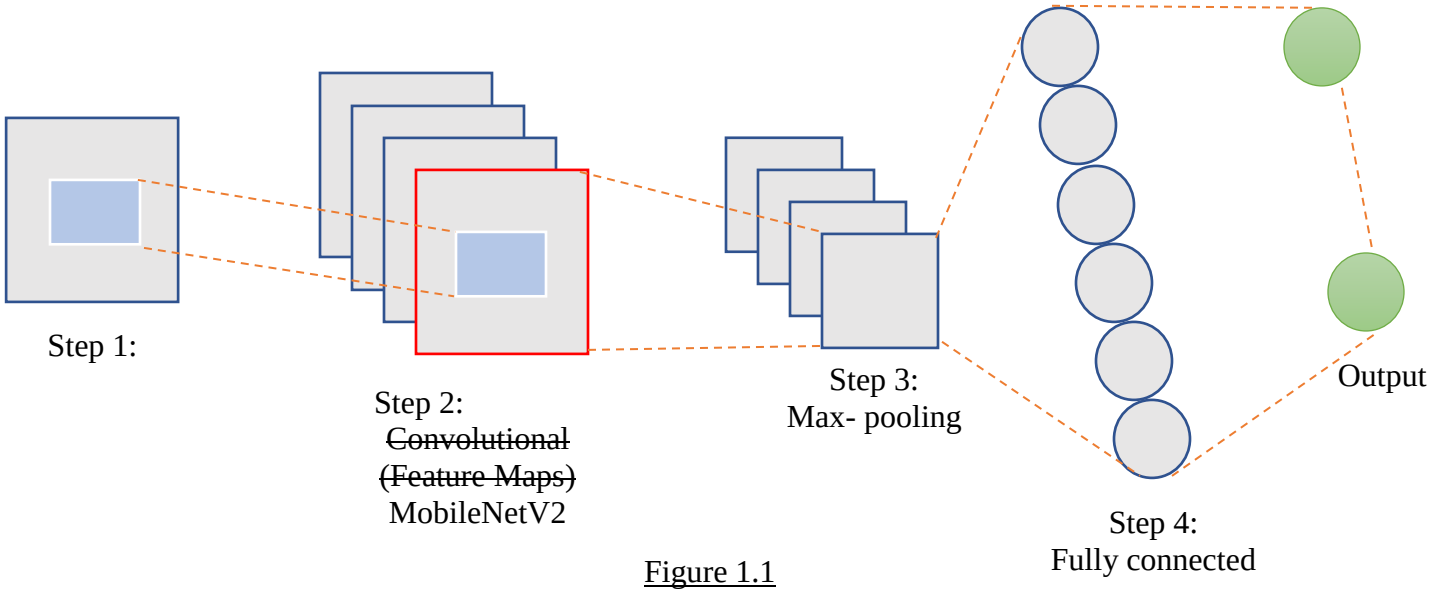


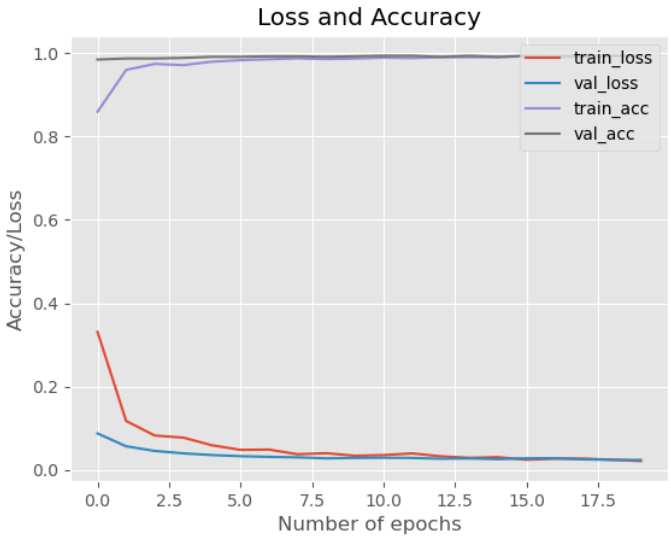
Figure 1.1

We divide MobileNet implementation into two models, i.e., the base model and the head model. The base model uses a weights parameter, `imagenet` and, has been frozen so it does not update in the first cycle of the epoch. There are pre-trained models for images, so `imagenet` helps us initialize those images to perform at better accuracy. In the head model, we construct our fully connected layer by pooling the images. The head model is placed at the top of our base model which pools the image, flattens the layer, dense the layer with ReLU as it is best for non-linear activation functions and lastly, it drops out any overfitting of the images. The rectified linear activation function, or ReLU for short, is a piecewise linear function that, if the input is positive, outputs the input directly; otherwise, it outputs zero.

One of the most important hyperparameter for the model is the learning rate. The learning rate determines how easily the model adapts to the situation. After multiple tries, we initialize the learning rate to 1e-4 (0.0001). Apparently, the lesser the learning rate we choose, the accurate the results are. The other hyperparameter we used is epochs which we set to 20 cycles. The number of epochs is a hyperparameter that controls how many times the learning algorithm runs over the entire training dataset. The third hyperparameter we used is the batch size, which we set to 32 samples. The batch size is a hyperparameter that specifies how many samples must be processed before the internal model parameters are updated. After compiling, our train model was ready within less than an hour and it creates two new files in our project folder. One is the graph plot for comparing the results for training loss and accuracy rates, which we will discuss at the end of this paper and the other is our mask_detector model. Now that our model is ready, we proceed to implement the trained model to our webcam using OpenCV.

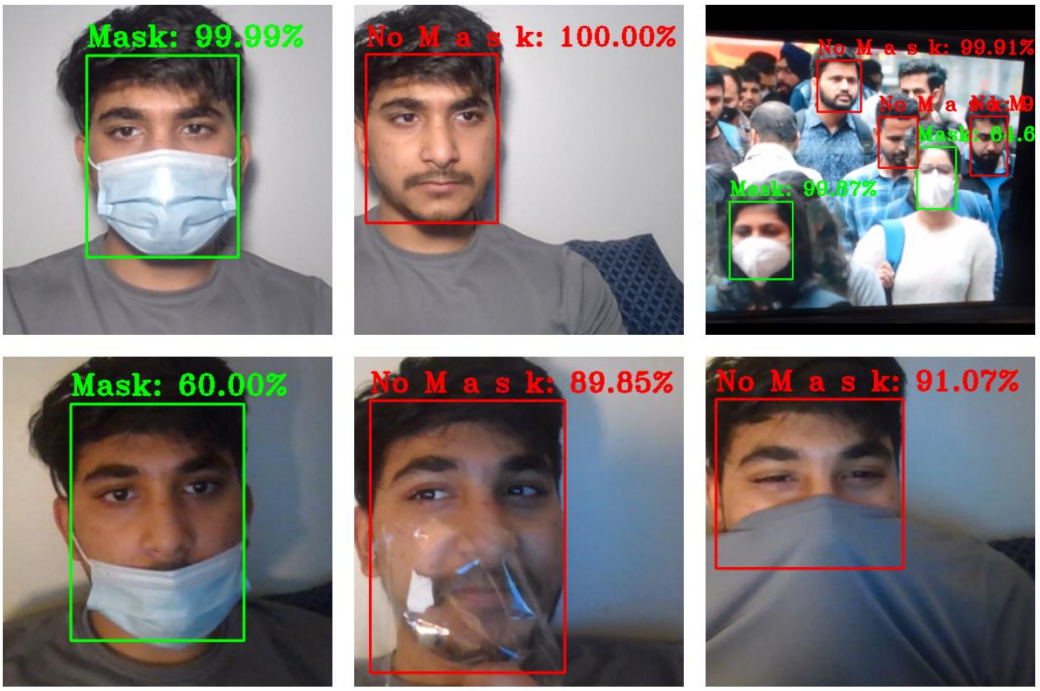
We need to import two models that are a face detector model which we imported from an online source and the train mask detector model that we built. To import the model, we use `dnn` (deep neural network) from cv2, which allows us to detect the face with the given parameters. Then we load our default camera and loop it using `imutils` to a maximum width of 800 pixels. Once we load the camera, we define a function with three parameters to grab the dimensions of the frame and create a blob from it. `BlobFromImage` lets us resize and crop pictures from the top, subtract mean values, scale values by the scale factor, and swap blue and red channels. This is the input picture we would like to preprocess before sending it to our deep neural network to be classified. In return, we get the location and prediction from it. Location return the x and y coordinates of the rectangle that is surrounding the face and prediction is the accuracy of the face

wearing a mask or not. Lastly, we label the rectangle using green color for mask and red color for without mask displaying the percentage of the accuracy we get from our train model.



We graph our mask_detector results using `matplotlib`, for the epochs and loss/accuracy rates. As we can see in the graph, the result that we got from our mask detector model is pretty good. The graph shows that the training accuracy and validation accuracy was slightly lower in the first two epochs and then went and stayed consistent throughout the 20 epochs we performed. Whereas, train loss and validation loss were higher in the beginning but then it lowered mostly after the 5th epoch.

Now we run our program and see how it detects masks through the webcam.



We tried to make it go through different possible scenarios and the results were pretty satisfying. The only place it was struggling majorly was in image 5 where I tried pretending as if I am wearing a transparent mask with a transparent piece of plastic to cover and even after multiple tries, it said no mask with an accuracy of 89.85%. However, I believe this can be fixed when we provide our train model with a wide variety of dataset. Also, in the third picture where we test the program with a group of people, it performed pretty well except the picture had to be adjusted in a certain angle several times.

The freedom we were given to complete this project helped me play with and learn new things. It would be great to continue this project and see how far it can be taken. We can try to implement this project in a real-life scenario by applying it to the embedded systems and see how well this performs. Projects similar to this such as helmet detection or safety goggles detection, and more can be used in different places to assure safety measures and avoid any miss happening.