Real-time Facemask Detector for COVID-19

(Tensorflow/Keras + OpenCV + Scikit-learn)

-Ashish Agarwal

Milestone Report

1. Motivation

Globally, the coronavirus stats say it has more than 16.3M confirmed cases and claimed over 649K lives so far, according to the Johns Hopkins University when I am writing this project repo. As many as 9.41 M people have recovered.

India's coronavirus cases are increasing in an unimaginable rate and is breaking the record in the highest single-day increase so far, every new day. The country's tally rose to 14,35,453 and the toll stood at 32,771. India is now the third worst-affected country by the pandemic and has overtaken Italy, according to Johns Hopkins University.

The World Health Organization said that new information showed that protective masks could be a barrier for potentially infectious droplets. The coronavirus primarily spreads through the transmission of respiratory droplets from infected people. On 5th day of June changed its guidelines about the use of protective face masks in public, saying that they must be worn at all places where physical distancing is not possible. The global health body had said in April that there was not enough evidence to show that healthy people should wear masks to shield themselves from the coronavirus.

Link: https://twitter.com/i/status/1268986094042992640

WHO also said that high-risk groups should wear medical grade masks in cases where physical distancing is not possible? Several countries, including India, have made wearing masks in public compulsory. In many states, people have been fined for not wearing masks. Maintaining hygiene and using protective equipment has become even more important ahead of the reopening of religious places, malls and restaurants in India from next week.

This motivated me to create a the COVID-19 Mask Detector with some of my ML/DL skills and making it such accurate that it could potentially be used to help ensure your safety and the safety of others (Leaving it on to the medical professionals to decide on, implement in public places).

2. Objective

The main objective of this project is(are) to:

- Collect images of people with and without wearing a face mask and preprocess it along with some data augmentation.
- Feed the compatible dataset to a Deep Learning Neural Network Model Architecture like MobileNetV2, VGG-16, ResNet-50 etc. and train it with pre-trained weights of ImageNet.
- Integrate the above model with Computer Vision libraries allowing it to use the web-cam of the computer to test the model in real-time.
- Carry out inferences from Model Evaluation.
- Plot the accuracies and losses in training and testing phase.

3. Overview

This is a simple image classification project trained on the top of Keras/Tensorflow API with MobileNetV2 deep neural network architecture having weights considered as pre-trained 'ImageNet' weights. The trained model (mask-detector-model.h5) takes the real-time video from webcam as an input and predicts if the face landmarks in Region of Interest (ROI) is 'Mask' or 'No Mask' with real-time on-screen accuracy.

4. Data Sources*

Data input in this project are images (with_mask and without_mask). Its easily available in Kaggle and if not able to find your custom dataset, we could generate our own dataset by constructing a face mask animation over the facial landmarks where the masks are usually gets wore.

Link: https://drive.google.com/drive/folders/1FHPJRCab-cyLq8IVz83LkU71gOc7gTS8?usp=sharing

5. Data Augmentation

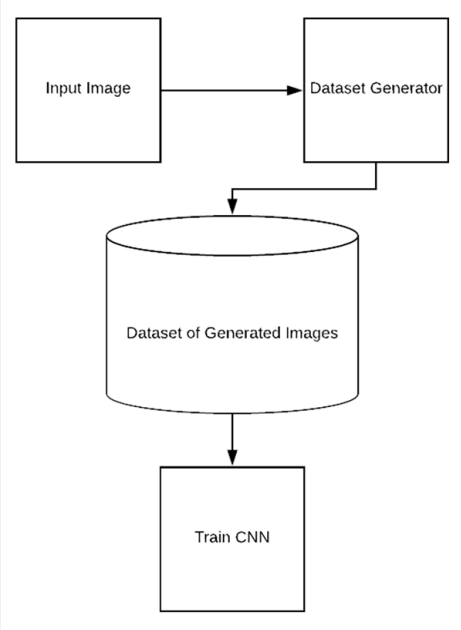
Data augmentation encompasses a wide range of techniques used to generate "new" training samples from the original ones by applying random jitters and perturbations (but at the same time ensuring that the class labels of the data are not changed).

- Our goal when applying data augmentation is to increase the generalizability of the model. At testing time, we do not apply data augmentation and simply evaluate our trained network on the unmodified testing data — in most cases, you'll see an increase in testing accuracy, perhaps at the expense of a slight dip in training accuracy.

- In the context of computer vision, data augmentation lends itself naturally. For example, we can obtain augmented data from the original images by applying simple geometric transforms, such as random:

  - Translations
  - Rotations
  - Changes in scale
  - Shearing
  - Horizontal (and in some cases, vertical) flips

Applying a (small) amount of the transformations to an input image will change its appearance slightly, but it does not change the class label — thereby making data augmentation a very natural, easy method to apply for computer vision tasks.



- OpenCV provides two functions to facilitate image preprocessing for deep learning classification i.e. `cv2.dnn.blobFromImage`. This function performs:

❖ Mean subtraction

Mean subtraction is used to help combat illumination changes in the input images in our dataset. We can therefore view mean subtraction as a technique used to aid our Convolutional Neural Networks.

Before we even begin training our deep neural network, we first compute the *average pixel intensity* across all images in the *training set* for each of the Red, Green, and Blue channels.

This implies that we end up with three variables:

μ_R, μ_G, and μ_B

Typically, the resulting values are a 3-tuple consisting of the mean of the Red, Green, and Blue channels, respectively.

For example, the mean values for the ImageNet training set are *R=103.93*, *G=116.77*, and *B=123.68* (you may have already encountered these values before if you have used a network that was pre-trained on ImageNet).

However, in some cases the mean Red, Green, and Blue values may be computed *channel-wise* rather than *pixel-wise*, resulting in an *MxN* matrix. In this case the *MxN* matrix for each channel is then subtracted from the input image during training/testing.

Both methods are perfectly valid forms of mean subtraction; however, we tend to see the pixel-wise version used more often, *especially* for larger datasets.

When we are ready to pass an image through our network (whether for *training* or *testing*), we subtract the *mean*, μ, from each input channel of the input image:

$$R = R - \mu_R$$
$$G = G - \mu_G$$
$$B = B - \mu_B$$

❖ Scaling

We may also have a *scaling factor*, σ, which adds in a normalization:

$$R = (R - \mu_R)/\sigma \qquad G = (G - \mu_G)/\sigma \qquad B = (B - \mu_B)/\sigma$$

The value of σ may be the standard deviation σ across the training set (thereby turning the preprocessing step into a standard score/z-score). However, may also be manually set (versus calculated) to scale the input image space into a particular range — it really depends on the architecture, how the network was trained, and the techniques the implementing author is familiar with.

It's important to note that not all deep learning architectures perform mean subtraction and scaling! Before you preprocess your images, be sure to read the relevant publication/documentation for the deep neural network you are using.

- **cv2.dnn.blobFromImage:**

It creates 4-dimensional blob from image. Optionally resizes and crops image from center, subtract mean values, scales values by scale factor, swap Blue and Red channels. The cv2.dnn.blobFromImage function returns a blob which is our input image after mean subtraction, normalizing, and channel swapping.

Informally, a blob is just a (potentially collection) of image(s) with the same spatial dimensions (i.e., width and height), same depth (number of channels), that have all be preprocessed in the same manner.

blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size, mean, swapRB=True)

I've provided a discussion of each parameter below:

Parameters:

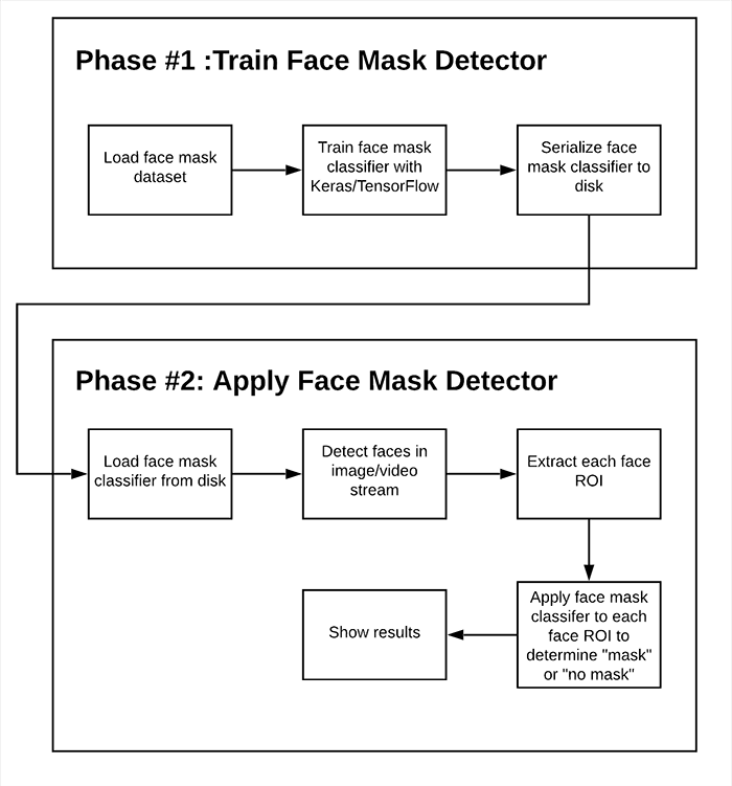
i. image: This is the input image we want to preprocess before passing it through our deep neural network for classification.
ii. scalefactor: After we perform mean subtraction, we can optionally scale our images by some factor. This value defaults to `1.0` (i.e., no scaling) but we can supply another value as well. It's also important to note that scalefactor should be 1 / \sigma as we're actually multiplying the input channels (after mean subtraction) by scalefactor.
iii. size: Here we supply the spatial size that the Convolutional Neural Network expects. For most current state-of-the-art neural networks this is either 224×224, 227×227, or 299×299.
iv. mean: These are our mean subtraction values. They can be a 3-tuple of the RGB means or they can be a single value in which case the supplied value is subtracted from every channel of the image. If you're performing mean subtraction, ensure you supply the 3-tuple in ` (R, G, B) ` order, especially when utilizing the default behavior of swapRB=True.
v. swapRB: OpenCV assumes images are in BGR channel order; however, the `mean` value assumes we are using RGB order. To resolve this discrepancy, we can swap the R and B channels in image by setting this value to `True`. By default, OpenCV performs this channel swapping for us.

6. Methodology

In order to train a Face Mask Detector, we need to break our project into two distinct phases, each with its own respective sub-steps:

- Training: Here we'll focus on loading our face mask detection dataset from disk, training a model (using Keras/TensorFlow) on this dataset, and then serializing the face mask detector to disk

- Deployment: Once the face mask detector is trained, we can then move on to loading the mask detector, performing face detection, and then classifying each face as with_mask or without_mask



7. Project structure

```
├── dataset
│   ├── with_mask [690 entries]
│   └── without_mask [686 entries]
├── examples
│   ├── example_01.png
│   ├── example_02.png
│   └── example_03.png
├── face detector
│   ├── deploy.prototxt
│   └── res10_300x300_ssd_iter_140000.caffemodel
```

```
├── detect_mask_image.py
├── detect_mask_video.py
├── mask_detector.model
├── evaluation.png
└── Data Augmentation and Model Training.ipynb
├── requirements.txt
└── mask-detector-model.model
```

- **Important Python Scripts:**

1. **Data Augmentation and Preprocessing.ipynb**: In this notebook Accepts our dataset is taken as input and fine-tuning is done with MobileNetV2 DNN architecture upon it to create our mask-detector-model. model. A training history evaluation.png containing accuracy/loss curves is also produced for better visualization of Model Evaluation through a plot. Some important processes which we performed here:

   - Data augmentation
   - Loading the MobilNetV2 classifier (fine-tune this model with pre-trained ImageNet weights)
   - Building a new fully-connected (FC) head
   - Pre-processing
   - Loading image data

- **Libraries Significance:**

   - scikit-learn: for binarizing class labels, segmenting our dataset, and printing a classification report.
   - imutils: To find and list images in our dataset.
   - matplotlib: To plot our training curves.

2. **detect_mask_from_webcam.py**: Using your webcam, this script applies face mask detection to every frame in the stream using webcam to read the real-time video.

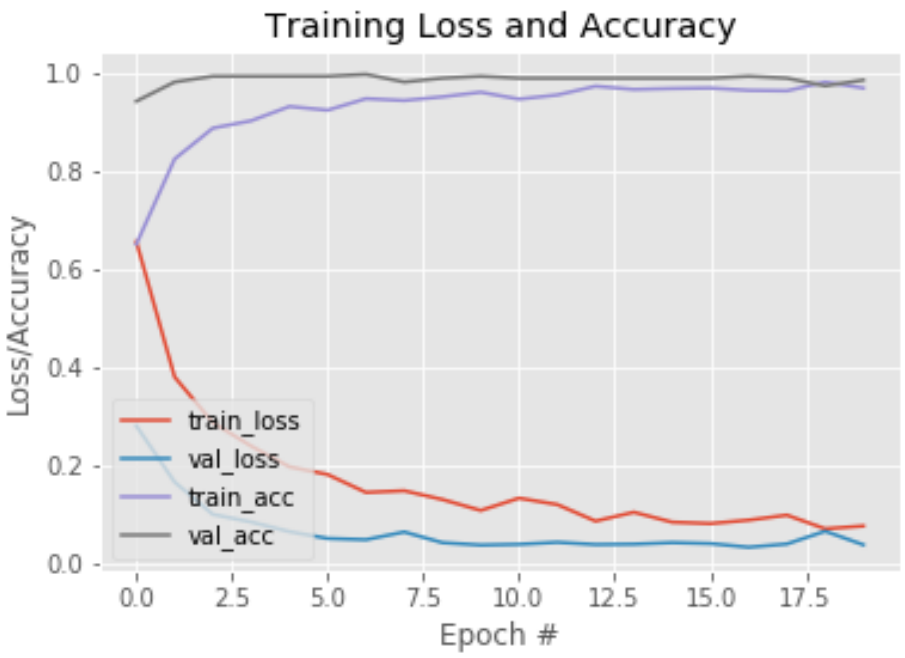
Some command line arguments in this script include:

```
--image: The path to the input image containing faces for inference
--face: The path to the face detector model directory (we need to localize faces prior to classifying them)
--model: The path to the face mask detector model that we trained earlier in this tutorial
--confidence: An optional probability threshold can be set to override 50% to filter weak face detections
```

8. Results/Classification Report

	precision	recall	f1-score	support
with_mask	0.97	1.00	0.99	138
without_mask	1.00	0.97	0.99	138
accuracy			0.99	276
macro avg	0.99	0.99	0.99	276
weighted avg	0.99	0.99	0.99	276

9. Accuracy/Loss Plot



10. To Do

- This approach reduces our computer vision pipeline to a single step — rather than applying face detection and then our face mask detector model, all we need to do is apply the object detector

to give us bounding boxes for people both with_mask and without_mask in a single forward pass of the network.

- An integration of this project to a web app/android app or any IoT devices in order to create a barricade like system to restrict the entry of peoples without mask at public places.

11. Technologies Used