Facemask detection using CNN and OpenCV

My GitHub repository link - https://github.com/adityaravichander/facemask_detector

Introduction

During this pandemic, it is essential for us to maintain social distance and use masks to prevent the further spread and protect ourselves. Our aim in this project is to use OpenCV, Deep learning, with Keras/Tensor flow to train a neural network and detect if a person is wearing a mask.

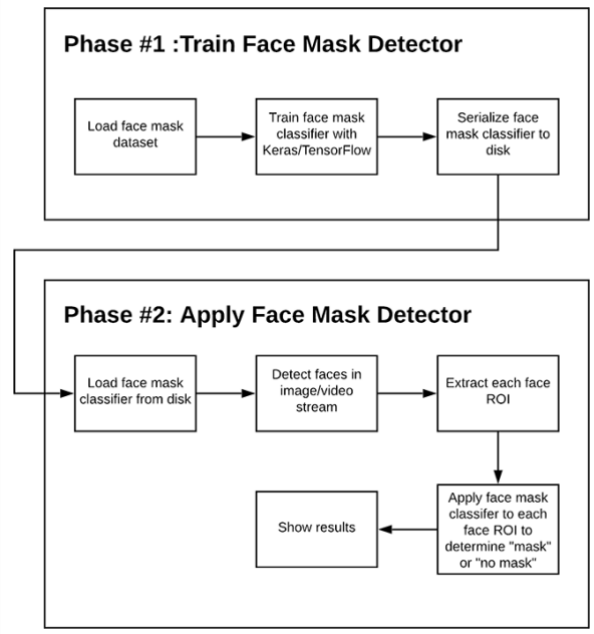
Data set

The data set used to train our network is a set of photos that have masks added to them artificially. The lack of proper huge data sets for people wearing masks is one of the reasons for using an augmented data set. The data set was found on a GitHub public repository. The link has been given in the references section. This dataset consists of 1,376 images belonging to two classes: with_mask: 690 images, without_mask: 686 images. If we use a set of images to create an artificial dataset of people wearing masks, you cannot "re-use" the images without masks in your training set — we will still need to gather non-face mask images that were not used in the artificial generation process!

If we include the original images used to generate face mask samples as non-face mask samples, the model will become heavily biased and fail to generalize well. Avoid that at all costs by taking the time to gather new examples of faces without masks.

Architecture

The program is split into two parts: training, and detection. The training of the network is done for 20 epochs. Considering that the training process is for 1,376 images over 20 epochs takes quite some time, we decided to save the model data into the disk and load it when we need to detect. The *model.save(),* method from the tensor flow library is used to save the trained model. This is a form of transfer learning.



The detection phase will load the network into the memory, then OpenCV is used to detect the Region of Interest in the image and then predict the presence of the mask. Once the prediction is obtained, a box of green/red colour is applied to the ROI based on if there is a mask detected or not.

This architecture is beneficial because we used the MobileNetV2 architecture, it's also computationally efficient, making it easier to deploy the model to embedded systems (Raspberry Pi, Google Coral, Jetson, Nano, etc.).

Neural Network

We use Convolutional Neural Network architecture for our network. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. In cases of extremely basic binary images, the method of using a Multi-Level perceptron might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet can successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

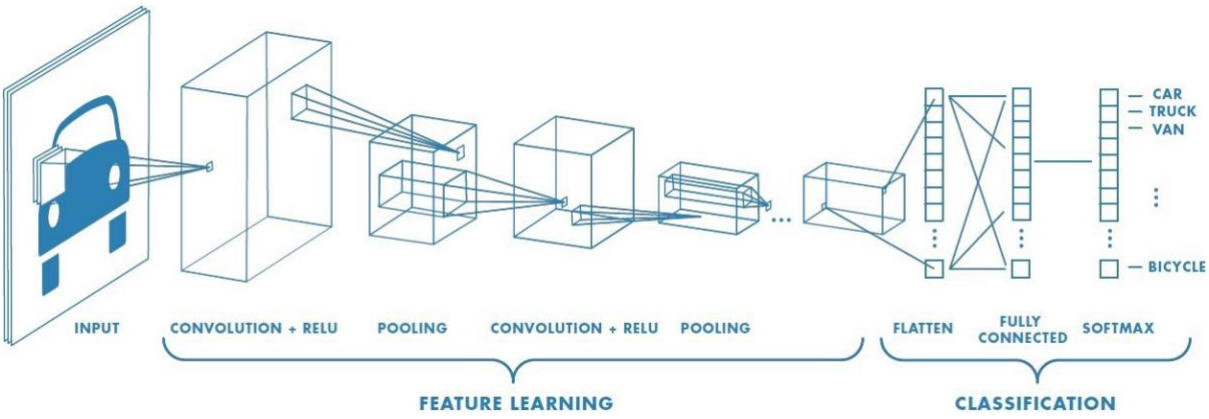


Figure 1 Architecture of CNN

Here, we will create the base model from the MobileNet V2 model developed at Google. MobileNet-v2 is a convolutional neural network that is 115 layers deep. This is pre-trained on the ImageNet dataset, a large dataset consisting of 1.4M images and 1000 classes. ImageNet is a research training dataset with a wide variety of categories like jackfruit and syringe. This base of knowledge will help us in classification.

First, we need to pick which layers of MobileNet V2 you will use for feature extraction. The very last classification layer (on "top", as most diagrams of machine learning models go from bottom to top) is not very useful. Instead, we will follow the common practice to depend on the very last layer before the flatten operation. This layer is called the "bottleneck layer". The bottleneck layer features retain more generality as compared to the final/top layer.

First, instantiate a MobileNet V2 model pre-loaded with weights trained on ImageNet. By specifying the *include_top=False* argument, you load a network that does not include the classification layers at the top, which is ideal for feature extraction. The network has an image input size of 224-by-224.

We then create our own head for the model by adding 5 more layers. These layers are AveragePooling2D, flatten, Dense(128 nodes), Dropout, and one more final Dense layer with 2 nodes for each of our class(with_mask and without_mask). These 2 parts are then combined to form our final model. We must then make the base layers of the mobilenetV2, as not trainable.

This is a form of transfer learning and will provide us with the full knowledge of the ImageNet dataset without the need for intensive computation. This also reduces the number of parameters we need to train by about 2.25 million, leading to reduction in the training times.

Training process

The data must be converted into a certain format to aid in more efficient training and better detection. The following sections give the working of the code used for training our network.

Pre-processing Input Data

Grabbing all the *imagePaths* in the dataset (Line 44). Initializing data and labels lists (Lines 45 and 46). Looping over the *imagePaths* and loading + pre-processing images (Lines 49-60).

Pre-processing steps include resizing to 224×224 pixels, conversion to array format, and scaling the pixel intensities in the input image to the range [-1, 1] (via the *preprocess_input* convenience function). Appending the pre-processed image and associated label to the data and labels lists, respectively (Lines 59 and 60). Ensuring our training data is in NumPy array format (Lines 63 and 64).

Lines 67-69 one-hot encode our class labels, meaning that our data will be in the following format. Using scikit-learn' s convenience method, Lines 73 and 74 segment our data into 80% training and the remaining 20% for testing.

During training, we will be applying on-the-fly mutations to our images to improve generalization. This is known as data augmentation, where the random rotation, zoom, shear, shift, and flip parameters are established on Lines 77-84. We will use the aug object at training time.

Finetuning MobileNetV2

Load MobileNet with pre-trained ImageNet weights, leaving off head of network (Lines 88 and 89). Construct a new FC head and append it to the base in place of the old head (Lines 93-102). Freeze the base layers of the network (Lines 106 and 107).

The weights of these base layers will not be updated during the process of backpropagation, whereas the head layer weights will be tuned.

Compilation and Validation

Lines 111-113 compile our model with the Adam optimizer, a learning rate decay schedule, and binary cross-entropy. If you are building from this training script with > 2 classes, be sure to use categorical cross-entropy. Face mask training is launched via Lines 117-122. Notice how our data augmentation object (aug) will be providing batches of mutated image data.

Once training is complete, we will evaluate the resulting model on the test set. Here, Lines 126-130 make predictions on the test set, grabbing the highest probability class label indices. Then, we print a classification report in the terminal for inspection. We also plot the accuracy and losses to look for overfitting. Line 138 serializes our face mask classification model to disk.

Implementation of the detector using OpenCV

Image detection

Here, we first load up the classifier model and the face detector network from the disk. This image is then pre-processed using the *blobfromImage()* function. Then the faces are detected, and their positions are used to narrow down our region of interest(ROI). There can be multiple faces detected in the same image and we will have to go through the detections and discard ones based on the confidence values that the function returns.

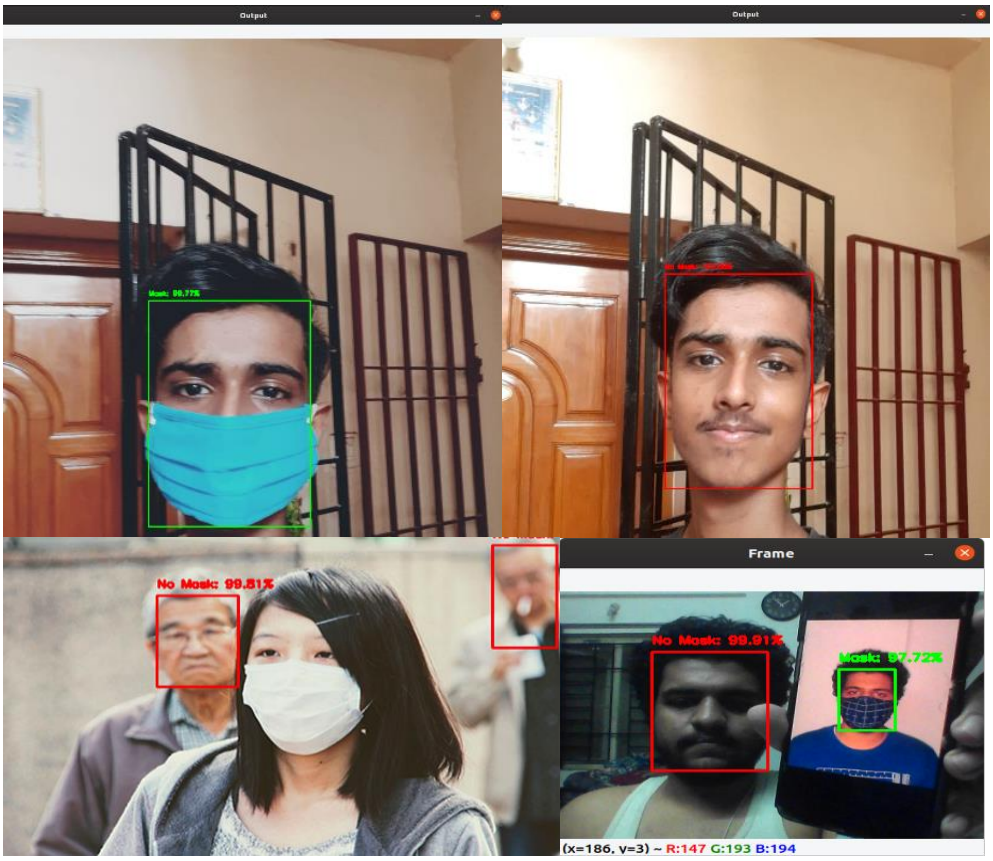
Next, we run our ROI through our classification net. We will apply similar pre-processing to the inputs as we did in the training and perform mask detection. Then, based on the returned *with_mask* and *without_mask* predictions we can assign the colour of the box and the confidence levels.

Video detection

The process is like the image detection but with only one difference. Instead of one image being passed as an input, we will extract the individual frames from the webcam video stream and apply our algorithm to it and draw the output to the output video stream.

Results:

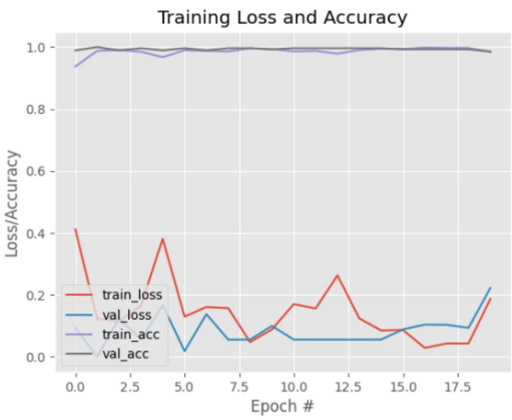
We tested the network on a couple of pictures of the internet and some of our own. The screenshots are attached below.

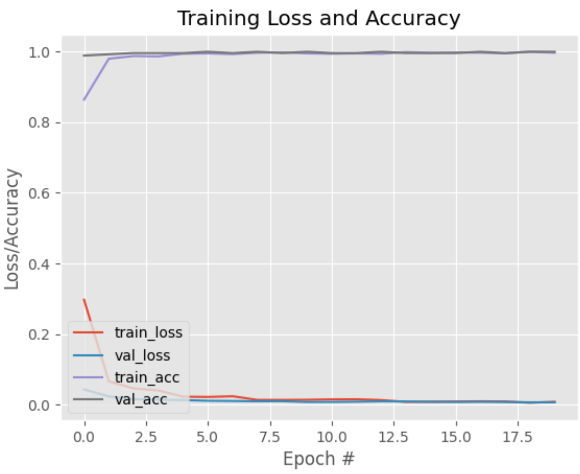


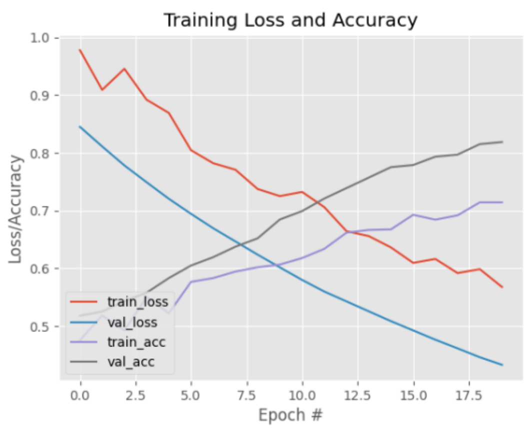
Varying the parameters

The initial parameters were varied and the respective losses per epoch was plotted to identify the optimal values for the parameters. The plots are attached in the same order as the table.

Epocs	Batch Size	Init_LR	Accuracy
20	32	1.00E-06	0.82
20	32	1.00E-05	0.99
20	32	1.00E-04	1
20	32	1.00E-02	0.99
20	32	1.00E-01	0.25







Scope for improvement

From the results sections above, our face mask detector is working quite well despite:

- Having limited training data
- The with_mask class being artificially generated (see the “Dataset” section above).

To improve our face mask detection model further, you should gather actual images (rather than artificially generated images) of people wearing masks. While our artificial dataset worked well in this case, there is no substitute for the real thing. Secondly, you should also gather images of faces that may “confuse” our classifier into thinking the person is wearing a mask when in fact they are not — potential examples include shirts wrapped around faces, bandana over the mouth, etc. All of these are examples of something that could be confused as a face mask by our face mask detector. Finally, you should consider training a dedicated two-class object detector rather than a simple image classifier.

Our current method of detecting whether a person is wearing a mask or not is a two-step process:

> Step #1: Perform face detection
>
> Step #2: Apply our face mask detector to each face

The problem with this approach is that a face mask, by definition, obscures part of the face. If enough of the face is obscured, the face cannot be detected, and therefore, the face mask detector will not be applied. To circumvent that issue, you should train a two-class object detector that consists of a with_mask class and without_mask class.

Combining an object detector with a dedicated with_mask class will allow improvement of the model in two respects. First, the object detector will be able to naturally detect people wearing masks that otherwise would have been impossible for the face detector to detect due to too much of the face being obscured. Secondly, 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. Not only is such a method more computationally efficient, it is also more “elegant” and end-to-end.

References:

- https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53#:~:text=A%20Convolutional%20Neural%20Network%20(ConvNet,differentiate%20one%20from%20the%20other.
- https://arxiv.org/pdf/1801.04381.pdf#:~:text=The%20architecture%20of%20MobileNetV2%20contains,computa%2D%20tion%20%5B27%5D.
- https://www.tensorflow.org/tutorials/images/transfer_learning
- https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c#:~:text=1.1.&text=In%20MobileNetV1%2C%20there%20are%202,convolutional%20filter%20per%20input%20channel.
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html