Report

Introduction and related work:

The corona virus COVID-19 pandemic is causing a worldly health disaster and there must exist some solution for its prevention. According to World Health Organization (WHO), wearing face masks is the best prevention mechanism. We can use an efficient approach of using AI to deal with this situation. AI can be used in keeping the crowded areas in safe environment. Therefore, a robust deep learning model is presented for face mask detection. The deep learning approaches used in detection is presented in succeeding section:

Body section:

The body elaborated the complete workflow about how the deep learning model is trained and evaluated with given dataset.

Dataset:

For most computer vision projects, the dataset is the most basic requirement and a quality dataset should possess consistency, diversity among the images (no redundancy), noiseless and good in number. The dataset we have used for training the machine learning model consists of two classes as it is a binary classification problem. The classes namely with_mask and without_mask each incorporating around 1700 images of diverse environments. Before feeding these two-class images to a machine learning which is explained briefly in the next section, all the images are resized to 224 by 224 to make the training process fast.

Methods:

The section focuses on the deep learning model architecture used in face mask detection. Building a machine learning model involves several trials and errors. Specifically, for those who are novices at the concept constantly tweak and alter their algorithms and models. During this time to make some right selection for the algorithm to use for building the machine learning model, challenges arise with handling data and tuning the architecture of the model. After experimenting different trails, the best suited machine learning model for face mask detection is based on Keras-api. So, the Keras-provided deep learning architecture we have used is MobileNet. The shorthand description of Keras is described next.

Programmatically speaking, Keras provides a fast, short and efficient way to read images from the directory and convert into a typical Keras object format compatible for training, testing and prediction. This is all done using the ImageDataGenerator built-in class provided by Keras. This class automatically configures all the operations and then instantiates its own generator object of augmented image batches. The instantiated generator objects are then used as input parameters in the Keras model method that is fit_generator function used for training. Specifically, in our case, this package is used for the following pre-processing operations:

- Rescaling of the images (224, 224)
- Dataset splitting (training: 80%, validation: 20%)
- Defining batch size (equals 32)
- Defining the class mode (binary, in this case)

Model:

The deep learning model is trained using the convolutional neural network architecture based on the dataset gathered as yet. Since the project requires the classification of the images, CNN is a go-to architecture. The neural network takes in input as multiple training and validation images which are processed in the convolutional hidden layers using the weights that are adjusted during the training phenomena. Then, the model spits out the prediction at the last output layer of

the architecture. During all this complex training process, there's no need to specify what patterns our model should learn during the transmissions in the hidden layers because neural network has the capability to learn on its own. This is all that what is going to interpret in this detailed section.

Model architecture:

The model used is MobileNetV2 which is a functional model that has allowed us to create a more flexible model for recognizing face with and without a mask. The internal structure of model is described in the next section:

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 1: Model summary

The above architecture is used to perform the face mask detection. Changes made in the architecture are in the input layer shape to 224 by 224 by 3 and in the output layer, changes the number of classes to 2.

Model compilation:

Once, the network or architecture for the model to train is defined, compilation is done before the training begin. Compilation transforms the architecture consisting of a sequence of several layers filled up with some hyper-parameters into a series of matrices ready for transformation depending on how Keras API configures this underlying computation in the backend. In short words, compilation configures and prepares the model for training. Compilation requires certain number of parameters to be specified which includes the optimization algorithm to train the network and loss function used to evaluate the model and this loss value is minimized by the optimization algorithm. Considering the case, 'Adam' is applied for optimization and 'binary_crossentropy' as a loss function because the model is based on binary outputs. The model compiled to make it ready for the training phenomena is represented in the succeeding section.

Model training process:

At this stage, the model instance is fitted to the fit_generator function to start the training process. This function trains the model for a fixed number of epochs (iterations on a dataset) using training and validation generators incorporating the pre-processed images and some other required attributes. The model had been trained for 10 epochs in almost 2 hours and took 84 steps to complete one epoch. There are certain approaches to analyze the trained model and makes some improvements in it.

Keras provides the capability to register callbacks when training the deep learning model. One of the most important callbacks for all deep learning models is the history callback. It records all the statistical analysis during each epoch and this includes training loss and training accuracy after trained on the trained dataset and validation loss and accuracy after cross-validated on the validation dataset. History object is returned after the completion of the an 2-hours long training process. We have utilized the graphical approach representing the whole training history of the model with epochs count on the x-axis and other parameters on the y-axis using the metrics stored in the history object. These parameters include validation loss, validation accuracy, training loss, and training accuracy. The following graphs are providing useful indications about how our model is trained. We can see from graphs that accuracies in the first graph and losses in the second graph are converging towards each other instantly just after the second epoch completion. There are no signs of overfitting or under fitting because lines are converging smoothly in a horizontal manner towards the end in the second half which is an indication that the model has learned weights or important features extracted from the images quietly well. The behavior can be seen graphically as follows:

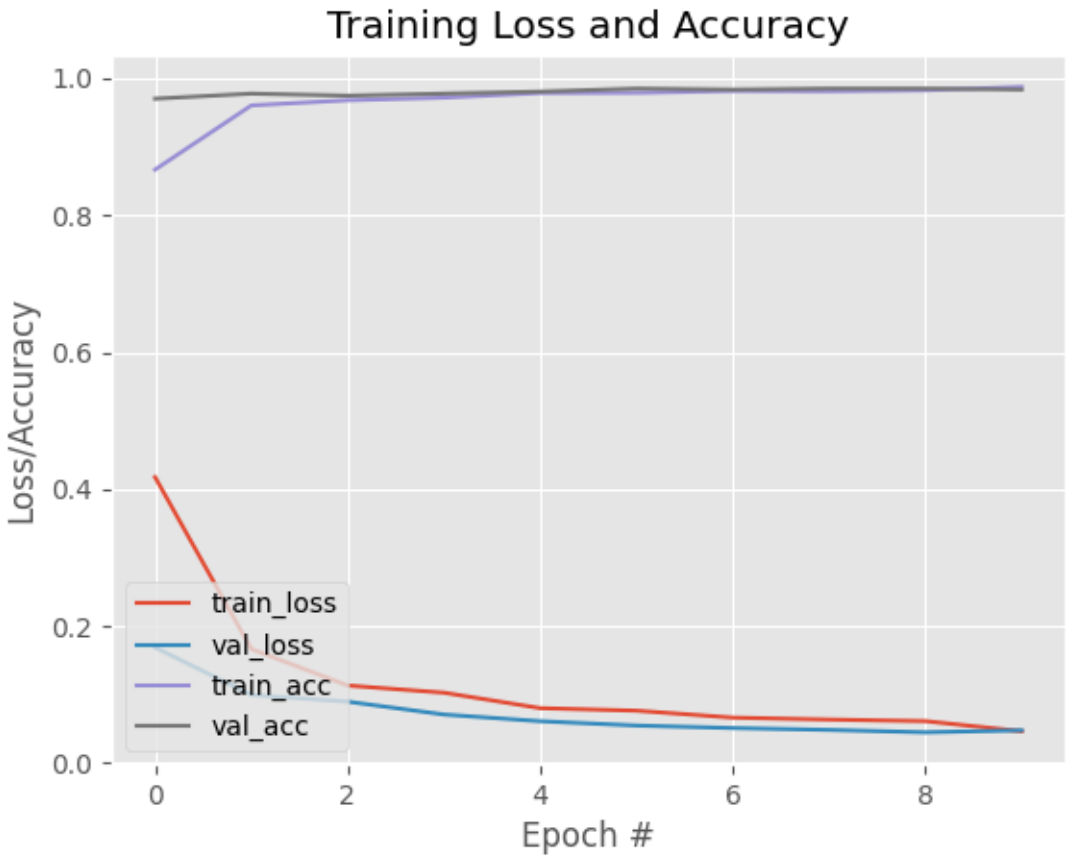

Figure 2: Training and validation accuracies and losses

After having the graphical analysis on the validation performance of the model, the next general step is to test the model on unforeseen data. This random and unforeseen evaluation will test the real intelligence of the model that how well it has learned from the input information. This unforeseen behavior is explained in the next section.

Testing and conclusion:

For the testing procedure, the model was tested on the test set incorporating around 200 images for each class. All models performed really well with 98> testing accuracies. As per-class analysis and evaluation, confusion matrix of the model is shown below:

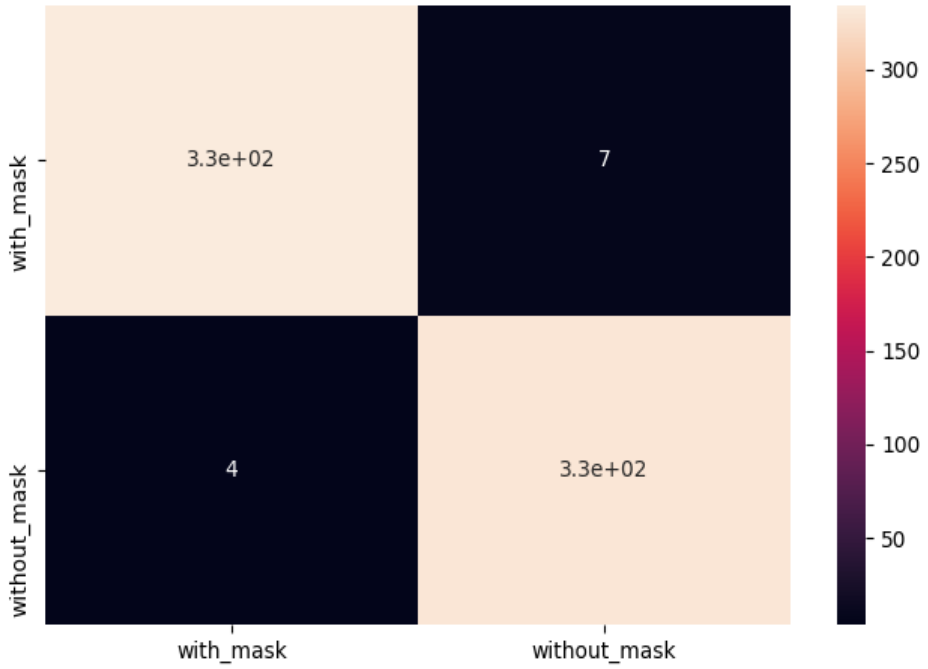

Figure 3: Confusion matrix

As per-class predictive analysis vs actual outputs, MobileNet also performed really well. But, overall performance of MobileNet is also cool surging to the accuracy of 99 percent. Statistically, performance measures of MobileNet are shown below through classification report.

	precision	recall	f1-score	support
with_mask	0.99	0.99	0.99	341
without_mask	0.99	0.99	0.99	334
accuracy			0.99	675
macro avg	0.99	0.99	0.99	675
weighted avg	0.99	0.99	0.99	675

Figure 4: Classification report: per-class analysis

References:

https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/

https://ieeexplore.ieee.org/document/9342585