# MASK DETECTION SECURITY

*"Wear मास्क, Curb कोविड - 19"*

**A PROJECT REPORT**

Submitted in partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering in Electronics and Communication Engineering.

Summited to

Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)

MINOR PROJECT II

Summited By

⁕ Yash Shrivastava (0176EC191085) ⁕ Pupul Bhatnagar (0176EC191057)

⁕ Simi Kushwaha (0176EC191074)

Under the Guidance Of

**Dr. Aparna Gupta**

(Project Incharge & Guide)

**Department of Electronics and Communication Engineering**

Lakshmi Narain College of Technology Excellence, Bhopal (Session 2021-22)

# Lakshmi Narain College of Technology Excellence, Bhopal

*Department of Electronics and Communication Engineering*

## CERTIFICATE

*This is to certify that the work embodied in this Minor Project* ***"MASK DETECTION SECURITY"*** *has been satisfactorily completed by Yash Shrivastava, Pupul Bhatnagar, Simi Kushwaha. It is a bonafide piece of work, carried out under our supervision and guidance in the* ***Department of Electronics and Communication Engineering, Lakshmi Narain College of Technology Excellence, Bhopal,*** *for partial fulfillment of the Bachelor of Engineering during the academic year 2021-22.*

Under Supervision Of

**Dr. Aparna Gupta**

(Project Incharge & Guide)

Approved by **Dr. Soheb Munir**

(Professor & Head of The Department)

Department of Electronics & Communication

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY BHOPAL
श्रद्धावान लभते ज्ञानम्

# DECLARATION

*We* ***Yash Shrivastava, Pupul Bhatnagar, Simi Kushwaha*** *students of* ***Bachelor of Technology, Department of Electronics and Communication Engineering, Lakshmi Narain College of Technology Excellence, Bhopal****, there by declare that the work presented in this* ***Minor Project Report II*** *is outcome of our own work, is bonafide, correct to the best of our knowledge and this work has been carried out taking care of Engineering Ethics. The work presented does not infringe any patented work and has not been submitted to any University for the award of any degree or any professional diploma.*

⁎ Yash Shrivastava (0176EC191085) ⁎ Pupul Bhatnagar (0176EC191057)

⁎ Simi Kushwaha (0176EC191074)

Date: ___/___/______

# ACKNOWLEDGEMENT

*Reasons can be given & they can be many but nothing can replace the efforts of numerous people behind the work, put in by the creator, giving us constant support all the way. We are greatly indebted to* ***Dr. Soheb Munir****, H.O.D of Electronics and Communication Engineering, LNCTE, Bhopal who has been there as our supervisor for the whole years of engineering and has been unstinting in her support and constructive critique throughout the progress of the dissertation. Also, we owe our profound gratitude to* ***Dr. Sunil Kumar Singh****, OSD, LNCTE College, Bhopal &* ***Dr. Ashok Kumar Rai****, Director of Administration, LNCT, Bhopal for providing us with all the necessary facilities as well as to* ***Dr. Vinay Kumar Sahu****, Principal, LNCTE for immense support. We would like to express our deep sense of gratitude to* ***Dr. Aparna Gupta*** *Minor Project Incharge of Electronics and Communication Engineering, for providing us all support and guidance all along, till the completion of our project work by providing all the necessary information for developing a good system. And Project Guide herself to started with us down this road and whose encouragement, suggestion and very constructive criticism have contributed immensely to the evolution of our ideas on project.*

⁕ Yash Shrivastava (0176EC191085) ⁕ Pupul Bhatnagar (0176EC191057)

⁕ Simi Kushwaha (0176EC191074)

# CONTENT

## Abstract

After the breakout of the worldwide pandemic COVID-19, there arises a severe need of protection mechanisms, face mask being the primary one. The basic aim of the project is to detect the presence of a face mask on human faces on live streaming video as well as on images. We have used deep learning to develop our face detector model. The architecture used for the object detection purpose is Single Shot Detector (SSD) because of its good performance accuracy and high speed. Alongside this, we have used basic concepts of transfer learning in neural networks to finally output presence or absence of a face mask in an image or a video stream. Experimental results show that our model performs well on the test data with 100% and 99% precision and recall, respectively.

# About us

My name is **YASH SHRIVASTAVA**. At present I am pursuing my degree in Bachelor of Technology in Electronics and Communication Engineering from Lakshmi Narain College of Technology Excellence, Bhopal. My current CGPA is 9.3. I have done my schooling Panini Jnanpeeth School (M.P.). I have excellent problem-solving skills and ability to perform well in a team. Passionate of coding and contribute for the best with my skills for my service. Work to Achieve the Highest Goal.

---

My name is **PUPUL BHATNAGAR**. At present I am pursuing my degree in Bachelor of Technology in Electronics and Communication from Lakshmi Narain College of Technology Excellence, Bhopal. My current CGPA is 8.42. I have done my schooling from Carmel Convent Senior Secondary School B.H.E.L. Bhopal (M.P.). I am able to handle multiple tasks on a daily basis and eager to learn new skills. Passionate to learn new skills and create by contribution towards the service.

---

My name is **SIMI KUSHWAHA**. At present I am pursuing my degree in Bachelor of Technology in Electronics and Communication Engineering from Lakshmi Narain College of Technology Excellence, Bhopal. My current CGPA is 8.55. I have done my schooling from St. George Sr. Sec School Bhopal (M.P.). I've an optimistic mindset and do not shy away from hard work and dense deadlines and have lunacy towards the goals.

# INTRODUCTION

The year 2020 has shown mankind some mind-boggling series of events amongst which the COVID19 pandemic is the most life-changing event which has startled the world since the year began. Affecting the health and lives of masses, COVID-19 has called for strict measures to be followed in order to prevent the spread of disease. From the very basic hygiene standards to the treatments in the hospitals, people are doing all they can for their own and the society's safety; face masks are one of the personal protective equipment. People wear face masks once they step out of their homes and authorities strictly ensure that people are wearing face masks while they are in groups and public places.

To monitor that people are following this basic safety principle, a strategy should be developed. A face mask detector system can be implemented to check this. Face mask detection means to identify whether a person is wearing a mask or not. The first step to recognize the presence of a mask on the face is to detect the face, which makes the strategy divided into two parts: to detect faces and to detect masks on those faces. Face detection is one of the applications of object detection and can be used in many areas like security, biometrics, law enforcement and more. There are many detector systems developed around the world and being implemented. However, all this science needs optimization; a better, more precise detector, because the world cannot afford any more increase in corona cases.

In this project, we will be developing a face mask detector that is able to distinguish between faces with masks and faces with no masks. In this report, we have proposed a detector which employs SSD for face detection and a neural network to detect presence of a face mask. The implementation of the algorithm is on images, videos and live video streams.

The rest of the report is organized as follows. In Chapter 2, we will go through the requirements review and related work. In Chapter 3, the methodology of our proposed solution is discussed in detail. In Chapter 4, the model is evaluated, and results discussed. Chapter 5, about the Source Code and Installation. In Chapter 6 about the Software interface and will talk further about the limitations, Use Case, Future Scope, and Final with the Conclusion.

## 1.1 Problem Statement

The world has been fighting the pandemic in great spirit, with the unlocking phases being motion. This is the time to be more proactive than ever. Governments all around the world have recognized the power of Artificial Intelligence & Machine Learning in order to battle the virus.

Since social distancing and wearing a mask are the only monitored way to avoid the infection till vaccination become accessible to all.

Computer Vision is the way of Mask Detection in a reviving factor to get of lives back on track. MASK DETECTION SECURITY can solve the monitoring issue in geographies in high population.

## 1.2. Objective

Face mask detection refers to detect whether a person is wearing a mask or not. In fact, the problem is reverse engineering of face detection where the face is detected using different machine learning algorithms for the purpose of security, authentication and surveillance. Face detection is a key area in the field of Computer Vision and Pattern Recognition. A significant body of research has contributed sophisticated to algorithms for face detection in past.

# Requirements

## 2.1 Hardware Requirement

- Monitoring Automation Screen
- Integrated CCTV camera
- Admin Database Setup
- DVR System
- Alert Alarm connected with DVR

**Higher the Specification Higher the Performance of Model**

## 2.1 Software Requirement

- **Front-End**
    - Alert Receiver
    - Database Handler

- **Back-End**
    - PYTHON 3.9.6
    - PyCharm\ VScode \ “Or Any Python IDE”
    - CMD
    - PowerShell

- **Dependencies**

  - OpenCV
  - TensorFlow
  - Keras
  - Imutils
  - MobileNetV2
  - ResNet50
  - SciPy
  - NumPy

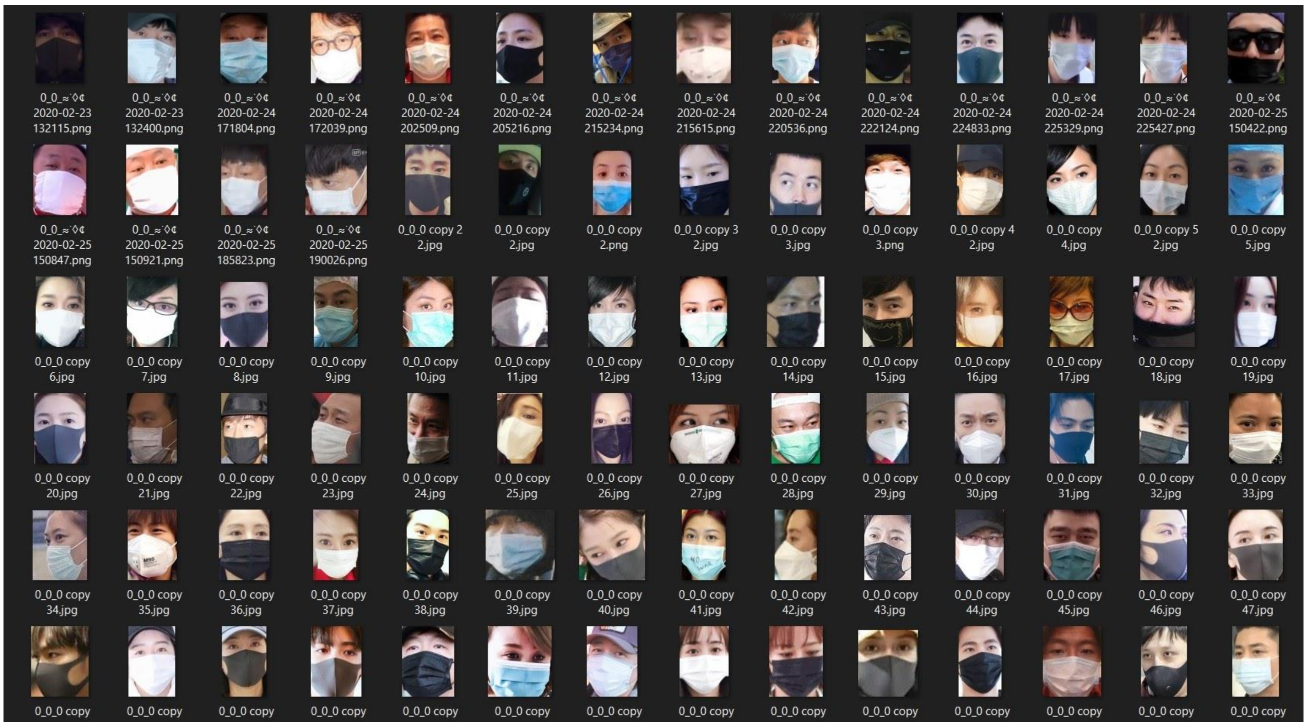
Dependencies Performance Can be improved by installing all the upgrades at that time.

# Methodology

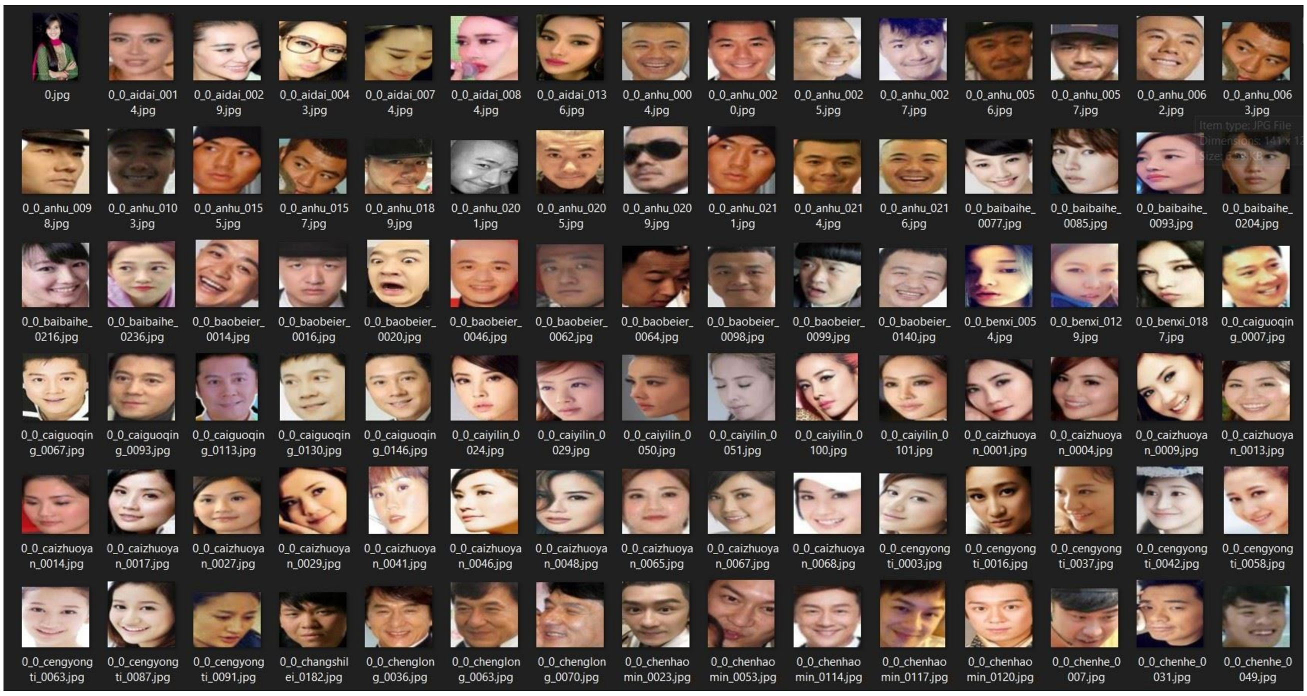
## 3.1 Dataset

The dataset which we have used consists of 3835 total images out of which 1916 are of people wearing mask and 1919 are of people not wearing mask.

As shown if Figure 3.1.1 and 3.1.2 All the images are actual images extracted from Bing Search API, Kaggle datasets and RMFD dataset. From all the three sources, the proportion of the images is equal. The images cover diverse races i.e. Asian, Caucasian etc. The proportion of masked to unmasked faces determine that the dataset is balanced.



(Figure 3.1.1: Dataset with Mask)



(Figure 3.1.2: Dataset Without Mask)

We need to split our dataset into three parts: training dataset, test dataset and validation dataset. The purpose of splitting data is to avoid overfitting which is paying attention to minor details/noise which is not necessary and only optimizes the training dataset accuracy. We need a model that performs well on a dataset that it has never seen (test data), which is called generalization. The training set is the actual subset of the dataset that we use to train the model. The model observes and learns from this data and then optimizes its parameters. The validation dataset is used to select hyperparameters (learning rate, regularization parameters). When the model is performing well enough on our validation dataset, we can stop learning using a training dataset. The test set is the remaining subset of data used to provide an unbiased evaluation of a final model fit on the training dataset. Data is split as per a split ratio which is highly dependent on the type of model we are building and the dataset itself. If our dataset and model are such that a lot of training is required, then we use a larger chunk of the data just for training which is our case. If the model has a lot of hyperparameters that can be tuned, then we need to take a higher amount of validation dataset. Models with a smaller number of hyperparameters are easy to tune and update, and so we can take a smaller validation dataset.

In our approach, we have dedicated 80% of the dataset as the training data and the remaining 20% as the testing data, which makes the split ratio as 0.8:0.2 of train to test set. Out of the training data, we have used 20% as a validation data set. Overall, 64% of the dataset is used for training, 16% for validation and 20% for testing.

## 3.2 Architecture

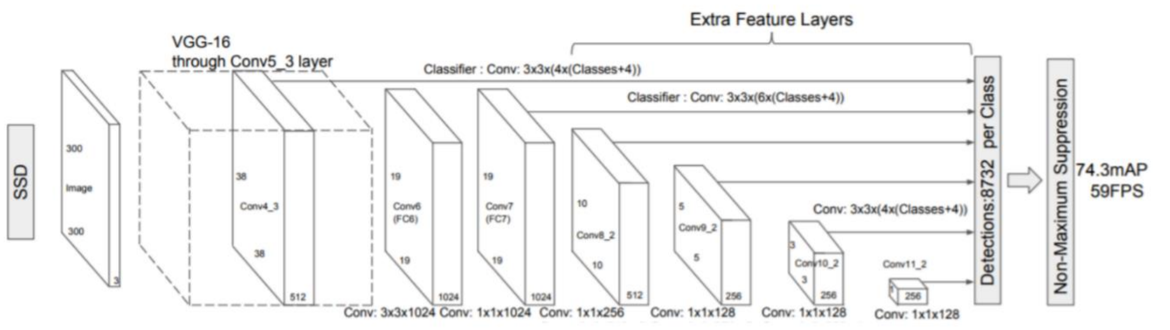
The working of the Single Shot Detector algorithm relies on an input image with a specified bounding box against the objects. The methodology of predicting an object in an image depends upon very renowned convolution fashion. For each pixel of a given image, a set of default bounding boxes (usually 4) with different sizes and aspect ratios are evaluated. Moreover, for all the pixels, a confidence score for all possible objects are calculated with an additional label of 'No Object'. This calculation is repeated for many different feature maps.

In order to extract feature maps, we usually use the predefined trained techniques which are used for high quality classification problems. We call this part of the model a base model. For the SSD, we have VGG-16 network as our base model. At the training time, the bounding boxes evaluated are compared with the ground truth boxes and in the back propagation, the trainable parameters are altered as per requirement. We truncate the VGG-16 model just before the classification layer and add feature layers which keep on decreasing in size. At each feature space, we use a kernel to produce outcomes which depicts corresponding scores for each pixel whether there exists any object or not and the corresponding dimensions of the resulting bounding box.

VGG-16 is a very dense network having 16 layers of convolution which are useful in extracting features to classify and detect objects. The reason for the selection is because the architecture consists of stacks of convolutions with 3x3 kernel size which thoroughly extract numerous feature information along with max-pooling and ReLU to pass the information flow in the model and adding non linearity respectively from the given image. For additional nonlinearity, it uses 1x1 convolution blocks which does not change the spatial dimension of the input. Due to the small size filters striding over the image, there are many weight parameters which end up giving an improved performance.

The block diagram Figure 3.2.1 shows the working functionality of SSD. At the input end, we can see the VGG-16 being used as the base model. Some additional feature layers are added at the end of the base model to take care of offsets and confidence scores of different bounding boxes. At end part of the figure, we can see the layers being flattened to make predictions for different bounding boxes. At the end, non-maximum suppression is used whose purpose is to remove duplicate or quite similar bounding boxes around same objects. There may be situations where the neighbouring pixel also predicts a bounding box for an object with a bit less confidence which is finally rejected.

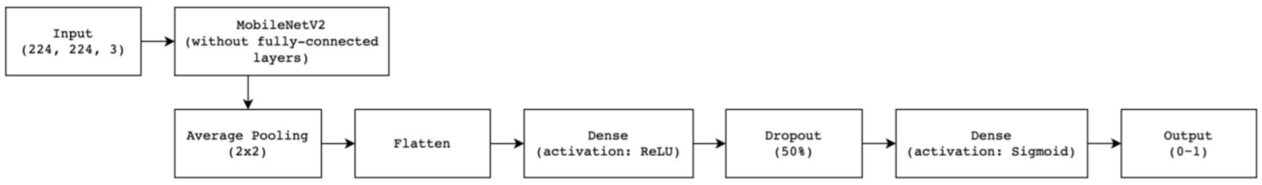
(Figure 3.2.1: Working Of SSD)



The problem can be solved in two parts: first detecting the presence of several faces in a given image or stream of video and then in the second part, detect the presence or absence of face mask on face. In order to detect the face, we have used the OpenCV library. The latest OpenCV includes a Deep Neural Network (DNN) module, which comes with a pre-trained face detection convolutional neural network (CNN). The new model enhances the face detection performance compared to the traditional models. Whenever a new test image is given, it is first converted into BLOBS (Binary Large Object refers to a group of connected pixels in a binary image) and then sent into the pretrained model which outputs the number of detected faces. Every face detected comes out with a level of confidence which is then compared with a threshold value to filter out the irrelevant detections. After we have the faces, we need to evaluate the bounding box around it and send it to the second part of the model to check if the face has a mask or not.

The second part of the model is trained by us using a dataset consisting of images with mask and without mask. We have used Keras along with TensorFlow to train our model. First part of the training includes storing all labels of the images in a Numpy array and the corresponding images are also reshaped (224, 244, 3) for the base model. Image augmentation is a very useful technique because it increases our dataset with images with a whole new perspective. Before inputting, we performed the following image augmentations randomly: rotations up to 20 degrees, zooming in and out up to 15%, width or height shift up to 20%, up to 15 degrees shear angle in the counter clockwise direction, flip inputs horizontally and points outside the boundaries of the inputs are filled from the nearest available pixel of the input. For the image classification, it is now a common practice to use transfer learning which means using a model which has been pre-trained on millions of labels before and it has been tested that this method results in significant increase in accuracy. Obviously, the assumption here is that both the problems have sufficient similarity. It uses a well-structured and deep neural network that has been trained on a large amount of data set. Due to somewhat same nature of the problem, we can use the same weights which have the capability to extract features and later in the deep layers, convert those features to objects.

The base model that we have used here is MobileNetV2 with the given ‘ImageNet’ weights. ImageNet is an image database that has been trained on hundreds of thousands of images hence it helps a lot in Image classification. For the base model, we truncate the head and use a series of our self-defined layers. We used an average pooling layer, a flatten layer, a dense layer with output shape (None, 128), and activation ReLU, a 50% dropout layer for optimization, finally another dense layer with output shape (None, 2), and Sigmoid activation is used. The overall process flow diagram of the algorithm is shown below IN Figure 3.2.2.

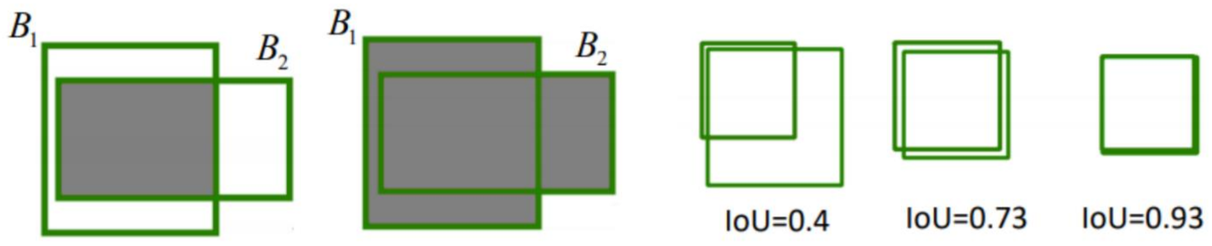


(Figure 3.2.2: Process Flow diagram of the Model)

## 3.3 Training

At the training time, for each pixel, we compare the default bounding boxes having different sizes and aspect ratios with ground truth boxes and finally use Intersection over Union (IoU) method to select the best matching box. IoU evaluates how much part of our predicted box match with the ground reality. The values range from 0 to 1 and increasing values of IoU determine the accuracies in the prediction; the best value being the highest value of IoU. The equation and pictorial description of IoU is given as follow:

$$IoU(B_1, B_2) = \frac{B_1 \cap B_2}{B_1 \cup B_2}$$



(Figure 3.3.1: Pictorial representation of IoU)

## 3.4 Hyperparameter

A hyperparameter is a parameter or a variable we need to set before applying an algorithm into a dataset. These parameters express the “High Level” properties of the model such as its complexity or how fast it should learn. Hyperparameters are fixed before the actual training process begins. They can be divided into two categories: optimizer hyperparameters and model hyperparameters.

Optimizer parameters help us to tune or optimize our model before the actual training process starts. Some common optimizer hyperparameters are as follows. Learning rate is a hyperparameter that controls how much we are adjusting the weights of our neural network with respect to the gradient. Mini-batch size is a hyperparameter that influences the resource requirements of the training and impacts training speed and number of iterations. Epochs are the hyperparameters that determine the frequency of running the model. One epoch is when an entire dataset is passed forward and backward through the neural network only once. Model hyperparameters are parameters that are more involved in the architecture or structure of the model. They help us to define our model complexity based on the different layers like the input layer, hidden layer, and output layer of a neural network.

Initially, we trained with different values of hyperparameters by changing one and keeping the other constant and noted down the results in each case. We selected the hyperparameters that produced better performance through evaluation metrics. We have chosen the hyperparameters as follows: initial learning rate is taken as 0.0001, batch size is taken to be 32 and number of epochs as 20. In our case, the target size is also one of the hyperparameters which we kept (224, 224, 3) as it is default input shape of MobileNetV2.

## 3.5 Loss Function

The loss of the overall detection problem can be broken into two main subsets: localization loss and confidence loss. The localization loss is just the difference between the default predicted bounding box and ground truth bounding box (g). For a given center (cx, cy), we try to alter the width and height of the box such as to decrease the loss. Respectively, the equations for the localization and confidence losses can be defined as:

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

and

$$L_{locf}(x,c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

<u>The notations used in above equation are as followed:</u>

- g: ground truth bounding box
- l: predicted box
- d: default bounding box
- $x_i^\rho j$ : matching $i^{th}$ predicted box with j th default box with “p” category.
- cx, cy: distance from centre of box in both x and y direction
- w, h = width and height with respect to image size
- c: confidence of presence of object or not.

Confidence loss is the just measure of how high the probability of the presence of an object is, when there exists an object. Similarly, the localization loss is the measure of how much a predicted box differs from the ground truth box in dimensions. Our model will try to push down both losses by predicting the presence of object and then correctly classifying it to the right class.

# Evaluation

## 4.1 Testing

We tried using three different base models for detecting ‘mask’ or ‘no mask’. The exercise was done to find the best fit model in our scenario. The evaluation process consists of first looking at the classification report which gives us insight towards precision, recall and F1 score. The equations of these three metrics are as follows:

$$Precision = \frac{True\ Positives}{Positives + False\ Positives}$$

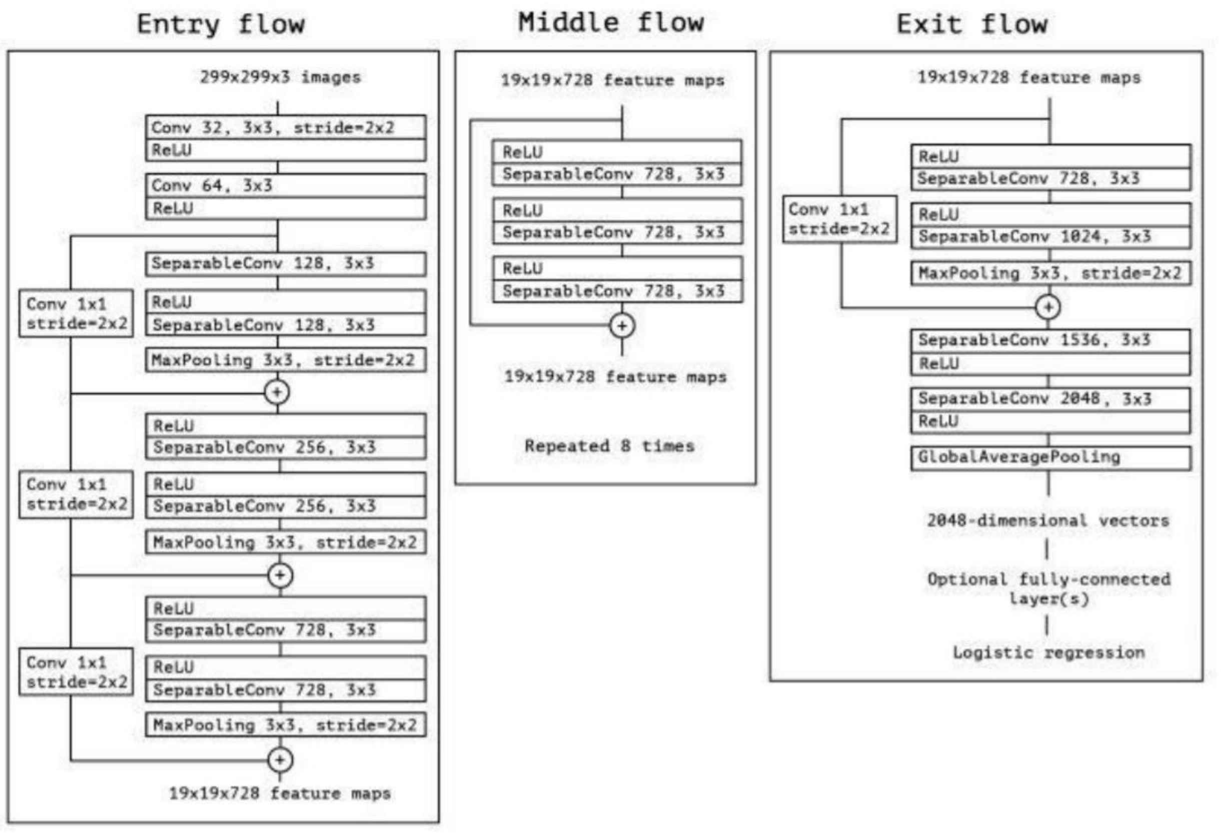
$$Recall = \frac{True\ Positives}{Positives + False\ Negatives}$$

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Positives + Negatives}$$

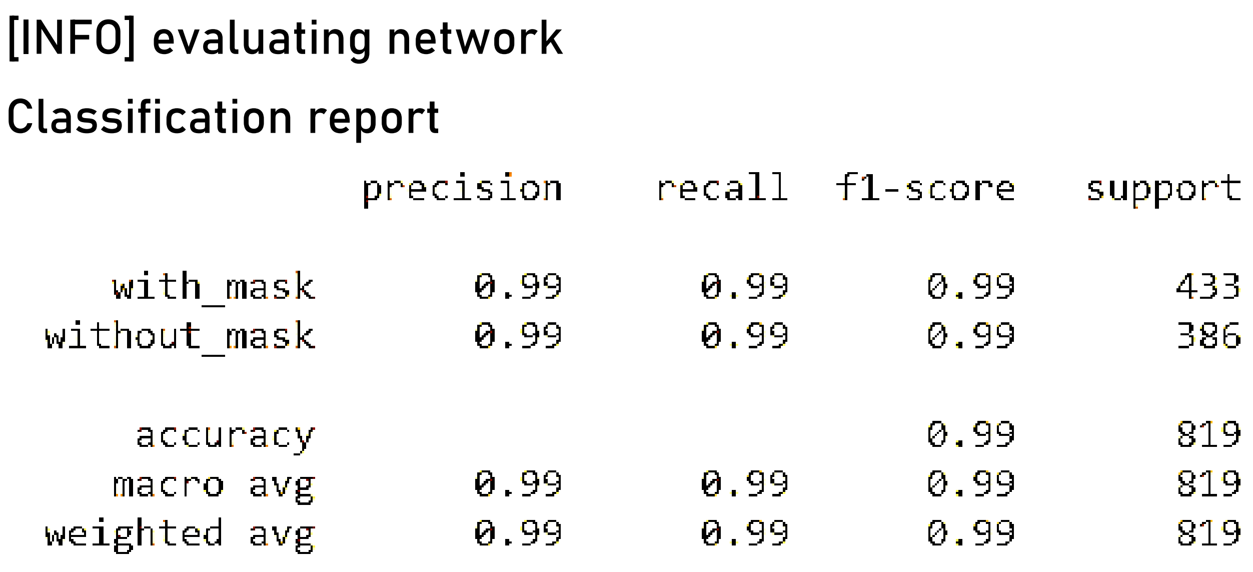
Using these three metrics, we can conclude which model is performing most efficiently. The second part consists of plotting the train loss, validation loss, train accuracy and validation accuracy which also proves helpful in choosing a final model. The results of different choices are shown below.

## 4.1.1 Xception

The complete form of Xception is “Extreme Inception”. Basically, the Xception architecture Figure 4.1.1.1 is designed with depth wise separable convolution layers with residual connections. These convolutional layers are linearly stacked with each other. This architecture is easy to define and easy to modify. If we compare it with Inception V2 and V3 then Inception V2 and V3 are much more complex to define.

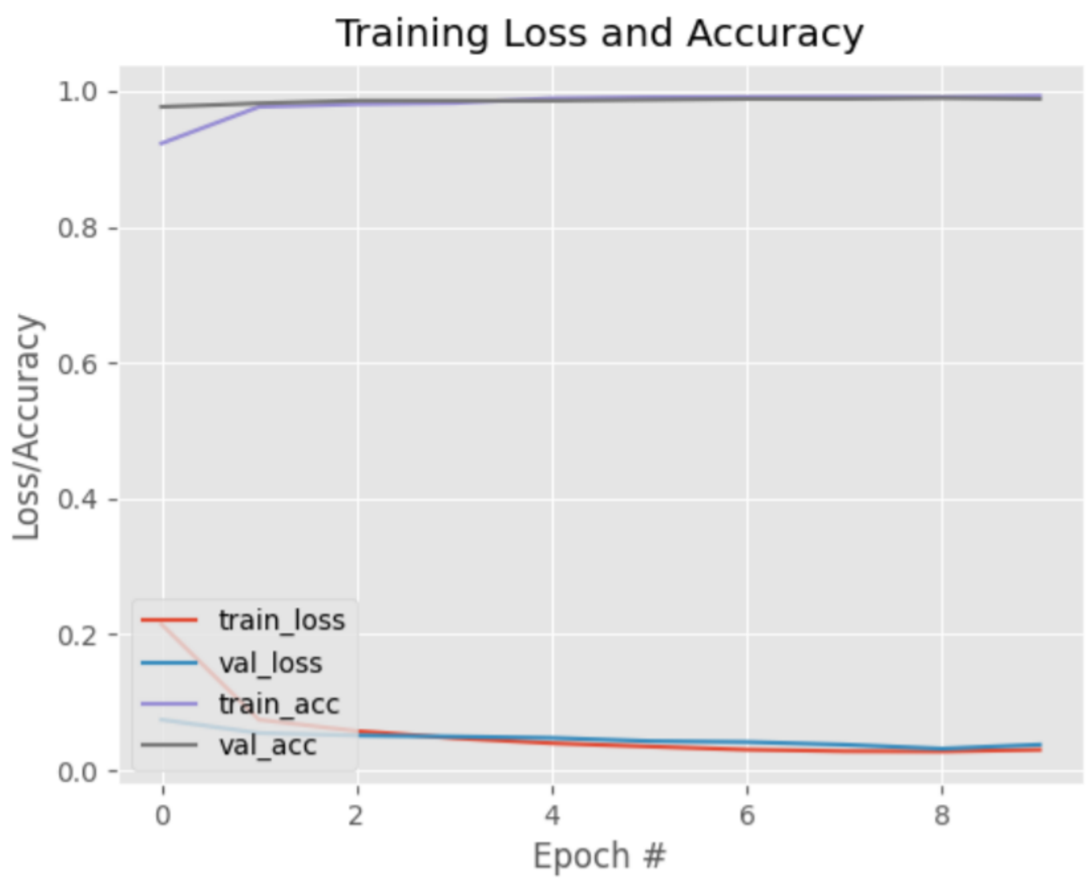


(Figure 4.1.1.1: The Xception Architecture)



```
[INFO] evaluating network
Classification report
              precision    recall  f1-score   support

   with_mask       0.99      0.99      0.99       433
without_mask       0.99      0.99      0.99       386

    accuracy                           0.99       819
   macro avg       0.99      0.99      0.99       819
weighted avg       0.99      0.99      0.99       819
```

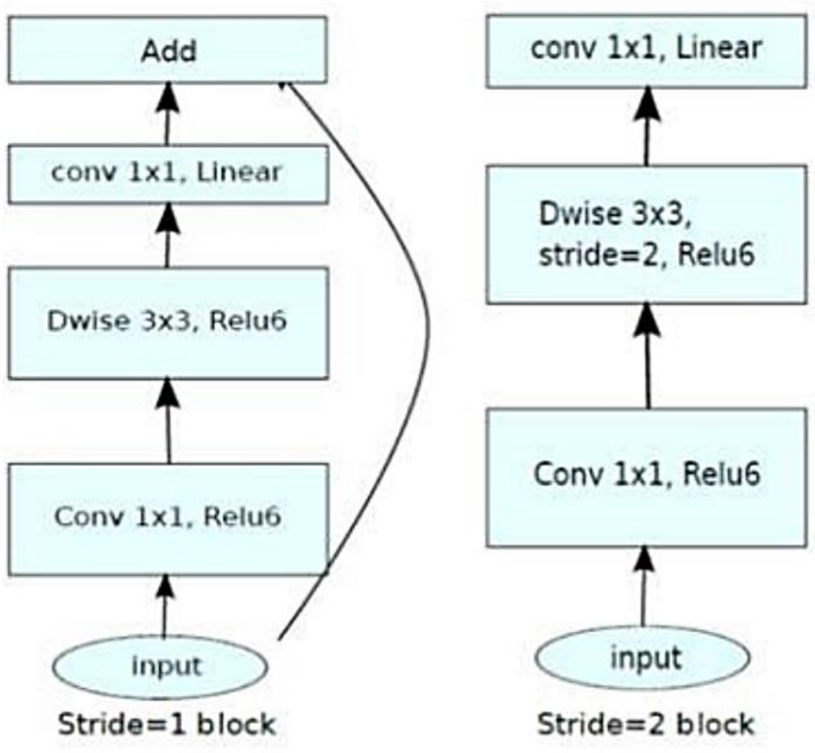
(Figure 4.1.1.2: Classification report for Xception)



(Figure 4.1.1.3: Loss Graph for Xception)

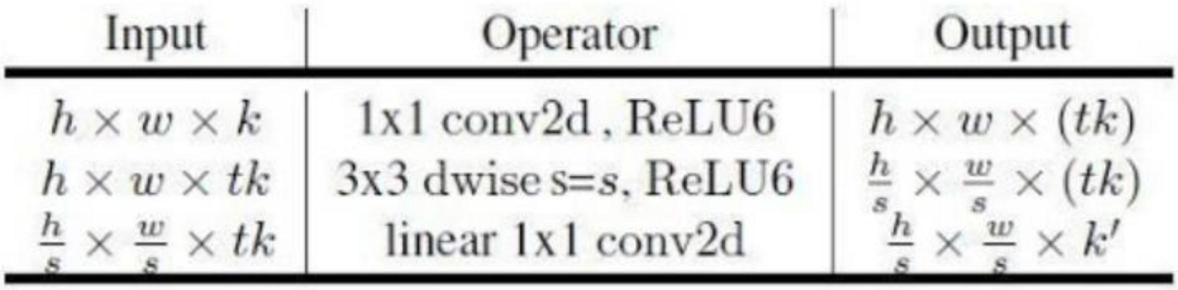
## 4.1.2 MobileNetV2

MobileNetV2 Figure 4.1.2.1 is an architecture of bottleneck depth-separable convolution building of basic blocks with residuals. It has two types of blocks. The first one is a residual block with stride of 1. Second one is also residual block with stride 2 and it is for downsizing.



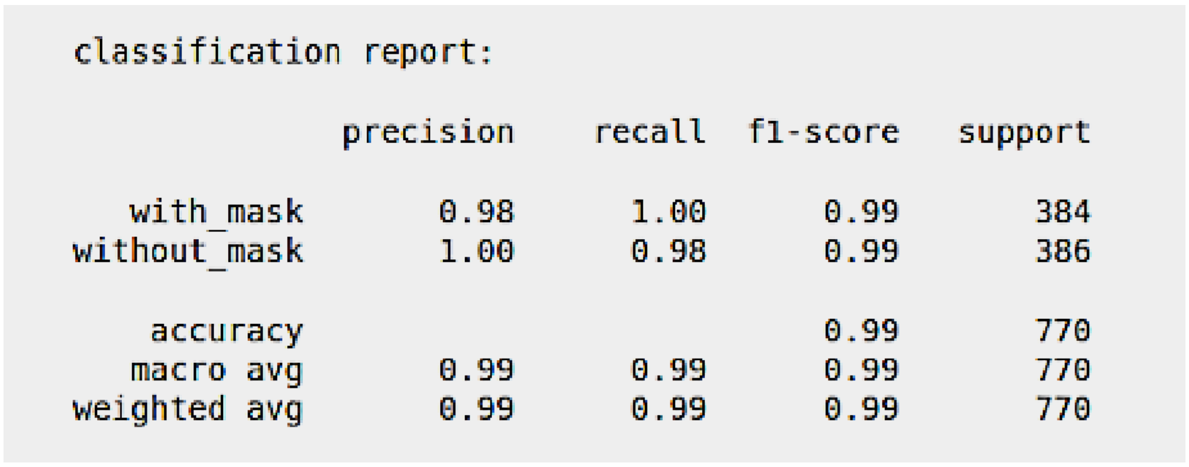
(Figure 4.1.2.1: Convolution Block of MobileNetV2)

In the discussion of the layer part, there are three layers for both blocks. First one is 1x1 convolution with ReLU6. Depth wise convolution is in the second layer and again in the third layer there is a 1x1 convolution but without any non-linearity.



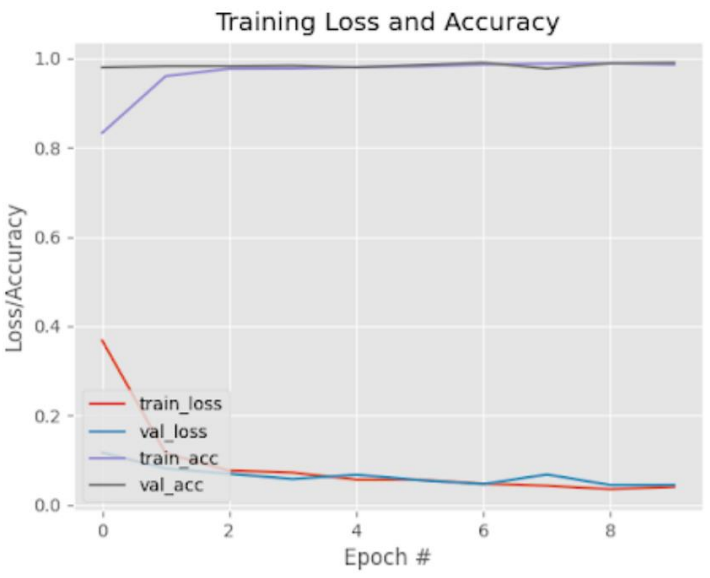
| Input | Operator | Output |
|---|---|---|
| $h \times w \times k$ | 1x1 conv2d , ReLU6 | $h \times w \times (tk)$ |
| $h \times w \times tk$ | 3x3 dwise s=$s$, ReLU6 | $\frac{h}{s} \times \frac{w}{s} \times (tk)$ |
| $\frac{h}{s} \times \frac{w}{s} \times tk$ | linear 1x1 conv2d | $\frac{h}{s} \times \frac{w}{s} \times k'$ |

(Figure 4.1.2.2: Bottleneck residual block transforming from k to k' channels, with stride s and factor t)



```
classification report:

              precision    recall  f1-score   support

   with_mask       0.98      1.00      0.99       384
without_mask       1.00      0.98      0.99       386

    accuracy                           0.99       770
   macro avg       0.99      0.99      0.99       770
weighted avg       0.99      0.99      0.99       770
```

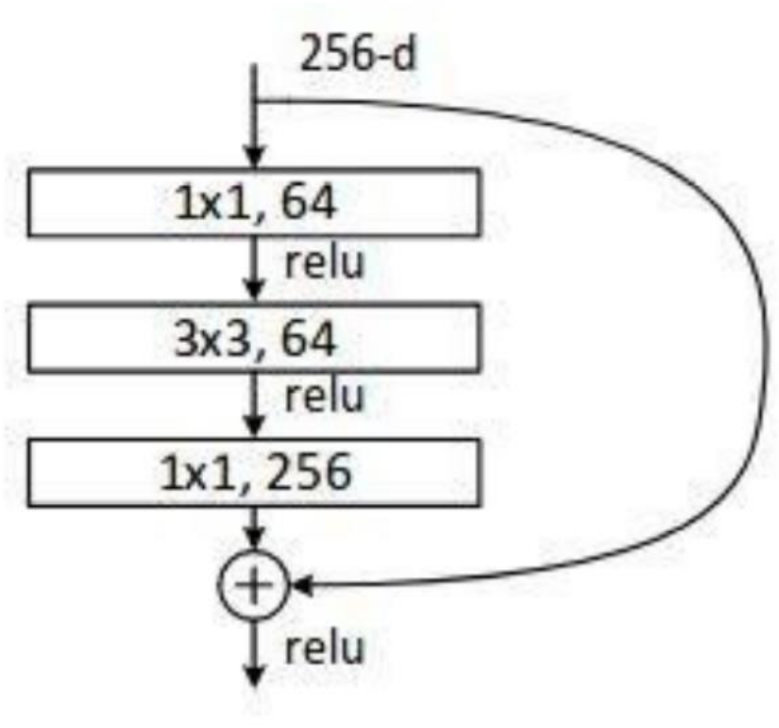
(Figure 4.1.2.3: Classification report of MobileNetV2)



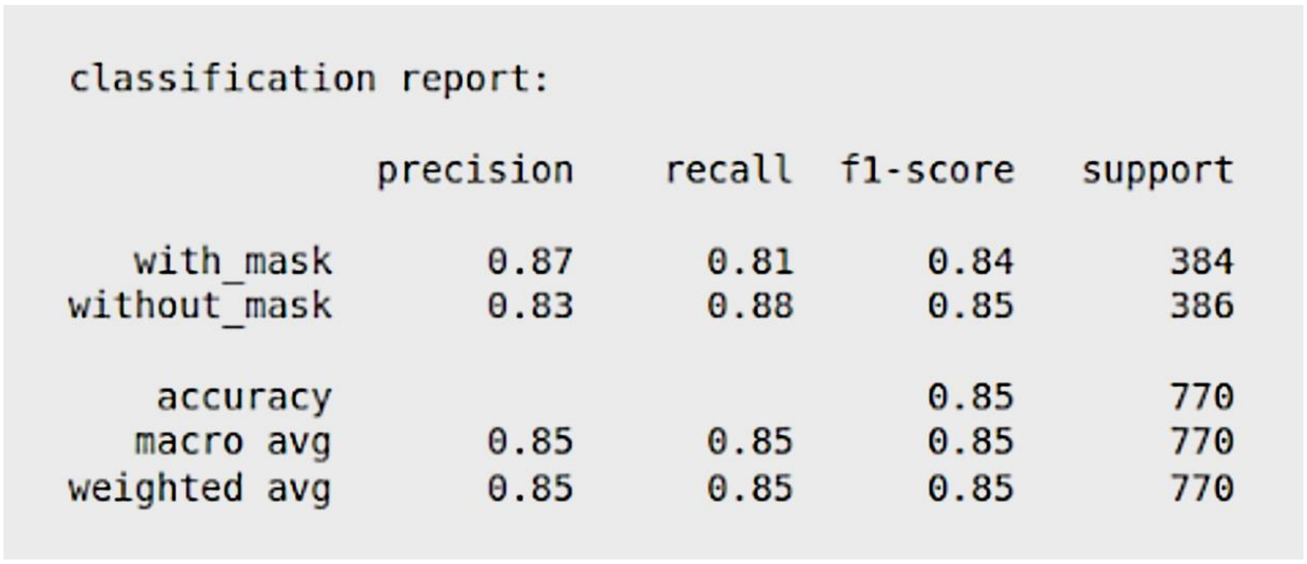
(Figure 4.1.2.3: Loss graph for MobileNetV2)

## 4.1.3 ResNet50

It is a convolutional neural network which is 3 layers deep. The bottleneck class implements this three-layer block. From the ImageNet database we can train millions of images and then load it as a pre-trained version of a network. This network can classify images into several object categories, such as face, car, bike and many animals.

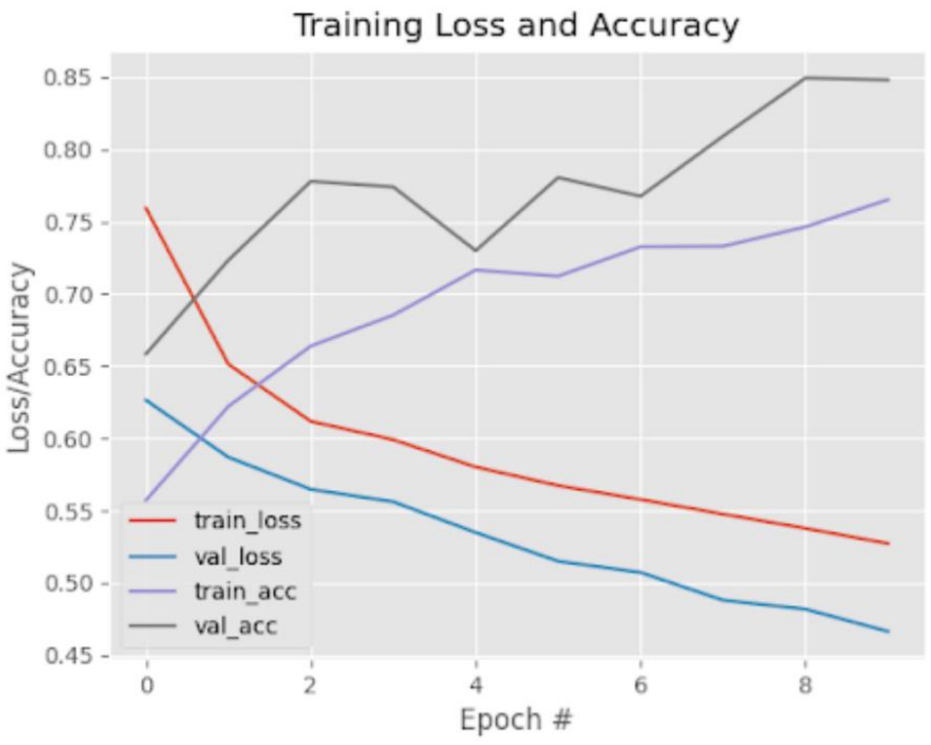


(Figure 4.1.3.1: ResNet50 3 Layer block)



```
classification report:
              precision    recall  f1-score   support

   with_mask       0.87      0.81      0.84       384
without_mask       0.83      0.88      0.85       386

    accuracy                           0.85       770
   macro avg       0.85      0.85      0.85       770
weighted avg       0.85      0.85      0.85       770
```

(Figure 4.1.3.2: Classification report of ResNet50)



(Figure 4.1.3.3: Loss graph for ResNet50)

After observing these results, we can conclude that the lowest performance amongst these was of ResNet50. Xception and MobileNetV2 both performed quite similar and to select the best from them was the 100% precision of unmasked faces by MobileNetV2. Since our detector can take no risk by falsely concluding that an unmasked face is a masked face, precision is the key metric to decide between these models. Since, the best performance was given by MobileNetV2, therefore, we decided to use MobileNetV2 as a base model for our face mask detector algorithm.

## 4.2 Inference

We implemented our model on images containing one and more faces. We also implemented it on videos and live video streams by removing and wearing masks one by one. Some screenshots of the results are shown here: Click Here

# Source Code & Repository

## 5.1 Source Code

### “Training Model”

```
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os

# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

DIRECTORY = r”E:\Minor_Project_2021\dataset”
CATEGORIES = [“with_mask”, “without_mask”]

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class images
print(“[INFO] loading images…”)

data = []
labels = []

for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
             img_path = os.path.join(path, img)
             image = load_img(img_path, target_size=(224, 224))
             image = img_to_array(image)
             image = preprocess_input(image)

             data.append(image)
             labels.append(category)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

data = np.array(data, dtype=”float32”)
labels = np.array(labels)

(trainX, testX, trainY, testY) = train_test_split(data, labels,
  test_size=0.20, stratify=labels, random_state=42)

# construct the training image generator for data augmentation
aug = ImageDataGenerator(
  rotation_range=20,
```

```python
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode=”nearest”)

# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights=”imagenet”, include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name=”flatten”)(headModel)
headModel = Dense(128, activation=”relu”)(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation=”softmax”)(headModel)

# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False

# compile our model
print(“[INFO] compiling model…”)
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss=”binary_crossentropy”, optimizer=opt,
    metrics=[“accuracy”])

# train the head of the network
print(“[INFO] training head…”)
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)

# make predictions on the testing set
print(“[INFO] evaluating network…”)
predIdxs = model.predict(testX, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
    target_names=lb.classes_))

# serialize the model to disk
print(“[INFO] saving mask detector model…”)
model.save(“mask_detector.model”, save_format=”h5”)

# plot the training loss and accuracy
N = EPOCHS
plt.style.use(“ggplot”)
plt.figure()
plt.plot(np.arange(0, N), H.history[“loss”], label=”train_loss”)
plt.plot(np.arange(0, N), H.history[“val_loss”], label=”val_loss”)
plt.plot(np.arange(0, N), H.history[“accuracy”], label=”train_acc”)
plt.plot(np.arange(0, N), H.history[“val_accuracy”], label=”val_acc”)
plt.title(“Training Loss and Accuracy”)
plt.xlabel(“Epoch #”)
plt.ylabel(“Loss/Accuracy”)
plt.legend(loc=”lower left”)
plt.savefig(“plot.png”)

```

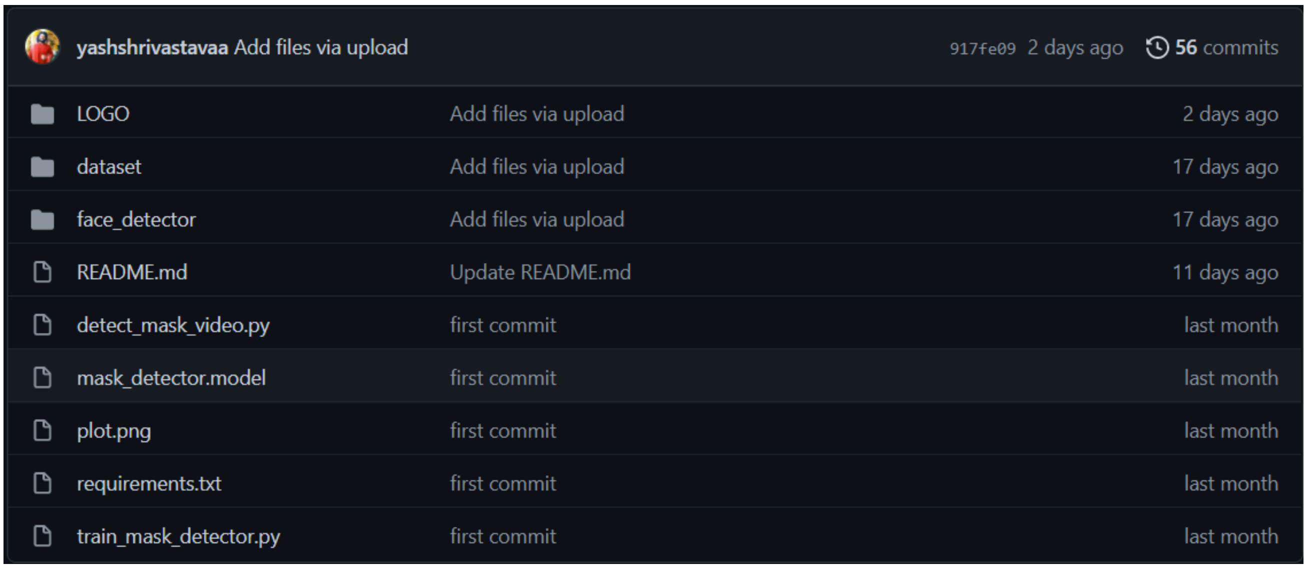
## “Mask Detect Video Stream”

```
# import the necessary packages
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
import os

def detect_and_predict_mask(frame, faceNet, maskNet):
  # grab the dimensions of the frame and then construct a blob
  # from it
  (h, w) = frame.shape[:2]
  blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
             (104.0, 177.0, 123.0))

  # pass the blob through the network and obtain the face detections
  faceNet.setInput(blob)
  detections = faceNet.forward()
  print(detections.shape)

  # initialize our list of faces, their corresponding locations,
  # and the list of predictions from our face mask network
  faces = []
  locs = []
  preds = []

  # loop over the detections
  for i in range(0, detections.shape[2]):
             # extract the confidence (i.e., probability) associated with
             # the detection
             confidence = detections[0, 0, i, 2]

             # filter out weak detections by ensuring the confidence is
             # greater than the minimum confidence
             if confidence > 0.5:
                        # compute the (x, y)-coordinates of the bounding box for
                        # the object
                        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                        (startX, startY, endX, endY) = box.astype(“int”)

                        # ensure the bounding boxes fall within the dimensions of
                        # the frame
                        (startX, startY) = (max(0, startX), max(0, startY))
                        (endX, endY) = (min(w – 1, endX), min(h – 1, endY))

                        # extract the face ROI, convert it from BGR to RGB channel
                        # ordering, resize it to 224x224, and preprocess it
                        face = frame[startY:endY, startX:endX]
                        face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                        face = cv2.resize(face, (224, 224))
                        face = img_to_array(face)
                        face = preprocess_input(face)

                        # add the face and bounding boxes to their respective
                        # lists
                        faces.append(face)
                        locs.append((startX, startY, endX, endY))

  # only make a predictions if at least one face was detected
  if len(faces) > 0:
             # for faster inference we’ll make batch predictions on *all*
             # faces at the same time rather than one-by-one predictions
             # in the above `for` loop
             faces = np.array(faces, dtype=”float32”)
             preds = maskNet.predict(faces, batch_size=32)

  # return a 2-tuple of the face locations and their corresponding
  # locations
  return (locs, preds)

```

```
# load our serialized face detector model from disk
prototxtPath = r”E:\Minor_Project_2021\face_detector\deploy.prototxt”
weightsPath = r”E:\Minor_Project_2021\face_detector\res10_300x300_ssd_iter_140000.caffemodel”
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
maskNet = load_model(“mask_detector.model”)

# initialize the video stream
print(“[INFO] starting video stream…”)
vs = VideoStream(src=0).start()

# loop over the frames from the video stream
while True:
  # grab the frame from the threaded video stream and resize it
  # to have a maximum width of 400 pixels
  frame = vs.read()
  frame = imutils.resize(frame, width=700)

  # detect faces in the frame and determine if they are wearing a
  # face mask or not
  (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

  # loop over the detected face locations and their corresponding
  # locations
  for (box, pred) in zip(locs, preds):
            # unpack the bounding box and predictions
            (startX, startY, endX, endY) = box
            (mask, withoutMask) = pred

            # determine the class label and color we’ll use to draw
            # the bounding box and text
            label = “Mask” if mask > withoutMask else “No Mask”
            color = (0, 255, 0) if label == “Mask” else (0, 0, 255)

            # include the probability in the label
            label = “{}: {:.2f}%”.format(label, max(mask, withoutMask) * 100)

            # display the label and bounding box rectangle on the output
            # frame
            cv2.putText(frame, label, (startX, startY – 10),
                      cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

  # show the output frame
  cv2.imshow(“Frame”, frame)
  key = cv2.waitKey(1) & 0xFF

  # if the `q` key was pressed, break from the loop
  if key == ord(“q”):
            break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()

```

## 5.2 GitHub Repository

GitHub Link: [https://github.com/yashshrivastavaa/Mask-Detection-Security.git](https://github.com/yashshrivastavaa/Mask-Detection-Security.git)



**yashshrivastavaa** Add files via upload 917fe09 2 days ago **56** commits

| | | |
|---|---|---|
| LOGO | Add files via upload | 2 days ago |
| dataset | Add files via upload | 17 days ago |
| face_detector | Add files via upload | 17 days ago |
| README.md | Update README.md | 11 days ago |
| detect_mask_video.py | first commit | last month |
| mask_detector.model | first commit | last month |
| plot.png | first commit | last month |
| requirements.txt | first commit | last month |
| train_mask_detector.py | first commit | last month |

### 5.2.1 Setup & Installation

- Dataset

The dataset used can be downloaded from here – Click to Download

This dataset consists of 3339 images belonging to two class:

1. with_mask – 1755 images
2. without_mask – 1584 images

## • Prerequisites

All the dependencies and required libraries are included in the file

requirements.txt Click here to see Or Check Dependencies

## • Installation

1. Clone this repository

```
$ git clone https://github.com/yashshrivastavaa/Mask-Detection-Security.git
```

2. Change your directory to clone repo

```
$ cd Face-Mask-Detection
```

3. Create a Python virtual environment named 'test' and activate it

```
$ virtualenv test
$ source test/bin/activate
```

4. Now, run the following command in your Terminal/ Command Prompt to install the libraries required

```
$ pip3 install -r requirements.txt
```

## • Working

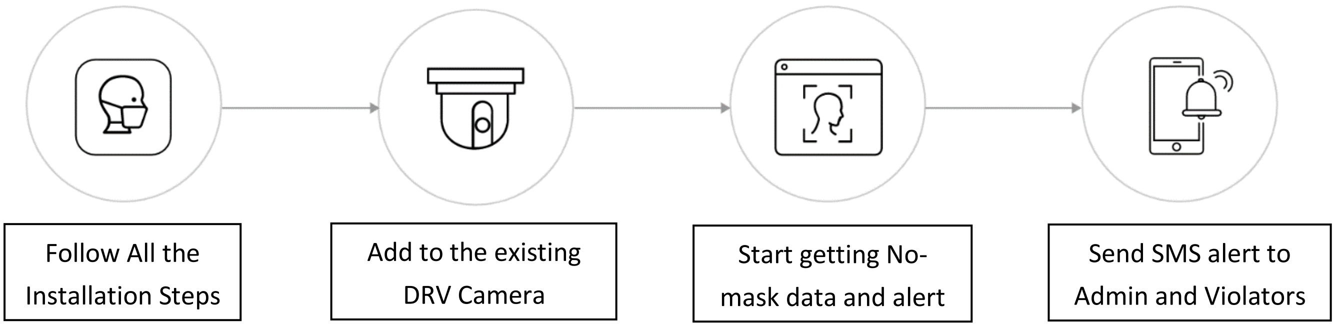
1. Open Terminal. Go into the cloned project directory and type the following command.

```
$ python3 train_mask_detector.py --dataset dataset
```

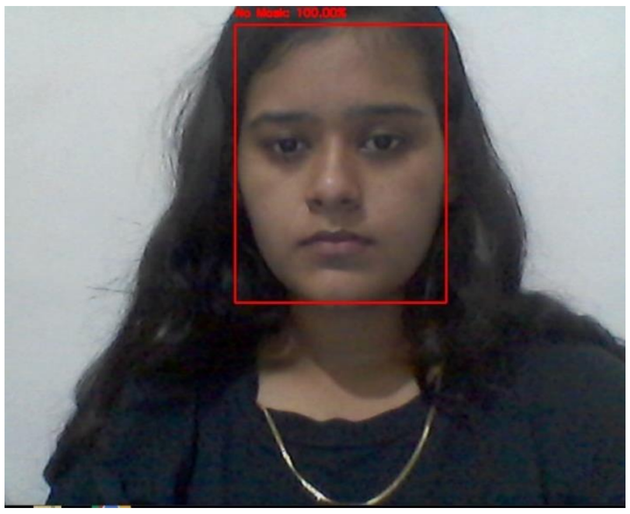
2. To detect face mask in real time video stream type the following command:

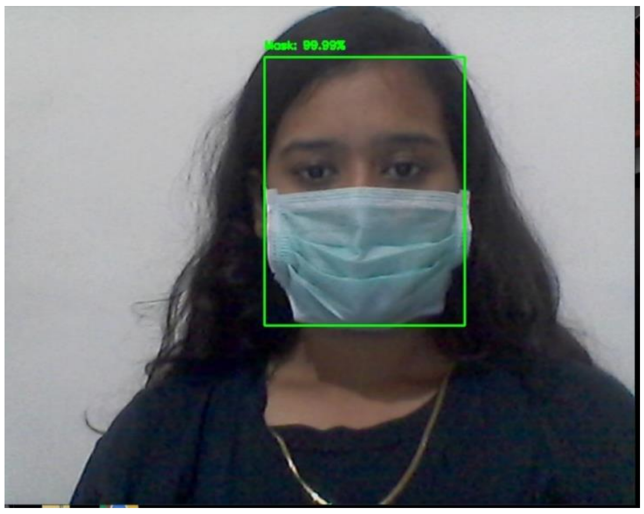
```
$ python3 detect_mask_video.py
```

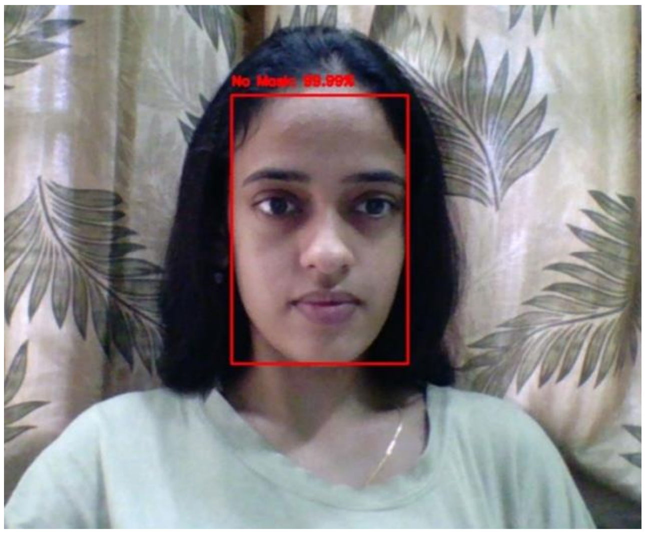
3. Model will be working properly.
4. Basic Working is shown in given Figure 5.2.1.1

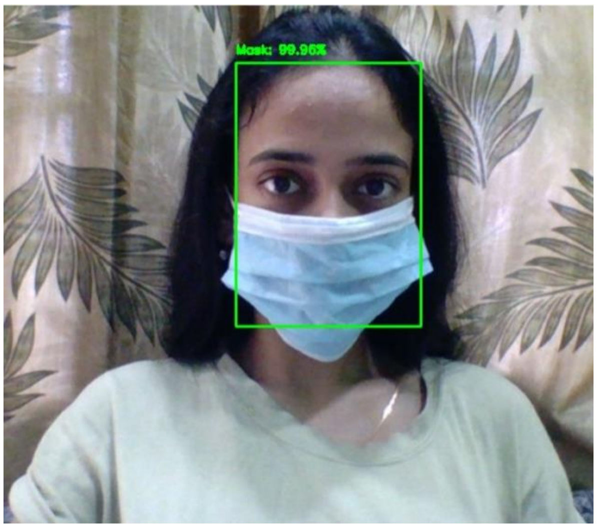

Follow All the Installation Steps
Add to the existing DRV Camera
Start getting No-mask data and alert
Send SMS alert to Admin and Violators

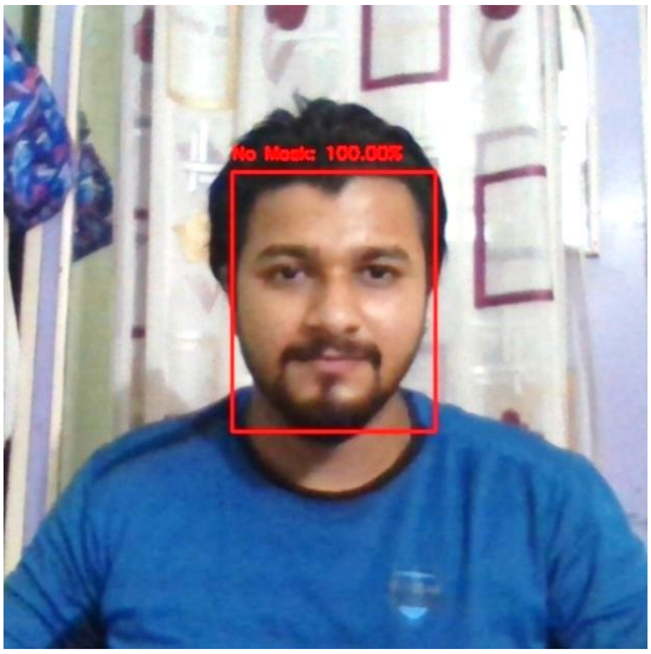
## 5.3 Sample Output

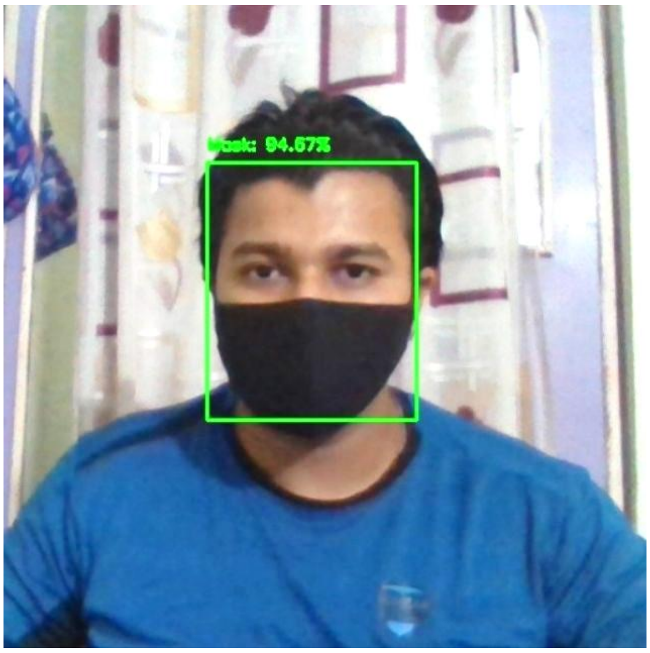





No Mask: 99.99%


Mask: 99.96%


No Mask: 100.00%

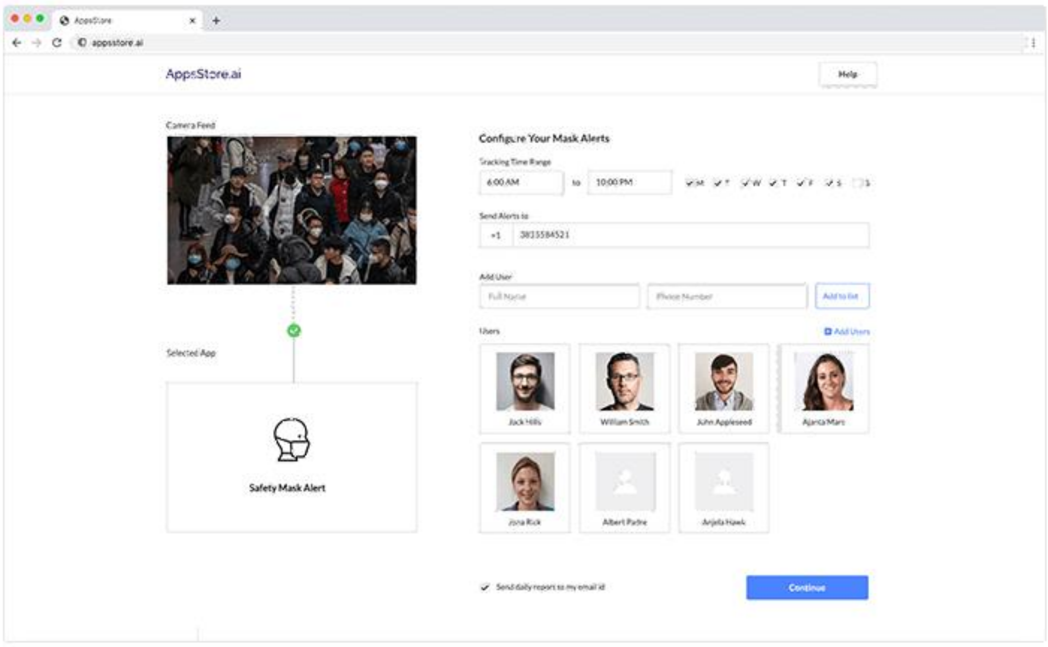

Mask: 94.67%

# Future Software Interface

- **LOGO**

- **Interface to Add Face Data**

Face Mask Detection Platform uses Artificial Network to recognize whether a user is wearing a mask or not. The app can be connected to any existing or new IP mask detection cameras to detect people without a mask. App users can also add faces and phone numbers to send them an alert in case they are not wearing a mask. If the camera captures an unrecognized face, a notification can be sent out to the administrator.



Configure Your Mask Alerts

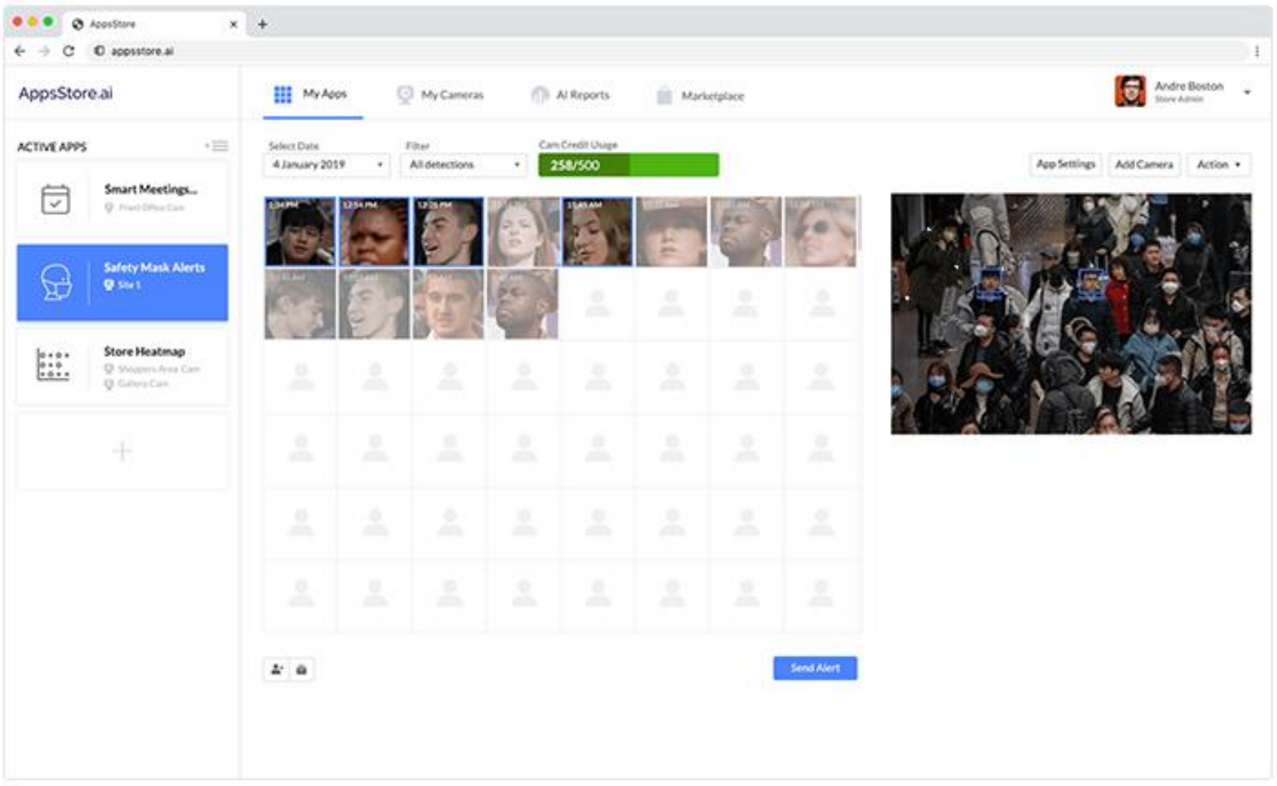
Tracking Time Range: 6:00 AM to 10:00 PM ✓M ✓T ✓W ✓T ✓F ✓S ☐S

Send Alerts to: +1 3835584521

✓ Send daily report to my email id

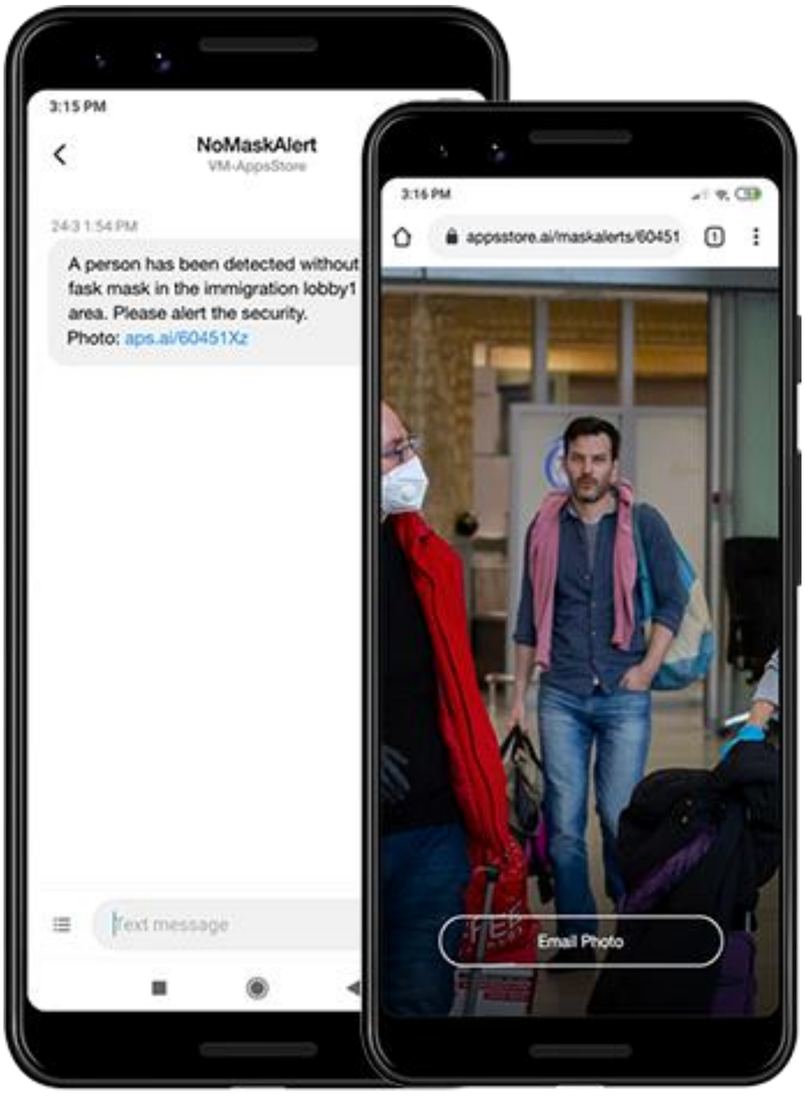
## • Real-Time Monitoring Dashboard

A user-friendly website allows the user to see who was not wearing a mask and see the photo or the video captured by the camera. User can also generate reports to download and integrate with any other third-party integration.



## • Notification Interface

If the face mask detector application identifies a user that he/she was not wearing a mask, AI alerts are sent with the picture of the person. It allows the application to run automatically and enforces the wearing of the mask.



3:15 PM

NoMaskAlert
VM-AppsStore

24-3 1:54 PM

A person has been detected without fask mask in the immigration lobby1 area. Please alert the security.
Photo: aps.ai/60451Xz

## Use Case

- **Example:**

### Airports

The Face Mask Detection System can be used at airports to detect travellers without masks. Face data of travellers can be

captured in the system at the entrance. If a traveller is found to be without a face mask, their picture is sent to the airport authorities so that they could take quick action. If the person's face is already stored, like the face of an Airport worker, it can send the alert to the worker's phone directly.

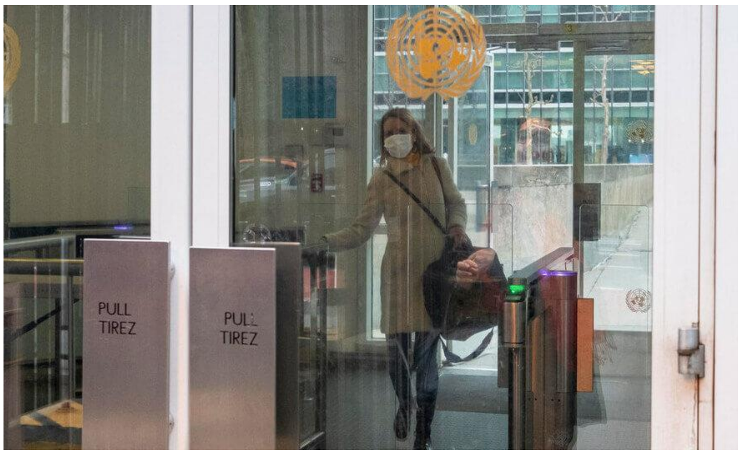
## Hospitals

Using Face Mask Detection System, Hospitals can monitor if their

staff is wearing masks during their shift or not. If any health worker is found without a mask, they will receive a notification with a reminder to wear a mask. Also, if quarantine people who are required to wear a mask, the system can keep an eye and detect if the mask is present or not and send notification automatically or report to the authorities.

## Offices

The Face Mask Detection System can be used at office premises to



detect if employees are maintaining safety standards at work. It monitors employees without masks and sends them a reminder to wear a mask. The reports can be downloaded or sent an email at the end of the day to capture people who are not complying with the regulations or the requirements.

## Limitations

There were not many challenges faced but the two problems that were time consuming and made the tasks tedious are discussed as follows. One was the excessive data loading time in Google Colab Notebook while loading the dataset into it. Since the runtime restarting refreshes all the cells, the cell for dataset loading took most of the time while running. Secondly, the access problem in Google Colab Notebook: it did not allow the access of webcam which posed a hurdle in testing images and live video stream through Google Colab Notebook. Therefore, we had to run the code locally on the computer through which we tested the code on the live video stream.

# Future Scope

More than fifty countries around the world have recently initiated wearing face masks compulsory. People have to cover their faces in public, supermarkets, public transports, offices, and stores. Retail companies often use software to count the number of people entering their stores. They may also like to measure impressions on digital displays and promotional screens. We are planning to improve our Face Mask Detection tool and release it as an open-source project. Our software can be equated to any existing USB, IP cameras, and CCTV cameras to detect people without a mask. This detection live video feed can be implemented in web and desktop applications so that the operator can see notice messages. Software operators can also get an image in case someone is not wearing a mask. Furthermore, an alarm system can also be implemented to sound a beep when someone without a mask enters the area. This software can also be connected to the entrance gates and only people wearing face masks can come in.

# Conclusion

To mitigate the spread of COVID-19 pandemic, measures must be taken. We have modelled a face mask detector using SSD architecture and transfer learning methods in neural networks. To train, validate and test the model, we used the dataset that consisted of 1916 masked faces images and 1919 unmasked faces images. These images were taken from various resources like Kaggle and RMFD datasets. The model was inferred on images and live video streams. To select a base model, we evaluated the metrics like accuracy, precision and recall and selected MobileNetV2 architecture with the best performance having 100% precision and 99% recall. It is also computationally efficient using MobileNetV2 which makes it easier to install the model to embedded systems. This face mask detector can be deployed in many areas like shopping malls, airports and other heavy traffic places to monitor the public and to avoid the spread of the disease by checking who is following basic rules and who is not.

# REFERENCES

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001, vol. 1. IEEE, 2001, pp. I–I.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587.

[3] R. Girshick, "Fast r-CNN," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-CNN: Towards real-time object detection with region proposal networks," in Advances in neural information processing systems, 2015, pp. 91–99.
[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in European conference on computer vision. Springer, 2016, pp. 21–37.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.

[7] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," 2017.

[8] Haddad, J., 2020. How I Built A Face Mask Detector For COVID-19 Using Pytorch Lightning. [online] Medium. Available at: https://towardsdatascience.com/how-i-built-a-face-mask-detector-for-covid-19-usingpytorch-lightning-67eb3752fd61

[9] Rosebrock, A., 2020. COVID-19: Face Mask Detector With OpenCV, Keras/TensorFlow, And Deep Learning - Pyimagesearch. [online] PyImageSearch. Available at: https://www.pyimagesearch.com/2020/05/04/covid19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/ .

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in European conference on computer vision. Springer, 2016, pp. 21–37.

[11] Francois Chollet "Xception: Deep Learning with Depthwise Separable Convolutions" in Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), 2017, pp. 1251-1258.

[12] M. Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), 2018. [13] Keras code inspiration taken from https://github.com/chandrikadeb7/Face-Mask-Detection

- **Project GitHub Repository**: Click Here to View

  Or Scan: (Note: Refer installation guide)

- **Presentation Link:** Click Here to View

Or Scan:

- **Feedback & Suggestion:** Click Here

  **Or Scan:**

******************