Digital Image Processing

PROJECT REPORT ON

FACE MASK DETECTION

Submitted by

GROUP 13

Sellamuthu Abishek - AP19110010087
Bobba Nikhila - AP19110010074
Manikanta Challa - AP19110010066
Hemasri Dadi - AP19110010026
Devika Jayavarapu - AP19110010027
Gauthami K - AP19110010023

for the course
CSE 314 – Digital Image Processing
Guided by
Dr. Sonam Mourya

ABSTRACT

The coronavirus COVID-19 pandemic is causing a global health crisis, so the effective protection method is wearing a face mask in public areas, according to the World Health Organization (WHO). The COVID-19 pandemic forced governments across the world to impose lockdown to prevent virus transmissions. Reports indicate that wearing facemasks while at work reduces the risk of transmission—an efficient and economical approach of using AI to create a safe environment in a manufacturing setup.

A hybrid model using deep and classical machine learning for face mask detection will be presented. A face mask detection dataset consists of with mask and without mask images. We are going to use OpenCV to do real-time facedetection from a live stream via our webcam. The proposed Retina Face Mask is a one-stage detector that consists of a feature pyramid network to fuse high-level semantic information with multiple feature maps and a novel context attention module to detect face masks. We also propose a novel cross-class object removal algorithm to reject predictions with low confidences and the union's high intersection.

Experiment results show that Retina Face Mask achieves state-of-the- art results on a public face mask dataset with 2.3% and 1.5% higher than the baselineresult in the face and mask detection precision, respectively, and 11.0% and 5.9% higher than baseline for recall. Besides, we also explore implementing Retina Face Mask with a light-weighted neural network Mobile Net for embedded ormobile devices. We will use the dataset to build a COVID-19 face mask detector withcomputer vision using Python, OpenCV, Tensor Flow, and Keras. Our goal is to identify whether the person on the image/video stream is wearing a face mask or not with computer vision and deep learning.

TABLE OF CONTENTS

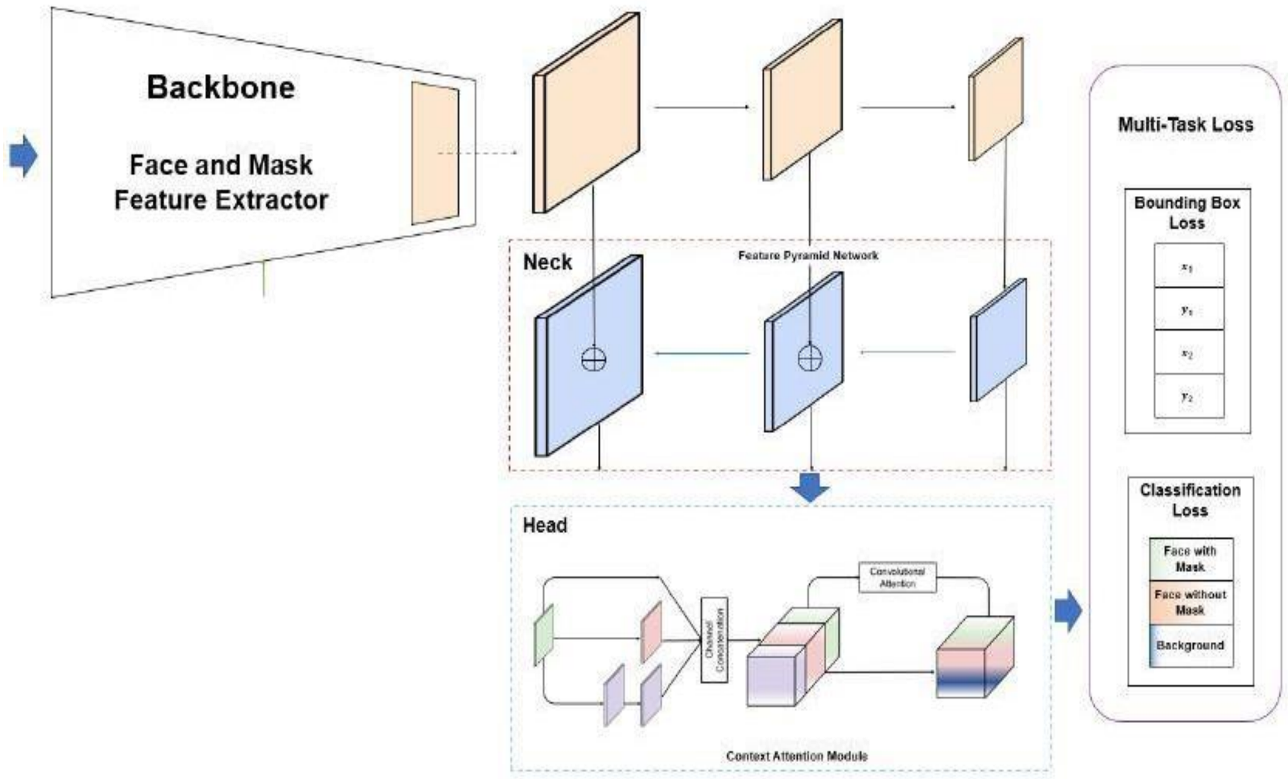
CHAPTER 1

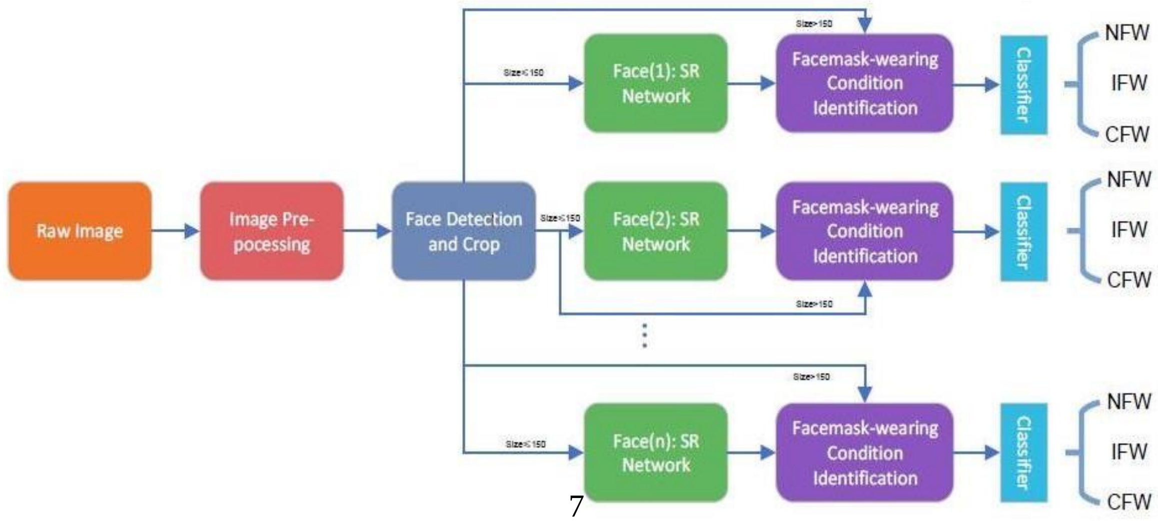
1.INTRODUCTION

The mask detection model can be said to be a combination of classification and face detection model. We use transfer learning with a model trained on the dataset witha modified final fully connected layer for classification. While using the face detection model, several different approaches were tried upon based on existing literature. We delve into our final mask detection model built using a combination of the classification and face detection models.

Face mask detection refers to detect whether a person wearing a mask or not and what is the location of the face. The problem is closely related to general object detection to detect the classes of objects, and face detection is to detect a particular class of objects, i.e. face. Applications of object and face detection can be found in many areas.

Traditional object detectors are usually based on handcrafted feature extractors. Face Mask Detection uses the integral image method while other works adopt different feature extractors, such as histogram of oriented gradients (HOG). Recently, deep learning-based object detectors demonstrated excellent performance and dominate the development of modern object detectors. Without using prior knowledge for forming feature extractors, deep learning allows neural networks to learn features in an end-to-end manner. There are one-stage and two-stage deep learning-based object detectors. One-stage detectors use a single neuralnetwork to detect objects, such as a single-shot detector (SSD). In contrast, two- stage sensors utilize two networks to perform a coarse-to-fine detection, such as a region-based convolutional neural network (R-CNN) and Faster R-CNN. Similarly, face detection adopts similar architecture as general object detector, but adds more face related features, such as facial landmarks in RetinaFace, to improve face detection accuracy.





1.1 Objectives

Traditional object detection uses a multi-step process. A well-known Face Mask Detection, which can achieve real-time detection. The algorithm extracts feature by Haar feature descriptor with an integral image method, selects valuable features, and detects objects through a cascaded detector. Although it utilizes integral image to facilitate the algorithm, In for human detection, a practical feature extractor called HOG is proposed, which computes the directions and magnitudes of orientedgradients over image cells. Later on, a deformable part-based model (DPM) detectsobjects parts Rather than using handcrafted features, the deep learning-based detector demonstrated excellent performance recently due to its robustness and high feature extraction capability. There are two popular categories, one-stage object detectors and two-stage object detectors. The two-stage detector generates regionproposals in the first stage and then fine-tune them in the second stage. The two- stage detector can provide high detection performance but with low speed. R-CNNuses selective search to propose some candidate regions which may contain objects.After that, the proposals are fed into a CNN model to extract features, and a supportvector machine (SVM) is used to recognize classes of objects. The network has to detect proposals in a one-by-one manner and use a separate SVM for final classification. Fast R-CNN solves this problem by introducing a region of interest (ROI) pooling layer to input all proposal regions at once. Finally, a region proposal network (RPN) is proposed in faster R-CNN to take selective search, limiting the speed of such detectors.

1.2 Background and Literature Survey

Various researchers and analysts mainly focused on grey-scale face image (Ojala, Pietikainen, & Maenpaa, 2002). While some were completely built on pattern identification models, possessing initial information of the face model while others were using AdaBoost (Kim, Park, Woo, Jeong, & Min, 2011), which was an excellent classifier for training purposes. Then came the Viola-Jones Detector, which provided a breakthrough in face detection technology, and real-time face detection got possible. It faced various problems like the orientation and brightness of the face, making it hard to intercept. So basically, it failed to work in dull and dim light. Thus, researchers started searching for a new alternative model that could easily detect faces and masks on the face.

Initially researchers focused on edge and gray value of face image was based on pattern recognition model, having a prior information of the face model. Adaboost was a good training classifier. The face detection technology got a breakthrough with the famous Viola Jones Detector, which greatly improved real time face detection. Viola Jones detector optimized the features of Haar, but failed to tackle the real world problems and was influenced by various factors like face brightness and face orientation. Viola Jones could only detect frontal well lit faces. It failed to work well in dark conditions and with non-frontal images. These issues have made the independent researchers work on developing new face detection models basedon deep learning, to have better results for the different facial conditions. We havedeveloped our face detection model using Multi Human Parsing Dataset, based on fully convolutional networks, such that it can detect the face in any geometric condition frontal or non-frontal for that matter. Convolutional Networks havealways been used for image classification tasks.

1.3 Organization of the Report

The remaining chapters of the project report are described as follows:

- **Chapter 2** contains the proposed system, methodology, hardware and software details.
- **Chapter 3** gives the cost involved in the implementation of the project.
- **Chapter 4** discusses the results obtained after the project was implemented.
- **Chapter 5** concludes the report.
- **Chapter 6** consists of codes.
- **Chapter 7** gives references.

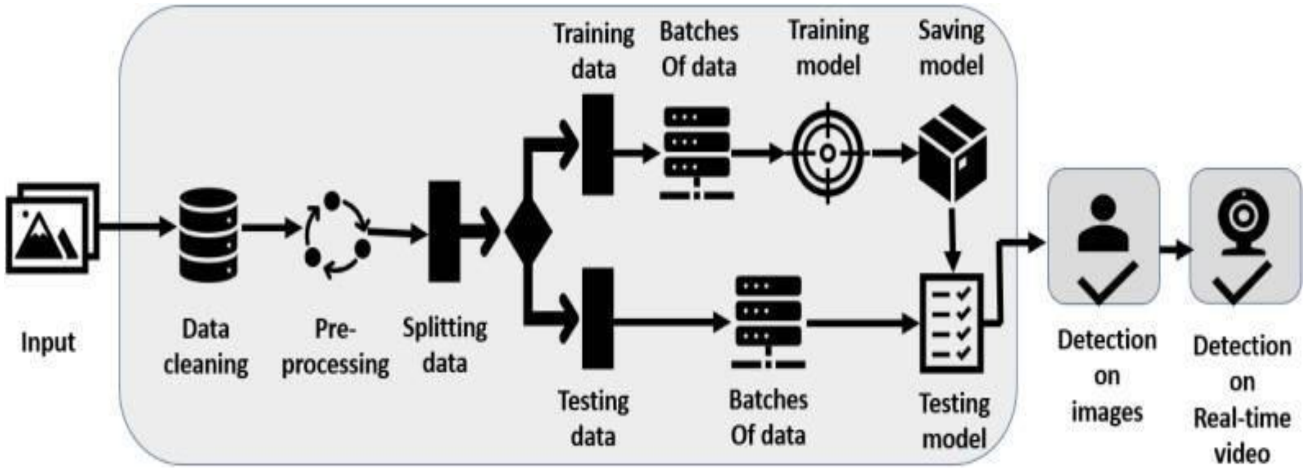
CHAPTER 2

FACE MASK DETECTION

This Chapter describes the proposed system, working methodology, software, and hardware details.

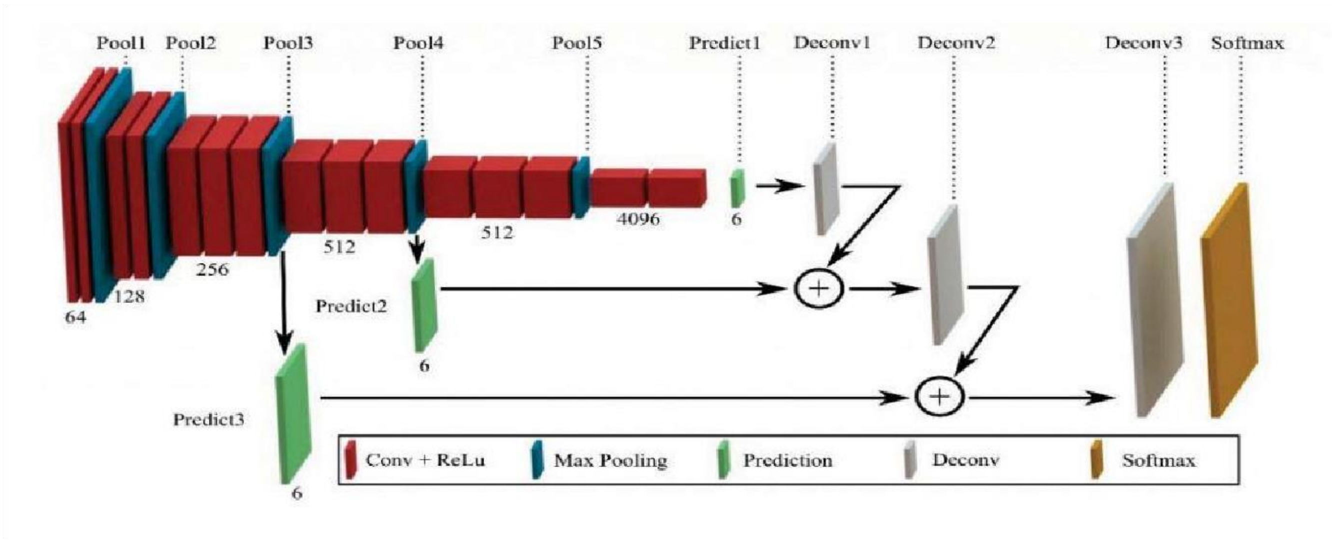
2.1 Proposed System

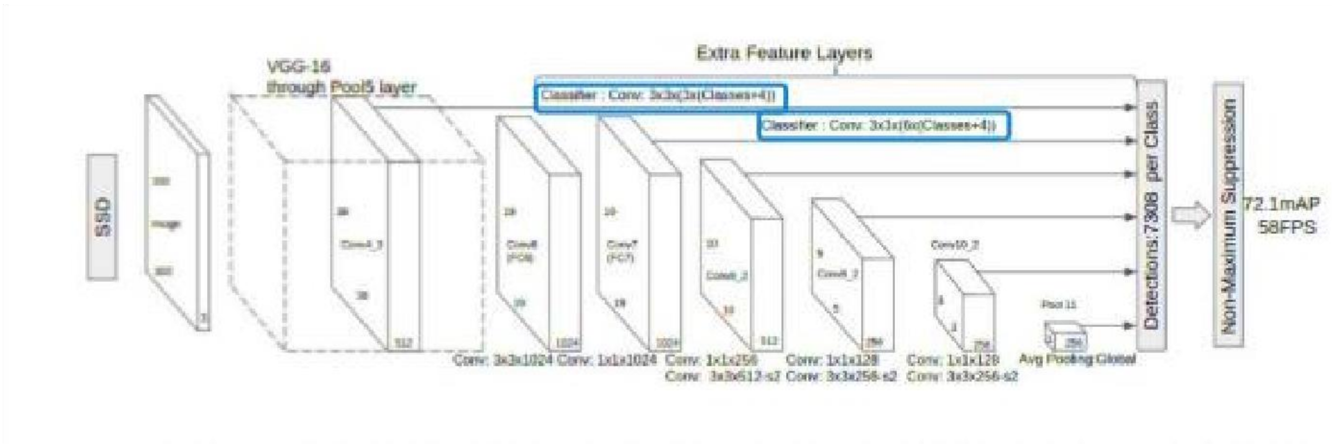
To predict whether a person has worn a mask correctly, the initial stage would be to train the model using a proper dataset. After training the classifier, an accurate face detection model is required to detect faces so that the SSDMNV2 model can classify whether the person is wearing a mask or not. The task in this paper is to raise the accuracy of mask detection without being too resource-heavy. For doing this task, the DNN module was used from OpenCV, which contains a 'Single Shot Multibox Detector (SSD)

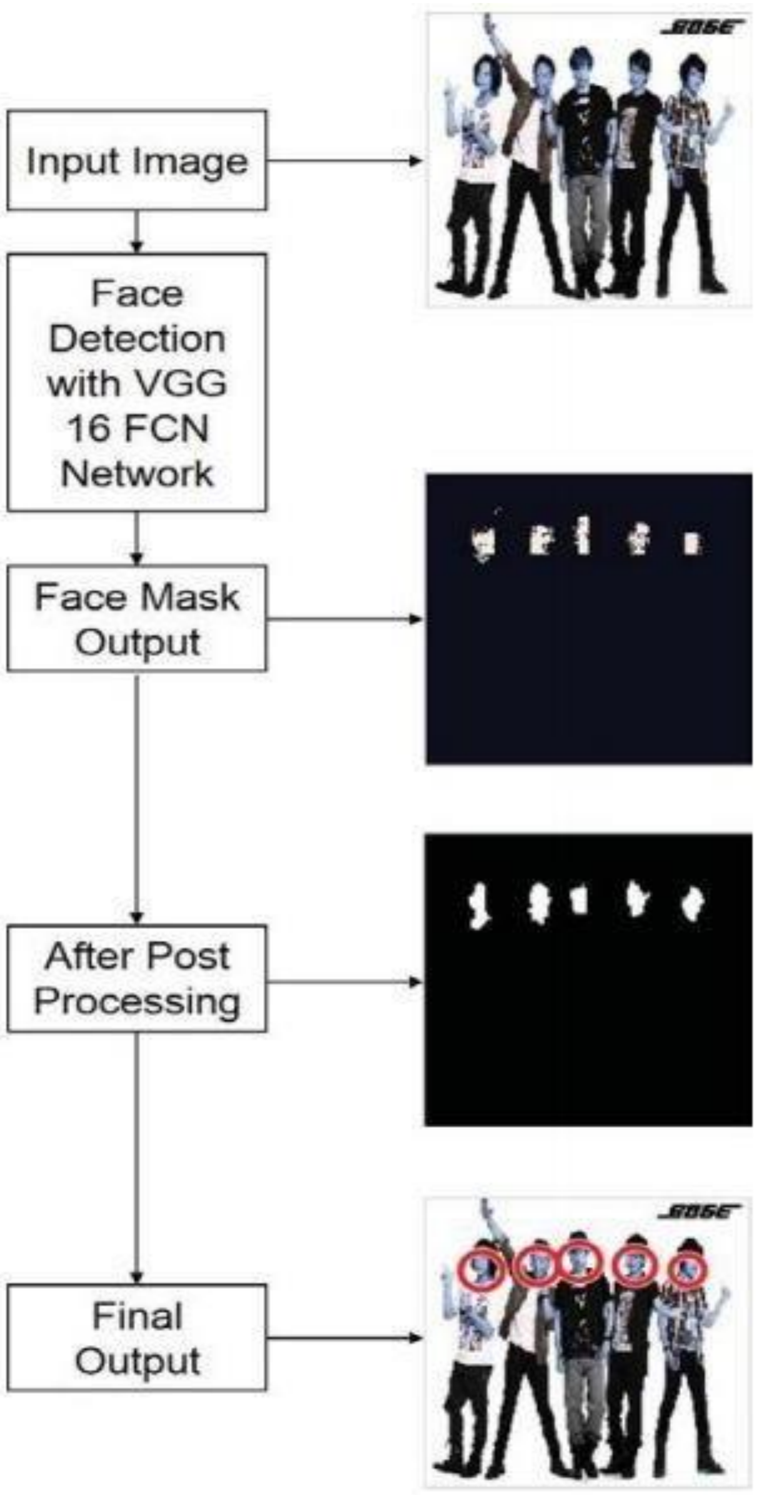


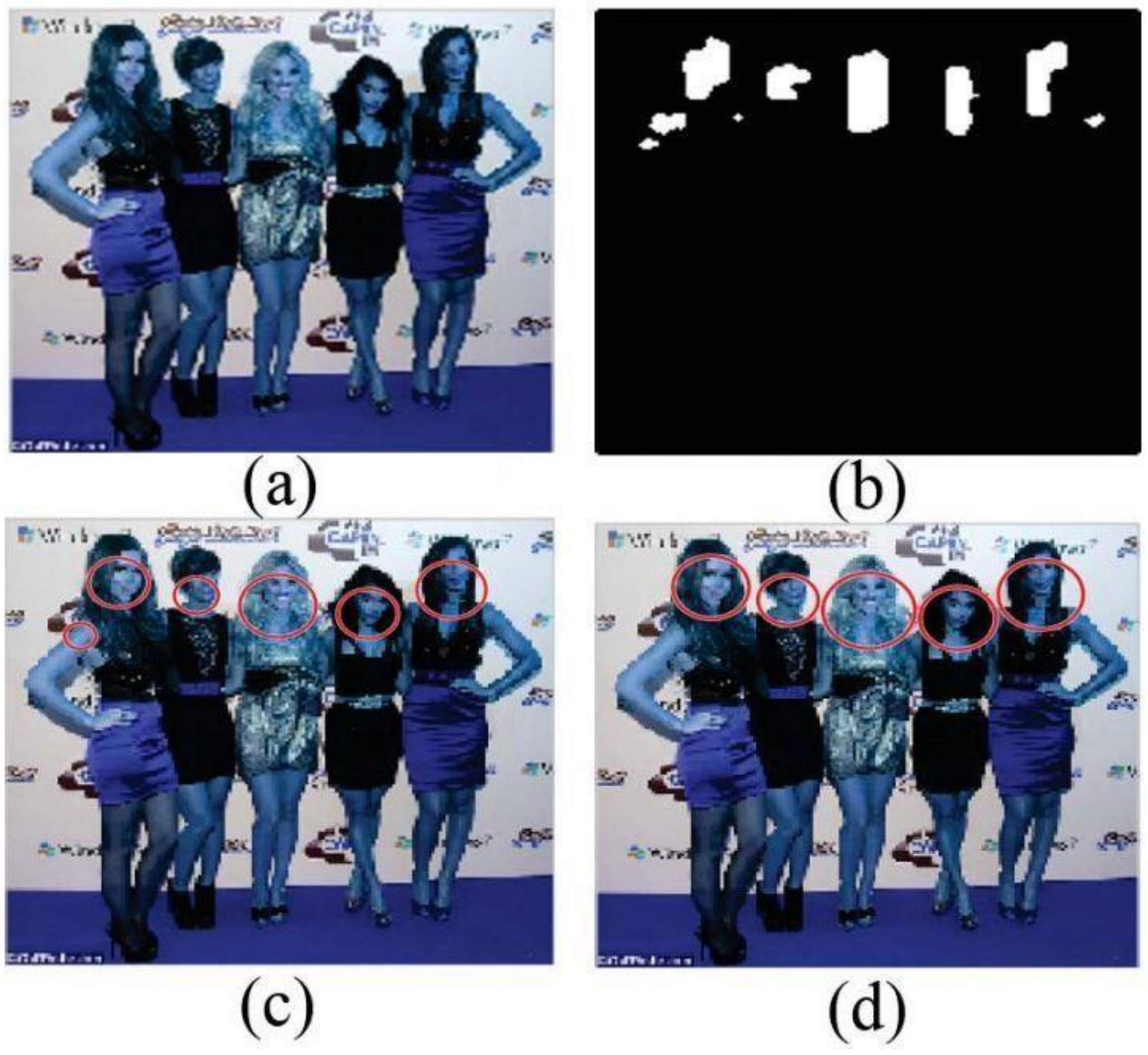
2.2 Working Methodology

We have developed our face detection model based on fully convolutional networks, such that it can detect the face in any geometric condition frontal or now We propose a method of obtaining segmentation masks directly from the images containing one or more faces in different orientation. The input image of any arbitrary size is resized to 224 × 224 × 3 and fed to the FCN network for feature extraction and prediction. The output of the network is then subjected to post processing. Initially the pixel values of the face and background are subjected to global thresholding. After that its passed through median filter to remove the high frequency noise and then subjected to Closing operation to fill the gaps in the segmented area. After this bounding box is drawn around the segmented area. n-frontal for that matter. Convolutional Networks have always been used for image classification tasks.






Input Image
Face Detection with VGG 16 FCN Network
Face Mask Output
After Post Processing
Final Output
BOSE



(a) Actual Image

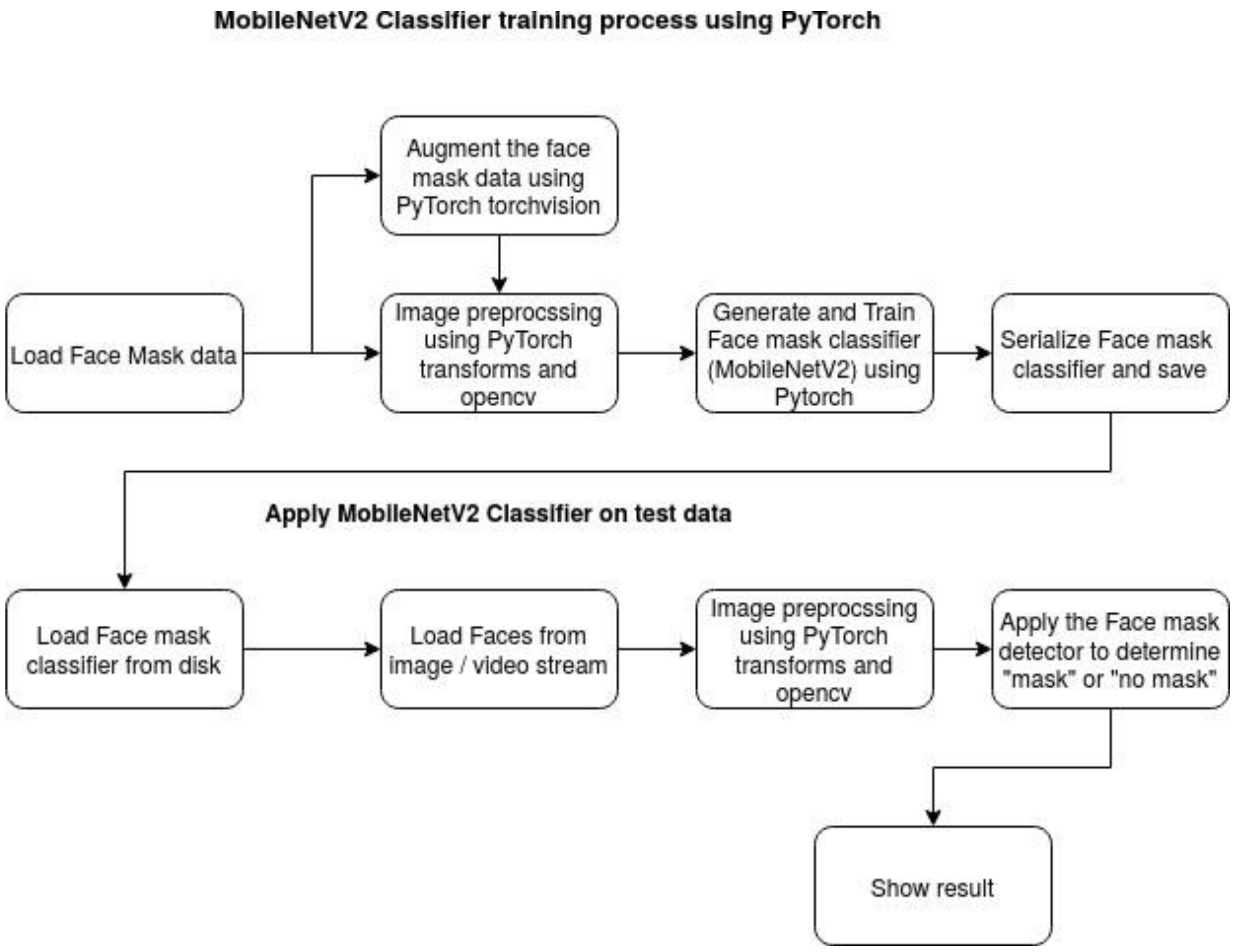
(b) Erroneous Prediction

(c) False Face Detection

(d) Correct Face Detection.

2.3 Work Flow

1.Train Deep learning model
(MobileNetV2)2.Apply mask detector over
images


MobileNetV2 Classifier training process using PyTorch
Load Face Mask data
Augment the face mask data using PyTorch torchvision
Image preprocssing using PyTorch transforms and opencv
Generate and Train Face mask classifier (MobileNetV2) using Pytorch
Serialize Face mask classifier and save
Apply MobileNetV2 Classifier on test data
Load Face mask classifier from disk
Load Faces from image / video stream
Image preprocssing using PyTorch transforms and opencv
Apply the Face mask detector to determine "mask" or "no mask"
Show result

The images were augmented by OpenCV. The set of images were already labeled "mask" and "no mask". The images that were present were of different sizes and resolutions, probably extracted from different sources or from machines (cameras)of different resolutions.

2.4 Data Preprocessing

Preprocessing steps was applied to all the raw input images to convert them into clean versions, which could be fed to a neural network machine learning model.

1.Resizing the input image (256 x 256)
2.Applying the color filtering (RGB) over the channels (Our model MobileNetV2supports 2D 3 channel image)
3.Scaling / Normalizing images using the standard mean of PyTorch build in weights
4.Center cropping the image with the pixel value of 224x224x3
5.Finally Converting them into tensors (Similar to NumPy array)

2.5 Deep Learning Frameworks

To implement this deep learning network, we have the following options.
1.TensorFlow
2.Keras
3.Pytorch

We are using the PyTorch because it runs on Python, which means that anyone witha basic understanding of Python can get started on building their deep learning models and it has the following advantage compared with TensorFlow

1. Data Parallelism
2. It looks like a Framework.

- In PyTorch, I have used the following module to develop the algorithm,

- **PyTorch Data Loader** – which is used to load the data from the Image Folder
- **PyTorch DataSets Image Folder** – which is used to locate the image sources and have a predefined module to label the target variable.
- **Pytorch Transforms** – helped to apply the preprocessing steps over thesource image while reading from the source folder.
- **PyTorch Device** – identifies the running system capabilities like CPU or GPU power to train the model. It will help us to switch the system usage.
- **Pytorch TorchVision** – it will help us to load the libraries which are created before. Like pre-trained models, image sources and so on. It is one of core in PyTorch
- **PyTorch nn** – it is one of the core modules. This module helps us to build our own Deep Neural Network (DNN) models. It has all the libraries needed to build the model. Like Linear layer, Convolution layer with 1D, conv2d, conv3d, sequence, Cross Entropy Loss (loss function), SoftMax, ReLu and so on.
- **PyTorch Optim** – help us to define the model optimizer. it will help the model to learn the data well. For example, Adam, SDG and so on.
- **Pytorch PIL** – helps to load the image from the source.
- **PyTorch AutoGrad** – another important module, it provides automatic differentiation for all operations on Tensors. For example, a single line of code. backward() to calculate the gradients automatically. This is very useful while implementing the backpropagation in DNN.

Image Classification Algorithm from PyTorch

CNN (Convolutional Neural Network) has many versions of pre-trained and well-architected networks for example Alex Net, ResNet, Inception, LeNet, Mobile Net and so on. In our case I have chosen the MobileNetV2 due to its lightweight and very efficient mobile-oriented model.

MobileNetV2

MobileNetV2 builds upon the ideas from MobileNetV1 using depth wise separable convolution as efficient building blocks. However, V2 introduces two new features to the architecture: 1) linear bottlenecks between the layers, and 2) shortcut connections between the bottlenecks

2.6. SOFTWARE DETAILS

- Anaconda Navigator
- Jupyter Notebook
- Google Colab
- Python (Keras, TensorFlow)

The project is developed using jupyter notebooks in anaconda navigator.
Various python libraries have been installed, and various model architectures have been imported from these libraries.

2.7 HARDWARE DETAILS

- At least 2.2GHz i3 Processor
- 2GB of RAM
- 10GB Hard drive
- Web Camera
- Internet Connectivity

2.8 Use Cases of Face Mask Detection System

The system can be used in the following places to identify people with or without masks:

- Offices – Manufacturers, retail, other SMEs, and corporate giants
- Hospitals/healthcare organizations
- Airports and railway stations
- Sports venues
- Entertainment and hospitality industry
- Densely populated areas
- Analyzing the current scenario, government and private organizations want to make sure that everyone working or visiting a public or private place is wearing masks throughout the day. The face mask detection platform can quickly identify the person with a mask, using cameras and analytics. Depending upon the requirements, the system is also adaptable to the latest technology and tools i.e.; you can add contact numbers or email addresses in the system to send an alert to the one who has not worn the mask. You can also send an alert to the person whose face is not recognizable in the system.

2.9 Features of the Face Mask Detection System

- The system is easy to implement in any existing organizational system.
- Custom alerts can be sent to the person with or without a face mask or the one whose face is unrecognizable in the admin system.
- No need to install any hardware as the system can relate to yourexisting surveillance system only.
- The system can be used easily with any camera or hardware like surveillance cameras.
- The system restricts access for those not wearing the masks and notifies the authorities.
- You can customize the face mask detection system based on your business requirements.
- You can check the analytics based on the system generated reports.
- Easy to access and control the movements from any device through face mask detection applications.
- Partially occluded faces either with mask or hair or hand, can be easily detected.

2.10 Attributes

Detecting masked faces is challenging because the system cannot detect incomplete or inaccurate facial features. But thanks to advanced technology and continuous research and studies performed by so many tech-leaders. Moreover, artificial intelligence (AI) and machine learning (ML) communities are developingvarious models withstanding the urgent need for detecting masked faces.

Detecting masked faces requires making a system that can identify the available datasets of a masked face, i.e., facial features. The advanced detection technology includes various attributes as mentioned hereunder:

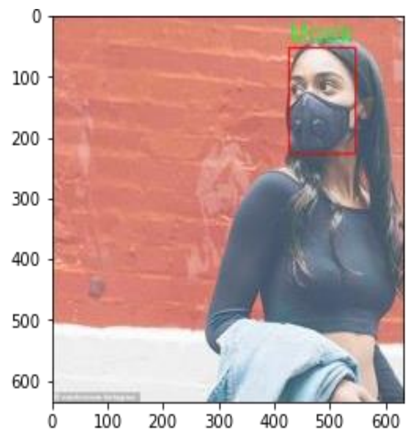
- Location of Faces, annotated by a shape i.e., square
- Face Orientation, includes front, left, left-front, right, right-front.
- Location of Eyes, need to mark eye centers.
- Location of Masks, annotated by a shape i.e., rectangles.
- Type of Masks, i.e., human-made masks with or without logo, face covered by hand, etc.
- Occlusion Degree, defining a face into four regions – eyes, nose, chin, and mouth

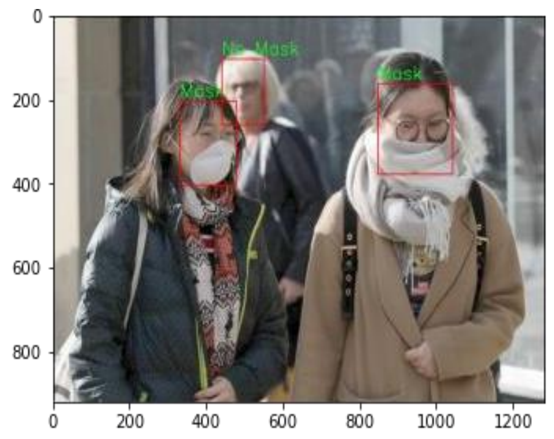
CHAPTER 3
COST ANALYSIS

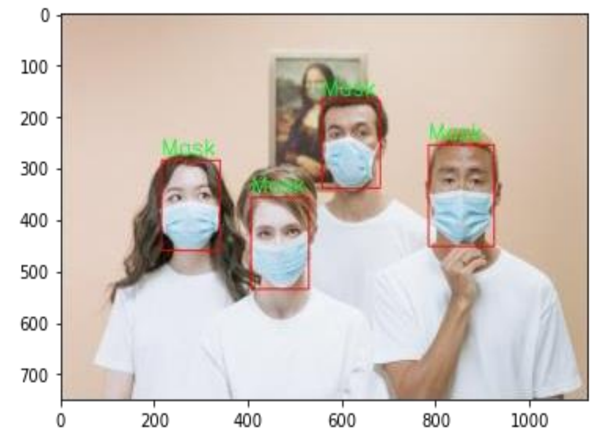
3.1 As the project does not require any Arduino, Raspberry pi or any other related hardware components expect any computer there is no cost involved in developing the project.

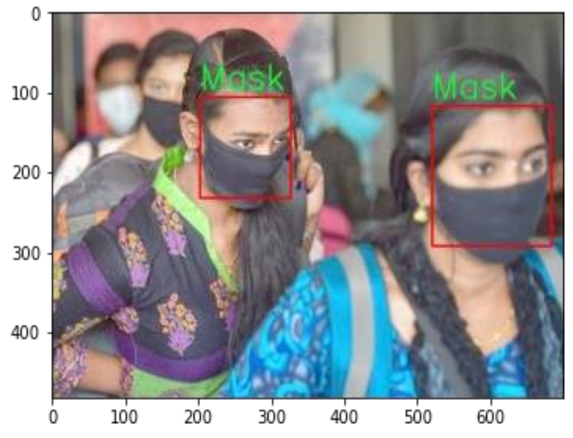
CHAPTER 4

RESULTS AND DISCUSSIONS









name	x1	x2	y1	y2	classname	
0	2756.png	69	126	294	392	face_with_mask
1	2756.png	505	10	723	283	face_with_mask
2	2756.png	75	252	264	390	mask_colorful
3	2756.png	521	136	711	277	mask_colorful
4	6098.jpg	360	85	728	653	face_no_mask

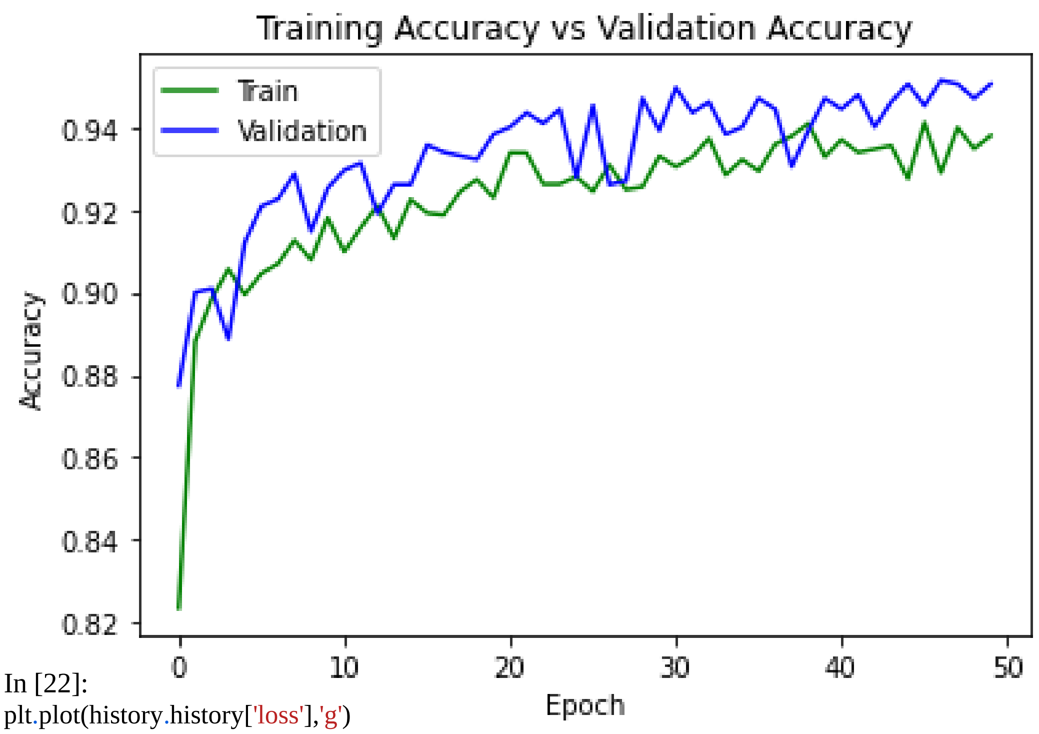
```
Epoch 1/50
143/143 [==============================] - 24s 165ms/step          - loss:   0.4193   - accura
cy: 0.8235 - val_loss: 0.2840 - val_accuracy: 0.8774
Epoch 2/50
143/143 [==============================] - 20s 138ms/step          - loss:   0.2781   - accura
cy: 0.8881 - val_loss: 0.2378 - val_accuracy: 0.9000
Epoch 3/50
143/143 [==============================] - 21s 144ms/step          - loss:   0.2622   - accura
cy: 0.8984 - val_loss: 0.2378 - val_accuracy: 0.9009
Epoch 4/50
143/143 [==============================] - 19s 131ms/step          - loss:   0.2466   - accura
cy: 0.9056 - val_loss: 0.2647 - val_accuracy: 0.8887
Epoch 5/50
143/143 [==============================] - 19s 136ms/step          - loss:   0.2510   - accura
cy: 0.8995 - val_loss: 0.2259 - val_accuracy: 0.9122
Epoch 6/50
143/143 [==============================] - 20s 143ms/step          - loss:   0.2363   - accura
cy: 0.9046 - val_loss: 0.2068 - val_accuracy: 0.9209
Epoch 7/50
143/143 [==============================] - 19s 131ms/step          - loss:   0.2254   - accura
cy: 0.9069 - val_loss: 0.1927 - val_accuracy: 0.9226
```

```
Epoch 8/50
143/143 [==============================] - 19s 131ms/step          - loss:    0.2325    - accura
cy: 0.9126 - val_loss: 0.1887 - val_accuracy: 0.9287
Epoch 9/50
143/143 [==============================] - 19s 135ms/step          - loss:    0.2246    - accura
cy: 0.9078 - val_loss: 0.2035 - val_accuracy: 0.9148
Epoch 10/50
143/143 [==============================] - 19s 134ms/step          - loss:    0.2062    - accura
cy: 0.9181 - val_loss: 0.1950 - val_accuracy: 0.9252
Epoch 11/50
143/143 [==============================] - 19s 135ms/step          - loss:    0.2254    - accura
cy: 0.9098 - val_loss: 0.1953 - val_accuracy: 0.9296Epoch 12/50

143/143 [==============================] - 19s 132ms/step          - loss:    0.2176    - accura
cy: 0.9157 - val_loss: 0.1751 - val_accuracy: 0.9313
Epoch 13/50
143/143 [==============================] - 19s 136ms/step          - loss:    0.2117    - accura
cy: 0.9210 - val_loss: 0.2099 - val_accuracy: 0.9191
Epoch 14/50
143/143 [==============================] - 18s 128ms/step          - loss:    0.2303    - accura
cy: 0.9131 - val_loss: 0.1858 - val_accuracy: 0.9261
Epoch 15/50
143/143 [==============================] - 19s 136ms/step          - loss:    0.2053    - accura
cy: 0.9225 - val_loss: 0.2028 - val_accuracy: 0.9261
Epoch 16/50
143/143 [==============================] - 19s 133ms/step          - loss:    0.2118    - accura
cy: 0.9192 - val_loss: 0.1740 - val_accuracy: 0.9357
Epoch 17/50
143/143 [==============================] - 18s 129ms/step          - loss:    0.2131    - accura
cy: 0.9188 - val_loss: 0.1685 - val_accuracy: 0.9339
Epoch 18/50
143/143 [==============================] - 19s 132ms/step          - loss:    0.2008    - accura
cy: 0.9245 - val_loss: 0.1724 - val_accuracy: 0.9330
Epoch 19/50
143/143 [==============================] - 19s 133ms/step          - loss:    0.2047    - accura
cy: 0.9273 - val_loss: 0.1828 - val_accuracy: 0.9322
Epoch 20/50
143/143 [==============================] - 19s 131ms/step          - loss:    0.2035    - accura
cy: 0.9229 - val_loss: 0.1772 - val_accuracy: 0.9383
Epoch 21/50
143/143 [==============================] - 18s 128ms/step          - loss:    0.1834    - accura
cy: 0.9338 - val_loss: 0.1678 - val_accuracy: 0.9400
Epoch 22/50
143/143 [==============================] - 19s 136ms/step          - loss:    0.1928    - accura
cy: 0.9337 - val_loss: 0.1526 - val_accuracy: 0.9435
Epoch 23/50
143/143 [==============================] - 19s 131ms/step          - loss:    0.1820    - accura
cy: 0.9262 - val_loss: 0.1515 - val_accuracy: 0.9409
Epoch 24/50
143/143 [==============================] - 19s 131ms/step          - loss:    0.1832    - accura
cy: 0.9262 - val_loss: 0.1599 - val_accuracy: 0.9443
Epoch 25/50
143/143 [==============================] - 20s 139ms/step          - loss:    0.1906    - accura
cy: 0.9280 - val_loss: 0.1711 - val_accuracy: 0.9278
Epoch 26/50
```

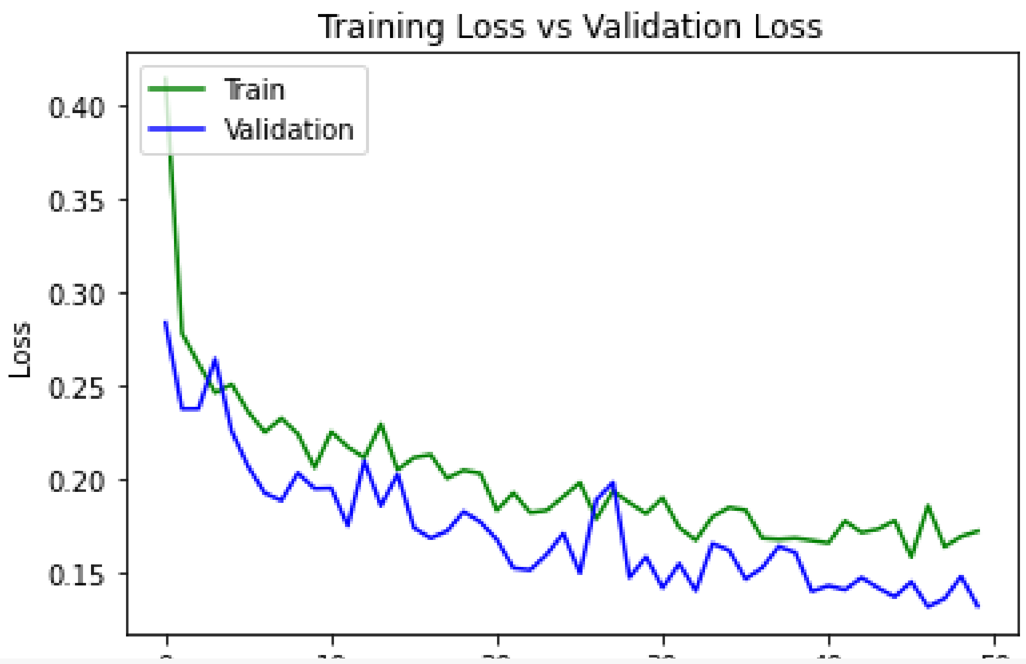
```
143/143 [==============================] - 18s 127ms/step - loss: 0.1986 - accura
cy: 0.9245 - val_loss: 0.1499 - val_accuracy: 0.9452
Epoch 27/50
143/143 [==============================] - 19s 133ms/step - loss: 0.1792 - accura
cy: 0.9308 - val_loss: 0.1892 - val_accuracy: 0.9261
Epoch 28/50
143/143 [==============================] - 20s 143ms/step - loss: 0.1935 - accura
cy: 0.9249 - val_loss: 0.1983 - val_accuracy: 0.9270
Epoch 29/50
143/143 [==============================] - 19s 130ms/step - loss: 0.1874 - accura
cy: 0.9256 - val_loss: 0.1476 - val_accuracy: 0.9470
Epoch 30/50
143/143 [==============================] - 19s 133ms/step - loss: 0.1818 - accura
cy: 0.9330 - val_loss: 0.1586 - val_accuracy: 0.9391
Epoch 31/50
143/143 [==============================] - 19s 134ms/step - loss: 0.1900 - accura
cy: 0.9304 - val_loss: 0.1420 - val_accuracy: 0.9496
Epoch 32/50
143/143 [==============================] - 19s 134ms/step - loss: 0.1751 - accura
cy: 0.9328 - val_loss: 0.1550 - val_accuracy: 0.9435
Epoch 33/50
143/143 [==============================] - 19s 132ms/step - loss: 0.1674 - accura
cy: 0.9373 - val_loss: 0.1404 - val_accuracy: 0.9461Epoch 34/50

143/143 [==============================] - 18s 128ms/step - loss: 0.1797 - accura
cy: 0.9284 - val_loss: 0.1657 - val_accuracy: 0.9383
Epoch 35/50
143/143 [==============================] - 20s 137ms/step - loss: 0.1849 - accura
cy: 0.9321 - val_loss: 0.1621 - val_accuracy: 0.9400
Epoch 36/50
143/143 [==============================] - 18s 129ms/step - loss: 0.1844 - accura
cy: 0.9294 - val_loss: 0.1467 - val_accuracy: 0.9470
Epoch 37/50
143/143 [==============================] - 19s 130ms/step - loss: 0.1686 - accura
cy: 0.9358 - val_loss: 0.1528 - val_accuracy: 0.9443
Epoch 38/50
143/143 [==============================] - 20s 138ms/step - loss: 0.1680 - accura
cy: 0.9378 - val_loss: 0.1641 - val_accuracy: 0.9304
Epoch 39/50
143/143 [==============================] - 18s 129ms/step - loss: 0.1685 - accura
cy: 0.9409 - val_loss: 0.1609 - val_accuracy: 0.9391
Epoch 40/50
143/143 [==============================] - 19s 133ms/step - loss: 0.1672 - accura
cy: 0.9328 - val_loss: 0.1401 - val_accuracy: 0.9470
Epoch 41/50
143/143 [==============================] - 19s 134ms/step - loss: 0.1659 - accura
cy: 0.9369 - val_loss: 0.1431 - val_accuracy: 0.9443
Epoch 42/50
143/143 [==============================] - 19s 132ms/step - loss: 0.1775 - accura
cy: 0.9339 - val_loss: 0.1409 - val_accuracy: 0.9478
Epoch 43/50
143/143 [==============================] - 18s 128ms/step - loss: 0.1715 - accura
cy: 0.9347 - val_loss: 0.1475 - val_accuracy: 0.9400
Epoch 44/50
```

```
143/143 [==============================] - 20s 139ms/step        - loss:   0.1732   - accura
cy: 0.9355 - val_loss: 0.1419 - val_accuracy: 0.9461
Epoch 45/50
143/143 [==============================] - 19s 131ms/step        - loss:   0.1780   - accura
cy: 0.9275 - val_loss: 0.1371 - val_accuracy: 0.9504
Epoch 46/50
143/143 [==============================] - 19s 129ms/step        - loss:   0.1587   - accura
cy: 0.9411 - val_loss: 0.1453 - val_accuracy: 0.9452
Epoch 47/50
143/143 [==============================] - 20s 139ms/step        - loss:   0.1859   - accura
cy: 0.9290 - val_loss: 0.1319 - val_accuracy: 0.9513
Epoch 48/50
143/143 [==============================] - 18s 127ms/step        - loss:   0.1641   - accura
cy: 0.9399 - val_loss: 0.1363 - val_accuracy: 0.9504
Epoch 49/50
143/143 [==============================] - 19s 133ms/step        - loss:   0.1697   - accura
cy: 0.9347 - val_loss: 0.1481 - val_accuracy: 0.9470
Epoch 50/50
143/143 [==============================] - 19s 133ms/step        - loss:   0.1723   - accura
cy: 0.9379 - val_loss: 0.1322 - val_accuracy: 0.9504
```

```
plt.plot(history. history['accuracy'],'g')
plt.plot(history.history['val_accuracy'],'b') plt.title('Training Accuracy vs
Validation Accuracy')plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')plt.show()
```

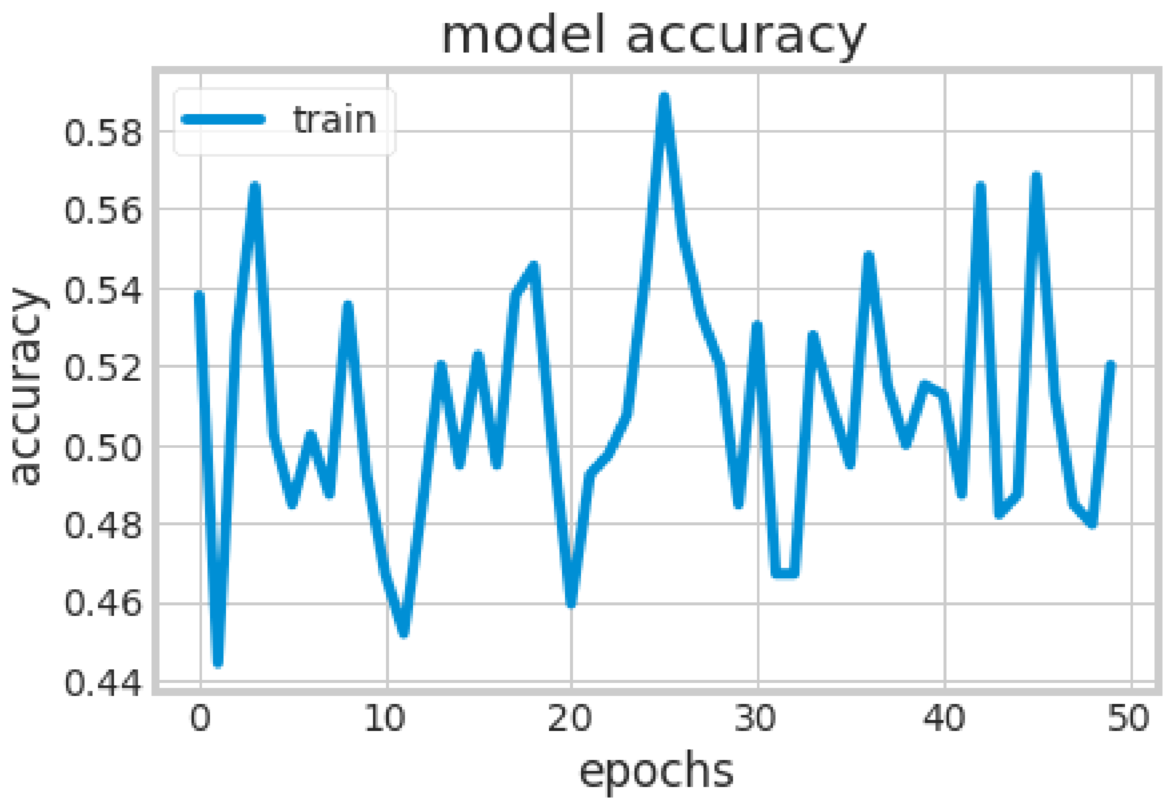


```
In [22]:
plt.plot(history.history['loss'],'g')
plt.plot(history.history['val_loss'],'b')
plt.title('Training Loss vs Validation Loss')plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')plt.show()
```

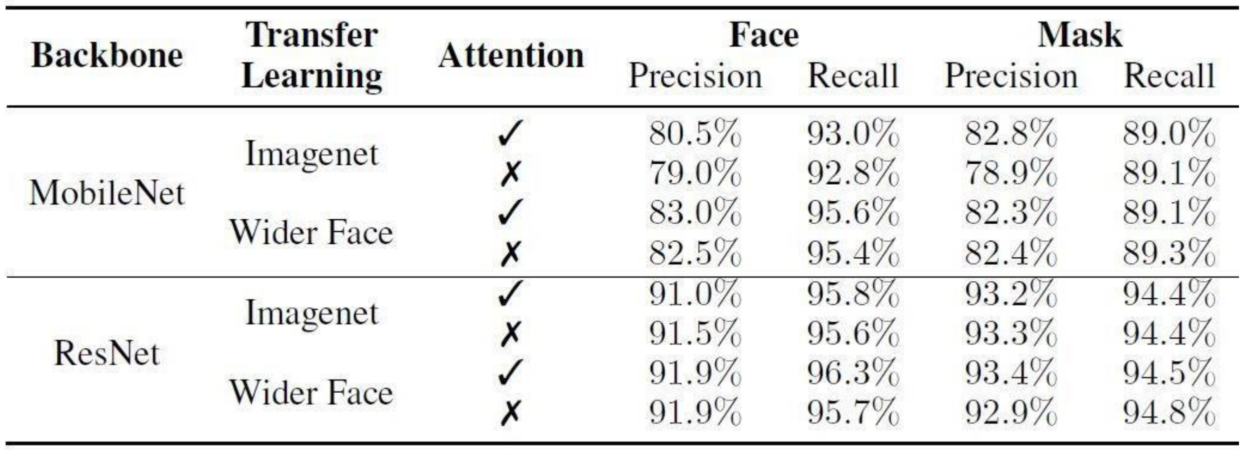


```
plt.plot(history.history['accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epochs')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.savefig('model_accuracy.png')
# summarize history for loss
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
plt.savefig('model_loss.png')
```

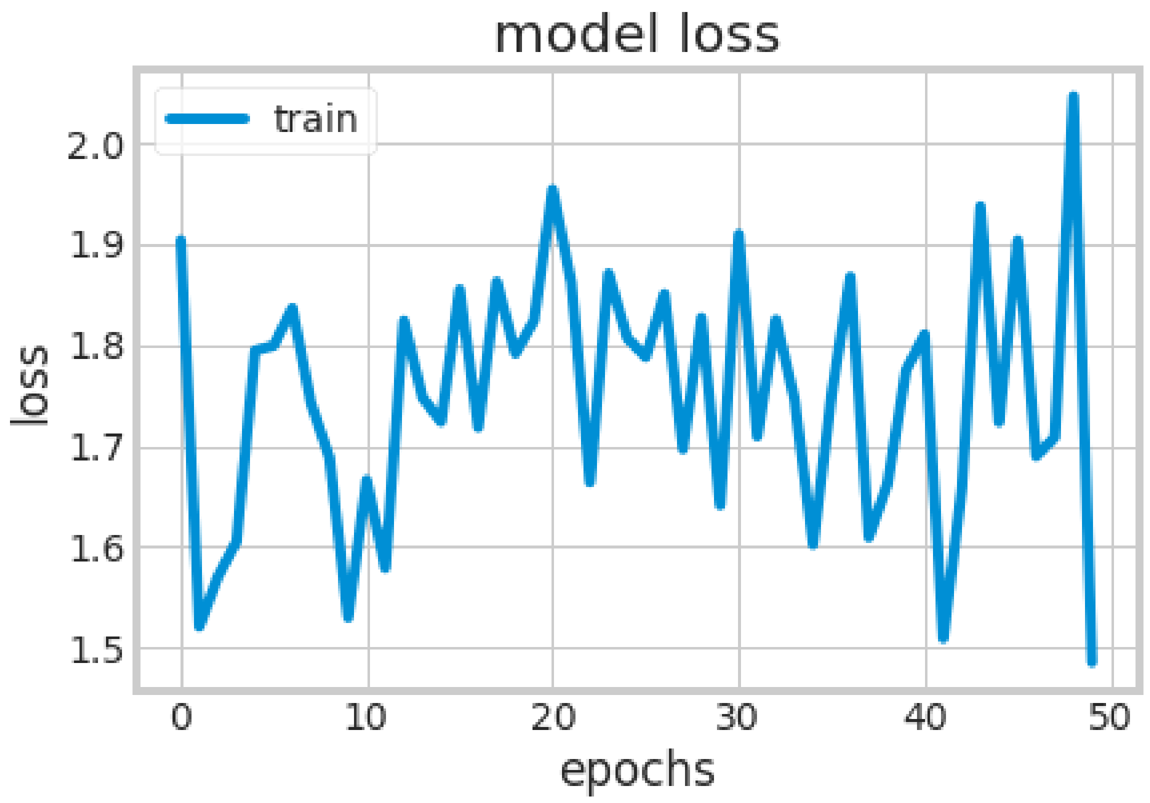
```
[<matplotlib.lines.Line2D at 0x7f7ec479ec90>]
Out[22]:
Text(0.5, 1.0, 'model accuracy')
Out[22]:
Text(0, 0.5, 'accuracy')
Out[22]:
Text(0.5, 0, 'epochs')
Out[22]:
<matplotlib.legend.Legend at 0x7f7ec47bc590>
```



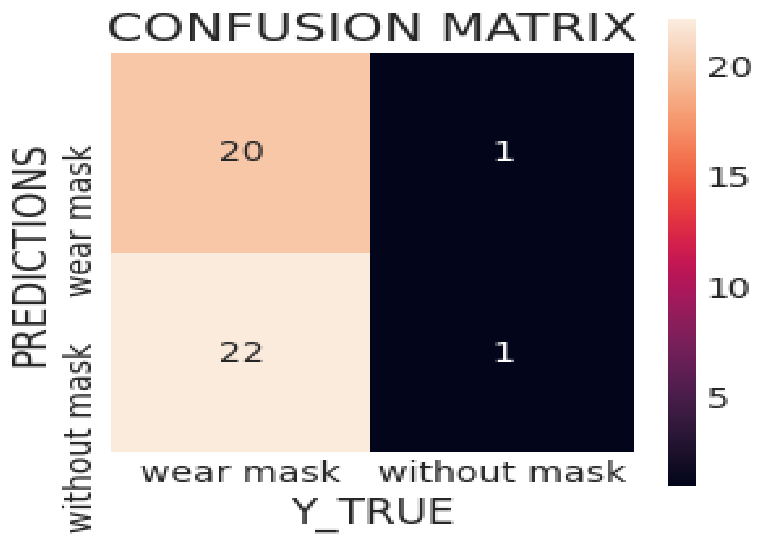
```
Out[22]:
Text(0.5, 1.0, 'model loss')
Out[22]:
Text(0, 0.5, 'loss')
Out[22]:
Text(0.5, 0, 'epochs')
Out[22]:
<matplotlib.legend.Legend at 0x7f7ec47218d0>
```



Backbone	**Transfer Learning**	**Attention**	**Face**		**Mask**	
			Precision	Recall	Precision	Recall
MobileNet	Imagenet	✓	80.5%	93.0%	82.8%	89.0%
		✗	79.0%	92.8%	78.9%	89.1%
	Wider Face	✓	83.0%	95.6%	82.3%	89.1%
		✗	82.5%	95.4%	82.4%	89.3%
ResNet	Imagenet	✓	91.0%	95.8%	93.2%	94.4%
		✗	91.5%	95.6%	93.3%	94.4%
	Wider Face	✓	91.9%	96.3%	93.4%	94.5%
		✗	91.9%	95.7%	92.9%	94.8%



```
confusion_mat = confusion_matrix(y_val, predictions)

plt.figure(figsize=(4,4))
sns.heatmap(confusion_mat, square=True, annot=True,
            yticklabels=['wear mask', 'without mask'],
            xticklabels=['wear mask', 'without mask']);
plt.title('CONFUSION MATRIX');
plt.xlabel('Y_TRUE');
plt.ylabel("PREDICTIONS");
```



CLASSIFICATION REPORT

```
              precision    recall  f1-score   support

           0       0.48      0.95      0.63        21
           1       0.50      0.04      0.08        23

    accuracy                           0.48        44
   macro avg       0.49      0.50      0.36        44
weighted avg       0.49      0.48      0.34        44
```

CHAPTER 5

CONCLUSION

Corporate giants from various verticals are turning to AI and ML, leveraging technology at humanity's service amid the pandemic. Digital product development companies are launching mask detection API services that enable developers to build a face mask detection system quickly to serve the community amid the crisis. The technology assures reliable and real-time face detection of users wearing masks. Besides, the system is easy to deploy into any business system while keeping the safety and privacy of users' data. So, the face mask detection system will be the leading digital solution for most industries, especially retail, healthcare, and corporate sectors. Discover how we can help you to serve the communities with the help of digital solutions.

CHAPTER 6

APPENDIX

```
import cv2
import os
from tensorflow.keras.preprocessing.image import img_to_arrayfrom
tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_inputimport numpy as np
from google.colab.patches import cv2_imshow

faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_alt2.xml")model =
load_model("mask_recog.h5")

def   face_mask_detector(frame): # frame =
    cv2.imread(fileName)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)faces =
    faceCascade.detectMultiScale(gray,
                                                                    scaleFactor=1.1,
                                                                    minNeighbors=5,
                                                                    minSize=(60, 60),
                                                                    flags=cv2.CASCADE_SCALE_IMAGE)
    faces_list=[]
    preds=[]
    for (x, y, w, h) in faces: face_frame =
            frame[y:y+h,x:x+w]
            face_frame = cv2.cvtColor(face_frame, cv2.COLOR_BGR2RGB)face_frame =
            cv2.resize(face_frame, (224, 224)) face_frame = img_to_array(face_frame)
            face_frame = np.expand_dims(face_frame, axis=0)face_frame =
            preprocess_input(face_frame) faces_list.append(face_frame)
            if len(faces_list)>0:
                    preds = model.predict(faces_list)for pred in
            preds:
                    (mask, withoutMask) = pred
            label = "Mask" if mask > withoutMask else "No Mask" color = (0, 255, 0) if label
            == "Mask" else (0, 0, 255)
            label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)cv2.putText(frame, label, (x,
            y- 10),
                                        cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)

            cv2.rectangle(frame, (x, y), (x + w, y + h),color, 3)# cv2_imshow(frame)
    return frame
```

Preprocessing

```
from tensorflow.keras.preprocessing.image import ImageDataGeneratortrain_data_generator =

ImageDataGenerator(rescale=1./255,
                                                    zoom_range=0.2,
                                                    shear_range=0.2,
                                                    rotation_range=0.2)

test_data_generator = ImageDataGenerator(rescale=1./255) validation_data_generator =

ImageDataGenerator(rescale=1./255,
                                                    zoom_range=0.2,
                                                    shear_range=0.2)
print("Flowing Train")
train_generator1 = train_data_generator.flow_from_directory(train_dirs[0],
        target_size=(HEIGHT,WEIGHT), batch_size=77,
        interpolation="nearest", class_mode='binary',
        classes=["without_mask","with_mask"])
train_generator2 = train_data_generator.flow_from_directory(train_dirs[1],
        target_size=(HEIGHT,WEIGHT),
        batch_size=46, interpolation="nearest",
        class_mode='binary', classes=["0","1"])
print("\nFlowing Test")
test_generator1 = test_data_generator.flow_from_directory(test_dirs[0],
        target_size=(HEIGHT,WEIGHT), batch_size=66,
        interpolation="nearest", class_mode='binary',
        classes=["without_mask","with_mask"])
test_generator2 = test_data_generator.flow_from_directory(test_dirs[1],
        target_size=(HEIGHT,WEIGHT),
        batch_size=11, interpolation="nearest",
        class_mode='binary', classes=["0","1"])
def genToTuple(gen):
        templist = []
        templist2 = []
        for i in range(gen._len_()): tempnext = gen.next()
            templist.append(tempnext[0])
            templist2.append(tempnext[1])
        x=np.concatenate(templist)
        y=np.concatenate(templist2)return (x,y)

def combine_tuple(*tuples):
        x=np.concatenate([tuples[i][0]  for  i  in  range(len(tuples))])y=np.concatenate([tuples[i][1]
        for i in range(len(tuples))])return (x,y.astype(int))


train_generator1_t  =  genToTuple(train_generator1) train_generator2_t  =
genToTuple(train_generator2)                train_generator3_t                  =
genToTuple(train_generator3)                train_generator4_t                  =
genToTuple(train_generator4)

test_generator1_t  =  genToTuple(test_generator1) test_generator2_t  =
genToTuple(test_generator2)                test_generator3_t                  =
genToTuple(test_generator3)                test_generator4_t                  =
genToTuple(test_generator4)



x_train,y_train = combine_tuple(train_generator1_t,train_generator2_t,train
_generator3_t,train_generator4_t)
x_test,y_test = combine_tuple(test_generator1_t,test_generator2_t,test_generator3_t,test_generator4_t)

x_val,y_val = genToTuple(validation_generator1)
```

Build Model

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dropout, SpatialDropout2D, BatchNormalization, Input,
Activation, Dense, Flatten
from keras.optimizers import Adam, RMSprop
from keras.callbacks import ReduceLROnPlateau, EarlyStoppingfrom keras.utils import
plot_model
from keras.losses import binary_crossentropy

def build_model():
```

```
model = Sequential()

model.add(Input(shape=(HEIGHT,WEIGHT,3,)))

model.add(Conv2D(filters=16,kernel_size=(2,2),padding="same"))
model.add(Activation("relu")) model.add(SpatialDropout2D(0.25))

model.add(MaxPool2D(pool_size=(4,4)))

model.add(Conv2D(filters=32,kernel_size=(2,2),padding="same"))
model.add(Activation("relu")) model.add(SpatialDropout2D(0.25))

model.add(MaxPool2D(pool_size=(4,4),strides=(4,4)))model.add(Flatten())

model.add(Dense(2048))
model.add(Activation("relu"))
model.add(Dropout(0.25))

model.add(Dense(1024))
model.add(Activation("relu"))
model.add(Dropout(0.2))


model.add(Dense(1)) model.add(Activation("sigmoid"))

optimizer = Adam(lr=0.001)
model.compile(optimizer = optimizer ,metrics=["accuracy"], loss = binary_crossentropy)

return model
```

Train Model

```
reducer = ReduceLROnPlateau(monitor='loss',patience=3,factor=0.75,min_lr=0.000001,verbose=1)
stopSign = EarlyStopping(monitor = "loss",patience=20,min_delta=0.000000000001,mode="min")

epochs = 50
batch_size = 32
steps_per_epoch = x_train.shape[0] // batch_sizehistory =
model.fit(x_train,y_train,
epochs = epochs, validation_data =
(x_val,y_val),verbose = 1,
batch_size=batch_size,
steps_per_epoch = steps_per_epoch,
callbacks=[reducer,stopSign])
```

```
Epoch 1/1000000
385/385 [==============================] - 3s 9ms/step - loss: 0.2668 -
accuracy: 0.8856 - val_loss: 0.0888 - val_accuracy: 0.9787
Epoch 2/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.1321 -
accuracy: 0.9520 - val_loss: 0.0453 - val_accuracy: 0.9850
Epoch 3/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.1037 -
accuracy: 0.9629 - val_loss: 0.0465 - val_accuracy: 0.9875
Epoch 4/1000000
385/385 [==============================] - 3s 9ms/step - loss: 0.0887 -
accuracy: 0.9671 - val_loss: 0.0462 - val_accuracy: 0.9775
Epoch 5/1000000
385/385 [==============================] - 3s 9ms/step - loss: 0.0806 -
accuracy: 0.9716 - val_loss: 0.0360 - val_accuracy: 0.9875
Epoch 6/1000000
385/385 [==============================] - 3s 9ms/step - loss: 0.0713 -
accuracy: 0.9735 - val_loss: 0.0313 - val_accuracy: 0.9850
Epoch 7/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0648 -
accuracy: 0.9759 - val_loss: 0.0231 - val_accuracy: 0.9925
Epoch 8/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0637 -
accuracy: 0.9771 - val_loss: 0.0258 - val_accuracy: 0.9900
Epoch 9/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0607 -
accuracy: 0.9781 - val_loss: 0.0251 - val_accuracy: 0.9900
Epoch 10/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0524 -
accuracy: 0.9815 - val_loss: 0.0135 - val_accuracy: 0.9962
Epoch 11/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0561 -
accuracy: 0.9793 - val_loss: 0.0179 - val_accuracy: 0.9937
Epoch 12/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0499 -
accuracy: 0.9811 - val_loss: 0.0184 - val_accuracy: 0.9950
Epoch 13/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0486 -
accuracy: 0.9811 - val_loss: 0.0319 - val_accuracy: 0.9875
Epoch 14/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0461 -
accuracy: 0.9825 - val_loss: 0.0292 - val_accuracy: 0.9862
Epoch 15/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0403 -
accuracy: 0.9847 - val_loss: 0.0299 - val_accuracy: 0.9900
Epoch 16/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0406 -
accuracy: 0.9838 - val_loss: 0.0186 - val_accuracy: 0.9950
Epoch 17/1000000
```

```
385/385 [==============================] - 3s 8ms/step - loss: 0.0418 -
accuracy: 0.9845 - val_loss: 0.0229 - val_accuracy: 0.9887
Epoch 18/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0379 -
accuracy: 0.9862 - val_loss: 0.0208 - val_accuracy: 0.9912
Epoch 19/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0377 -
accuracy: 0.9863 - val_loss: 0.0187 - val_accuracy: 0.9900
Epoch 20/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0356 -
accuracy: 0.9863 - val_loss: 0.0155 - val_accuracy: 0.9937
Epoch 21/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0350 -
accuracy: 0.9872 - val_loss: 0.0197 - val_accuracy: 0.9950
Epoch 22/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0382 -
accuracy: 0.9863 - val_loss: 0.0125 - val_accuracy: 0.9950
Epoch 23/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0323 -
accuracy: 0.9881 - val_loss: 0.0212 - val_accuracy: 0.9912
Epoch 24/1000000
385/385 [==============================] - 4s 10ms/step - loss: 0.0284 -
accuracy: 0.9894 - val_loss: 0.0221 - val_accuracy: 0.9962Epoch 25/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0240 -
accuracy: 0.9916 - val_loss: 0.0259 - val_accuracy: 0.9925Epoch 26/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0273 -
accuracy: 0.9898 - val_loss: 0.0108 - val_accuracy: 0.9975Epoch 27/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0249 -
accuracy: 0.9915 - val_loss: 0.0143 - val_accuracy: 0.9975Epoch 28/1000000
380/385 [============================>.] - ETA: 0s - loss: 0.0265 -
accuracy: 0.9895
Epoch 00028: ReduceLROnPlateau reducing learning rate to
0.0007500000356230885.
385/385 [==============================] - 3s 8ms/step - loss: 0.0264 -
accuracy: 0.9897 - val_loss: 0.0144 - val_accuracy: 0.9987Epoch 29/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0233 -
accuracy: 0.9913 - val_loss: 0.0119 - val_accuracy: 0.9962Epoch 30/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0174 -
accuracy: 0.9942 - val_loss: 0.0122 - val_accuracy: 0.9962Epoch 31/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0227 -
accuracy: 0.9925 - val_loss: 0.0145 - val_accuracy: 0.9962Epoch 32/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0167 -
accuracy: 0.9941 - val_loss: 0.0174 - val_accuracy: 0.9962Epoch 33/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0218 -
accuracy: 0.9920 - val_loss: 0.0146 - val_accuracy: 0.9987Epoch 34/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0178 -
accuracy: 0.9936 - val_loss: 0.0124 - val_accuracy: 0.9962
```

```
Epoch 35/1000000
381/385 [============================>.] - ETA: 0s - loss: 0.0176 -
accuracy: 0.9933
Epoch 00035: ReduceLROnPlateau reducing learning rate to
0.0005625000048894435.
385/385 [==============================] - 3s 8ms/step - loss: 0.0174 -
accuracy: 0.9934 - val_loss: 0.0114 - val_accuracy: 0.9962Epoch 36/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0142 -
accuracy: 0.9950 - val_loss: 0.0113 - val_accuracy: 0.9975Epoch 37/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0144 -
accuracy: 0.9955 - val_loss: 0.0092 - val_accuracy: 0.9987Epoch 38/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0149 -
accuracy: 0.9945 - val_loss: 0.0125 - val_accuracy: 0.9975Epoch 39/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0114 -
accuracy: 0.9960 - val_loss: 0.0103 - val_accuracy: 0.9987Epoch 40/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0123 -
accuracy: 0.9949 - val_loss: 0.0076 - val_accuracy: 0.9975Epoch 41/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0134 -
accuracy: 0.9955 - val_loss: 0.0160 - val_accuracy: 0.9987Epoch 42/1000000
383/385 [============================>.] - ETA: 0s - loss: 0.0122 -
accuracy: 0.9962
Epoch 00042: ReduceLROnPlateau reducing learning rate to
0.0004218749818392098.
385/385 [==============================] - 3s 8ms/step - loss: 0.0124 -
accuracy: 0.9962 - val_loss: 0.0163 - val_accuracy: 0.9937Epoch 43/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0111 -
accuracy: 0.9959 - val_loss: 0.0107 - val_accuracy: 0.9987Epoch 44/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0068 -
accuracy: 0.9972 - val_loss: 0.0126 - val_accuracy: 0.9987Epoch 45/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0101 -
accuracy: 0.9965 - val_loss: 0.0181 - val_accuracy: 0.9950Epoch 46/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0093 -
accuracy: 0.9967 - val_loss: 0.0127 - val_accuracy: 0.9987Epoch 47/1000000
383/385 [============================>.] - ETA: 0s - loss: 0.0102 -
accuracy: 0.9962
Epoch 00047: ReduceLROnPlateau reducing learning rate to
0.00031640623637940735.
385/385 [==============================] - 3s 8ms/step - loss: 0.0102 -
accuracy: 0.9962 - val_loss: 0.0182 - val_accuracy: 0.9962Epoch 48/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0088 -
accuracy: 0.9974 - val_loss: 0.0095 - val_accuracy: 0.9975Epoch 49/1000000
385/385 [==============================] - 3s 8ms/step - loss: 0.0075 -
accuracy: 0.9975 - val_loss: 0.0123 - val_accuracy: 0.9987
```

Prediction

```
from sklearn.metrics import confusion_matrix, accuracy_scoreimport seaborn as sns

ypred = model.predict_classes(x_test)
plt.subplots(figsize=(18,14))
sns.heatmap(confusion_matrix(ypred,y_test),annot=True,fmt="1.0f",cbar=False
,annot_kws={"size": 20})
plt.title(f"CNN Accuracy: {accuracy_score(ypred,y_test)}",fontsize=40)plt.xlabel("Target",fontsize=30)
plt.show()

plt.figure(figsize=(50,50))
tempc = np.random.choice(x_test[y_test == ypred.ravel()].shape[0],35,replace=False)
d = 0
for i in tempc: plt.subplot(7, 5, d+1)
        d += 1
        tempc = np.random.randint(x_test[y_test == ypred.ravel()].shape[0]) plt.imshow(x_test[y_test ==
        ypred.ravel()][tempc]) plt.title(f"True:{withWithoutMask[str(y_test[y_test == ypred.ravel()][t
empc])]}\nPredicted:{withWithoutMask[str(ypred.ravel()[y_test == ypred.ravel()][tempc])]}",
                        fontsize=40)
        plt.axis("off")
plt.subplots_adjust(wspace=-0.1, hspace=0.3)plt.show()
```

REFERENCES

- 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.
- P. Felzenszwalb, D. McAllester, and D. Ramanan, “A discriminatively trained, multiscale, deformable part model,” in 2008 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2008, pp. 1–8.
- L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, “Deep learning for generic object detection: A survey,” International journal of computer vision, vol. 128, no. 2, pp. 261–318, 2020.
- 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.
- 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.
- 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.
- 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.