

FACE MASK DETECTION USING MOBILENET V2

A PROJECT REPORT

Submitted

In the partial fulfillment of the requirements for

the award of the degree of

Bachelor of Technology in Electronics and Communication Engineering

By

JETTI SURESH

[Regd.No:171FA05026]

PONNURU GIRIDHAR RAM SAI

[Regd.No:171FA05324]

RAJALAPUDI KARTHIK

[Regd.No:171FA05329]

Under the Guidance of

Mrs. A. Divya

Assistant Professor



VIGNAN'S

Foundation for Science, Technology & Research

(Deemed to be University)

-Estd. u/s 3 of UGC Act 1956

(ACCREDITED BY **NAAC** WITH 'A' GRADE)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

**VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY AND RESEARCH
(Deemed to be University)**

VADLAMUDI, GUNTUR – 522 213, INDIA

JUNE 2021

CERTIFICATE

This is to certify that the PROJECT report entitled “**FACE MASK DETECTION USING MOBILENET V2**”, that is being submitted by **SURESH JETTI** bearing **Regd.No.171FA05026**, **GIRIDHAR PONNURU** bearing **Regd. No. 171FA05324** and **KARTHIK RAJALAPUDI** bearing **Regd.No.171FA05329** in partial fulfillment for the award of B.Tech degree in Electronics and Communication Engineering to Vignan’s Foundation for Science Technology and Research (deemed to be University), is a record of bonafide work carried out by them at University under the supervision of **Mrs. A. Divya** Assistant Professor of ECE Department.

Signature of the Faculty Guide

Mrs. A. Divya
Assistant Professor

Signature of Head of the Department

Dr. T. Pitchaiah, M.E, Ph.D,MIEEE
Professor, HoD

DECLARATION

We hereby declare that the project work entitled “**FACE MASK DETECTION USING MOBILENET V2**” is being submitted to Vignan’s Foundation for Science, Technology and Research (Deemed to be University) in partial fulfillment for the award of B. Tech degree in Electronics and Communication Engineering. The work was originally designed and executed by us under the guidance of our supervisor **Mrs.A.Divya** Assistant Professor at Department of Electronics and Communication Engineering, Vignan’s Foundation for Science Technology and Research (Deemed to be University) and was not a duplication of work done by someone else. We hold the responsibility of the originality of the work incorporated into this thesis.

Signature of the candidates

SURESH JETTII

[171FA05026]

GIRIDHAR PONNURU

[171FA05324]

KARTHIK RAJALAPUDI

[171FA05329]

ACKNOWLEDGEMENT

It's our privilege to express deep sense of gratitude to our beloved guide, **Mrs.A.Divya**, Assistant Professor, VFSTR (Deemed to be University) for his able guidance and useful suggestions, which helped us in completing the project work, in time.

We would like to specially thank **Dr. T. Pitchaiah**, Head of the Department, ECE and **Mr. S. Sivaji & Mr. M. Sekhar** Project Coordinator, Department of ECE, VFSTR (Deemed to be University) for their help and support during the program.

We wish to express our gratitude to **Dr. M. Y. S. Prasad** Vice-Chancellor, VFSTR (Deemed to be University) for providing us the greatest opportunity to have a great exposure and to carry out the Project.

We wish to express our heart full thanks to our parents for their support and encouragement throughout our life.

NAME OF THE STUDENTS

SURESH JETTI
GIRIDHAR PONNURU
KARTHIKRAJALAPUDI

ABSTRACT

This project intends to develop a Face Mask Detection system using OpenCV, Keras/Tensor Flow and Deep Learning. The system can be easily integrated to various embedded devices with limited computational capacity as it uses MobileNetV2 architecture. It will detect face masks in images as well as in real-time videos.

In recent times, where Covid-19 has impacted a domino effect on manufacturing, travel, tourism, hospitality, crippling the global economy. In addition to it is the growing curve of human deaths across the globe due to the pandemic, this project which relies on computer vision and deep learning, intends to make an impact and solve the real-world problem of safety measures at some significant level.

This project can be used at airports, offices, hospitals and many more public places to ensure that the safety standards are maintained and people are abiding by the rules and regulations to wear protective masks at public places. If the detection system classifies as 'No Mask', reminders can be given as well as actions can be taken against such individuals.

TABLE OF CONTENTS

CHAPTER-1: INTRODUCTION	Page. No
1.1 Introduction	2
1.2 Problem statement	2
1.3 Aim and objectives	2
1.4 Excepted outcomes	3
1.5 Algorithms	4
1.5.1 Feature Extraction using ConvNets	4
1.5.2 Transfer Learning Algorithms	4
1.5.3 Transfer Learning with pre-trained ConvNet	5
1.6 Methodlogies	7
1.6.1 OpenCV's DNN module	7
1.6.2 OpenCV's Face detector based on framework combind with mobileNet V2	7
1.6.3 OpenCV's blobFromImage and blobFromImages to facilitate image preprocessing	7
1.6.4 Keras Image Data Generator	9
CHAPTER-2: LITERATURE SURVEY	
2.1 Deep Learning based on Face Recognition System	11
2.2 Facemask detection using mobilenet V2 in era of covid-19	11
2.3 Facemask detection with deep learning and computer vision	11
2.4 Covid 19 Facemask detection using Tensorflow,keras and opencv	11
2.5 An automated System to limit covid-19 using facemask detection in smart city network	12
CHAPTER-3: TOOLS AND LIBRARIES USED	
3.1 Tools Required	14
3.1.1 Hardware requirements	14
3.1.2 Software requirements	14
3.2 Libraries Used	14
3.2.1 OpenCV	14
3.2.2 Sklearn	15
3.2.3 NumPy	15
3.2.4 Imutils	16
3.2.5 Operating System	16
CHAPTER-4: BLOCK DIAGRAM	
4.1 Block Diagram	18
4.1.1 Block Diagram Explanation	18
4.2 Steps Involved	19
4.2.1 Data Collection	19
4.2.2 Pre-Processing	19

4.2.3 Split Data	20
4.2.4 Building a Model	20
4.2.5 Testing Model	20
4.2.6 Model Implementation	20
4.3 Dataset Pre-Processing	21
4.4 Training with dataset	21
4.5 Activity Diagram	23
4.6 Class Diagram	25
CHAPTER-5: CODE AND EXPLANATION	
5.1 Directory Structure	28
5.2 Train the Model	28
5.3 Implementation of our Covid—19 facemask detector model for Static images	33
5.4 Implementation of our Covid—19 facemask detector model for Real-time video stream	36
CHAPTER-6: OUTPUT	
6.1 Output	42
6.2 Results and model evaluation	43
CHAPTER-7: ADVANTAGES, LIMITATIONS AND APPLICATIONS	
7.1 Advantages	45
7.2 Limitations	46
7.3 Applications	47
CONCLUSION	48
FUTURE SCOPE	49
REFERENCES	50

LIST OF FIGURES

Figure No	Figure Name	Page No
4.1	Block Diagram	18
4.2	Steps Involved	19
4.5	Activity Diagram	24
4.6	Class Diagram	26
6.1	Output of with-out Mask	42
	Output of with Mask	42

LIST OF TABLES

Table No	Table Name	Page No
6.2	Results of Model Evaluation	43

CHAPTER -1

INTRODUCTION

1.1 INTRODUCTION

The main objective of this project is To develop a Face Mask Detection system with OpenCV, Keras/TensorFlow and Deep Learning in order to detect face masks in static images as well as in real-time video streams.

1.2 PROBLEM STATEMENT

In the present scenario due to Covid-19, there are no efficient face mask detection applications, which are now in high demand for transportation means, densely populated areas, residential districts, large-scale manufacturers and other enterprises to ensure that the safety guidelines are strictly followed. Also, the absence of large datasets of 'with_mask' images has made this task more cumbersome and challenging.

Therefore, the need of the hour is to generate a huge custom dataset of 'with_mask' images with the help of existing datasets and search APIs followed by developing a face mask detection system. This system is need of the hour as India tries to battle the novel corona virus that has infected nearly 1,00,000 and has caused more than 1000 deaths per day with the figures still increasing at a rapid pace.

1.3 AIM & OBJECTIVES

The objective of this project is to develop face mask detection using mobilenet v2. Expected achievements in order to fulfill the objectives are:

- To train a custom deep learning model to detect whether a person is wearing a mask or not.
- To develop a custom dataset of with_mask images with the help of Bing Search API, few existing Kaggle datasets and RMFD dataset.
- Datasets will be divided into two classes: 'with_mask' and 'without_mask'.
- To train the face mask detector on the custom dataset using Keras and TensorFlow.
- To implement the trained model to detect masks in static input images and also in real-time videos.
- The model is expected to give an accuracy of 98% and above.

1.4 EXPECTED OUTCOMES

- A two-step system to first **"detect"** faces on an image/live video stream, using a trained facedetector from OpenCV and after that to pass the found faces on the **"mask predictor"** that returns if the face is wearing a mask or not.
- A custom dataset of 'with_mask' with 3000+ images for both 'with_mask' and 'without_mask'.
- A face mask detector training script in Python, which accepts the input dataset, loads and pre-processes the images, and labels them using TensorFlow.keras.
- It would also fine-tune the model with MobileNetV2 classifier using pre-trained ImageNet weights.
- Training history plot with accuracy and loss curves produced with the help of matplotlib.
- Perform face mask detection correctly in static images present in folders.
- Using your webcam, the system developed applies face mask detection correctly to every frame in the real-time video stream.
- Outputs the confidence probability of "Mask" or "No Mask" in the static images and real-time video streams examined.
- The system can be easily integrated to various embedded devices with limited computational capacity as it uses MobileNetV2 architecture.

1.5 ALGORITHMS:

1.5.1 Feature Extraction using ConvNets

Traditional machine learning approach uses feature extraction for images using Global feature descriptors such as Local Binary Patterns (LBP), Histogram of Oriented Gradients (HoG), Color Histograms etc. or Local descriptors such as SIFT, SURF, ORB etc. These are hand-crafted features that require domain level expertise.

But here comes Convolutional Neural Networks (CNN). Instead of using hand-crafted features, Deep Neural Nets automatically learns these features from images in a hierarchical fashion. Lower layers learn low-level features such as Corners, Edges whereas middle layers learn color, shape etc. and higher layers learn high-level features representing the object in the image.

Instead of making a CNN as a model to classify images, we can use it as a Feature Extractor by taking the activations available before the last fully connected layer of the network. These activations will be acting as the feature vector for a machine learning model (classifier) which further learns to classify it. This type of approach is well suited for Image Classification problems, where instead of training a CNN from scratch (which is time-consuming and tedious), a pre-trained CNN could be used as a Feature Extractor - **Transfer Learning**.

1.5.2 Transfer Learning Algorithm

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is the idea of overcoming the isolated learning paradigm and utilizing the knowledge acquired for one task to solve related ones.

Traditional learning is isolated and occurs purely based on specific tasks, datasets and training separate isolated models on them. No knowledge is retained which can be transferred from one model to another. In transfer learning, you can leverage knowledge (features, weights etc.) from previously trained models for training newer models and even tackle problems like having less data for the newer task!

Learning is not an easy process, not for humans and not for machines either. It is a heavy-duty, resource-consuming and time-consuming process and hence it was important to devise a method that would prevent a model from forgetting the learning curve that it attained from a specific dataset and also lets it learn more from new and different datasets.

Transfer learning is simply the process of using a pre-trained model that has been trained on a dataset for training and predicting on a new given dataset.

“A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task such as the **ImageNet**.”

Applications of Transfer Learning:

- 1. Transfer learning for NLP:** Textual data presents all sorts of challenges when it comes to ML and deep learning. These are usually transformed or vectorized using different techniques. Embedding, such as Word2vec and FastText, have been prepared using different training datasets. These are utilized in different tasks, such as sentiment analysis and document classification, by transferring the knowledge from the source tasks.
- 2. Transfer learning for Audio/Speech:** Similar to domains like NLP and Computer Vision, deep learning has been successfully used for tasks based on audio data. For instance, Automatic Speech Recognition (ASR) models developed for English have been successfully used to improve speech recognition performance for other languages, such as German.
- 3. Transfer learning for Computer Vision:** Deep learning has been quite successfully utilized for various computer vision tasks, such as object recognition and identification, using different CNN architectures.

1.5.3 Transfer Learning with a pre-trained ConvNet

We can have two ways to customize a pre-trained model:

- 1. Feature Extraction:** Use the representations learned by a previous network to extract meaningful features from new samples. We simply add a new classifier, which will be trained from scratch, on top of the pre-trained model so that we can repurpose

the feature maps learned previously for the dataset.

We do not need to retrain the entire model. The base convolutional network already contains features that are generically useful for classifying pictures. However, the final, classification part of the pre-trained model is specific to the original classification task, and subsequently specific to the set of classes on which the model was trained.

2. **Fine-Tuning:** Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task.

We will create the base model from the **MobileNetV2** model developed at Google. This is pre-trained on the **ImageNet dataset**, a large dataset consisting of 1.4M images and 1000 classes. ImageNet is a research training dataset with a wide variety of categories. This base of knowledge will help us classify "Mask" and "No Mask" from our specific dataset.

It is important to freeze the convolutional base before we compile and train the model. Freezing (*by setting `layer.trainable = False`*) prevents the weights in a given layer from being updated during training. MobileNetV2 has many layers, so setting the entire model's trainable flag to False will freeze all the layers.

1.6 METHODOLOGIES:

1.6.1 OpenCV’s “deep neural networks” (dnn) module: OpenCV (3.3 or later) comprises of the highly efficient dnn module supported by a number of deep learning frameworks such as Caffe, TensorFlow, and Torch/PyTorch. This module has a more accurate Caffe-based face detector. In this project, we will be training our deep learning model using Caffe, hence we need the following files:

- The **.prototxt** file(s) which will define the *model architecture* (i.e., the layers)
- The **.caffemodel** file which will contain the *weights* for the actual layers

1.6.2. OpenCV’s face detector based on the Single Shot Multibox Detector (SSD) framework combined with MobileNetV2 architecture: In order to obtain the bounding box (x , y) coordinates for an object (mask in this case) in an image we need to apply object detection.

- SSDs, originally developed by Google, are a balance between R-CNNs and YOLO methods of object detection. The algorithm is more straightforward than Faster R-CNNs.
- Network architectures such as VGG or Resnet are unsuitable for resource constrained devices due to their sheer size and resulting number of computations. Instead, we use MobileNets. They are designed for resource constrained devices such as your smartphone.
- Hence, if we combine both the MobileNet architecture and the Single Shot Detector (SSD) framework, we will have a fast, efficient deep learning-based method for object detection.

1.6.3 OpenCV’s blobFromImage and blobFromImages to facilitate image pre-processing: cv2.dnn.blobFromImage and cv2.dnn.blobFromImages functions of OpenCV’s dnn module facilitates image pre-processing for deep learning classification. These two functions perform:

1. Mean subtraction - Mean subtraction is used to help tackle illumination changes in the input images in our dataset. For example, the mean values for the ImageNet training set are **R=103.93, G=116.77, and B=123.68**
2. Scaling - The scaling factor aids in normalization.
3. And optionally channel swapping

A '*blob*' is just a collection of image(s) with the same spatial dimensions (width and height), same depth (number of channels), that have to be pre-processed in the same manner.

[blobFromImage] creates 4-dimensional blob from image. Optionally resizes and crops image from center, subtracts mean values, scales values by scale-factor, swaps Blue and Red channels.

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

Each parameter is described below:

- **Image:** This is the input image we want to pre-process before passing it through our deep neural network for classification.
- **Scale factor:** After we perform mean subtraction we can optionally scale our images by some factor. This value defaults to `1.0` (i.e., no scaling) but we can supply another value as well. It's also important to note that scalefactor should be $1/\sigma$ as we're actually multiplying the input channels (after mean subtraction) by scale factor.
- **Size:** Here we supply the spatial size that the Convolutional Neural Network expects. For most current state-of-the-art neural networks this is either 224×224 , 227×227 , or 299×299 . We will be using 224×224 .
- **mean:** These are our mean subtraction values. They can be a 3-tuple of the RGB means or they can be a single value in which case the supplied value is subtracted from every channel of the image.
- **swapRB:** OpenCV assumes images are in BGR channel order; however, the `mean` value assumes we are using RGB order. To resolve this discrepancy, we

can swap the R and B channels in image by setting this value to `True`. By default, OpenCV performs this channel swapping for us.

- The `cv2.dnn.blobFromImage` function returns a blob which is our input image after mean subtraction, normalizing, and channel swapping.
- The **`cv2.dnn.blobFromImages`** function is exactly the same:

```
blob = cv2.dnn.blobFromImages(images,    scaleFactor=1.0,    size,    mean,  
swapRB=True)
```

The only exception is that we can pass in multiple images, enabling us to batch process a set of images.

1.6.4 Keras ImageDataGenerator: Generate batches of tensor image data with real time data augmentation. The data will be looped over (in batches).

Keras ImageDataGenerator class works by:

- Accepting a batch of images used for training.
- Taking this batch and applying a series of random transformations to each image in the batch (including random rotation, resizing, shearing, etc.).
- Replacing the original batch with the new, randomly transformed batch.
- Training the CNN on this randomly transformed batch (i.e., the original data **itself is not** used for training).

CHAPTER-2

LITERATURE SURVEY

2.1 Deep learning based on face recognition system

The paper by Marko Arsenovic, Andras Anderla in the year 2019, proposed the important steps in face detection, image preprocessing like finding face landmarks, face positioning, generating face embeddings and classification and then attendance marking. This paper has used Support vector machine algorithm for face recognition purpose. This paper has used small number of images for training the model.

2.2 Face Mask Detection Using MobileNetV2 in The Era of COVID-19

The paper by Isunuri, B Venkateswarulu, Jagadeesh Kakarla, Shree Prakash in the year 2020, has implemented LBPH with haarcascade Lib for face detection and Using Open CV Lib for image capturing, and tells us the way in which there is no physical interaction of human with the devices for capturing the faces with cameras and to process every image into datasets in an efficient manner.

2.3 Facemask detection with Deep learning and Computer Vision

The paper by Vinitha, Velanitha in the year 2020, has several steps like image capture, image processing, CNN technique, comparison from database, Output displaying. In this, faces are captured from cc camera and there after applying techniques for face detection and recognition which can decrease the manual work from human and can also increase the security safety. This paper has used CNN algorithm.

2.4 Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV

The paper by Mashhood Arijya Das, Rohini basak in the year 2020, we briefly explained the motivation of the work at first. Then, we illustrated the learning and performance task of the model. Using basic ML tools and simplified techniques the method has achieved reasonably high accuracy. It can be used for a variety of applications. Wearing a mask may be obligatory in the near future, considering the Covid-19 crisis. Many public service providers will ask the customers to wear masks correctly to avail of their services.

2.5 An Automated System to Limit COVID-19 Using Facial Mask Detection in Smart City Network

The paper by Mohammad Marufur Rahman, Md. Milon Islam, Jong-Hoon Kim in the year 2020, presents a system for a smart city to reduce the spread of coronavirus by informing the authority about the person who is not wearing a facial mask that is a precautionary measure of COVID-19. The motive of the work comes from the people disobeying the rules that are mandatory to stop the spread of coronavirus. The system contains a face mask detection architecture where a deep learning algorithm is used to detect the mask on the face. To train the model, labeled image data are used where the images were facial images with masks and without a mask.

CHAPTER 3

TOOLS AND LIBRARIES USED

3.1 TOOLS REQUIRED:

3.1.1 Hardware Requirements:

- Processor: 1.6 GHz Intel Core i5/Pentium IV 2.4 GHz
- RAM: at least 2 GB RAM
- Speed: 500 MHz
- Hard Disk: 80 GB minimum

3.1.2 Software Requirements:

Operating System:

- Windows 8 or later
- Mac OS 10.13.6 or later (preferable)
- Ubuntu 16.04 or later (64-bit) (preferable)
- PyCharm/ VSCode editor/ TensorFlow GPU (optional)

Programming Language:

- Python (3.7.6)

3.2 LIBRARIES USED:

3.2.1 OPENCV:

- OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.
- The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.
- These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.
- The library is used extensively in companies, research groups and by governmental bodies.

- It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

3.2.2 SKLEARN:

- Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.
- Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms.
- We can use Sklearn datasets by first importing datasets which holds all the seven datasets. Each dataset has corresponding function used to load the dataset. These functions follow the same format: "load_DATASET()", where DATASET refers to the name of the dataset.

3.2.3 NUMPY:

- NumPy is a module for Python. The name is an acronym for "Numeric Python" or "Numerical Python". Furthermore, NumPy enriches the programming language Python with powerful data structures, implementing multi-dimensional arrays and matrices. These data structures guarantee efficient calculations with matrices and arrays.
- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- It also has functions for working in domain of linear algebra, Fourier transform, and matrices.
- NumPy is a Python library used for working with arrays.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.
- NumPy arrays are stored at one continuous place in memory unlike lists, so

processes can access and manipulate them very efficiently. This behavior is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also, it is optimized to work with latest CPU architectures

- Arrays are very frequently used in data science, where speed and resources are very important.

3.2.4 IMUTILS:

Imutils are a series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and both Python 2.7 and Python 3. Contributing to the open-source community. PyPI, the Python Package Index repository is a wonderful thing. It makes downloading, installing, and managing Python libraries and packages a breeze. And with all that said, one can push their own personal imutils package online.

3.2.5 OPERATING SYSTEMS:

Operating System can be defined as an interface between user and the hardware. It provides an environment to the user so that, the user can perform its task in convenient and efficient way.

In the Computer System (comprises of Hardware and software), Hardware can only understand machine code (in the form of 0 and 1) which doesn't make any sense to a naive user. We need a system which can act as an intermediary and manage all the processes and resources present in the system. An Operating System can be defined as an interface between user and hardware. It is responsible for the execution of all the processes, Resource Allocation, CPU management, File Management and many other tasks.

CHAPTER 4
BLOCK DIAGRAM
ACTIVITYDIAGRAM
AND
CLASS DIAGRAM

4.1 BLOCK DIAGRAM:

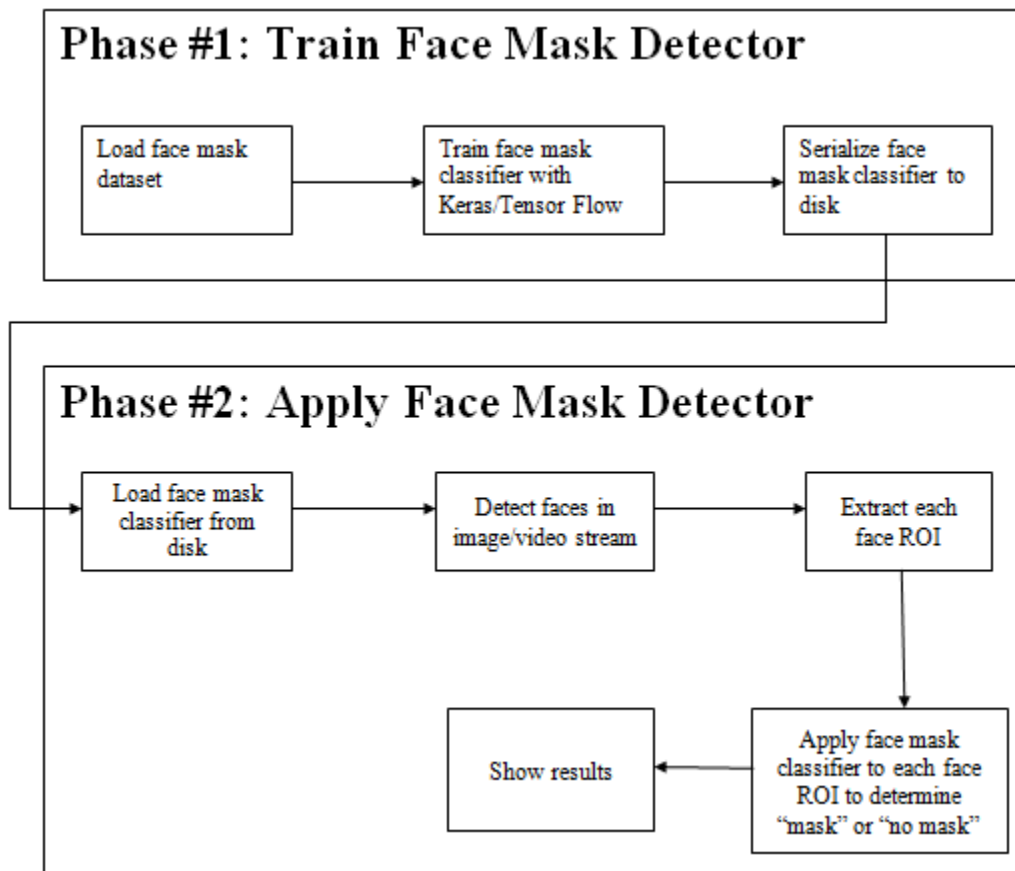


Figure 4.1 Block Diagram

4.1.1 Block Diagram Explanation:

1. Training:

- Loading the face mask detection dataset from disk
- Training a model using Keras/TensorFlow on this loaded dataset
- Serializing the face mask detector back to disk

2. Deployment:

- Loading the face mask detector
- Performing face detection
- Classifying each face as "Mask" or "No Mask"

4.2 STEPS INVOLVED:

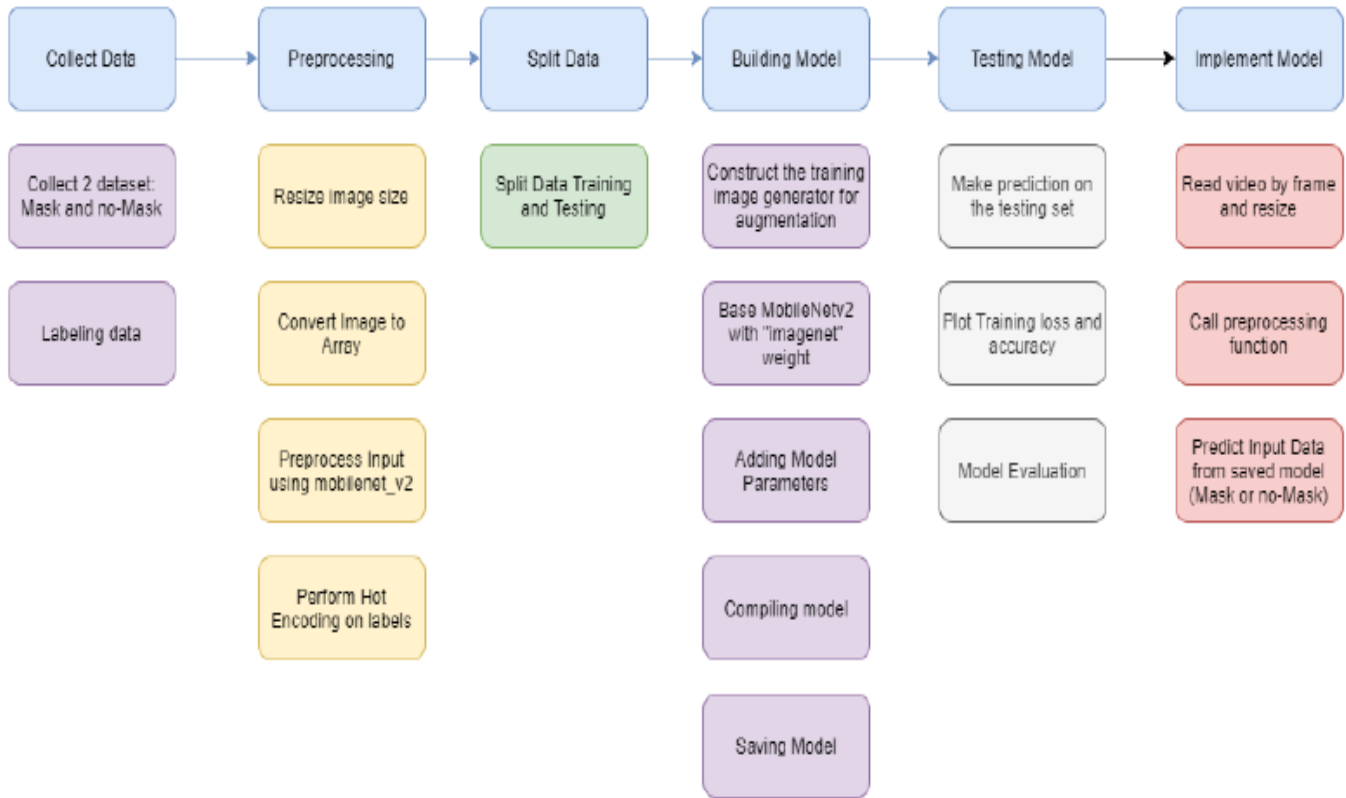


Figure 4.2 Steps Involved.

4.2.1 Data Collection:

The development of the Face Mask Recognition model begins with collecting the data. The dataset train data on people who use masks and who do not. The model will differentiate between people wearing masks and not. For building the model, this study uses 1.916 data with mask and 1.930 data without a mask. At this step, the image is cropped until the only visible object is the face of the object. The next step is to label the data. The data which has been collected labeled into two groups; with and without a mask. After the data has been labeled, it is grouped into those two groups.

4.2.2 Pre-Processing:

The pre-processing phase is a phase before the training and testing of the data. There are four steps in the pre-processing which are resizing image size, converting the image to the array, pre-processing input using MobileNetV2, and the last is performing hot encoding on labels. The resizing image is a critical pre-processing step in computer vision due to the effectiveness of training models. The smaller size of the image, the better the model will run. In this study, the resizing an image is making the image into 224 x 224 pixels. The next step is to process all the

images in the dataset into an array. The image is converted into the array for calling them by the loop function. After that, the image will be used to pre-process input using MobileNetV2. And the last step in this phase is performing hot encoding on labels because many machine learning algorithms cannot operate on data labeling directly. They require all input variables and output variables to be numeric, including this algorithm. The labeled data will be transformed into a numerical label, so the algorithm can understand and process the data.

4.2.3 Split Data:

After the pre-processing phase, the data is split into two batches, which are training data namely 75 percent, and the rest is testing data. Each batch is containing both of with-mask and without-mask images

4.2.4 Building a model:

The next phase is building the model. There are six steps in building the model which are constructing the training image generator for augmentation, the base model with MobileNetV2, adding model parameters, compiling the model, training the model, and the last is saving the model for the future prediction process.

4.2.5 Testing Model:

In this step we are going to test the model using testing dataset and then evaluate the model built. This step is followed by implementation of the model.

4.2.6 Model Implementation:

The model implemented in the video. The video read from frame to frame, then the face detection algorithm works. If a face is detected, it proceeds to the next process. From detected frames containing faces, reprocessing will be carried out including resizing the image size, converting to the array, preprocessing input using MobileNetV2. The next step is predicting input data from the saved model. Predict the input image that has been processed using a previously built model. Besides, the video frame will also be labeled that the person is wearing a mask or not along with the predictive percentage.

4.3 DATASET PREPROCESSING

- We have two sets which are divided into with_mask and with_out_mask folders.
- This folder consists of real faces wearing protective masks and not wearing masks.
- No morphed faces with masks are used.
- After collection of datasets we deleted some images which are being blurred and not fruitful and done with the data pruning.
- This dataset consists of 4095 images belonging to two classes:
with_mask: 2165 images
without_mask: 1930 images
- The dataset will be split into 80% training and 20% testing data with the help of sklearn library.
- The training set has roughly 3276 images and the testing set has roughly 819 images.

4.4 Training with Dataset:

- We are having algorithms like R-CNN, YOLO, SSD, etc.,
- But here we are using single shot multibox detection for its speed and accuracy and it better suits our model.
- Network architectures like RESNET, VGG are occupying large amount of space about 200mb-500mb.
- So we are using MobileNetV2 architecture which is specially designed for memory constrained devices like mobiles.
- We are combinely using both MobileNet architecture and SSD framework for faster and memory efficient performance.
- Data augmentation of images are done by using keras image generator module for generalization.
- Here we are using Convnets which is our classifier, these deep neural nets automatically learns features from images in hierarchical fashion.
- Instead of using it as a model we are making it as a feature extractor by taking activation before the last fully connected layer of the network.
- These activation acts as feature vector for our model(classifier).

- It is based on transfer learning algorithm.
- By using MobileNetV2 we are creating a base model and do finetuning with our model(classifier).
- This MobileNetV2 is pretrained on 'ImageNet dataset' which consists of 1.4 million images and 100 classes.
- This base as a knowledge will help our model to detect whether mask or no mask.

4.5 ACTIVITY DIAGRAM:

Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require coordination.

Purpose of Activity Diagram:

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

Activity diagram can be used for:

- Modeling work flow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigating business requirements at a later stage.

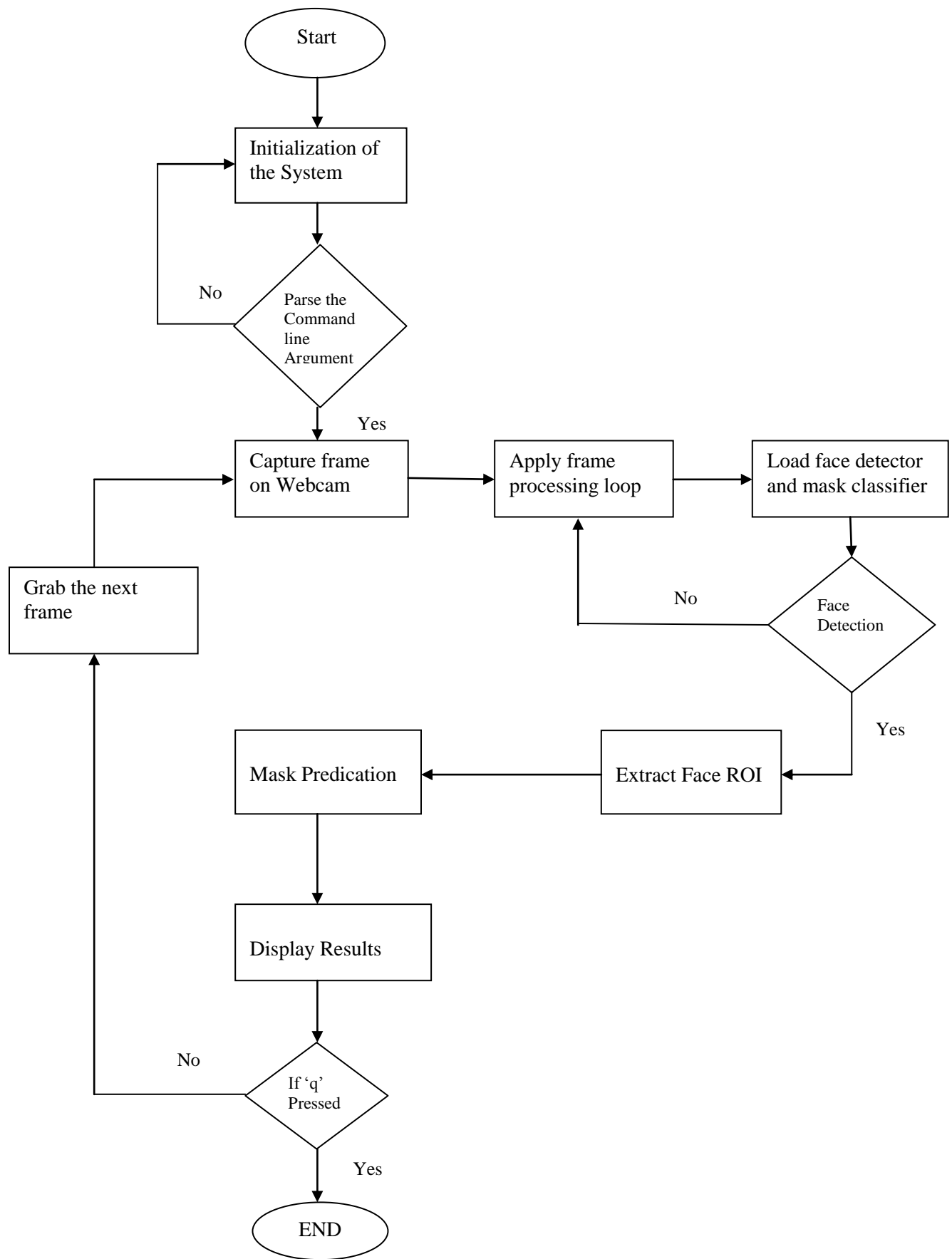


Fig 4.5 Activity Diagram

4.6 CLASS DIAGRAM

Class diagrams are the main building blocks of every object-oriented methods. The class diagram can be used to show the classes, relationships, interface, association, and collaboration.

Class diagrams are a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Purpose of Class Diagram:

- Shows static structure of classifiers in a system
- Diagram provides a basic notation for other structure diagrams prescribed by UML
- Helpful for developers and other team members too
- Business Analysts can use class diagrams to model systems from a business perspective

A UML class diagram is made up of:

- A set of classes and
- A set of relationships between classes

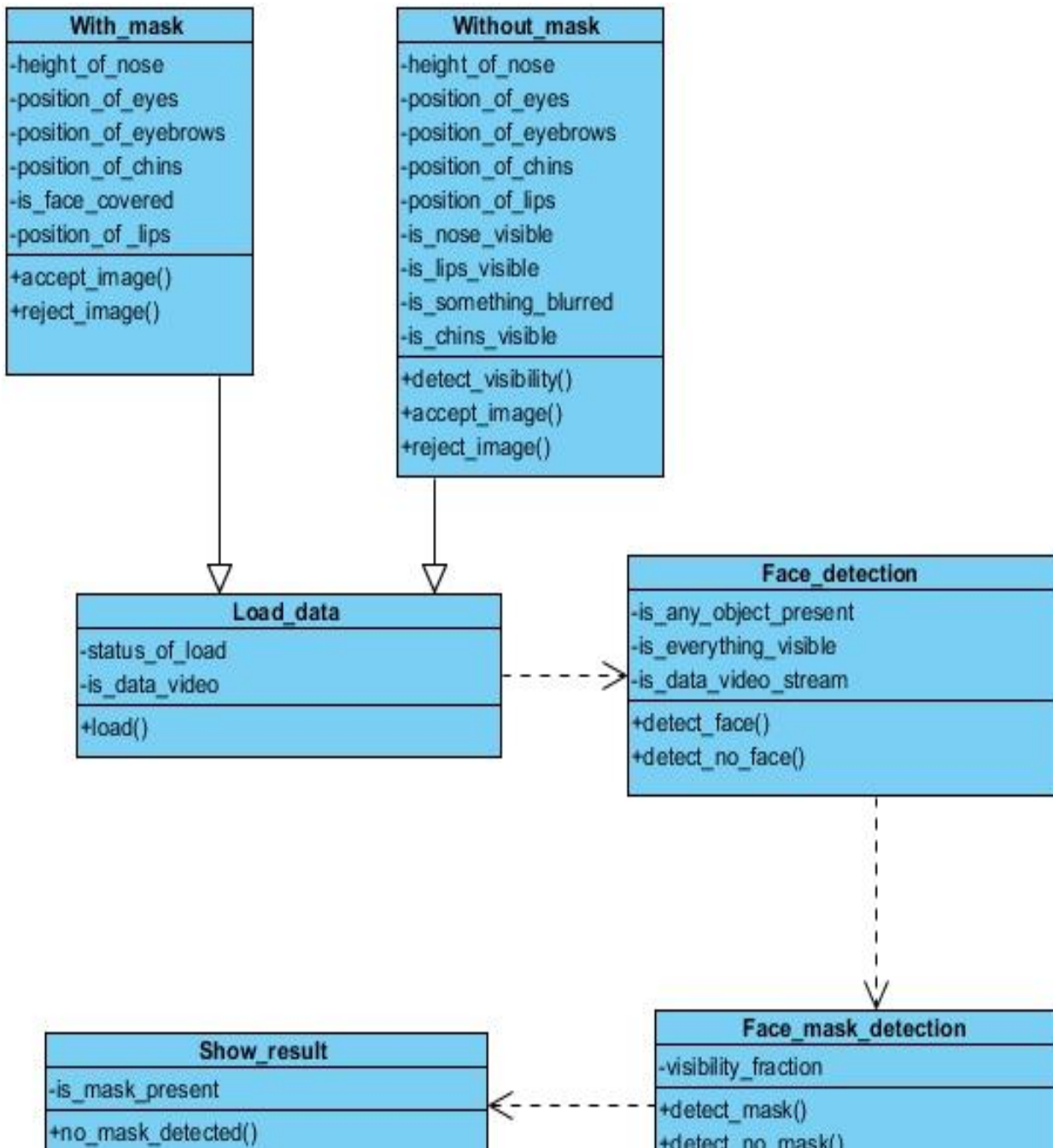












Fig 4.6 Class Diagram

CHAPTER – 5

CODE IMPLEMENTATION AND EXPLANATION

5.1 Directory structure

 <code>__pycache__</code>	File folder
 <code>css</code>	File folder
 <code>dataset</code>	File folder
 <code>face_detector</code>	File folder
 <code>detect_mask_image</code>	Python File
 <code>detect_mask_video</code>	Python File
 <code>mask_detector.model</code>	MODEL File
 <code>plot</code>	PNG image
 <code>search</code>	Python File
 <code>train_mask_detector</code>	Python File

5.2 Train the model

Step 1: Training the model with Keras and TensorFlow

In this step, we will train our face mask detector model using Keras, TensorFlow, and Deep Learning.

- ❖ The following file accepts our input dataset and creates our custom mask detector model. The explanation of the code is also given partwise.

File: `train_mask_detector.py`

```
# import the necessary packages
from TensorFlow.keras.preprocessing.image import ImageDataGenerator
from TensorFlow.keras.applications import import MobileNetV2
from TensorFlow.keras.layers import import AveragePooling2D
from TensorFlow.keras.layers import import Dropout
from TensorFlow.keras.layers import import Flatten
from TensorFlow.keras.layers import import Dense
from TensorFlow.keras.layers import import Input
from TensorFlow.keras.models import 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 argparse
import os

```

Explanation: We perform the following imports to allow the below mentioned tasks:

- Keras' ImageDataGenerator class to perform data augmentation.
- Loading the MobileNetV2 classifier for fine-tuning this model with pre-trained ImageNet weights.
- Building a new fully-connected (FC) head
- Preprocessing and loading image data
- scikit-learn (sklearn) for binarizing class labels, segmenting our dataset, and printing a classification report.
- imutils paths implementation will help us to find and list images in our dataset.
- And we'll use matplotlib to plot our training curves, argparse module to write user-friendly command-line interfaces.

```

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
    help="path to input dataset")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
    help="path to output loss/accuracy plot")
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path to output face mask detector model")
args = vars(ap.parse_args())

```

Explanation: We parse the following command line arguments to launch our script via terminal. It includes:

- -d/--dataset: The path to the input dataset of faces with and without masks
- -p/--plot: The path to the output training history plot, which will be generated using matplotlib
- -m/--model: The path to the resulting serialized face mask classification model

```

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

```

Explanation: The learning hyperparameter constants are declared including:

- Initial learning rate (INIT_LR)
- Number of training epochs (EPOCHS)
- Batch Size (BS)

```

# 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...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []

# loop over the image paths
for imagePath in imagePaths:
    # extract the class label from the filename
    label = imagePath.split(os.path.sep)[-2]

    # load the input image (224x224) and preprocess it
    image = load_img(imagePath, target_size=(224, 224))
    image = img_to_array(image)
    image = preprocess_input(image)

    # update the data and labels lists, respectively
    data.append(image)
    labels.append(label)

# convert the data and labels to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)

```

Explanation: In this section of code we do the following tasks:

- We get hold of all the ‘image path’ in the dataset.
 - Initialize the data[] and labels[] list.
 - We loop over the imagepaths to extract the class label, load and preprocess the input image into 224px*224px, also convert it to array format and scale the pixel intensities in the range of [-1,1].
 - We convert our training data into NumPy array format.
-

```

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

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(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,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")

```

Explanation: In this section of code, we do the following tasks:

- ‘one-hot’ encoding of the class labels: One-hot encoding is essentially the representation of categorical variables as binary vectors. These categorical values are first mapped to integer values. Each integer value is then represented as a binary vector that is all 0s (except the index of the integer which is marked as 1).
- dataset partitioning: Using `train_test_split()` method of scikit-learn we partition our dataset into 80% for training and the rest 20% for testing purposes.
- prepare for data augmentation: Now, we apply on the fly data mutation methods to our image dataset in order to improve generalization. The Keras `ImageDataGenerator` class accepts the original data, randomly transforms it (via rotation, zoom, shear, shift and flip), and returns only the new, transformed data.

```
# 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
```

Explanation: In this code block, we will prepare the MobileNetV2 architecture for fine-tuning by a three-step procedure:

- Load MobileNet with pre-trained ImageNet weights, leaving off the head of network.
- Replace the fully connected old head (i.e., where the actual class label predictions are made) with new FC head.
- Freezing the base layers, so that they won't be updated during the process of backpropagation, and only tuning the new head layer weights.

```
# 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)
```

Explanation: In this section of code, we are ready to compile and train the face mask detector network.

- The model is compiled using Adam optimizer, time-based learning rate scheduler via ‘decay’ parameter and ‘binary cross-entropy’ (because 2 classes classifier).
 - We then train the head of the network using the ‘aug’ object which provides batches of augmented data.
-

```
# 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(args["model"], save_format="h5")
```

Explanation: In this code-section:

- We make predictions on the test set, grabbing the highest probability class label indices.
 - Then, we print a classification report in the terminal for inspection.
 - We then serialize our face mask classification model to disk.
-

```
# 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_accuracy")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_accuracy")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

Explanation: In this code block, we plot our learning curves of the training and validation accuracy/loss curves and save the plot to disk.

Step 2: Execute the following command in Terminal after going into the project folder to train the model.

```
$ python train_mask_detector.py --dataset dataset
```

We now obtain our training history containing loss/accuracy curves as a separate 'plot.png'. We can refer that the curves demonstrate high accuracy with little signs of overfitting.

We can now proceed to apply our trained model to detect face masks in static images and real-time video streams appropriately outside our input dataset.

5.3 Implementing our COVID-19 face mask detector model for static images with OpenCV

❖ The following file loads our model and hence detects faces and classifies them as 'Mask' or 'No Mask' in static images given as an input.

File: detect_mask_image.py

```
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
import numpy as np
import argparse
import cv2
import os
```

Explanation: In this code block, we require the TensorFlow/Keras imports for loading the model and pre-processing the input image. We import OpenCV for image display and modifications.

```
-----
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
ap.add_argument("-f", "--face", type=str,
    default="face_detector",
    help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

Explanation: We parse the following command line arguments to launch our script via terminal. It includes:

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

```

# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
    "res10_300x300_ssd_iter_140000.caffemodel"])
net = cv2.dnn.readNet(prototxtPath, weightsPath)

# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
model = load_model(args["model"])

```

Explanation: In this code block, we load both the face detector and face mask classifier models.

```

# load the input image from disk, clone it, and grab the image spatial
# dimensions
image = cv2.imread(args["image"])
orig = image.copy()
(h, w) = image.shape[:2]

# construct a blob from the image
blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300),
    (104.0, 177.0, 123.0))

# pass the blob through the network and obtain the face detections
print("[INFO] computing face detections...")
net.setInput(blob)
detections = net.forward()

```

Explanation: In this code section, we load and pre-process an input image.

- We make a copy of the input image and grab the frame dimensions for scaling and display.
 - Image pre-processing is done by OpenCV's 'blobFromImage' method. We resize the image to 300px*300px and mean subtraction is performed.
 - Then we perform face detection to locate all the faces in the input image provided.
-

```

# 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 > args["confidence"]:
        # 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))

```

Explanation: In this section of code, we do the following:

- Loop over detections and extract the confidence parameter to measure against the confidence threshold.
- Compute bounding box value for a particular face and ensure that the box dimension falls within the boundary of the image.

```

# extract the face ROI, convert it from BGR to RGB channel
# ordering, resize it to 224x224, and preprocess it
face = image[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)
face = np.expand_dims(face, axis=0)

# pass the face through the model to determine if the face
# has a mask or not
(mask, withoutMask) = model.predict(face)[0]

```

Explanation: In this code block, we run the face ROI (Region of Interest) through our MaskNet model in the following ways:

- Extract the face ROI with the help of NumPy slicing.
- Pre-process the face ROI the same way we did during training.
- Perform prediction for “Mask” or “No Mask”.

```

# 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(image, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)

# show the output image
cv2.imshow("Output", image)
cv2.waitKey(0)

```

Explanation: In this block, we display our result.

- We determine the class label of “Mask” or “No Mask” based on the probabilities returned by the mask detector model.
- We then assign the color “GREEN” for “with_mask” and “RED” for “without_mask”.
- Then using OpenCV functions we draw a bounding box rectangle along and display the class as well as probability.
- Then finally image output is shown in an output window.

Execute the following command in Terminal to detect masks in static images present in ‘images’ folder.

```
$ python detect_mask_image.py --image images/pic1.jpeg
```

5.4 Implementing our COVID-19 face mask detector model in real-time video streams with OpenCV

- ❖ The following file loads our model and hence detects faces and classifies them as ‘Mask’ or ‘No Mask’ in real-time video streams.

File: detect_mask_video.py

```
# 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 argparse
import imutils
import time
import cv2
import os
```

Explanation: In this code block, we allow processing of every frame of our webcam stream.

- So, we import the VideoStream class along with the time module.
- The imutils module helps for its aspect aware resizing method.

```
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, (300, 300),
    (104.0, 177.0, 123.0))

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

```

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

```

Explanation: In this code block, we apply our frame processing loop in the following way:

- The `detect_and_predict_mask` function detects faces and then applies the face mask classifier to each face ROI.
- This function accepts three parameters:
 - `frame`: A frame from our stream
 - `faceNet`: The model used to detect where in the image faces are
 - `maskNet`: Our COVID-19 face mask classifier model
- We then construct a blob, detect faces and initialize the lists including faces (i.e., ROIs), locs (the face locations), and preds (the list of mask/no mask predictions).

```

# 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 > args["confidence"]:
        # 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))

```

Explanation: In this code section, we do the following:

- we filter out the weak detections by extracting the confidence and measuring it against the confidence threshold.
- We also compute the bounding boxes and ensure the dimensions are within the frame.

```

# 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)
face = np.expand_dims(face, axis=0)

# add the face and bounding boxes to their respective
# lists

```

```
faces.append(face)
locs.append((startX, startY, endX, endY))
```

Explanation: In this code block, we do the following:

- extract the face ROI and pre-process it
- append the face ROI and bounding boxes to their lists.

```
# 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
    preds = maskNet.predict(faces)

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

Explanation: Here we'll run the faces through the mask detector.

- We perform the inference on entire batch of faces in the frame so that our pipeline is faster.
- At last, we return the face bounding box locations and corresponding mask/non-mask predictions to the caller function.

```
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
    default="face_detector",
    help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
    default="mask_detector.model",
    help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())
```

Explanation: We parse the following command line arguments to launch our script via terminal. It includes:

- --face: The path to the face detector directory
- --model: The path to our trained face mask classifier
- --confidence: The minimum probability threshold to filter out the weak face detections

```
# load our serialized face detector model from disk
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
    "res10_300x300_ssd_iter_140000.caffemodel"])
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
```

```
# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
maskNet = load_model(args["model"])

# initialize the video stream and allow the camera sensor to warm up
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)
```

Explanation: In this code block, we have initialized the following:

- Face detector
- Face Mask detector
- Webcam video stream

```
# 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=400)

    # 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)
```

Explanation: In this code block, we loop over the frames and resize it and apply our mask prediction function.

```
# 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)
```

Explanation: In this code section, we post-process the face mask detection results:

- Unpack a face bounding box and with mask/without mask prediction
- Determine the label and color (GREEN and RED)
- Annotate the label and face bounding box

```
# 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()
```

Explanation: Finally, we capture the key presses once the frame is displayed. If the user presses the key 'q', the system breaks out of the loop and stops.

Execute the following command in Terminal to detect masks in real-time video streams using webcam.

```
$ python detect_mask_video.py
```

CHAPTER – 6

OUTPUT

6.1 Output:



Fig 6.1 Output of with-out mask



Fig 6.2 Output of with mask

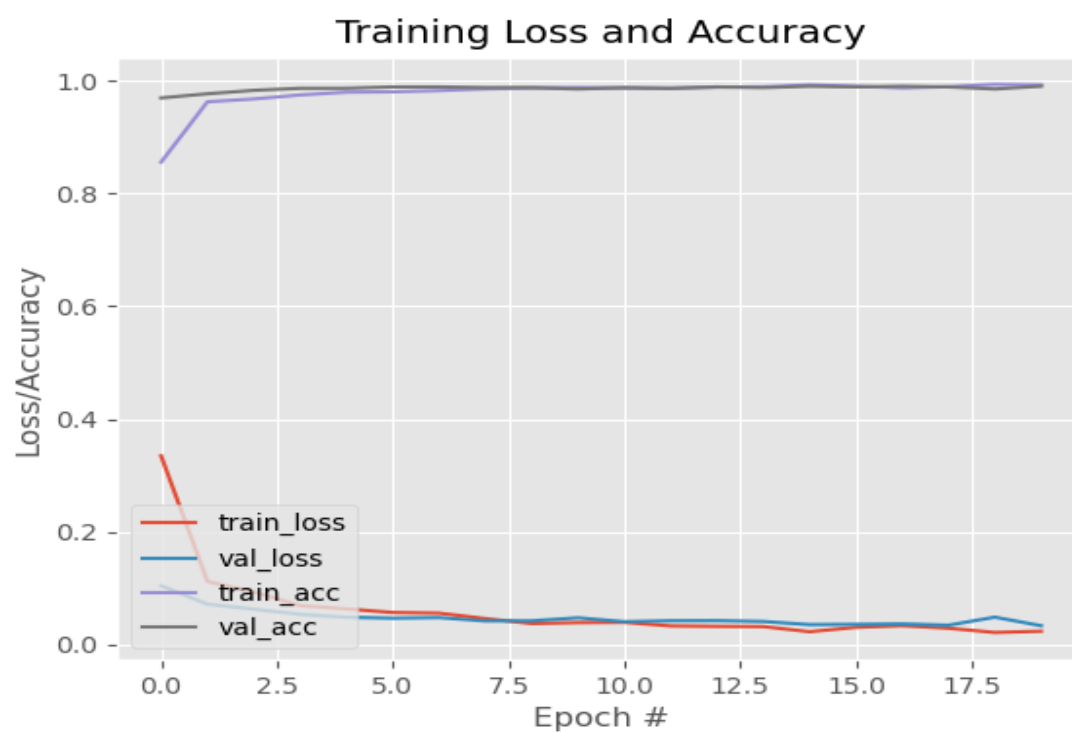
6.2 Results of Model Evaluation:

```
[INFO] evaluating network...
           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

[INFO] saving mask detector model...
```



CHAPTER – 7
ADVANTAGES, LIMITATIONS AND APPLICATIONS

7.1 ADVANTAGES:

- **Automated Tracking System**

We can track the crowded areas where people wore a mask or not without any person's help.

- **Cost Effective**

Since the whole process will be done by a computer, it means the total will be automated and done by the system itself, therefore, saving us the money which would have been otherwise spent on the labor cost to do that.

- **Easy to Manage**

Since the artificial intelligence-based attendance system is fully automated, managing the records and keeping a track of day-to-day activities will become much easier than the manual system. Everything will be done by the system. Many software's are programmed in such a way that it shows the exact time of Students Present at the class. All this can be done on a very large scale.

7.2 LIMITATIONS:

- **Data Privacy Breach**

Data in these systems means, billions of pictures, 1000s of hours of video footage saved into a hard drive. This calls for the point when this hard drive gets hacked or smuggled to get data of a particular person. The data can be easily tampered and used to in against a particular organization or person. Which can be very much dangerous. Therefore, companies that use Artificial intelligence-based attendance system have a very strict security to safeguard the information of their employees.

- **Low Reliability**

Sometimes there have been instances where the identity of a person is not able to get verified. There have been also cases where the identity of a person is verified with another person's identity. This means a person who is "X" is recognized as "Y" instead of "X". Times like this can be misused as criminals can loot or commit a crime in the name of other people. This means, that even though we have achieved a lot of advancement in the technology sector, there will always be a small gap between us and the ideal system.

7.3 APPLICATIONS:

- Schools & Colleges.
- Industries.
- Hospitals.
- Office.

CONCLUSION

An accurate and efficient face mask detection system has been developed which achieves comparable metrics with the existing state-of-the-art system. This project uses recent techniques in the field of computer vision and deep learning. Custom dataset was made from scratch using Bing Search API, Kaggle datasets and RMFD dataset, and the evaluation of the model on test dataset was found consistent. The system correctly detected the presence of face masks on human faces that it detected in static images as well as real-time video streams.

To create our face mask detector, we trained a two-class model with images of people *wearing masks* and *not wearing masks*.

We then fine-tuned our model using MobileNetV2 on our *mask/no mask* dataset and obtained an image classifier that was **98% accurate**.

We then took this face mask classifier and applied it to both *images* and *real-time video streams* by:

1. Detecting faces in the images/video. .
2. Extracting each individual face ROI. .
3. Applying our face mask classifier. .

Our face mask detector is accurate, and since we used the MobileNetV2 architecture, it's also computationally efficient and thus making it easier to deploy the model to embedded systems (Raspberry Pi, Google Coral, etc.).

This system can therefore be used in real-time applications which require face-mask detection for safety purposes due to the outbreak of Covid-19. This project can be integrated with embedded systems for application in airports, railway stations, offices, schools, and public places to ensure that public safety guidelines are followed.

SCOPE FOR IMPROVEMENT AND FUTURE WORK

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

- **Step 1:** Perform face detection
- **Step 2:** Apply our face mask detector to each face

The problem with this approach is that a face mask, by definition, obscures part of the face. If enough of the face is obscured, the face cannot be detected, and therefore, the face mask detector will not be applied.

To circumvent that issue, we should train a two-class object detector that consists of a **‘with_mask’** class and **‘without_mask’** class. Combining an object detector with a dedicated **‘with_mask’** class will allow improvement of the model in two respects.

First, the object detector will be able to naturally detect people wearing masks that otherwise would have been impossible for the face detector to detect due to too much of the face being obscured.

Secondly, this approach reduces our computer vision pipeline to a single step — rather than applying face detection and then our face mask detector model, all we need to do is apply the object detector to give us bounding boxes for people both **‘with_mask’** and **‘without_mask’** in a single forward pass of the network.

Not only is such a method more computationally efficient, it’s also more “elegant” and end-to-end. Also, we can develop an Android and/or web application for the same in future.

REFERENCES

- [1] <https://keras.io/api/applications/mobilenet/#mobilenetv2-function>
- [2] <https://arxiv.org/abs/1704.04861>
- [3] <https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/>
- [4] <https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset>
- [5] <https://github.com/AIZOOTech/FaceMaskDetection>
- [6] <https://www.pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/>
- [7] <https://gogul.dev/software/flower-recognition-deep-learning>
- [8] https://www.tensorflow.org/tutorials/images/transfer_learning
- [9] <https://towardsdatascience.com/detecting-faces-with-python-and-opencv-face-detection-neural-network-f72890ae531c>
- [10] <https://www.indulgenceexpress.com/msociety/2020/may/04/hyderabad-company-launches-ai-based-surveillance-tech-to-detect-face-mask-and-social-distancing-viol-24656.html>
- [11] <https://www.engadget.com/uber-face-mask-detection-technology-184137514.html>
- [12] <https://www.leewayhertz.com/face-mask-detection-system/>
- [13] <https://www.thehansindia.com/telangana/hyderabad-cameras-to-detect-face-masks-test-temp-soon-619505>
- [14] <https://www.kaggle.com/chandrikadeb/face-mask-dataset>
- [15] <https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/>