

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award
of Bachelor of Technology Degree in

Information Technology

By

VAJRAPU ULLAS
(Reg.No.37120080)



DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600119

AUGUST 2020



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" Grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119.

Phone: 044 - 2450 3150 / 51 / 52 / 54 / 55 Fax: 044 - 2450 2344

www.sathyabama.ac.in



DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Professional Training Report is the bonafide work of **VAJRAPU ULLAS (Reg. No. 37120080)** who carried out the project entitled "**FACE MASK DETECTION**" under our supervision from July 2020 to September 2020.

Internal Guide

Ms. Indira.K, M.Tech

Head of the Department

Dr. R. SUBHASHINI M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **VAJRAPU ULLAS** (Reg.No. **37120080**) hereby declare that the Project Report entitled “**FACE MASK DETECTION**” done by me under guidance of **Ms.INDIRA.K** at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE :

PLACE:

SIGNATURE OF THECANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of SATHYABAMA or their encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T SASIKALA M.E Ph.D**, Dean, School Of Computing and **Dr .R. SUBHASHINI M.E Ph.D** ,Head of the Department, Dept .of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms.INDIRA.K,M.Tech** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Information Technology who were helpful in many ways for the completion of the project.

This certificate is computer generated and can be verified by scanning the QR code given below. This will display the certificate from the NPTEL repository, <https://nptel.ac.in/noc/>

Roll No: NPTEL20CS36S1087337

To
VAJRAPU ULLAS
H.NO165,MAIN
ROAD,MADHAVARAM(V),MUNAGALA(M),SURYAPET(D),TELANGANA(S).
SURYAPET
TELANGANA-508213
PH. NO :9866624912



No. of credits recommended by NPTEL:1

An additional 1 credit may be awarded if the University deems it fit, based on the actual student effort involved.



NPTEL Online Certification

(Funded by the Ministry of HRD, Govt. of India)



This certificate is awarded to

VAJRAPU ULLAS

for passing the course

Python for Data Science

with Score* **82** %



Devendra Jalihal

Prof. Devendra Jalihal
Chairman
Centre for Continuing Education, IITM

Jan-Feb 2020
(4 week course)

Prof. Andrew Thangaraj

Prof. Andrew Thangaraj
NPTEL Coordinator
IIT Madras



Indian Institute of Technology Madras



***Continuous online assessment score**

To validate and check scores: <https://nptel.ac.in/noc>



INTERNSHIP CERTIFICATE

This certificate is awarded to

VAJRAPU ULLAS

In appreciation for your outstanding accomplishment
in the company as intern at
DAURINDIA TECHNOLOGIES
in **Machine Learning and CNN.**
from **JUNE 01,2020 to JULY 15,2020.**

We wish you best of luck for your future.



Date: JULY 16,2020

SURENDRA TYAGI
Administrator

DAURINDIA TECHNOLOGIES

614-Janakpuri Bareilly-243122

 www.daurindia.com

 care@daurindia.com

 daurindiatech@gmail.com

+91-8307329272

+91-7800695381

Certificate Id :DIT20570

ABSTRACT

The corona virus COVID-19 pandemic is causing a global health crisis so the effective protection methods 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 lock down to prevent virus transmissions. Reports indicate that wearing facemasks while at work clearly reduces the risk of transmission. An efficient and economic 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 face detection from a live stream via our webcam. We will use the dataset to build a COVID-19 face mask detector with computer vision using Python, OpenCV, and Tensor Flow and Keras. Our goal is to identify whether the person on image/video stream is wearing a face mask or not with the help of computer vision and deep learning .By using this model we can check the whether the persons are wearing the mask or not to prevent from the virus. By using this model we can reduce the time to check whether the person is wearing the mask or not.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
1	INTRODUCTION	
	1.1 Machine Learning	1
	1.2 Computer Vision	2
	1.3 Deep Learning	4
	1.4 OpenCV	5
	1.5 Tensorflow	6
	1.6 Keras	6
2	AIM AND SCOPE	
	2.1 Aim	8
	2.2 Inputs and Outputs	8
	2.3 Scope	8
3	ALGORITHM AND METHODS	
	3.1 Data Preprocessing	10
	3.2 Deep Learning Frameworks	10
	3.3 MobileNetV2	10
	3.3.1 MobileNetV2 Convolutional Blocks	11
	3.3.2 Overall Architecture	12
4	REQUIREMENTS SPECIFICATION	
	4.1 Software Requirements	14
	4.2 Hardware Requirements	14
5	RESULT AND CONCLUSION	
	5.1 Result	15

5.1.1 Face Mask Detection Using the images	15
5.1.2 Face Mask Detection Using the Streaming Video	16
5.2 Accuracy of Trained Model	17
5.2.1 Precision	17
5.2.2 Recall	17
5.2.3 Accuracy	18
5.2.4 F1 score	18
5.3 Plot Diagram of Model	19
5.4 Conclusion	19
REFERENCES	21
APPENDIX	22

LIST OF FIGURES

FIGURE.NO	TITLE OF THE FIGURE	PAGE.NO
1.1	Machine Learning Outlook	2
1.2	Computer Vision	3
1.3	Deep Learning	4
3.1	Flow Chart	9
3.2	MobileNetV2 Convolutional Blocks	11
3.3	MobileNetV2 Domain	11
3.4	MobileNetV2 Building Block	12
5.1	Image Without Mask	15
5.2	Image With Mask	15
5.3	Video Without Mask	16
5.4	Video With Mask	16
5.5	Report of Trained Model	17
5.6	Graphical Representation of Model	20

LIST OF ABBREVIATIONS

ACRONYM

EXPANSION

OPENCV

Open Source Computer Vision Library

TP

True Positives

FP

False Positives

FN

False Negatives

FP

False Positives

CNN

Convolution Neural Network

CHAPTER 1

INTRODUCTION

Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry. Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java. Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

1.1 Machine Learning:

Machine learning (ML) is the study of computer algorithms that improve automatically through experience .It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks. Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Machine Learning

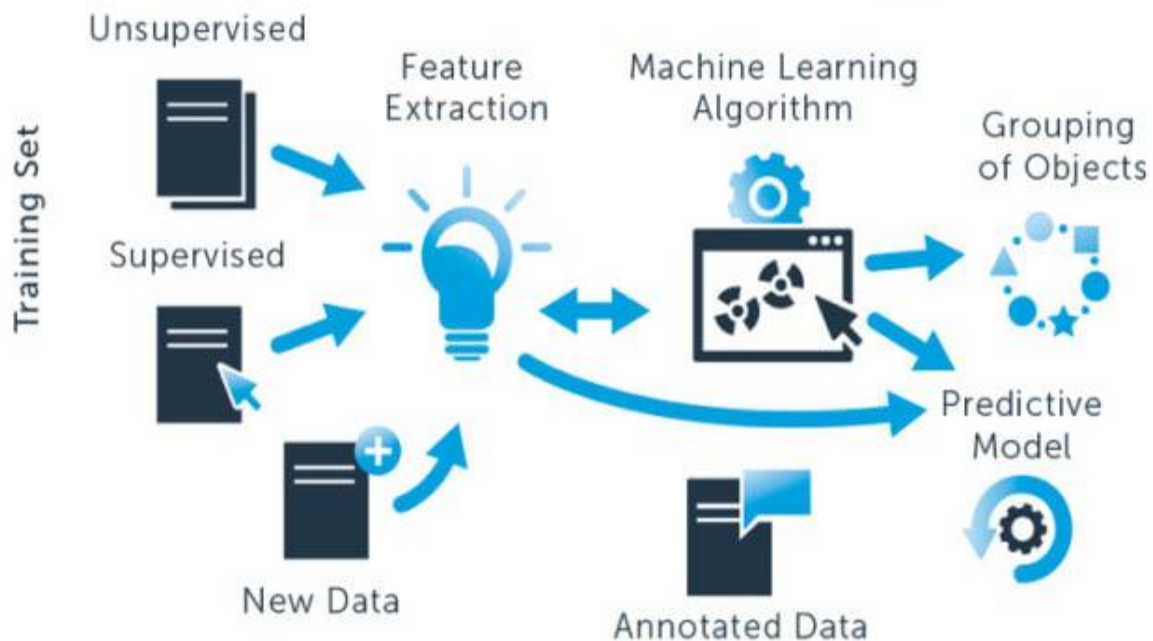


Fig-1.1: Machine Learning Outlook

Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the “signal” or “feedback” available to the learning system:

Supervised learning: The computer is presented with example inputs and their desired outputs, given by a “teacher”, and the goal is to learn a general rule that maps inputs to outputs.

Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

Reinforcement learning: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). As it navigates its problem space, the program is provided feedback that's analogous to rewards, which it tries to maximize.

1.2 Computer Vision:

Computer vision is an interdisciplinary scientific field that deals with how computers can

gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do, Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high dimensional data from the real world in order to produce numerical or symbolic information, e.g. in the forms of decisions, Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that make sense to thought processes and can elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory. The scientific discipline of computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, multidimensional data from a 3D scanner or medical scanning device. The technological discipline of computer vision seeks to apply its theories and models to the construction of computer vision systems. Computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.



Fig-1.2: Computer Vision

Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. It involves the development of a theoretical and algorithmic basis to achieve automatic visual

understanding. As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multidimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

1.3 Deep Learning:

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features. Deep learning algorithms seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features.

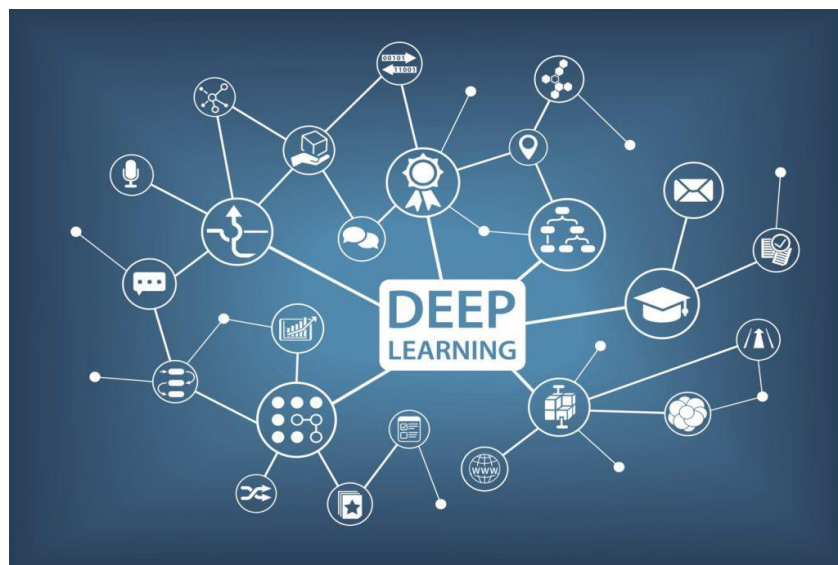


Fig-1.3: Deep Learning

The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning. Deep learning excels on problem domains where the inputs (and even output) are analog. Meaning, they are not a few quantities in a tabular

format but instead are images of pixel data, documents of text data or files of audio data. Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

1.4 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. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies .Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, Video Surf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

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 OpenCV 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 template interface that works seamlessly with STL containers.

1.5 Tensorflow:

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google, TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units).

Tensor Flow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

The name Tensor Flow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google. Unlike other numerical libraries intended for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems, not least RankBrain in Google search and the fun Deep Dream project .It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machines.

1.6 Keras:

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides. Keras contains

numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. Keras is a minimalist Python library for deep learning that can run on top of Theano or Tensor Flow. It was developed to make implementing deep learning models as fast and easy as possible for research and development .It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license.

Keras was developed and maintained by François Chollet, a Google engineer using four guiding principles:

- a) Modularity: A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.
- b) Minimalism: The library provides just enough to achieve an outcome, no frills and maximizing readability.
- c) Extensibility: New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.
- d) Python: No separate model files with custom file formats. Everything is native Python. Keras is designed for minimalism and modularity allowing you to very quickly define deep learning models and run them on top of a Theano or TensorFlow backend.

CHAPTER 2

AIM AND SCOPE

2.1 Aim:

The main to identify the person on image/video stream wearing face mask with the help of computer vision and deep learning algorithm.

2.2 Inputs and Outputs:

a)Inputs:

Importing all the required packages like keras, OpenCV, Tensorflow, matplotlib, numpy, sklearn and other required packages.

The dataset contains two class of different images:

- a) With_mask: 690 images
- b) Without_mask: 686 images

The set of images for testing and the streaming video which has the camera.

b)Output:

It will detect mask or the without mask in the images or the streaming video.

2.3 Scope:

This project helps us to spread the awareness among people using face mask properly. It detects the face mask on your face whether the person is hiding his/her face by mask or not. it also check the face mask is properly cover your face both nose and mouth.

CHAPTER 3

ALGORITHM AND METHODS

The system focuses on how to identify the person on image/video stream wearing face mask with the help of computer vision and deep learning algorithm by using the OpenCV, Tensor flow, Keras and PyTorch library.

Approach:

- 1) Train Deep learning model (MobileNetV2)
- 2) Apply mask detector over images / live video stream

Flow Chart:

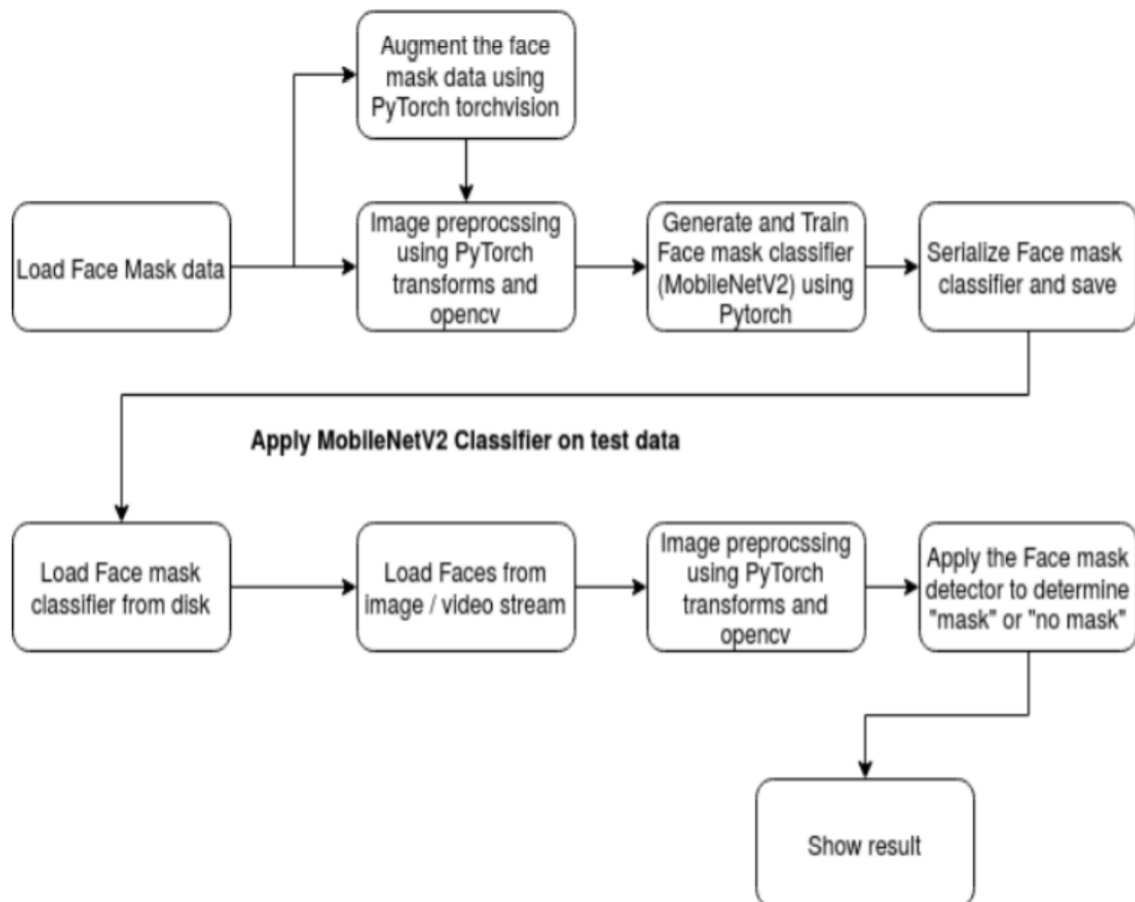


Fig 3.1 Flow Chart

The majority of 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.

3.1 Data preprocessing

In the preprocessing the raw input images to convert them into clean versions, which could be feed to a neural network machine learning model.

3.2 Deep Learning Frameworks

- a) TensorFlow
- b) Pytorch
- c) Keras
- d) MXNet
- e) MobileNetV2,etc.

3.3 MobileNetV2

As a lightweight deep neural network, MobileNet has fewer parameters and higher classification accuracy. In order to further reduce the number of network parameters and improve the classification accuracy, dense blocks that are proposed in DenseNets are introduced into MobileNet. In Dense-MobileNet models, convolution layers with the same size of input feature maps in MobileNet models are taken as dense blocks, and dense connections are carried out within the dense blocks. The new network structure can make full use of the output feature maps generated by the previous convolution layers in dense blocks, so as to generate a large number of feature maps with fewer convolution cores and repeatedly use the features. By setting a small growth rate, the network further reduces the parameters and the computation cost. Two Dense-MobileNet models, Dense1-MobileNet and Dense2-MobileNet, are designed. Experiments show that Dense2-MobileNet can achieve higher recognition accuracy than MobileNet, while only with fewer parameters and computation cost.

3.3.1 MobileNetV2 Convolutional Blocks

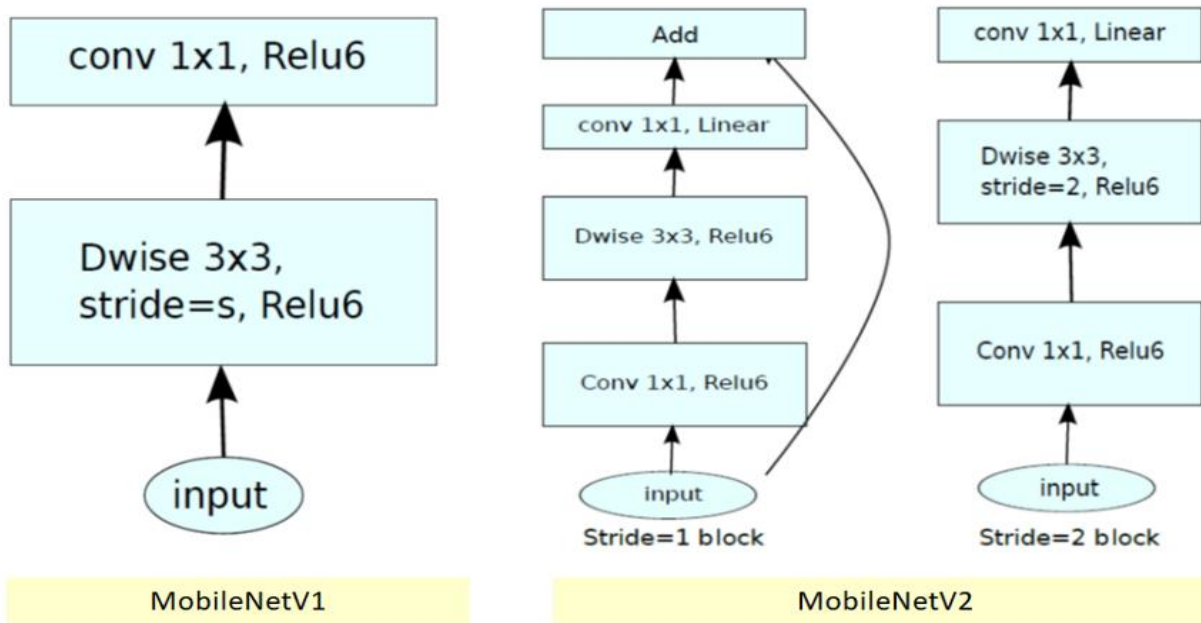


Fig 3.2 MobileNetV2 Convolutional Blocks

In MobileNetV2, there are two types of blocks. One is residual block with stride of 1. Another one is block with stride of 2 for downsizing. There are 3 layers for both types of blocks. This time, the first layer is 1x1 convolution with ReLU6. The second layer is the depthwise convolution. The third layer is another 1x1 convolution but without any non-linearity. It is claimed that if ReLU is used again, the deep networks only have the power of a linear classifier on the non-zero volume part of the output domain.

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

Fig 3.3 MobileNetV2 Domain

If the input got 64 channels, the internal output would get $64 \times t = 64 \times 6 = 384$ channels.

3.3.2 Overall Architecture

The MobileNetV2 introduces the two new features to the architecture.

- a) Linear bottlenecks between the layers and
- b) Shortcut connections between the bottlenecks.

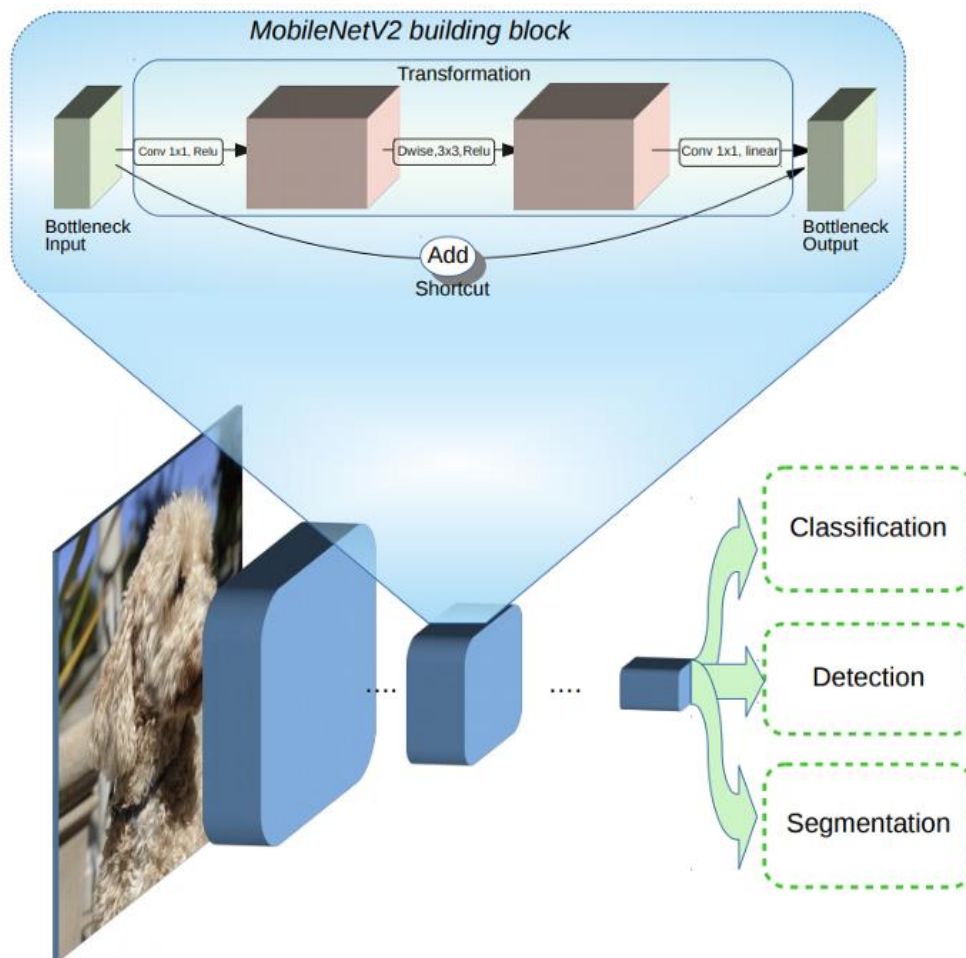


Fig 3.4 MobileNetV2 Building Block

Typical MobilenetV2 architecture has as many layers listed below, In Pytorch we can use the models library in TorchVision to create the MobileNetV2 model instead of defining/building our own model. The weights of each layer in the model are predefined

based on the ImageNet dataset. The weights indicate the padding, strides, kernel size, input channels and output channels. MobileNetV2 was chosen as an algorithm to build a model that could be deployed on a mobile device. A customized fully connected layer which contains four sequential layers on top of the MobileNetV2 model was developed.

The layers of the MobileNetV2 are

1. Average Pooling layer with 7x7 weights
2. Linear layer with ReLu activation function
3. Dropout Layer
4. Linear layer with Softmax activation function with the result of 2 values.

The final layer softmax function gives the result of two probabilities each one represents the classification of “mask” or “not mask”

CHAPTER 4

REQUIREMENTS SPECIFICATION

4.1 SOFTWARE REQUIRMENTS

The software requirements are description of features and functionality of the target system. There are different types of packages to be installed separately before running the project. The prerequisites of the system must be there like windows, etc .The RAM of the cpu must be minimum of 4GB to run smartly and quickly .If the RAM is high the application will respond with high speed.

Software Requirements are:

Operating System : Windows 7/8/10

CPU RAM : Minimum 4 GB required

Packages : Keras,Python,Numpy,Matplotlib,Basic Software.

4.2 HARDWARE REQUIREMENTS

The hardware requirements of the system like camera with high quality to capture the image in the streaming videos. The minimum harddisk is required to install or run the application. We should have the images to set and the mask for the streaming video.

Hardware Requirements are:

Hard drive space : 10GB for 64-bit

Images : with mask or without mask and mask

CHAPTER 5

RESULT AND CONCLUSION

5.1 Result

5.1.1 Face Mask Detection Using the images:



Fig 5.1 Image Without Mask



Fig 5.2 Image With Mask

5.1.2 Face Mask Detection Using the Streaming Video:

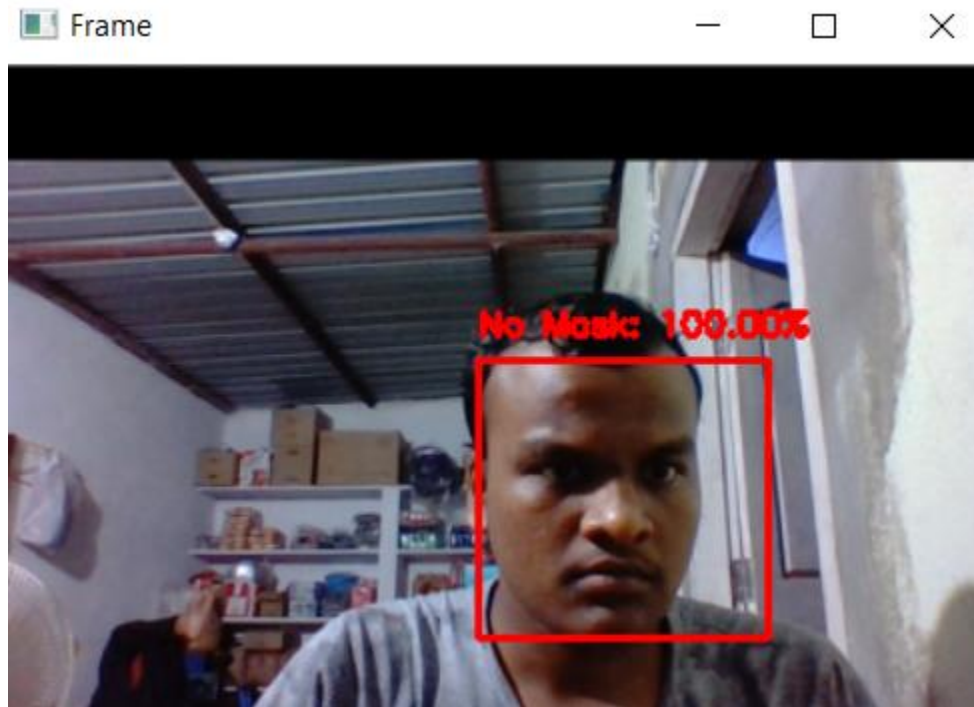


Fig 5.3 Video Without Mask



Fig 5.4 Video With Mask

5.2 Accuracy of Trained Model

	precision	recall	f1-score	support
with_mask	0.98	1.00	0.99	138
without_mask	1.00	0.98	0.99	138
accuracy			0.99	276
macro avg	0.99	0.99	0.99	276
weighted avg	0.99	0.99	0.99	276

Fig 5.5 Report of Trained Model

5.2.1 Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to the low false positive rate. We have got 0.98 precision with mask and 1.00 precision for without mask which is pretty good.

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

Where,

TP=True Positives

FP=False Positives

5.2.2 Recall

Recall is the ratio of correctly predicted positive observations to the all observations in actual class. We have got recall for with mask of 1.00 and recall for without mask of 0.98 which is good for this model as it is above 0.5.

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

Where,

TP=True Positives

FN=False Negatives

5.2.3 Accuracy

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have

high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. We have got 0.99 is which is good for this model.

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$

Where,

TP=True Positive

TN=True Negatives

FP=False Positives

FN=False Negatives

5.2.4 F1 score

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall. We got 0.99 as the F1 score.

$$\text{F1 score} = \frac{2(\text{Recall} * \text{Precision})}{(\text{Recall} + \text{Precision})}$$

5.2.5 True Positives (TP)

These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes.

5.2.6 True Negatives (TN)

These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no.

5.2.7 False Positives (FP)

When actual class is no and predicted class is yes.

5.3.8 False Negatives (FN)

When actual class is yes but predicted class in no.

5.2.9 Macro-average

Macro-average method can be used when you want to know how the system performs overall across the sets of data. You should not come up with any specific decision with this average. The macro average will take the average of precision and

recall of the system on different sets.

5.2.10 Weighted Average

Weighted average is a calculation that takes into account the varying degrees of importance of the numbers in a data set. In calculating a weighted average, each number in the data set is multiplied by a predetermined weight before the final calculation is made. The formula for finding the weighted average is the sum of all variables multiplied by its weight, then divided by the sum of the weights.

5.3 Plot Diagram of Model

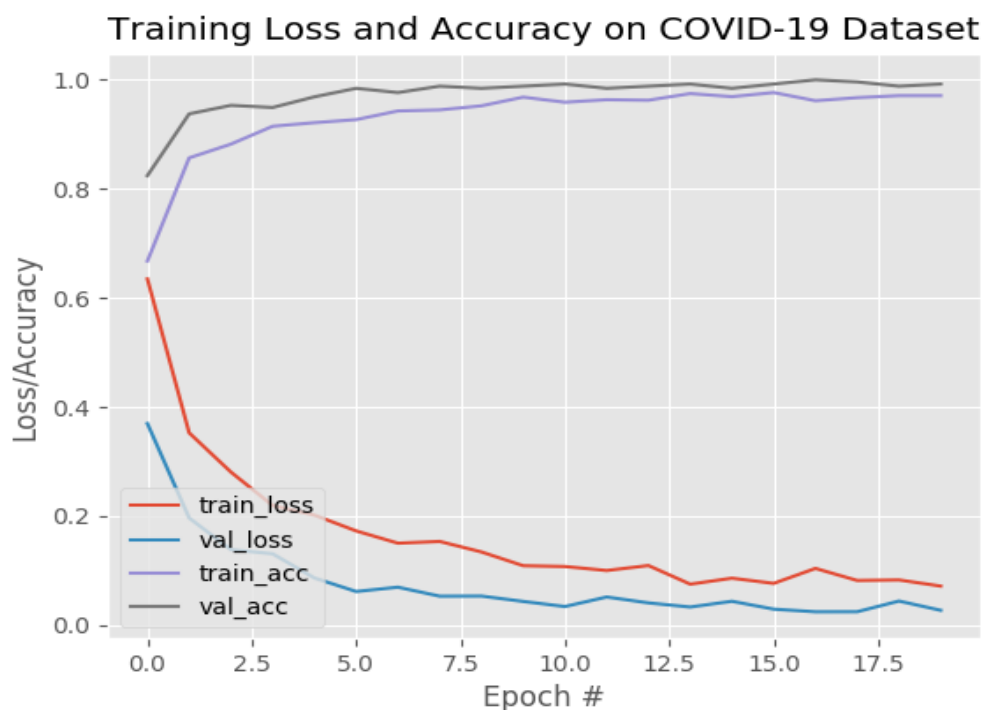


Fig 5.6 Graphical Representation of Model

5.4 Conclusion:

As the technology are blooming with emerging trends the availability so we have novel face mask detector which can possibly contribute to public healthcare. The architecture consists of MobileNet as the backbone it can be used for high and low computation scenarios. In order to extract more robust features, we utilize transfer learning to adopt weights from a similar task face detection, which is trained on a very large dataset.

We used OpenCV, tensor flow, keras , Pytorch and CNN to detect whether

people were wearing face masks or not. The models were tested with images and real-time video streams. The accuracy of the model is achieved and, the optimization of the model is a continuous process and we are building a highly accurate solution by tuning the hyper parameters. This specific model could be used as a use case for edge analytics. Furthermore, the proposed method achieves state-of-the-art results on a public face mask dataset. By the development of face mask detection we can detect if the person is wearing a face mask and allow their entry would be of great help to the society.

REFERENCES

- 1) P. A. Rota, M. S. Oberste, S. S. Monroe, W. A. Nix, R. Campagnoli, J. P. Icenogle, S. Penaranda, B. Bankamp, K. Maher, M.-h. Chen et al., "Characterization of a novel coronavirus associated with severe acute respiratory syndrome," *science*, vol. 300, no. 5624, pp. 1394–1399, 2003.
- 2) S. Feng, C. Shen, N. Xia, W. Song, M. Fan, and B. J. Cowling, "Rational use of face masks in the covid19 pandemic," *The Lancet Respiratory Medicine*, 2020.
- 3) Y. Fang, Y. Nie, and M. Penny, "Transmission dynamics of the covid-19 outbreak and effectiveness of government interventions: A data driven analysis," *Journal of medical virology*, vol. 92, no. 6, pp. 645–659, 2020.
- 4) P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features", *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I-I, Dec 2001.
- 5) S. Yang, Y. Xiong, C. C. Loy et al., "Face detection through scale-friendly deep convolutional networks," 2017.

APPENDIX

Train_mask_Detector.py

USAGE

python train_mask_detector.py --dataset dataset

import the necessary packages

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.layers import AveragePooling2D

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Input

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.applications.mobilenet_v2 import preprocess_input

from tensorflow.keras.preprocessing.image import img_to_array

from tensorflow.keras.preprocessing.image import load_img

from tensorflow.keras.utils import to_categorical

from sklearn.preprocessing import LabelBinarizer

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report

from imutils import paths

import matplotlib.pyplot as plt

import numpy as np

import argparse

import os

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())

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

# 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)

# 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")

# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,

```

```

input_tensor=Input(shape=(224, 224, 3)))

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

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

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

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

# train the head of the network
print("[INFO] training head...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,

```

```

validation_data=(testX, testY),
validation_steps=len(testX) // BS,
epochs=EPOCHS)

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

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

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

# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save(args["model"], save_format="h5")

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

```

```
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

Detect_mask_image.py

```
# USAGE
```

```
# python detect_mask_image.py --image examples/example_01.png
```

```
# 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
```

```
import numpy as np
```

```
import argparse
```

```
import cv2
```

```
import os
```

```
# 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())
```

```
# 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"])

# 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()

# 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))

    # 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]

    # 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(10000)

```

Detect_mask_video.py

```

# USAGE
# python 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

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 = []

# 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))

# 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))

# 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)

# 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())

# 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(0.02)

# 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)

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

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

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

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

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

```

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

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop(10000)
```