iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** VI **Month of publication:** June 2022

DOI: https://doi.org/10.22214/ijraset.2022.44512

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VI June 2022- Available at www.ijraset.com

Face Mask Detection Alert System Using Deep Learning

Dr. A. Jagan[1], P. Ganesh[2], R. Balapriya[3]
[1]Associate Professor & Head, [2]Assistant Professor, [3]PG Student, Computer Science Department, Surya Groups Of Institution, Villupuram, Tamilnadu, India

***Abstract:** Covid-19 pandemic has brought significant by spreading as of recently everywhere on the world. The effect of COVID-19 has been fallen on practically all areas of development. The medical services framework is going through an emergency. Numerous prudent steps have been taken to lessen the spread of this illness where wearing a mask is one of them. In this paper, an effective way is proposed that confine the development of Corona virus by discovering individuals who are not wearing any facial mask in a brilliant city network where Closed-Circuit Television (CCTV) cameras make close observance. While an individual without a mask is identified, the relating authority is educated through the city organization.*

I. INTRODUCTION

A. Overview

COVID-19 pandemic has had a long-lasting impact in many countries worldwide since December 2019. It originated in Wuhan, China. the planet Health Organization (WHO) as on March 11, 2020, declared it as a deadly diseases that gained its roots across the world and severely affected 114 countries. Every medical professionals, healthcare organizations, medical practitioners and researchers are in explore for a correct vaccines and medicines to beat this deadly disease. The virus spreads through air space to opposite person when the infected person communicate or sneezers, the water droplets from their nose or mouth disseminate through the air and affect other peoples within the vicinity . While other problems of social distancing and sanitization are addressed as yet, the difficulty of mask detection has not yet been adequately addressed. Wearing a mask during this pandemic may be a critical defense and is most important step in times when social distancing is tough to keep up. Wearing a mask is important, particularly for those who are at a higher risk of extreme illness from COVID-19 diseases. it's found that the spread of COVID-19 is principally among folks that are in immediate contact with each other (nearly about 6 feet), it is spread by people that don't have symptoms and are unaware of the actual fact that they're infected. So Centers for Disease Control and Prevention recommended all people 2 years old and older to wear a mask publicly areas especially when other social distancing measures are difficult to keep up. Hence by reducing the danger of transmission of this deadly virus from an infected person to a healthy, the virus' spread and disease severity is reduced to a good extent.

To monitor that individuals are following this basic safety principle, a technique should be developed. A face mask detector system are often implemented to test this. mask detection means to spot whether someone is wearing a mask or not. the primary step to acknowledge the presence of a mask on the face. Face detection is one amongst the applications of object detection and might be employed in many areas like security, biometrics, enforcement and more. There are many detector systems developed round the world and being implemented. However, all this science needs optimization; a higher, more precise detector, because the globe cannot afford any longer increase in corona cases.

In this project we are training our model in such a way that it should recognize or find the person without mask in the premises, schools, colleges, office, etc. We are training our model by real world dataset and then by live video streaming. In this way we will be able to identify the person not wearing a mask. So, to take care of this problem we don't need any guard or person who keeps a watch on people. This system aims at classifying whether a person is wearing a mask or not by taking input from Images and Real time Video streaming. If person is weared a mask then it will show a green frame around the face of that person and give the signal "Mask with percentage" and if person is not wearing a mask, then it will show a red frame around the faces of that person and give the signal "No Mask with percentage" by sounding an alarm or sending an email warning to police, authority, or an observer. This system allows them to see who isn't wearing a mask on their faces. The Face Mask Detection model is based on computer vision and deep learning. The model is integration between deep learning and classical machine learning techniques with OpenCV, tensor flow and Keras.

B. Objectives

The main objective of this project is to provide some effective technology for preventing the spread of Corona virus.
Primary objectives behind the development of this system are as follows: -

1) Prevent the spread of Corona virus by promoting the use of face masks with the help of effective technology to detect the face mask.
2) Help to take necessary precautions for the safety of society by predicting the future outbreaks of COVID-19.
3) Ensure a safe working environment.
4) Save the lives of people

C. Scope Of The Project

This project uses deep learning techniques to distinguish if the person is wearing a face mask or not using a pre-trained AI model through TensorFlow and other Python libraries. Footage is recorded via webcam or surveillance camera and processed though the Python program in real time. Through protocols such as SMTP and MIME, an alert is logged and sent via email to the administrator/authorities with an image attachment of the person's face who violated mask guidelines if the conditions are met for someone not wearing a mask.

II. SYSTEM ANALYSIS

A. Existing System

In Existing system the device might be deployed in high-traffic areas to keep a close eye on people. If we consider the cost estimation for implementing the project, it will be almost of no cost as most of the metropolitan cities already have cameras installed in public places. Camera; which is the only main requirement of the model is already available.This model is based on neural networks. A neural network is a network or circuit of neurons, which is also called an artificial neural network and is made up of artificial neurons or nodes. Some of the important types of neural networks are Artificial Neural Networks (ANN), Convolution Neural Networks (CNN), Recurrent Neural Networks (RNN). Different fields including Math, Physics, and Neuroscience have all had an impact on Artificial Neural Networks. Neuroscience made a significant contribution to the original inspiration for Artificial Neural Networks. Consider the recent success of biologically inspired Convolutional Neural Networks (CNN).
The traditional CNN is the most commonly used deep learning algorithm, where the structure is mainly composed of convolution layers, pooling layers, fully connected layer s, and the activation functions. Each convolution kernel is connected to the feature maps parts. The input is connected to the output elements in the fully connected layer.

B. Proposed System

The proposed system focuses on how to identify the person on image/video stream wearing face mask or not with the help of computer vision and deep learning algorithm by using Open CV, Tensorflow, Keras, Pytorch, Numpy, MobileNetV2 and Pygame where, Open CV: It is an open-source library for machine learning used to process images, videos to identify faces.
Tensorflow: It is an open-source artificial intelligence library, using data flow graphs to build models.
Keras: It is a free open-source python library used for developing and evaluating deep learning models.
Pytorch: It is an open-source machine learning library used in deep learning application.
Numpy: It is used to do mathematical operations on array MobileNetV2: It is used to extract the data fast and easily.
Pygame: This library is used to create a peep sound if person is not wearing a mask
MobileNetV2 architecture is used which are similar to Convolutional neural network where the images are processed through different layers.
For building the base model using Mobilenet, we use imagenet which has predefined weights for images to give better results.
MobileNetv2 architecture consists of three convolutional layers.

- 1×1 convolution
- Depth wise CNN
- Bottleneck layer

The methodology of the proposed system is comprised of two main steps: The first step is the creation of a face-matching model using deep learning and traditional machine learning techniques. The main challenge was to create a dataset that is composed of faces with and without face masks. A computer vision-based face detector was built using the created dataset, OpenCV, andPython with TensorFlow, withal in our custom machine learning framework. The computer vision and deep learning techniques were used to identify whether the person is wearing a face mask or not. This helps in expediting the proliferation of computer vision in the currently nascent areas such as digital signage, autonomous driving, video recognition, customer service, language translation, and mobile apps.

1) Advantages

- MobileNets are faster than CNN.
- They use lesser parameters and produce higher accuracy.
- It's suitable for mobile devices, or devices with low computing power.
- Linear bottlenecks between the layers: Experimental evidence suggests that using linear layers is crucial as it prevents nonlinearities from destroying too much information. Using non-linear layers in bottlenecks indeed hurts the performance by several percent, further validating our hypothesis

Shortcut connections between the bottlenecks.

III. SYSTEM REQUIREMENTS

A. Introduction

System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently. Failure to meet these requirements can result in installation problems or performance problems. The former may prevent a device or application from getting installed, whereas the latter may cause a product to malfunction or perform below expectation or even to hang or crash.

1) Software Requirements: A software requirement specification is a description of a software system to be developed. It lays out functional and non-functional requirements and it also describes the operating system and tool used in the system and they are:

Operating System : Any Operating with Internet Connectivity
Language : Python 3.6
Packages required :

- tensorflow>=2.3.0
- numpy==1.18.2
- scikit-learn==0.22.2
- pandas==1.0.3

2) Hardware Requirements: Hardware specifications are technical description of the computer's components and capabilities. Processor speed, model and manufacturer, etc.,. So the hardware components required for the proposed system are:

Processor : Intel i3
Hard Disk : 1TB
RAM : 4-8 GB

B. Specifications

1) Python

Python could be a high-level, interpreted, interactive and object-oriented scripting language. Python is intended to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it's fewer syntactical constructions than other languages. Python could be a MUST for college students and dealing professionals to become an excellent coder specially once they are working in Web Development Domain.

Key advantages of learning Python:

- Python is Interpreted − Python is processed at runtime by the interpreter.
- Python is Interactive − Python prompt can be used to interact with the interpreter on to write your programs.
- Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python may be a Beginner's Language − Python could be a great language for the beginner-level programmers and supports the event of a large range of applications from simple text processing to WWW browsers to games.

Following are important characteristics of Python Programming −

- It supports functional and structured programming methods still as OOP.
- It may be used as a scripting language or is compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking. It supports automatic trash collection.
- It is easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

2) Deep Learning

Deep learning relies on the branch of machine learning, which may be a subset of computer science. Since neural networks imitate the human brain then deep learning will do. In deep learning, nothing is programmed explicitly. Basically, it's a machine learning class that produces use of various nonlinear processing units so on perform feature extraction likewise as transformation.

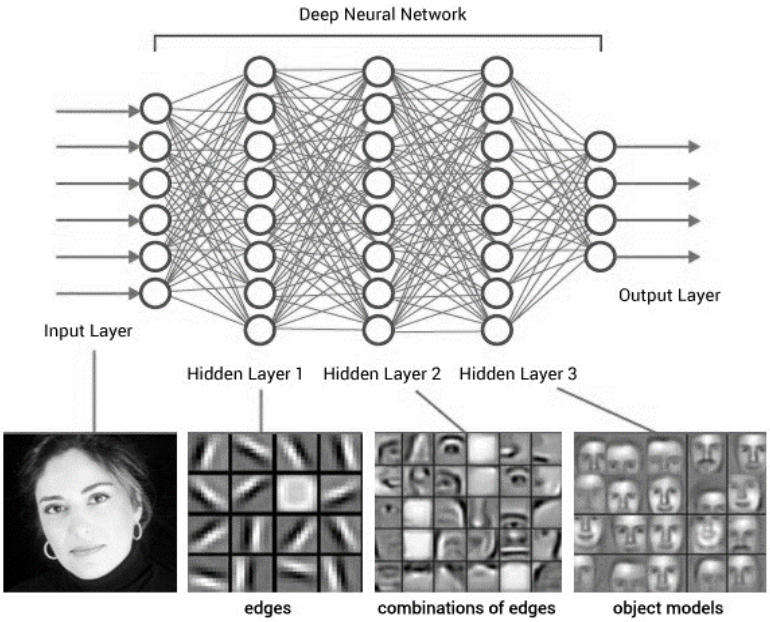


Fig 3.1: Deep learning neural network

The output from each preceding layer is taken as input by all of the successive layers. Deep learning models are capable enough to specialize in the accurate features themselves by requiring a bit guidance from the programmer and are very helpful in solving out the matter of dimensionality. Deep learning algorithms are used, especially after we have an enormous no of inputs and outputs.

3) Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.
One of the great challenges of developing CNNs is adjusting the weights of the individual neurons to extract the right features from images. The process of adjusting these weights is called “training” the neural network.

In the beginning, the CNN starts off with random weights. During training, the developers provide the neural network with a large dataset of images annotated with their corresponding classes. The ConvNet processes each image with its random values and then compares its output with the image's correct label.

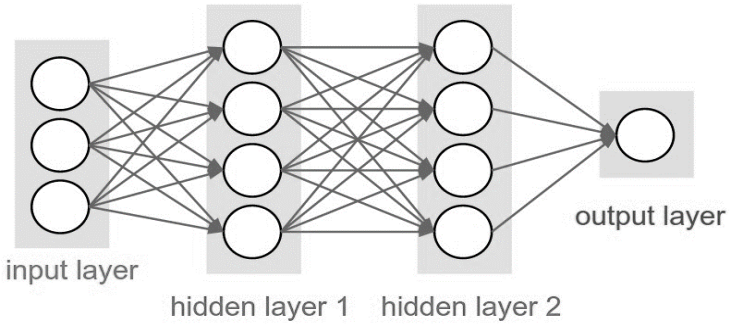


Fig 3.2: Convolutional network

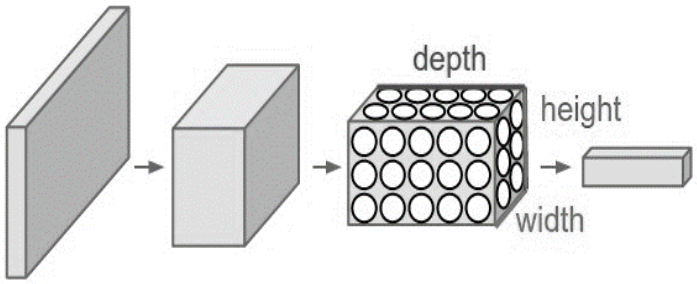


Fig 3.3: Layers of CNN

We use three main types of layers to build ConvNet architectures: Convolutional Layer**,** Pooling Layer**, and** Fully-Connected Layer.

- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the max(0,x)max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores.

4) MobileNetV2

MobileNetV2 is a significant improvement over MobileNetV1 and pushes the state of the art for mobile visual recognition including classification, object detection and semantic segmentation.

MobileNetV2 builds upon the ideas from MobileNetV1, using depthwise separable convolution as efficient building blocks. However, V2 introduces two new features to the architecture: 1) linear bottlenecks between the layers, and 2) shortcut connections between the bottlenecks. The basic structure is shown below.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VI June 2022- Available at www.ijraset.com

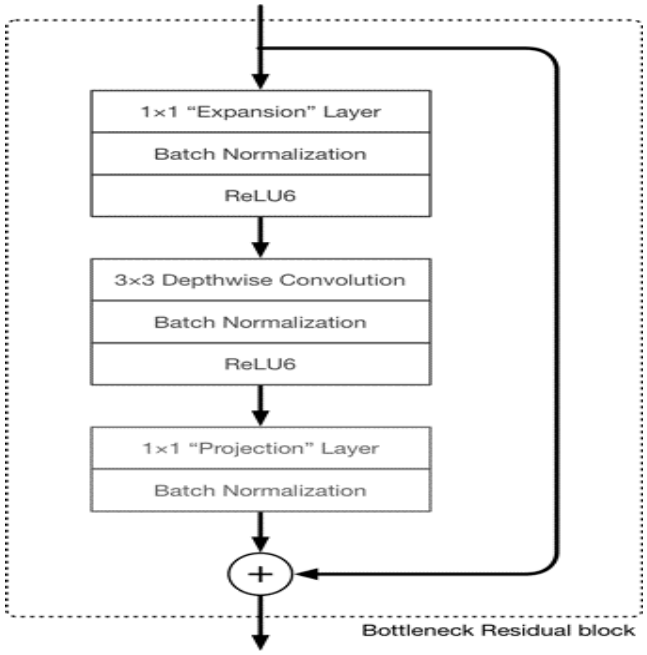


Fig 3.4: MobileNetV2 -Bottleneck residual block

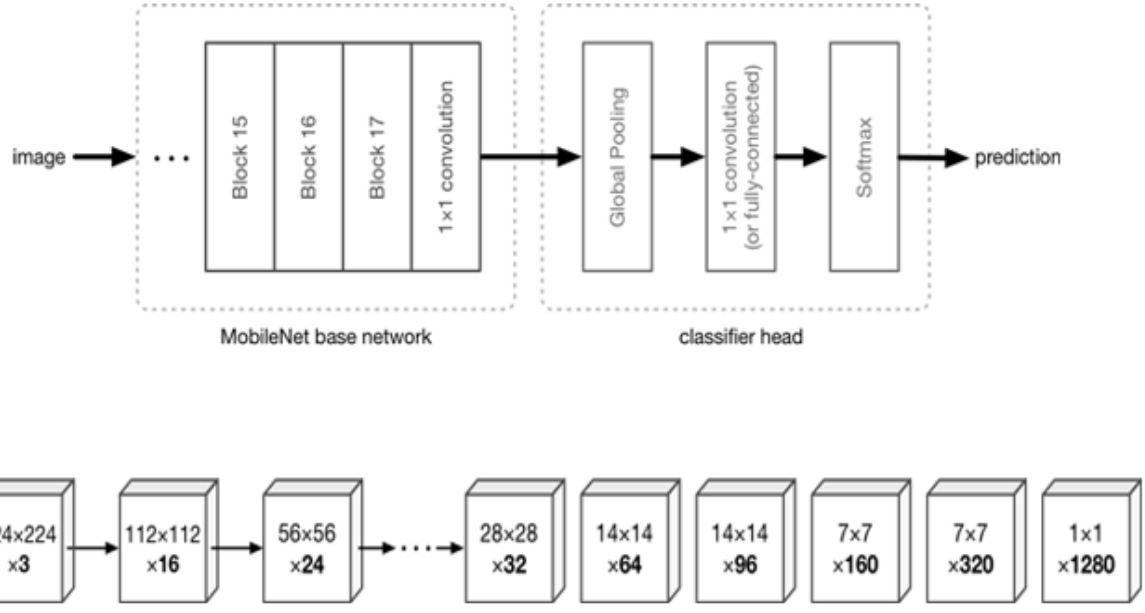


Fig 3.5: MobileNetV2 Layers

The intuition is that the bottlenecks encode the model's intermediate inputs and outputs while the inner layer encapsulates the model's ability to transform from lower-level concepts such as pixels to higher level descriptors such as image categories. Finally, as with traditional residual connections, shortcuts enable faster training and better accuracy.

MobileNetV2 models are faster for the same accuracy across the entire latency spectrum. In particular, the new models use 2x fewer operations, need 30% fewer parameters and are about 30-40% faster. MobileNetV2 is a very effective feature extractor for object detection and segmentation.

5) Packages

a) Tensorflow

TensorFlow is a Python library for fast numerical computing created and released by Google.

It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.

TensorFlow is an open source library for fast numerical computing.It can run on single CPU systems, GPUs as well as mobile devics and large scale distributed systems of hundreds of machines.

TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy.If you need more flexibility, eager execution allows for immediate iteration and intuitive debugging. TensorFlow has always provided a direct path to production. Whether it's on servers, edge devices, or the web, TensorFlow lets you train and deploy your model easily, no matter what language or platform you use.

b) Numpy

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

c) Scikit-Learn

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.The sklearn library contains a lot of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction.

It should not be used for reading the data, manipulating and summarizing it. One of the main reasons behind using opensource tools is the huge community it has. Same is true for sklearn as well.

d) Pandas

Pandas is one of the tools in Machine Learning which is used for data cleaning and analysis. It has features which are used for exploring, cleaning, transforming and visualizing from data.

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. Pandas is hands down one of the best libraries of python. It supports reading and writing excel spreadsheets, CVS's and a whole lot of manipulation. It is more like a mandatory library to know if dealing with datasets from excel files and CSV files. i.e., for Machine learning and data science.

IV. SYSTEM DESIGN

A. Architecture Diagram

Dataset of nearly 2000 images are gathered and used for training and building the model. The images are augmented using Image Data Generator and processed using Tensorflow and scikit-learn. The MobileNetV2 classifier is generated and the model file is saved as h5 file after training.

The trained model is loaded and the captured image is processed and predicted with good accuracy.

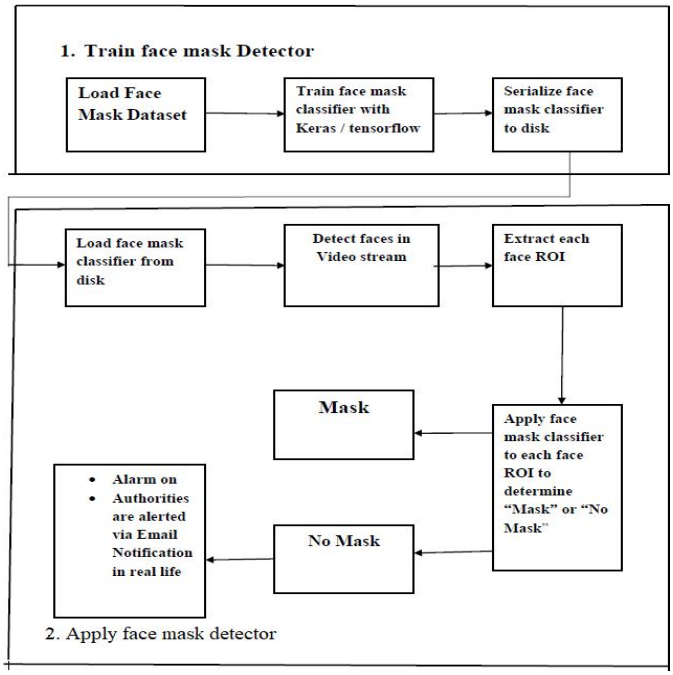


Fig 4.1: Architecture diagram

1) MobileNetV2 Architecture

MobileNetv2 architecture consists of three convolutional layers.

- 1×1 convolution
- Depth wise CNN
- Bottleneck layer

The first layer is a 1×1 convolution layer. The input channel initially has 3 channels- Red, Green and Blue for images. Its purpose is to increase the number of channels in the data before it goes into the depth wise convolution.

Depthwise CNN consists of a regular 3×3 convolution as the very first layer, followed by 13 times the 1x1 convolution.

Here it reduces the spatial dimensions of the data step by step. When that happens, the corresponding layer also doubles the number of output channels. For example, if the input image is 224×224×3, then the output of the network is a 7×7×1024 feature map. The convolution layers are followed by batch normalization and the activation function used by MobileNet is ReLU6.

Third is the bottleneck layer. It makes the number of channels smaller. It is also known as the projection layer as it projects data with a high number of dimensions into a tensor with a lower number of dimensions.

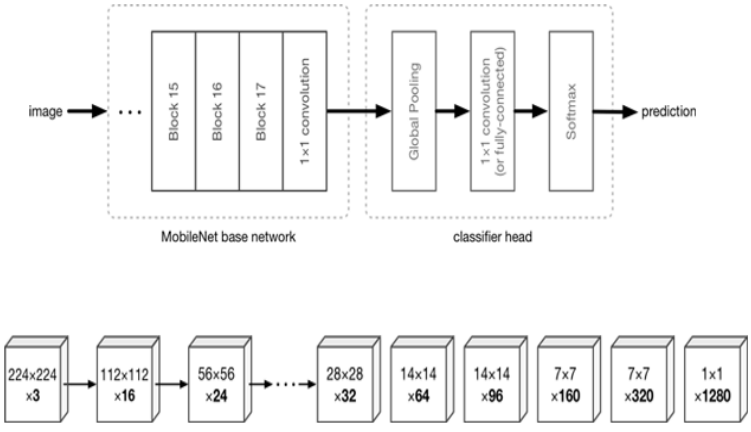


Fig 4.2: MobileNetV2 Architecture

B. Algorithm

With the help of a combination of deep learning and CNN techniques, a real-time face mask detection system with an alert system has been developed. The image segmentation method produces efficient and accurate results for face detection. Face-reading showed that over half of the participants could accurately determine if the individual was wearing a mask or not. Algorithm explains the methodology behind the face mask detection process.

Face Mask Detection Algorithm

- Step 1: Create Dataset (Arrays).
- Step 2: Create python file Train Mask Detector and import libraries.
- Step 3: Append images to data List.
- Step 4: Go through Path and Create Loops for Images with and without Mask.
- Step 5: Use Keras and MobilenetV2 for preprocessing.
- Step 6: Use Label by arrays method which includes sklearn module.
- Step 7: For deep learning Model Convert into numpy Arrays.
- Step 8: Track the Accuracy Matrix to save generated model.
- Step 9: Plot the accuracy using matplotlib. Run and check the Accuracy.
- Step 10: Train the model by images/Live Webcam to check whether the person is wearing a mask or not.
- Step 11: A beep sound is generated and alert message has send to the authorities whether the person is wearing a mask or not.

C. UML Diagrams

1) Use Case Diagram

A use case diagram is a graphic depiction of the interactions among the elements of a system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. In our use case diagram, we will discuss the interaction between different user roles and system.

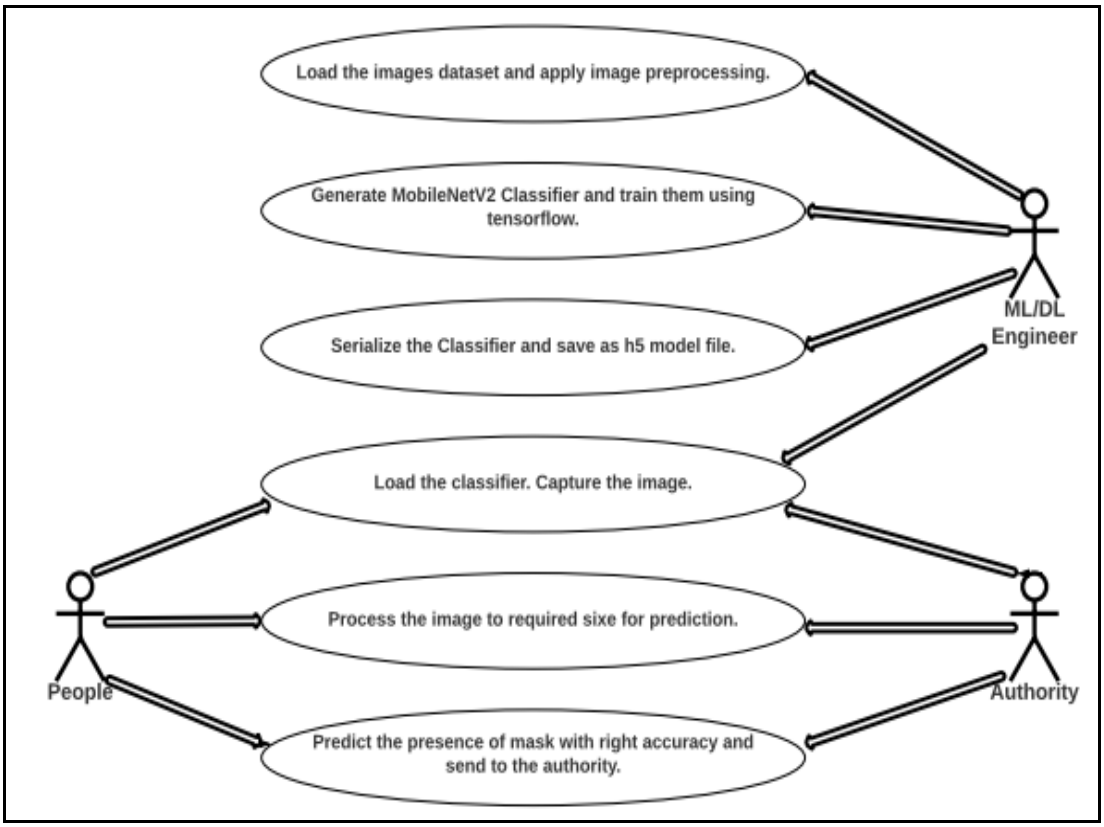


Fig 4.3: Use Case Diagram

2) Class Diagram

A class diagram is an illustration of the relationships and source code dependencies among classes in the UML. In our class diagram we have specified the details of methods which are related to the training process of images before prediction.

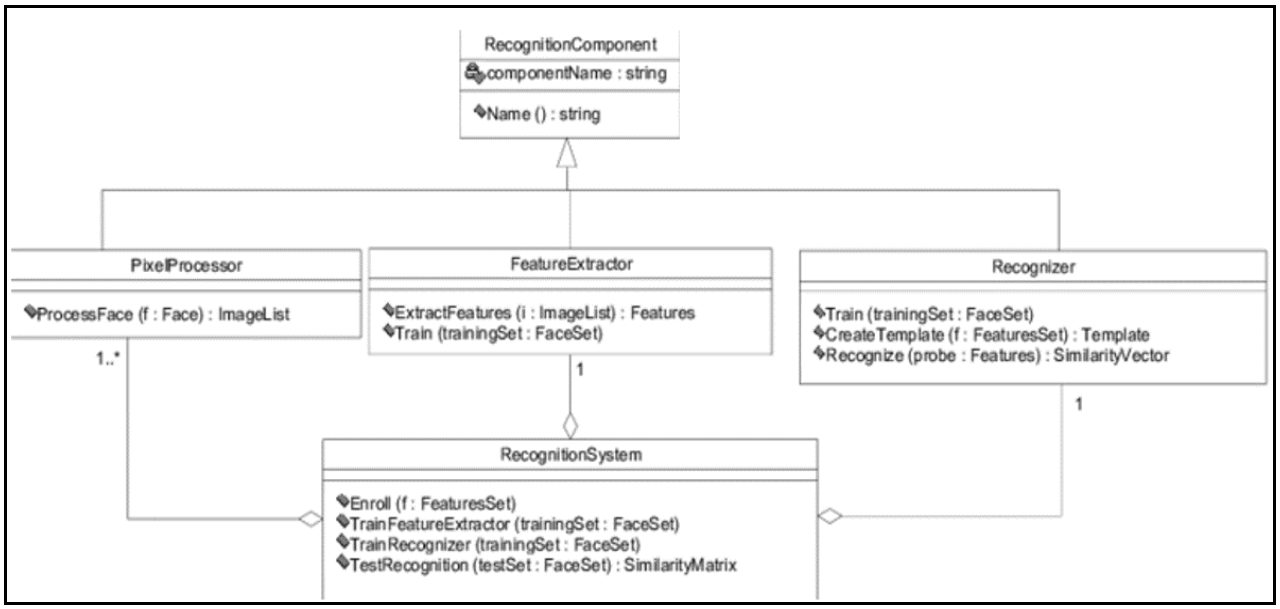


Fig 4.4: Class Diagram

3) Sequence Diagram

Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of collaboration and they show the order in which operations are carried out. In our sequence diagram we discuss, the steps that take place during the training process of images using mobilenetv2 and how the user interacts with the system.

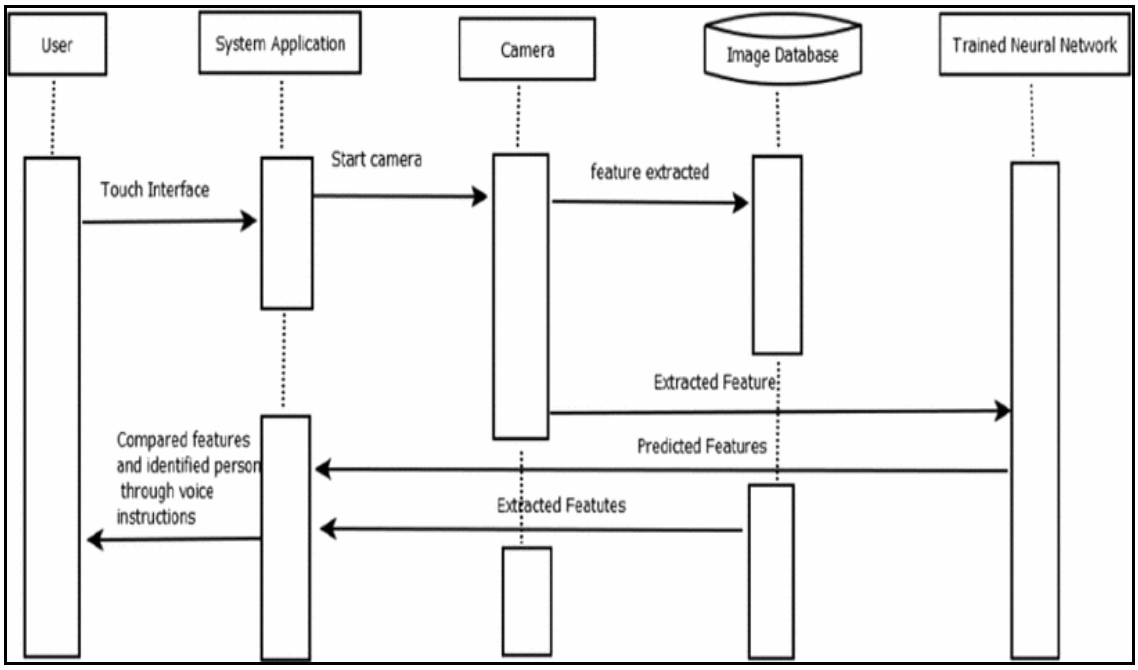


Fig 4.5: Sequence Diagram

4) Data Flow Diagram

A Data Flow Diagram is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. In data flow diagram, we have discussed the flow of image processing across various stages in the training and prediction.

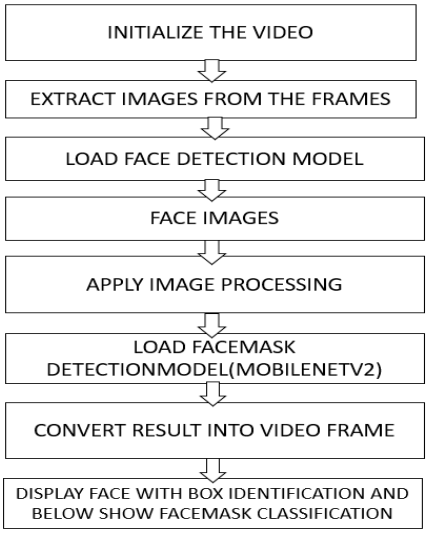


Fig 5.6: Data Flow Diagram

V. SYSTEM IMPLEMENTATION

A. List Of Modules

1) Gathering the image dataset.

Dataset of nearly 2000 images are gathered and used for training and building the model. As our training dataset contains many images, we would begin by plotting the images that fell into the most categories that we could find. There are approximately 690 images with a face mask that have been marked as “yes” in the database and approximately 690 photographs of people without face masks are been marked “no,”.

Fig 5.1: with mask

Fig 5.2: without mask

2) Image Preprocessing

Image processing can be defined as the technical analysis of an image by using complex algorithms. Here, image is used as the input, where the useful information returns as the output.

Numpy − For image manipulation and processing.

Scikit-learn − Provides lots of algorithms for image processing.

OpenCV − Image processing library mainly focused on real-time computer vision with application in wide-range of areas like 2D and 3D feature toolkits, facial & gesture recognition, Human-computer interaction, Mobile robotics, Object identification and others.

3) Generating the MobileNetV2 Classifier.

- Mobilenets are a type of Convolutional neural network where the images are processed through different layers.
- For building the base model using Mobilenet, we use imagenet which has predefined weights for images to give better results and we give inputshape as 224x224 with 3 channels for colored images (RGB).
- We then create fully connected layer by passing the created base model to the head model.
- We apply pooling layer, flatten layer, dense layer and dropout layer.
- In dense layer, we use 128 neurons and relu activation function for nonlinear use cases.
- Dropout layer is used to avoid overfitting of models.
- In the output layer, we use either softmax or sigmoid activation function as they are probability based functions that gives 0s & 1s.
- Adam optimizer is used in compiling the model and argmax is used to index the images.

4) Training the dataset using Tensorflow and saving it in H5 format.

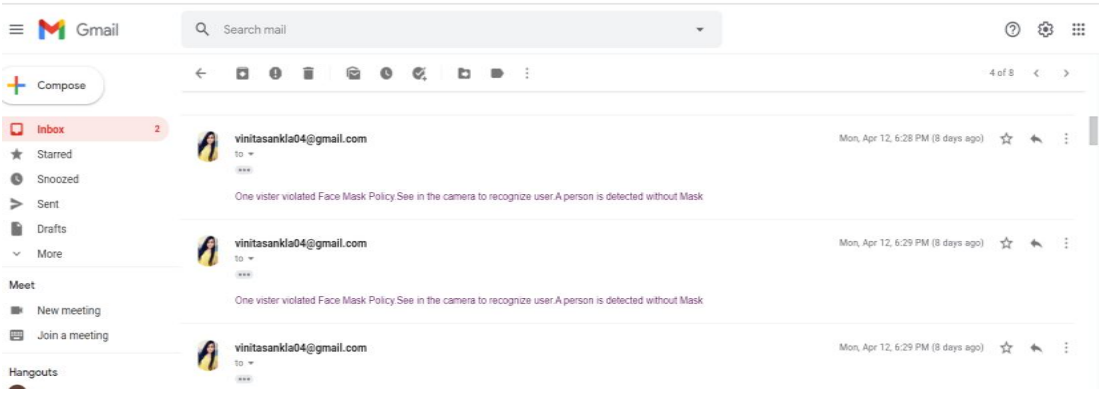
The created model is trained using model.fit by giving epochs and batch size. The created model is finally saved as a h5 file. H5 is a hierarchical data format which is a multidimensional array structure that can hold our data.

```
[INFO] training head...
Epoch 1/20
102/102 [==============================] - 45s 351ms/step - loss: 0.6336 - accuracy: 0.6823 - val_loss: 0.1864 - val_accuracy: 0.9621
Epoch 2/20
102/102 [==============================] - 34s 336ms/step - loss: 0.1790 - accuracy: 0.9513 - val_loss: 0.1119 - val_accuracy: 0.9719
Epoch 3/20
102/102 [==============================] - 34s 336ms/step - loss: 0.1094 - accuracy: 0.9673 - val_loss: 0.0856 - val_accuracy: 0.9731
Epoch 4/20
102/102 [==============================] - 34s 336ms/step - loss: 0.0899 - accuracy: 0.9705 - val_loss: 0.0798 - val_accuracy: 0.9731
Epoch 5/20
102/102 [==============================] - 34s 337ms/step - loss: 0.0728 - accuracy: 0.9764 - val_loss: 0.0641 - val_accuracy: 0.9756
Epoch 6/20
102/102 [==============================] - 34s 336ms/step - loss: 0.0607 - accuracy: 0.9814 - val_loss: 0.0596 - val_accuracy: 0.9768
Epoch 7/20
102/102 [==============================] - 34s 336ms/step - loss: 0.0601 - accuracy: 0.9824 - val_loss: 0.0620 - val_accuracy: 0.9756
Epoch 8/20
102/102 [==============================] - 34s 337ms/step - loss: 0.0461 - accuracy: 0.9853 - val_loss: 0.0584 - val_accuracy: 0.9768
Epoch 9/20
102/102 [==============================] - 34s 337ms/step - loss: 0.0401 - accuracy: 0.9870 - val_loss: 0.0695 - val_accuracy: 0.9731
Epoch 10/20
102/102 [==============================] - 34s 336ms/step - loss: 0.0506 - accuracy: 0.9817 - val_loss: 0.0540 - val_accuracy: 0.9780
Epoch 11/20
102/102 [==============================] - 34s 334ms/step - loss: 0.0291 - accuracy: 0.9918 - val_loss: 0.0465 - val_accuracy: 0.9792
Epoch 12/20
102/102 [==============================] - 34s 336ms/step - loss: 0.0342 - accuracy: 0.9909 - val_loss: 0.0530 - val_accuracy: 0.9792
Epoch 13/20
102/102 [==============================] - 34s 335ms/step - loss: 0.0368 - accuracy: 0.9877 - val_loss: 0.0447 - val_accuracy: 0.9792
Epoch 14/20
102/102 [==============================] - 34s 336ms/step - loss: 0.0339 - accuracy: 0.9889 - val_loss: 0.0480 - val_accuracy: 0.9805
Epoch 15/20
102/102 [==============================] - 34s 336ms/step - loss: 0.0288 - accuracy: 0.9890 - val_loss: 0.0460 - val_accuracy: 0.9817
Epoch 16/20
102/102 [==============================] - 34s 337ms/step - loss: 0.0307 - accuracy: 0.9883 - val_loss: 0.0455 - val_accuracy: 0.9829
Epoch 17/20
102/102 [==============================] - 35s 338ms/step - loss: 0.0279 - accuracy: 0.9894 - val_loss: 0.0452 - val_accuracy: 0.9817
Epoch 18/20
102/102 [==============================] - 34s 337ms/step - loss: 0.0387 - accuracy: 0.9904 - val_loss: 0.0411 - val_accuracy: 0.9805
Epoch 19/20
102/102 [==============================] - 34s 337ms/step - loss: 0.0252 - accuracy: 0.9925 - val_loss: 0.0430 - val_accuracy: 0.9817
Epoch 20/20
102/102 [==============================] - 34s 337ms/step - loss: 0.0283 - accuracy: 0.9905 - val_loss: 0.0427 - val_accuracy: 0.9829
```

Fig 5.3: Training the dataset

5) Prediction of Result

The face is captured, detected and the model predicts the output by providing us bounding boxes around the region showing us whether the person is wearing mask or not. If the camera captures a face without a mask an Email notification will be sent out to the administrator and the system alarm will ring.



VI. RESULTS AND DISCUSSION

The result is visualized in the form of graph. When we want 2 or more plots in a single image, we can make use of Subplot in Matplotlib, Pl. Subplot (xyz). xyz is a 3 digit integer where x-No of rows, y =No of columns, z= index number of that plot. This is one of the most useful feature when we need to compare two or more plots hand to hand instead of having them in separate images.

In this face mask detection project, the accuracy and loss of the trained model is plotted in the form of a graph.

In the x-axis, the number of epochs are taken and in the y-axis, the loss or accuracy probability is taken.

Here, the training accuracy and validation accuracy goes up nearing the probability of 1.0. And the training and validation loss goes quite lesser than the probability of 0.1.

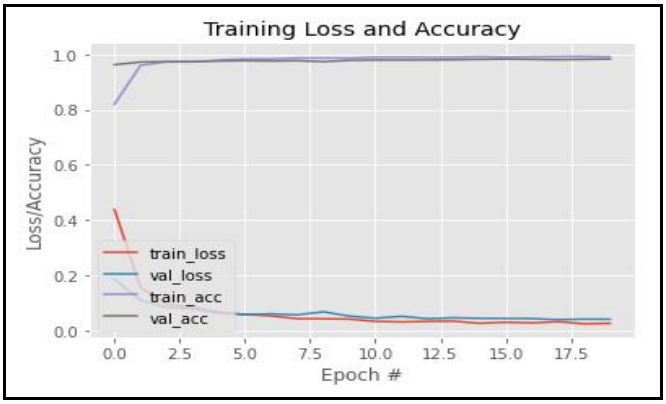


Fig 6.1: Training loss and accuracy

VII. CONCLUSION

The pandemic of COVID-19 has made the world to face a huge health crisis. According to the COVID-19 statistics, it is said that the transmission of the virus is more in crowded areas. Research studies have proved that wearing a mask in public places will reduce the transmission rate of the virus. Therefore, it's made it mandatory to wear masks in public places and crowded areas. So in this paper, The model was built using MobileNetV2 architecture. Successfully built alarm system to alert a person who didn't wear masks and implemented an Email notification system to notify respected authorities.We were able to generate accurate face masks for human objects from RGB channel images containing localized objects. Also erroneous problems in predictions has been solved and a proper bounding box has been drawn around the segmented region. Proposed network can detect non frontal faces and multiple faces from single image.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VI June 2022- Available at www.ijraset.com

APPENDIX A

Sample Coding

train.py

```
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="model-facemask.h5",
    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
DIRECTORY = list(paths.list_images(args["dataset"]))
CATEGORIES = ["with_mask", "without_mask"]
# grab the list of images in our dataset directory, then initialize
print("[INFO] loading images...")
data = []
labels = []
for category in CATEGORIES:
```



```
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)
        # update the data and labels lists, respectively
        data.append(image)
        labels.append(category)
# convert the data and labels to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)
print(labels)
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
print(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(arg  s["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["acc"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_acc"], 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"])
```

predictmask.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
```

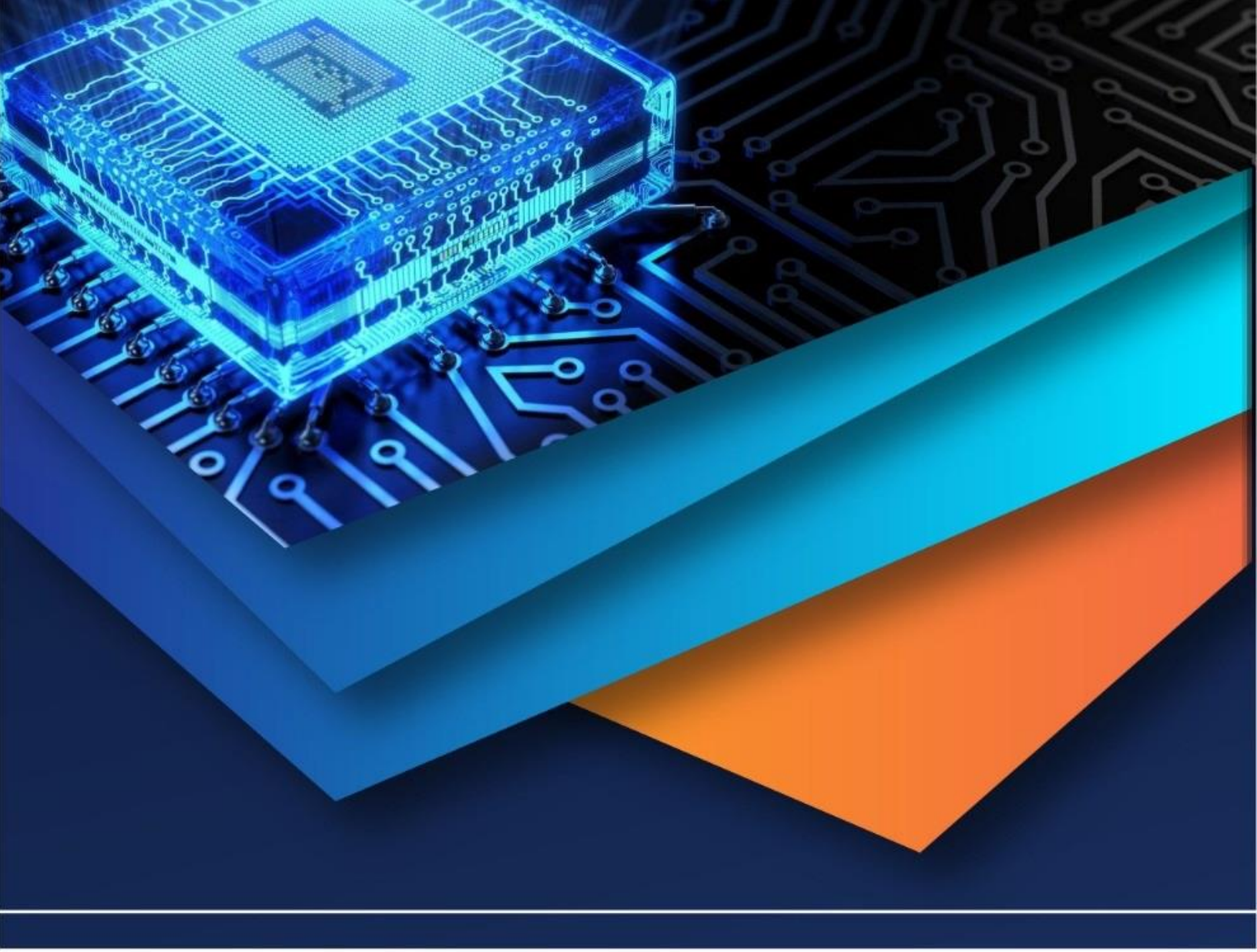
```
import numpy as np
import argparse
import imutils
import time
import cv2
import os
def detect_and_predict_mask(frame, faceNet, maskNet):
    # grab the dimensions of the frame and then construct a blob
    # from it
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
        (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)
            # add the face and bounding boxes to their respective
            # lists
            faces.append(face)
```

```
                locs.append((startX, startY, endX, endY))
        # only make a predictions if at least one face was detected
        if len(faces) > 0:
                # for faster inference we'll make batch predictions on *all*
                # faces at the same time rather than one-by-one predictions
                # in the above `for` loop
                faces = np.array(faces, dtype="float32")
                preds = maskNet.predict(faces, batch_size=32)
        # return a 2-tuple of the face locations and their corresponding
        # locations
        return (locs, preds)
# 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(2.0)
# 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()
```

REFERENCES

[1] Amit Chavda, Jason Dsouza, Sumeet Badgujar, Ankit Damani,“ Multi- Stage CNN Architecture for Face Mask Detection ”, iPing Data Labs LLP, Mumbai.
[2] Amrit Kumar Bhadani, Anurag Sinha,“ A FACEMASK DETECTOR USING MACHINE LEARNING AND IMAGE PROCESSING TECHNIQUES”, MuktShabd Journal.
[3] G. Jignesh Chowdary, Narinder Singh Punn, Sanjay Kumar Sonbhadra, Sonali Agarwal ,“ Face Mask Detection using Transfer Learning of InceptionV3n”, Journal paper.
[4] Kaihan Lin, Huimin Zhao , JujianLv , Canyao Li, Xiaoyong Liu, RongjunChen,andRuoyan Zhao ,“Face Detection and Segmentation Based on Improved Mask R-CNN”, Discrete Dynamics in Nature and Society.
[5] KALANGI B SUBRAMANYAM , S N S K KANTH, D J KUMAR, “COVID-19 Face Mask Detector with OpenCV ”, IRJET Journal.
[6] Marco Grassi , Marcos Faundez-Zanuy, “Face Recognition with Facial Mask Application and Neural Networks ”, Computational and Ambient Intelligence, 9th International Work-Conference.
[7] Mingjie Jiang , Xinqi Fan, Hong Yan, “RETINA FACEMASK: A FACE MASK DETECTOR ”, Journal Paper.
[8] Mohammad Marufur Rahman, MotalebHossenManik, Milon Islam, Saifuddin Mahmud ,Jong-Hoon Kim, “An Automated System to Limit COVID-19 Using Facial Mask Detection in Smart City Network ”, IEEE.
[9] Rafiuzzaman Bhuiyan, SharunAkter Khushbu, Sanzidul Islam, “A Deep Learning Based Assistive System to Classify COVID-19 Face Mask for Human Safety with YOLOv3 ”, IEEE.
[10] RamotLubis , “MACHINE LEARNING (CONVOLUTIONAL NEURAL NETWORKS) FOR FACE MASK DETECTION IN IMAGE AND VIDEO”, Bina Nusantara University Research paper.
[11] Shiming Ge, Jia Li, Qiting Ye , Zhao Luo, “Detecting Masked Faces in the Wild with LLE-CNNs ”, IEEE.
[12] Sneha Sen, Dr. Harish Patidar,“ Face Mask Detection System for COVID_19 Pandemic Precautions using Deep Learning method ”, Journal of Emerging Technologies and Innovative Research .
[13] T Subhamastan Rao ,“ A Novel Approach To Detect Face Mask To Control Covid Using Deep Learning ”, European Journal of Molecular & Clinical Medicine .
[14] ToshanMeenpal, Amit Verma , Ashutosh Balakrishnan,“ Facial Mask Detection using Semantic Segmentation ”, 2019 4th International Conference on Computing, Communications and Security (ICCCS) .
[15] Vinitha , Velantina, “COVID-19 FACEMASK DETECTION WITH DEEP LEARNING AND COMPUTER VISION”, IRJET Journal.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY