Designing of Face Mask Detection

A PROJECT REPORT

Submitted by

JATIN CHAUHAN

IN

MASTER OF COMPUTER APPLICATIONS

Chandigarh University, India.

Dec 2023

CERTIFICATE

Certified that this project report "**Designing of Face Mask Detection**" is the work of "**JATIN CHAUHAN**" who carried out the project work under our supervision.

Submitted for the project viva-voce examination held on *29, Nov. 2023*.

Mr. Keshav
Assistant Professor
Project Supervisor
University Institute Of Computing,
Chandigarh University, India

Dr. Abdullah
Head Of Department
University Institute Of Computing,
Chandigarh University, India

ABSTRACT

Face Mask Detection is currently a hot topic that has piqued the interest of researchers all over the world. Today, the entire world is dealing with the COVID-19 pandemic. To control the spread of the Coronavirus, people are taking a variety of measures. There are numerous critical measures required to combat COVID-19, one of the most important of which is the use of a face mask. There is still a lot of research and study being done on COVID-19. Several studies have also shown that wearing a face mask significantly reduces the problem of viral transmission. In addition, a person wearing a face mask perceives a sense of protection. When we are at home, we take care of everything, but when we are in public places such as offices, malls, and colleges, it becomes more difficult to keep people safe. Machine Learning and Data Mining are a collection of technologies that provide effective solutions to complex problems in a variety of fields. We attempted to develop a face mask recognition system using machine learning in order to prevent the spread of the Coronavirus. This is a good system for detecting a face mask in newspaper and news channel images and videos. It can recognize both Mask and No Mask faces. With the advancement of this system, it will be possible to detect whether or not a person is wearing a face mask. If the person is not wearing a face mask, it will display a message such as "No Mask," otherwise it will display "Mask Detected."

Keywords: - Face and Face mask detection, Data Train, Machine learning, Data mining, MobileNetV2.

Table of Contents

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The trend of wearing face masks in public is rising due to the COVID- 19 corona virus epidemic all over the world. Before Covid-19, People used to wear masks to protect their health from air pollution. While other people are self-conscious about their looks, they hide their emotions from the public by hiding their faces. Scientists proofed that wearing face masks works on impeding COVID-19 transmission. COVID19 (known as corona virus) is the latest epidemic virus that hit the human health in the last century. In 2020, the rapid spreading of COVID-19 has forced the World Health Organization to declare COVID- 19 as a global pandemic.

More than five million cases were infected by COVID-19 in less than 6 months across 188 countries. The virus spreads through close contact and in crowded and overcrowded areas.

The corona virus epidemic has given rise to an extraordinary degree of worldwide scientific cooperation. Artificial Intelligence (AI) based on Machine learning and Deep Learning can help to fight Covid-19 in many ways. Machine learning allows researchers and clinicians evaluate vast quantities of data to forecast the distribution of COVID-19,to serve as an early warning mechanism for potential pandemics, and to classify vulnerable populations.The provision of healthcare needs funding for emerging technology such as artificial intelligence, IoT, big data and machine learning to tackle and predict new diseases. In order to better understand infection rates and to trace and quickly detect infections, the AI’s power is being exploited to address the Covid-19 pandemic. People are forced by laws to wear face masks in public in many countries. These rules and laws were developed as an action to the exponential growth in cases and deaths in many areas.

However, the process of monitoring large groups of people is becoming more difficult. The monitoring process involves the detection of anyone who is not wearing a face mask.

1.1 BACKGROUND

It is a problem of object detection and classification with two distinct classes (Mask and No Mask). For detecting face masks, A model based on deep and classical machine learning will be presented. We present a face mask detection model based on computer vision, deep learning and Data mining. The proposed model can be integrated with Picture or videos to detect people wearing masks and those who are not wearing masks. The model was created by combining deep learning and traditional machine learning techniques. It uses an approach to split the base layer's feature map into two sections, then merges them using a crossstage hierarchy. The adoption of a split-and-merge approach allows the network to have more gradient flow. We presented a comparison of three machine learning algorithms in order to identify the most appropriate algorithm with the highest accuracy. The spread of the COVID-19 virus has slowed, but it is far from over. If everyone follows all of the safety precautions, it may come to an end.

1.2 MOTIVATION

The world has not yet fully recovered from this pandemic and the vaccine that can effectively treat COVID-19 is yet to be discovered. However, to reduce the impact of the pandemic on the country's economy, several governments have allowed a limited number of economic activities to be resumed once the number of new cases of COVID-19 has dropped below a certain level. As these countries cautiously restart their economic activities, concerns have emerged regarding workplace safety in the new post-COVID-19 environment.

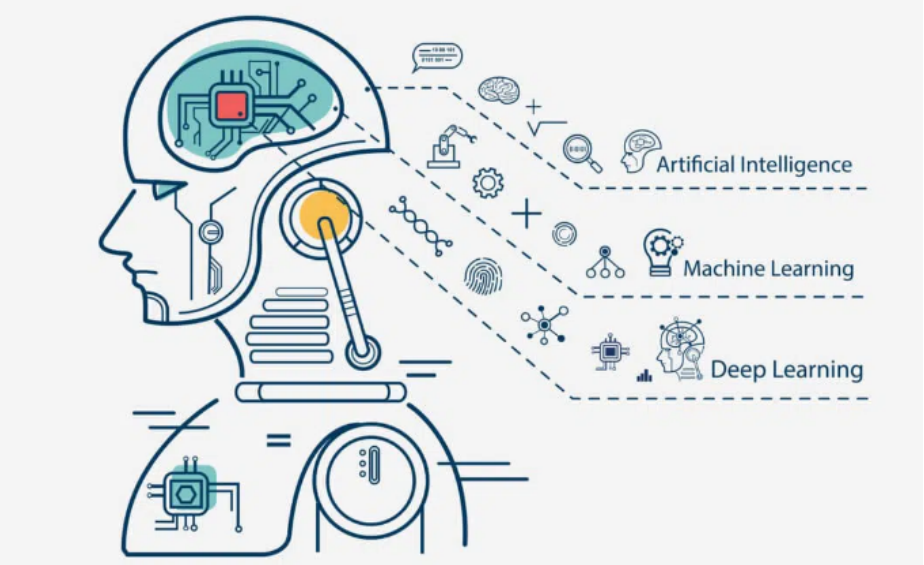
To reduce the possibility of infection, it is advised that people should wear masks and maintain a distance of at least 1 meter from each other. Deep learning has gained more attention in object detection and was used for human detection purposes and to develop a face mask detection tool that can detect whether the individual is wearing a mask or not. This can be done by evaluating the classification results by analyzing real-time streaming from the camera. In deep learning projects, we need a training data set. It is the actual dataset used to train the model for performing various actions.

1.3 PROBLEM STATEMENT

The main objective of the face detection model is to detect the face of individuals and conclude whether they are wearing masks or not at that particular moment when they are captured in the image.

1.4 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.

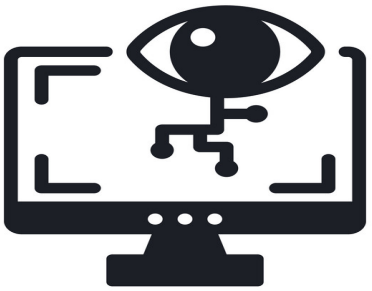
Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system:

- ***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 achieve 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 with feedback that's analogous to rewards, which it tries to maximize. Other approaches have been developed which don't fit neatly into this three-fold categorization, and sometimes more than one is used by the same machine learning system.

1.5 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 highdimensional 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



COMPUTER VISION

images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.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 to the construction of computer vision systems.

1.6 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.

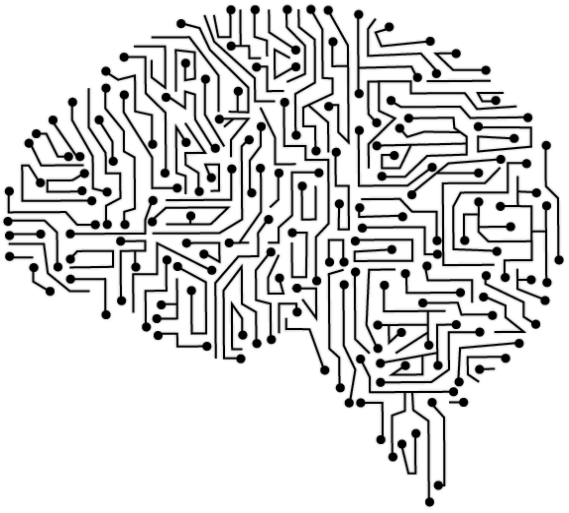


Fig. – 3: Deep learning

CHAPTER 2

LITERATURE SURVEY

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

COVID-19 pandemic caused by novel coronavirus is continuously spreading until now all over the world. The impact of COVID-19 has been fallen on almost all sectors of development. The healthcare system is going through a crisis. Many precautionary measures have been taken to reduce the spread of this disease where wearing a mask is one of them. In this paper, we propose a system that restrict the growth of COVID-19 by finding out people who are not wearing any facial mask in a smart city network where all the public places are monitored with Closed-Circuit Television (CCTV) cameras. While a person without a mask is detected, the corresponding authority is informed through the city network. A deep learning architecture is trained on a dataset that consists of images of people with and without masks collected from various sources. The trained architecture achieved 98.7% accuracy on distinguishing people with and without a facial mask for previously unseen test data. It is hoped that our study would be a useful tool to reduce the spread of this communicable disease for many countries in the world.

2.2 Masked Face Recognition Using Convolutional Neural Network :

Recognition from faces is a popular and significant technology in recent years. Face alterations and the presence of different masks make it too much challenging. In the real-world, when a person is uncooperative with the systems such as in video surveillance then masking is further common scenarios. For these masks, current face recognition performance degrades. An abundant number of researches work has been performed

for recognizing faces under different conditions like changing pose or illumination, degraded images, etc. Still, difficulties created by masks are usually disregarded. The primary concern to this work is about facial masks, and especially to enhance the recognition accuracy of different masked faces. A feasible approach has been proposed that consists of first detecting the facial regions. The occluded face detection problem has been approached using Deep Learning and Computer Vision concepts.

2.3 EXISTING SYSTEM

Face detection problems have been approached using Deep Learning and Computer Vision concepts. Then facial features extraction is performed using the keras .This system is capable of training the dataset of both persons wearing masks and without wearing masks. After training the model the system can predict whether the person is wearing the mask or not wearing the mask.

…

CHAPTER 3

METHODOLOGY

3.1 PROPOSED SYSTEM

1.This system is capable of training the dataset of both persons wearing masks and without wearing masks.
2.After training the model, the system can predicting whether the person is wearing the mask or not .
3.It can also access the webcam and predict the result.

3.2 TENSORFLOW FRAMEWORK:

TensorFlow is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. TensorFlow was originally developed for large numerical computations without keeping deep learning in mind. However, it proved to be very useful for deep learning development as well, and therefore Google open-sourced it.

TensorFlow accepts data in the form of multi-dimensional arrays of higher dimensions called tensors. Multi-dimensional arrays are very handy in handling large amounts of data.

TensorFlow works on the basis of data flow graphs that have nodes and edges. As the execution mechanism is in the form of graphs, it is much easier to execute TensorFlow code in a distributed manner across a cluster of computers while using GPUs.

- **TensorFlow Architecture:** The Tensorflow architecture works in three significant steps:
    - *Data preprocessing:* structuring the data and brings it under one limiting value.

- *Building the model:* build the model for the data
- *Training and estimating the model:* use the data to train the model and test it with unknown data.

3.3 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.

The code is hosted on GitHub, and community support forums include the GitHub issues page, and a Slack channel. 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.

Four principles:

- 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.
- Minimalism: The library provides just enough to achieve an outcome, no frills and maximizing readability.
- Extensibility: New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.

- 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.

3.4 Imutils

A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and *both* Python 2.7 and Python 3.

Installation method:
$ pip install imutils

Confirm that numpy, SciPy, Matplotlib, and openCV are installed before installation.

If a missing package error occurs, you can use the following installation command to install all packages.

$ pip install NumPy SciPy opencv-python matplotlib imutils

Usage of imutils

- image translation
- Image scaling
- Image rotation
- Skeletal extraction (edge extraction)
- to RGB, and displayed using Matplotlib.
- detection of the OpenCV version

3.5 Numpy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an *ndarray* will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

3.6 OpenCV-python

OpenCV (Open Source Computer Vision Library) is an open source software library for computer vision and machine learning. OpenCV was created to provide a shared infrastructure for applications for computer vision and to speed up the use of machine perception in consumer products. OpenCV, as a BSD-licensed software, makes it simple for companies to use and change the code. There are some predefined packages and libraries that make our life simple and OpenCV is one of them.

Gary Bradsky invented OpenCV in 1999 and soon the first release came in 2000. This library is based on optimized C / C++ and supports Java and Python along with C++ through an interface. The library has more than 2500 optimized algorithms, including an extensive collection of computer vision and machine learning algorithms, both classic and state-of-the-art.Using OpenCV it becomes easy to do complex tasks such as identify and recognise faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D object models, generate 3D point clouds from stereo cameras, stitch images together to generate an entire scene with a high resolution image and many more.

Python is a user friendly language and easy to work with but this advantage comes with a cost of speed, as Python is slower to languages such as C or C++.So we extend Python with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. Doing this, the code is fast, as it is written in original C/C++ code (since it is the actual C++ code working in the background) and also, it is easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

Computer vision allows the computer to perform the same kinds of tasks as humans with the same efficiency. There are two main tasks, which are defined below:

- **Object Classification:** In object classification, we train a model on a dataset of particular objects, and the model classifies new objects as belonging to one or more of your training categories.
- **Object Identification:** In object identification, our model will identify a particular instance of an object.

3.7 Matplotlib

There are numerous libraries present in **Python**. And **Matplotlib** is one of the most successful and commonly used libraries, that provides various tools for data visualization in Python.

It is one of the most powerful plotting libraries in Python. It is a **cross-platform** library that provides various tools to create 2D plots from the data in lists or arrays in python.

It was created and coded by **John D. Hunter** in **Python** programming language in 2003. The current stable version of matplotlib is **3.4.2**, which was released on **May 8, 2021**. It makes use of **NumPy**, a library that provides the numerical mathematical extension for Python.

It also provides an object-oriented API that enables it to extend the functionality to put the static plots in applications by using various Python GUI toolkits available (Tkinter, PyQt, etc).

It allows a user to visualize data using a variety of different types of plots to make the data understandable. You can use these different types of plots (scatterplots, histograms, bar charts, error charts, box plots, etc.) by writing a few lines of code in Python.

Features of matplotlib

- It is used as a data visualization library for the Python programming language.
- It provides quite possibly the simplest and most common way to plot data in Python.
- It provides such tools that can be used to create publication-standard plots and figures in variety of export formats and various environments (pycharm, jupyter notebook) across platforms.
- It provides a **procedural interface** called **Pylab**, which is used designed to make it work like **MATLAB**, a programming language used by scientists, researchers. MATLAB is a paid application software and not open source.

3.8 Argparse

Python argparse is a command-line parsing module that is recommended to work with the command line argument. This module was released as a part of the standard library with Python on February 20th, 2011.

It is similar to the getopt module, but it is slightly harder to use and requires more code lines to perform the same task. However, the argparse module is a better replacement for the **Python getopt** and **optparse modules**. It provides a few important features that are given below.

- It allows us to use positional arguments.
- It allows us to customize the prefix characters.
- It supports a variable number of parameters for a single option.
- It supports subcommands.

The argparse module makes it easy to write user-friendly command-line interfaces. It parses the defined arguments from the *sys.argv*.

The *argparse* module also automatically generates help and usage messages, and issues errors when users give the program invalid arguments.

The *argparse* is a standard module; we do not need to install it.

A parser is created with *ArgumentParser* and a new parameter is added with *add_argument*. Arguments can be optional, required, or positional.

3.9 Scipy

The SciPy is an open-source scientific library of Python that is distributed under a BSD license. It is used to solve the complex scientific and mathematical problems. It is built on top of the Numpy extension, which means if we import the SciPy, there is no need to import Numpy. The **Scipy** is pronounced as **Sigh pi**, and it depends on the Numpy, including the appropriate and fast N-dimension array manipulation.

It provides many user-friendly and effective numerical functions for numerical integration and optimization.

The **SciPy** library supports **integration, gradient optimization, special functions, ordinary differential equation solvers, parallel programming tools**, and many more. We can say that **SciPy** implementation exists in every complex numerical computation.

The **scipy** is a data-processing and system-prototyping environment as similar to MATLAB. It is easy to use and provides great flexibility to scientists and engineers.

- SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
- SciPy package in Python is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
- Easy to use and understand as well as fast computational power.
- It can operate on an array of NumPy library.

3.10 Scikit-Learn

Scikit Learn or Sklearn is one of the most robust libraries for machine learning in Python. It is open source and built upon NumPy, SciPy, and Matplotlib. It provides a range of tools for machine learning and statistical modeling including dimensionality reduction, clustering, regression, and classification, through a consistent interface in Python. Additionally, it provides many other tools for evaluation, selection, model development, and data preprocessing.

Scikit-learn is one of NumFOCUS's fiscally sponsored projects. It also integrates well with many other Python libraries, such as Matplotlib, Plotly, NumPy, Pandas, SciPy, etc. Although the library is fairly new, it has quickly become one of the most popular libraries on GitHub. A number of big organizations such as Spotify, Evernote, JP Morgan, Inria, AWeber, and many more use Sklearn.

Note: Sklearn is used to build machine learning models.

- Open Source − It is an open-source library and commercially usable under the BSD license.
- Clustering − It can be used for grouping unlabeled data.
- Supervised Learning algorithms − It contains almost all the popular supervised learning algorithms such as Decision Tree, Linear Regression, Support Vector Machine (SVM), etc.
- Unsupervised Learning algorithms − It also contains all the popular unsupervised learning algorithms such as clustering, principal component analysis, factor analysis, unsupervised neural networks, etc.
- Feature selection − It can identify useful attributes to create supervised models.
- Feature extraction − It can extract features from data to define the attributes in image and text data.

- Cross-Validation − It can check the accuracy of supervised models on unseen data.
- Dimensionality Reduction − It can reduce the number of attributes in data which can be further used for summarization, visualization, and feature selection.
- Ensemble methods − It can combine the predictions of multiple supervised models.

3.11 Pillow

Some of the most common image processing libraries are: OpenCV, Python Imaging Library (PIL), Scikit-image, Pillow. However, in this tutorial, we are only focusing on Pillow module and will try to explore various capabilities of this module.

Pillow is built on top of PIL (Python Image Library). PIL is one of the important modules for image processing in Python. However, the PIL module is not supported since 2011 and doesn't support python 3.

Pillow module gives more functionalities, runs on all major operating system and support for python 3. It supports wide variety of images such as "jpeg", "png", "bmp", "gif", "ppm", "tiff". You can do almost anything on digital images using pillow module. Apart from basic image processing functionality, including point operations, filtering images using built-in convolution kernels, and color space conversions.

Image Archives

The Python Imaging Library is best suited for image archival and batch processing applications. The Python pillow package can be used for creating thumbnails, converting from one format to another and print images, etc.

Image Display

You can display images using Tk PhotoImage, BitmapImage, and the Windows DIB interface, which can be used with PythonWin and other

Windows-based toolkits and many other graphical user interface (GUI) toolkits.

For debugging purposes, there is a show () method to save the image to disk, which calls the external display utility.

Image Processing

The Pillow library contains all the basic image processing functionality. You can do image resizing, rotation, and transformation.

The Pillow module allows you to pull some statistical data out of image using the histogram method, which later can be used for statistical analysis and automatic contrast enhancement.

3.12 Streamlit

Streamlit is an open-source python framework for building web apps for Machine Learning and Data Science. We can instantly develop web apps and deploy them easily using Streamlit. Streamlit allows you to write an app the same way you write a python code. Streamlit makes it seamless to work on the interactive loop of coding and viewing results in the web app.

The framework fills a vital void between data scientists who want to develop a new analytics widget or app and the data engineering typically required to deploy these at scale. Data scientists can build web apps to access and explore machine-learning models, advanced algorithms, and complex data types without having to master back-end data engineering tasks.

Streamlit cofounder and CEO Adrien Treuille told VentureBeat that “the combination of the elegant simplicity of the Streamlit library and the fact that it is all in Python means developers can do things in hours that normally took weeks.”

Examples of this increased productivity boost include reducing data app development time from three and a half weeks to six hours or reducing 5,000 lines of JavaScript to 254 lines of Python in Streamlit.

Streamlit's architecture allows you to write apps the same way you write plain Python scripts. To unlock this, Streamlit apps have a unique data flow: any time something must be updated on the screen, Streamlit reruns your entire Python script from top to bottom.
This can happen in two situations:

- Whenever you modify your app's source code.
- Whenever a user interacts with widgets in the app. For example, when dragging a slider, entering text in an input box, or clicking a button.

3.14 Pandas

Pandas is an open-source library that is built on top of NumPy library. It is a Python package that offers various data structures and operations for manipulating numerical data and time series. It is mainly popular for importing and analyzing data easily. Pandas is fast, and they have high-performance & productivity for users.
Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open-source data analysis/manipulation tool available in any language. It is already well on its way toward this goal.
Explore data analysis with Python. Pandas Data Frames make manipulating your data easy, from selecting or replacing columns and indices to reshaping your data.

- Fast and efficient for manipulating and analyzing data.
- Data from different file objects can be loaded.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Data set merging and joining.
- Flexible reshaping and pivoting of data sets
- It provides time-series functionality.
- Powerful group by functionality for performing split-apply-combine operations on data sets.

3.15 PYTHON CONCEPTS

You can skip to the next chapter if you are not interested in how and why Python. In this chapter, I will try to explain why I think Python is one of the best programming languages. languages available and why it is such a great place to start.

Python was developed into an easy-to-use programming language. It uses English words instead of punctuation, and has fewer syntax than other languages. Python is a highly developed, translated, interactive, and object-oriented language.

Python translated: Interpreter processing Python during launch. Before using your software, you do not need to install it. This is similar to PERL and PHP editing languages.

Python interactive: To write your own applications, you can sit in Python Prompt and communicate directly with the interpreter.

Python Object-Oriented: Python supports the Object-Oriented program style or method, encoding the code within objects.

Python is a language for beginners: Python is an excellent language for beginners, as it allows for the creation of a variety of programs, from simple text applications to web browsers and games.

3.15.1 Python Features

Python features include -
1.Easy-to-learn - Python includes a small number of keywords, precise structure, and well-defined syntax. T This allows the student to learn the language faster

2.Easy to read - Python code is clearly defined and visible to the naked eye.

3.Easy-to-maintain - Python source code is easy to maintain.

4.Standard General Library - Python's bulk library is very portable and shortcut compatible with UNIX, Windows, and Macintosh.

5.Interaction mode - Python supports interaction mode that allows interaction testing and correction of captions errors.

6.Portable - Python works on a variety of computer systems and has the same user interface for all.

7.Extensible - Low-level modules can be added to Python interpreter. These modules allow system developers to improve the efficiency of their tools either by installing or customizing them.

8.Details - All major commercial information is provided by Python ways of meeting.

9.GUI Programming - Python assists with the creation and installation of a user interface for images of various program phones, libraries, and applications, including Windows MFC, Macintosh, and Unix's X Window.

10.Scalable - Major projects benefit from Python building and support, while Shell writing is not.

Aside from the characteristics stated above, Python offers a long list of useful features, some of which are described below. −

● It supports OOP as well as functional and structured programming methodologies.
● It can be used as a scripting language or compiled into byte-code for large-scale application development.
● It allows dynamic type verification and provides very high-level dynamic data types.
● Automatic garbage pickup is supported by IT.

3.15.2 Python Numbers

Numeric values are stored in number data types. When you give a number a value, it becomes a number object.

3.15.3 Python Strings

In this python uses a string is defined as a collection set of characters enclosed in quotation marks. Python allows you to use any number of quotes in pairs. The slice operator ([] and [:]) can be used to extract subsets of strings, with indexes starting at 0 at the start of the string and working their way to -1 at the end.

3.15.4 Python Lists

The most diverse types of Python data are lists. Items are separated by commas and
placed in square brackets in the list ([]). Lists are similar to C-order in some ways.
Listings can be for many types of data, which is one difference between them. The slide operator ([] and [:]) can be used to retrieve values stored in the list, the
indicators start with 0 at the beginning of the list and work their way to the end of the list. The concatenation operator of the list is a plus sign (+), while the repeater is an asterisk (*).

3.15.5 Python Tuples

A cone is a type of data type similar to a sequence of items. Cone is a set of values separated by commas. The pods, unlike the list, are surrounded by parentheses. Lists are placed in parentheses ([]), and the elements and sizes can be changed, but the lumps are wrapped in brackets (()) and cannot be sorted. Powders are the same as
reading lists only.

3.15.6 Python Dictionary

Python dictionaries in Python are a way of a hash table. They are similar to Perl's combination schemes or hashes, and are made up of two key numbers.
The dictionary key can be any type of Python, but numbers and strings are very common. Prices, on the other hand, can be anything you choose Python.

Curly braces () surround dictionaries, and square braces ([) are used to assign and access values.

Different modes in Python
Python Normal and interactive are the two basic Python modes.
The scripted and completed.py files are executed in the Python interpreter in the regular manner. Interactive mode is a command line shell that

provides instant response for each statement while simultaneously running previously provided statements in active memory. The feed programme is assessed in part and whole as fresh lines are fed into the interpreter.

CHAPTER 4
DESIGN & IMPLEMENTATION

4.1 PROPOSED SYSTEM

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

Approach

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

4.2 Proposed methodology

To predict whether a person has worn a mask correctly, the initial stage would be to train the model using a proper dataset. After training the classifier, an accurate face detection model is required to detect faces, so that the SSDMNV2 model can classify whether the person is wearing a mask or not. The task in this paper is to raise the accuracy of mask detection without being too resource-heavy. For doing this task, the DNN module was used from OpenCV, which contains a ‘Single Shot Multibox Detector’ (SSD) object detection model with ResNet-10 as its backbone architecture. This approach helps in detecting faces in real-time, even on embedded devices like Raspberry Pi. The following classifier uses a pre-trained model MobileNetV2 to predict whether the person is wearing a mask or not. The approach used in this paper is depicted in the flow diagram in Fig-4.1

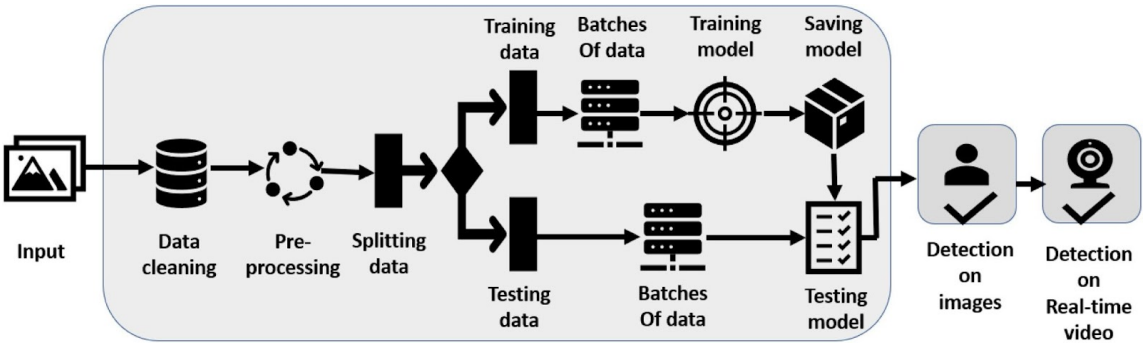


Fig- 4.1(Flow Diagram of the SSDMNV2 model)

4.3 Dataset used

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.

There are only a few datasets available for the detection of face masks. Most of them are either artificially created, which doesn't represent the real world accurately, or the dataset is full of noise and wrong labels. So, choosing the right dataset which would work best for the SSDMNV2 model required a little effort. The dataset used for training the model in a given approach was a combination of various open-source datasets and pictures, which included data from Kaggle datasets, Bing Search API(*search.py*). Data were also collected using the dataset provided by the masked face recognition dataset and application. The Kaggle dataset contains pictures of people wearing medical masks and XML files containing their descriptions and masks. This dataset had a total of 4095 images. The dataset includes 4095 images separated into two classes with wearing masks, 2165 pictures, and without wearing a mask, 1930 pictures.

In the end, a dataset which included 4095 images was obtained, having the label "*with_mask*" and "*without_mask*" also contained 5521 images to make a balanced dataset. The distribution between the two classes is visualized in Fig. 4.2. The created dataset can also be used for detecting assailants who cover their faces while performing unlawful deeds.

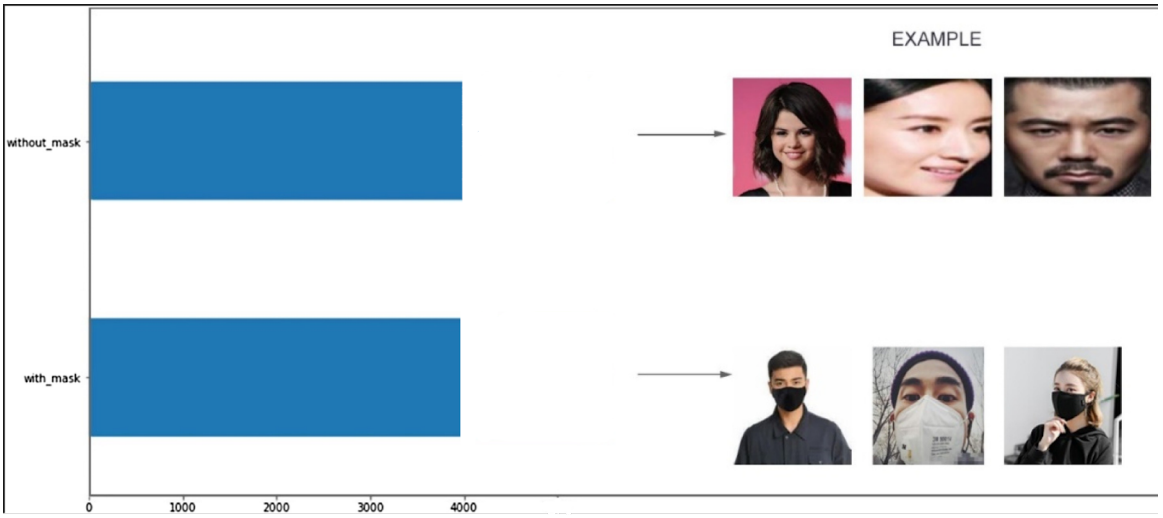


Fig- 4.2(Dataset Visualization.)

4.4 Pre-processing

The Dataset from Masked face recognition and application contained a lot of noise, and a lot of repetitions were present in the images of this dataset. Since a good dataset dictates the accuracy of the model trained on it, so the data from the above-specified datasets were taken. They were then processed, and also all the repetitions were removed manually. The data cleaning was done manually to remove the corrupt images which were found in the said dataset. Finding these images was a vital part. As it is well known, the corrupt image was a tricky task, but due to valid efforts, we divided the work and cleaned the data set with mask images and without mask images. Cleaning, identifying, and correcting errors in a dataset removes adverse effects from any predictive model.

This part explains the procedure of pre-processing the data and then training on data. First, we define a function name sorted alphanumerically

to sort the list in lexicographical order. A function pre-processing is defined, which takes the folder to the dataset as input, then loads all the files from the folder and resizes the images according to the SSDMNV2 model. Then the list is sorted using sorted alphanumerically, and then the images are converted into tensors. Then the list is converted to NumPy array for faster calculation. After this, the process of data augmentation is applied to increase the accuracy after training the model.

Preprocessing steps as mentioned below was applied

1. Resizing the input image (256 x 256)

2. Applying the color filtering (RGB) over the channels (Our model MobileNetV2 supports 2D 3 channel image).

3. convert the data and labels to NumPy arrays.

4. perform one-hot encoding on the labels.

4.5 Data augmentation

For the training of SSDMNV2 model, an enormous quantity of data is necessary to perform training effectively since due to the non-availability of an adequate amount of data for training the proposed model. The method of data augmentation is used to solve this issue. In this technique, methods like rotation, zooming, shifting, shearing, and flipping the picture are used for generating numerous versions of a similar picture. In the proposed model, image augmentation is used for the data augmentation process. A function image data generation is created for image augmentation, which returns test and train batches of data.

4.6 Face detection using OPEN-CV DNN

This model is included in the GitHub repository of OpenCV, starting from the latestversion. It is established on the 'Single Shot Multi-box Detector’

(SSD) and uses ‘ResNet-10’ architecture as the base-model. The Single Shot Multi-box Detector is similar to YOLO technique which takes only one shot to detect multiple objects present in an image using Multibox. It is significantly faster in speed and high-accuracy object detection algorithm. The images from which the model is trained have not been disclosed. Two versions of the model are made available by OpenCV:

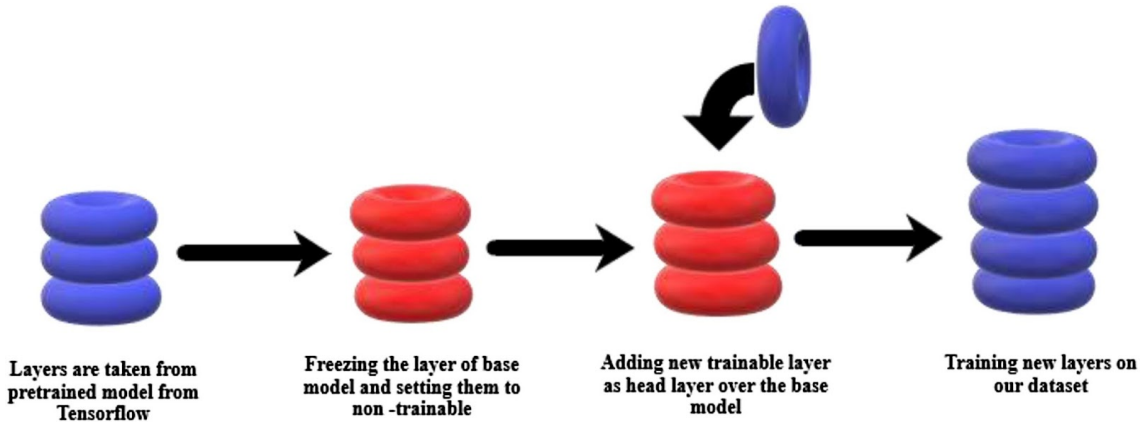
i.)CaffeImplementation (Jia et al., 2014) (Floating point 16 version)
ii.) Original TensorFlow Implementation (8-bit quantized version)

Caffe is a deep learning framework developed as a more faster, powerful and efficient alternative as compared to other object detection methods and is created and managed by Berkeley AI Research (BAIR) and community contributors.We have used the Caffemodel for our implementation for SSDMNV2 model to detect faces for the detection of facial masks. For this, the Caffemodel and prototxt files were loaded using *cv2.dnn.readNet (“path/to/prototxtfile”, “path/to/caffemodelweights”)* After applying the face detection model, we get the number of faces detected, the location of their bounding boxes, and the confidence score in those predictions. These outputs are then used as input for the face mask classifier. Using this approach to detect faces allows for real-time detection without much resource usage. It can also detect faces in different orientations, i.e., left, right, top, and bottom, with good accuracy. It is also able to detect the face mask of different sizes, i.e., big or small.

4.7 Classification of images using MobileNetV2

MobileNetV2 is a Deep Neural Network that has been deployed for the classification problem. Pretrained weights of ImageNet were loaded from TensorFlow. Then the base layers are frozen to avoid impairment of already learned features. Then new trainable layers are added, and these

layers are trained on the collected dataset so that it can determine the features to classify a face wearing a mask from a face not wearing a mask. Then the model is fine-tuned, and then the weights are saved. Using pre-trained models helps avoid unnecessary computational costs and helps in taking advantage of already biased weights without losing already learned features



. *Fig- 4.3(Pipeline of using Pretrained Model.)*

4.7.1 Building blocks of MobileNetV2

MobileNetV2 is a deep learning model based on Convolutional Neural Network, which uses the following layers and functions. The structure of MobileNetV2 has been shown in Fig. 4.4.

- ***Convolutional Layer***

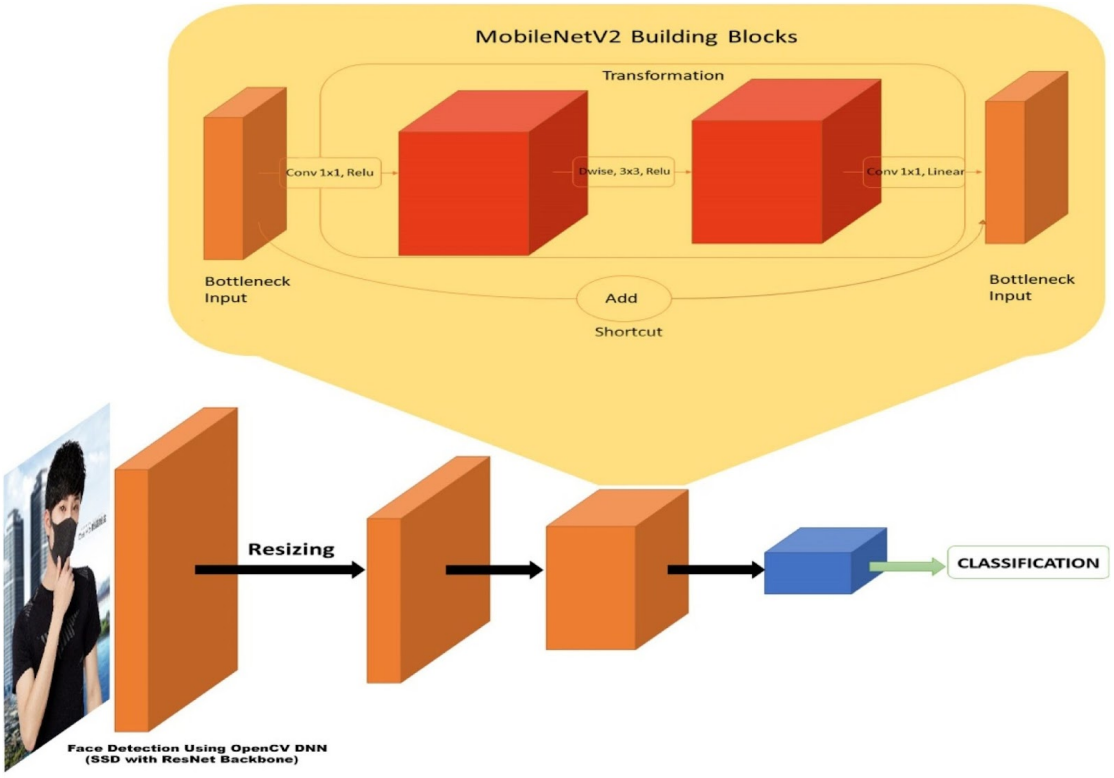


Fig- 4.4(Architecture of MobileNetV2)

This layer is the fundamental block of the Convolutional Neural Network. The term convolution implies a mathematical combination of two functions to get the third function. It works on a sliding window mechanism, which helps in extracting features from an image. This helps in generation feature maps. The convolution of two functional matrices, one being the input image matrix A and the other being convolutional kernel B give us the output C as:

$$C(T) = (A * B)(x) = \int \infty - \infty A(T) \times B(T - x)\, dT$$

- ***Pooling Layer***

Applying the pooling operations can make calculations go faster by allowing a reduction in the size of the input matrix without losing many

features. Different kind of pooling operations can be applied out of which some are explained below:

i.)Max Pooling: It takes the maximum value present in the selected region where the kernel is currently at as the value for the output of matrix value for that cell.

ii.) Average Pooling: It takes the average of all the values that are currently in the region where the kernel is at and takes this value as the output for the matrix value of that cell. Fig. 4.6 shows the average-pooling operation.

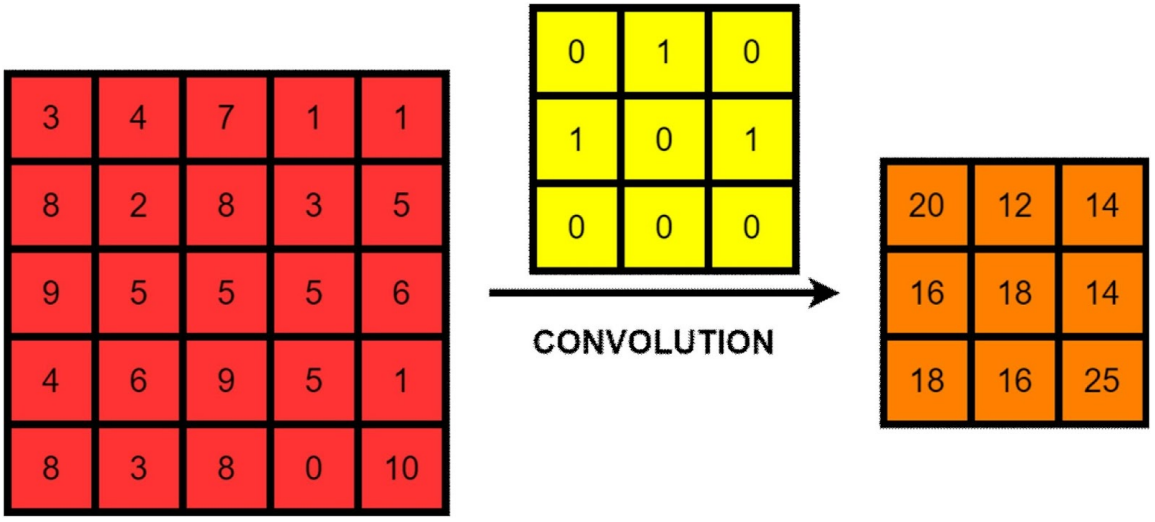


Fig-4.5(Convolutional Operation)

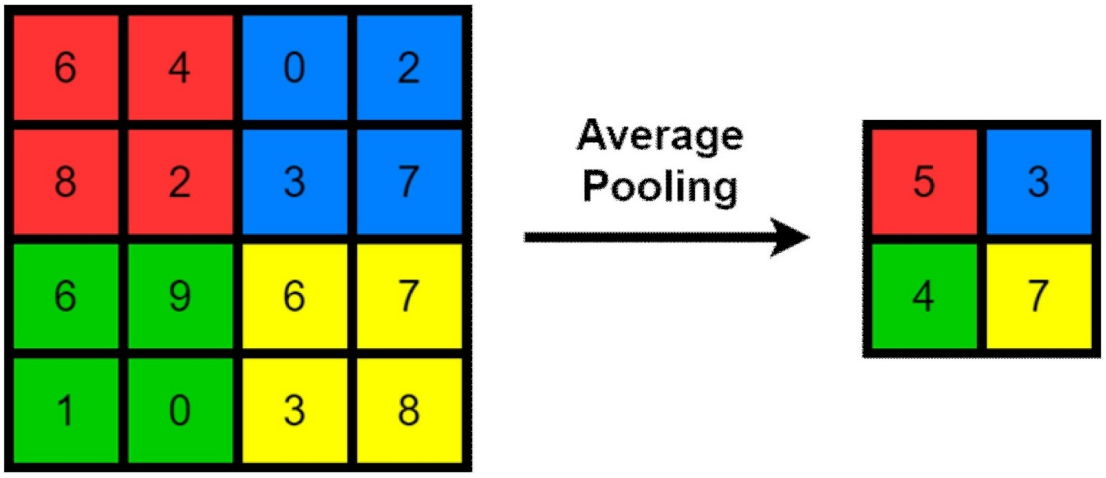


Fig-4.6 (Average-Pooling Operation)

- ***Dropout Layer***

This helps reduce the overfitting, which may occur while training by dropping random biased neurons from the model. These neurons can be a part of hidden layers as well as visible layers. The likelihood for a neuron to be dropped can be changed by changing the dropout ratio.

- ***Fully-Connected Layer***

These layers are appendedin the model and have full connections to activations layers. These layers help in classifying the given images in multi-class or binary classification. SoftMax is an example of activation functions used in these layers, and it provides the result of predicted output classes in terms of probability.

- ***Linear Bottlenecks***

As multiple matrix multiplications cannot be diminished to a single numerical operation, non-linear activation functions like ReLU6 are applied in the neural networks, so several discrepancies can be easily removed. Through this a multilayer neural network can be built. Since the ReLU activation function abandons the values which are less than 0. The dimensions of the network increase by escalating the number of channels to challenge the loss of information.

For a reversed residual block, layers of the blocks are compressed, and the contrary procedure is done as done above. This is done at the point where the skip connections are linked; this could affect the execution of the network. To deal with this, the concept of the linear bottleneck was introduced in which before adding the block to initial activation, the last convolution of the left-over block is given a linear output.

- ***Non-Linear Layer***

These layers usually follow the convolutional layers. Most commonly used non-linear functions include different kinds of Rectified Linear Unit (ReLU). Exponential ReLU , sigmoid function as well as tanh functions. Different kinds of Non-Linear Functions and their equations are shown below.

Sigmoid: $\sigma(x) = 1/1 + e- x$ (2)

Leaky Relu: $f(x) = \max(0.1x, x)$ (3)

Tanh: $f(x) = \tanh(x)$ (4)

Maxout : $f(x) = \max(wT1\ x + b1 , wT\ \ 2\ x + b2)$ (5)

Relu: $f(x) = \max(0, x)$ (6)

ELU: $f(x) =\{ x\ x \geq 0\ \alpha(ex - 1)\ x < 0\}$ (7)

4.8 Algorithms explaining the complete pipeline

The proposed SSDMNV2 methodology has been clearly explained using the two algorithms as shown below. First the images were pre-processed and were trained on the whole dataset. Second the model trained in the first part was used to detect the face mask with the appropriate accuracy. In Algorithm 1 shown below, images along with their pixel values were taken as an input, resized and normalized. Data augmentation technique was then applied on the images in order to get more accuracy. Data was then splitted into training and testing batches and MobilenetV2 model was applied on it. Adam optimizer was used to compile the whole model. In Algorithm 2 the model trained in previous part was then deployed on both classification on static images and on real time webcam. If the faces are detected using SSD, a bounding box showing the face of the person wearing a mask is shown in the output.

ALGORITHM 1: Pre-processing and Training on Dataset

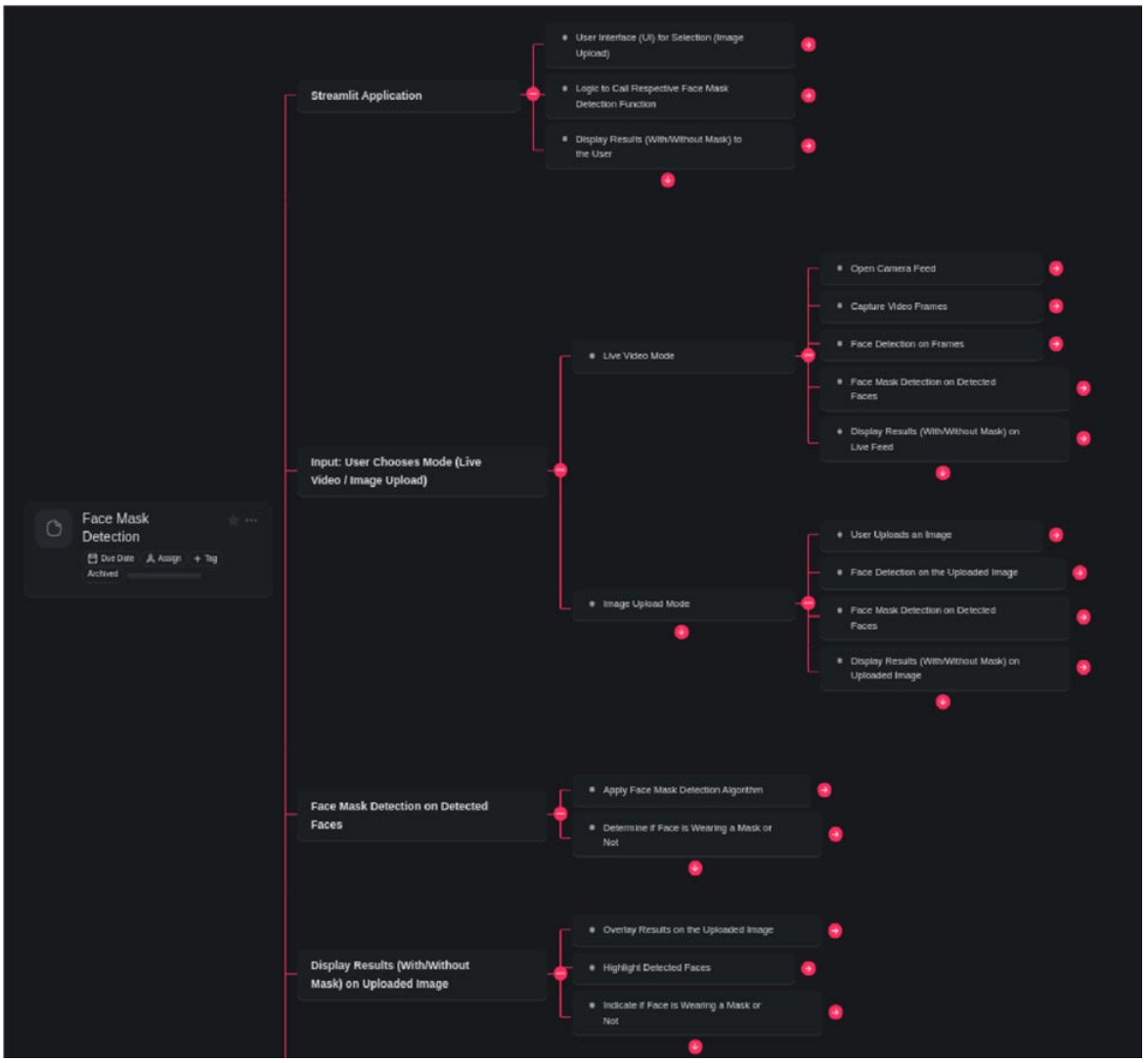
INPUT: Images along with their pixels values
OUTPUT: Trained Model

STEP 1: Load Images and their pixel values.
STEP 2: Process the images, i.e., resizing, normalization, and conversion to a 1D array.
STEP 3: Load the Filenames and their respective labels.
STEP 4: Perform Data augmentation and then split data into training and testing batches.
STEP 5: Load MobilenetV2 model from Keras. Train it on training batches and compile it using Adam optimizer.
STEP6: Save the model for future use.

ALGORITHM 2: Deployment of Face Mask Detector

INPUT: Choice of deployment and Files(optional).
OUTPUT: Images classified into the mask and no mask or Classification in Real-time.

STEP 1: Load saved classifier from disk. Also, load face detector from OpenCV.
STEP 2: If the choice is classification on image:
Load Image(s)
STEP 2.1: Apply face detection model to Detect faces in an image
STEP 2.2: If Faces are detected:
Crop face to bounding box coordinates from face detection model
Get predictions from the face classifier model.
Show predictions and save resultant image.
Else:
Show no output
STEP 3: If the choice is classification in real-time:
Load real-time feed from OpenCV
Read the feed frame by frame.
STEP 3.1: Apply face detection model to Detect faces in Frames read in real-time
STEP 3.2: If Faces are detected:
Crop face to bounding box coordinates from face detection model
Get predictions from the face classifier model.
Show output in a real-time feed
Else:
Show normal feed
STEP 4: End stream when q is pressed


Face Mask Detection
Due Date
Assign
Tag
Archived
Streamlit Application
User Interface (UI) for Selection (Image Upload)
Logic to Call Respective Face Mask Detection Function
Display Results (With/Without Mask) to the User
Input: User Chooses Mode (Live Video / Image Upload)
Live Video Mode
Open Camera Feed
Capture Video Frames
Face Detection on Frames
Face Mask Detection on Detected Faces
Display Results (With/Without Mask) on Live Feed
Image Upload Mode
User Uploads an Image
Face Detection on the Uploaded Image
Face Mask Detection on Detected Faces
Display Results (With/Without Mask) on Uploaded Image
Face Mask Detection on Detected Faces
Apply Face Mask Detection Algorithm
Determine if Face is Wearing a Mask or Not
Display Results (With/Without Mask) on Uploaded Image
Overlay Results on the Uploaded Image
Highlight Detected Faces
Indicate if Face is Wearing a Mask or Not

Deployment on Cloud

Cloud

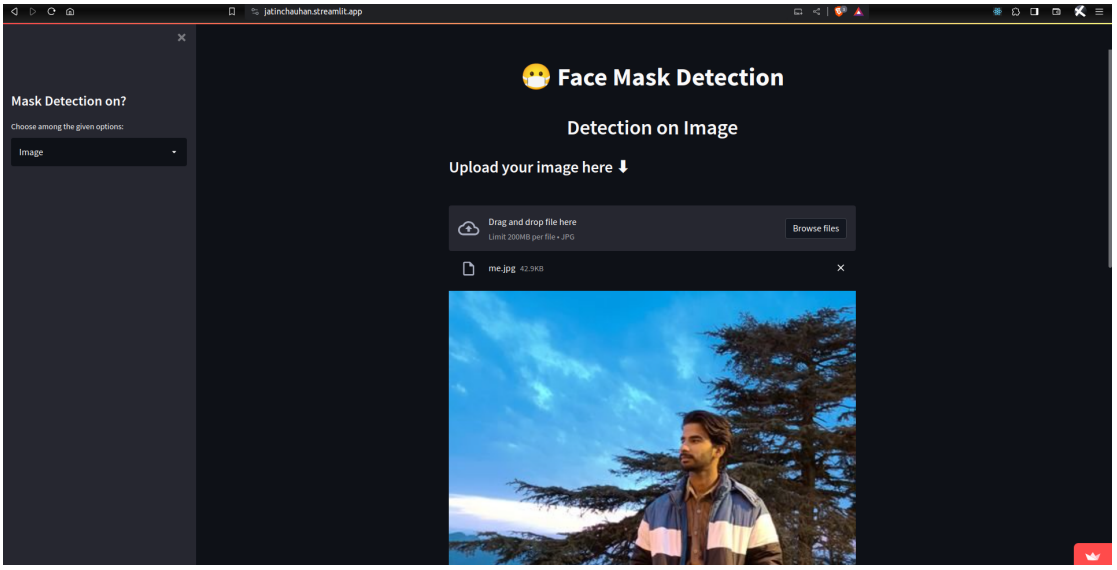
Simply put, cloud computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet (“the cloud”) to offer faster innovation, flexible resources, and economies of scale. You typically pay only for cloud services you use, helping lower your operating costs, run your infrastructure more efficiently and scale as your business needs change.

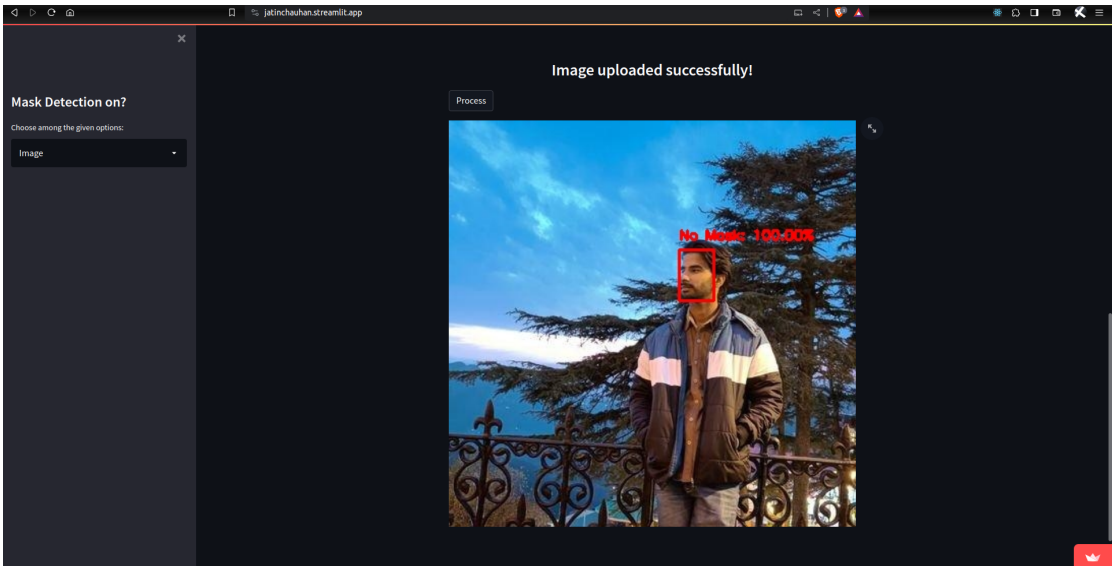
Wrapping up everything and Deploying on Cloud:

Now we have reached the last step which is deploying everything on the cloud.

We have just deployed **app.py** on cloud and we are good to go.

We deploy it on the cloud because it provides good performance.




jatinchauhan.streamlit.app
Mask Detection on?
Choose among the given options:
Image
Image uploaded successfully!
Process
No Mask: 100.00%

IMPLEMENTATION

MODEL Training (train_mask_detector.py)

```
# 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"])
```

Model deployment:

- **Detect live on Video Model** *(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 = []
```

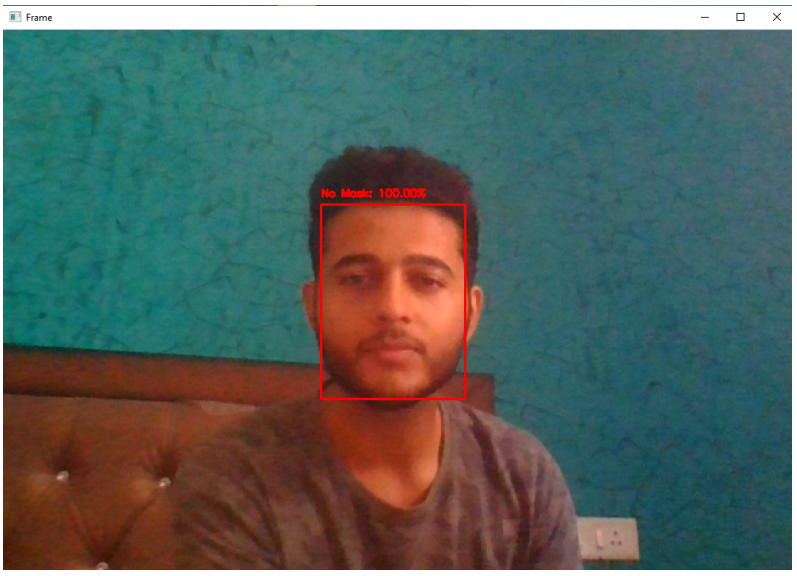
```
# loop over the detections
for i in range(0, detections.shape[2]):
# extract the confidence (i.e., probability) associated with
# the detection
confidence = detections[0, 0, i, 2]
# filter out weak detections by ensuring the confidence is
# greater than the minimum confidence
if confidence > args["confidence"]:
# 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]
if face.any():
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)
```

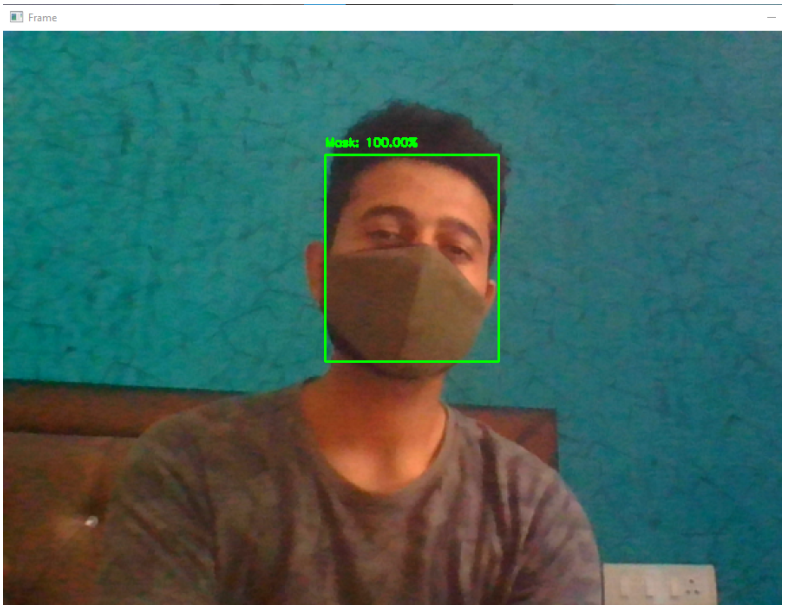
```
# load the face mask detector model from disk
print("[INFO] loading face mask detector model...")
maskNet = load_model(args["model"])

# 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()
```

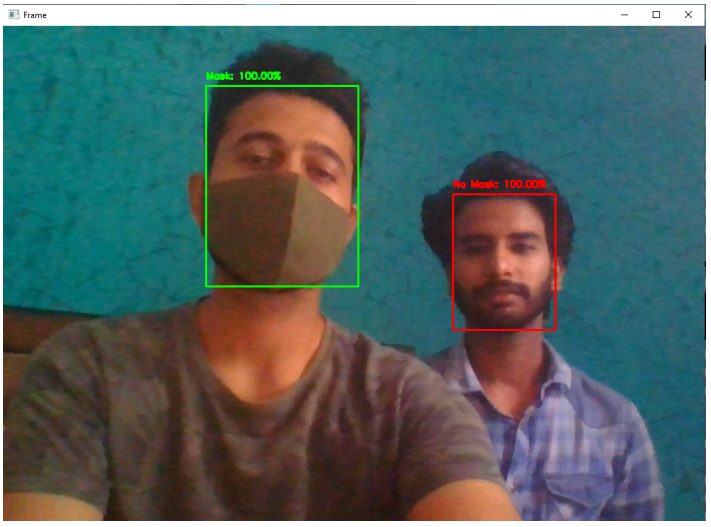
OUTPUT->
Usage: $ *python detect_mask_video.py*



Without Mask



With Mask



Multiface detection

- **Detect Image Model(*detect_mask_image.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
import numpy as np
import argparse
import cv2
import os
def mask_image():
    # 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"],
```

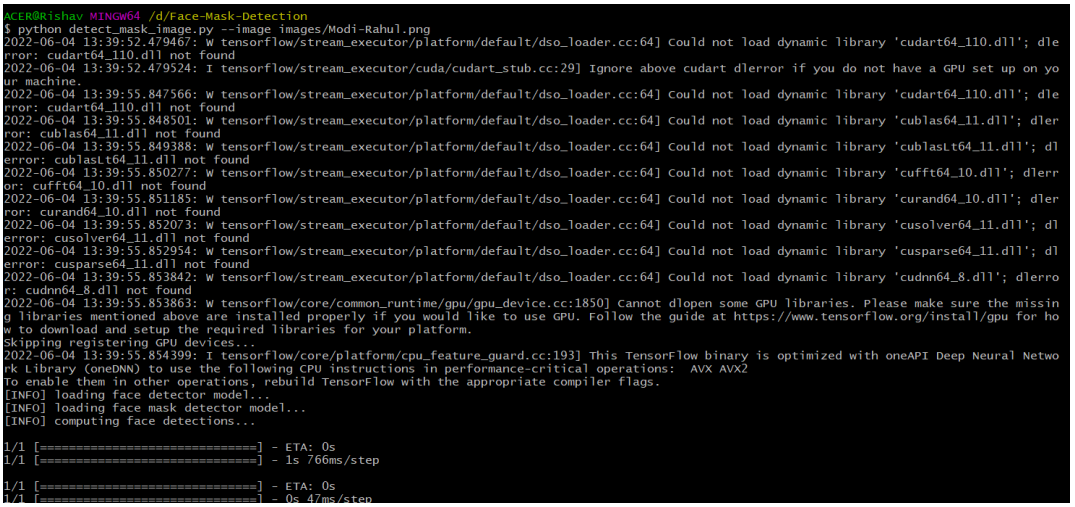
```
    "res10_300x300_ssd_iter_140000.caffemodel"])
net = cv2.dnn.readNet(prototxtPath, weightsPath)

# 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
```

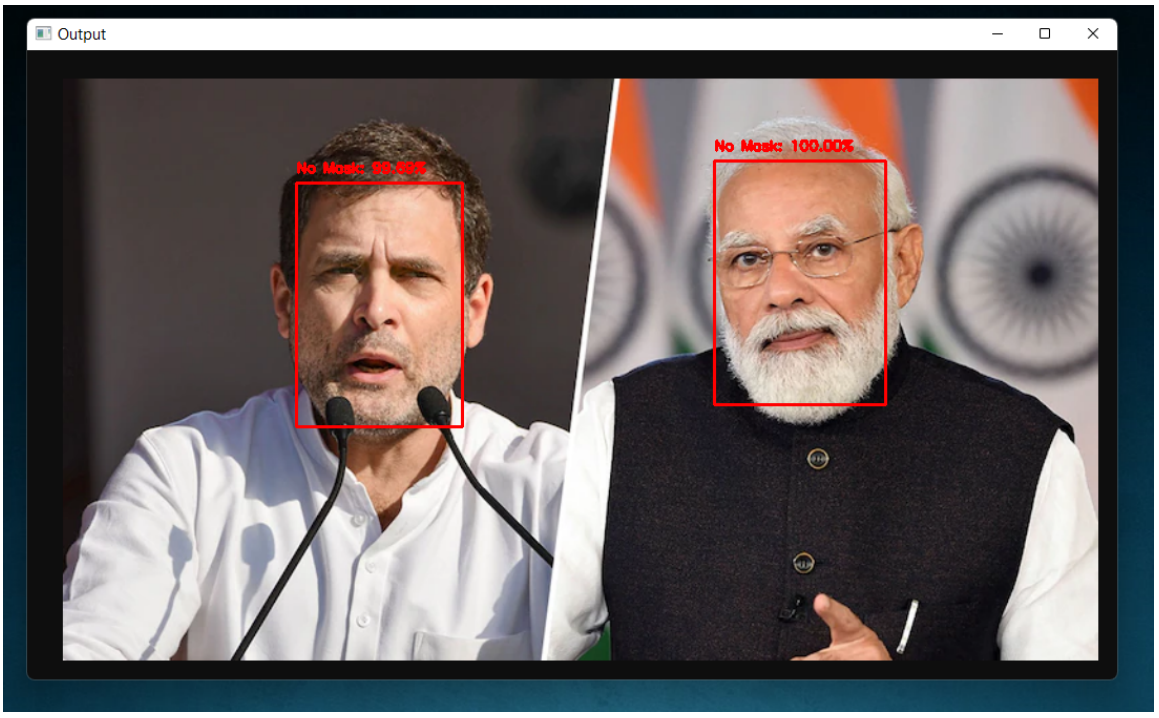
```
        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) *
100)

        # 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(0)

if __name__ == "__main__":
    mask_image()
```

Output:

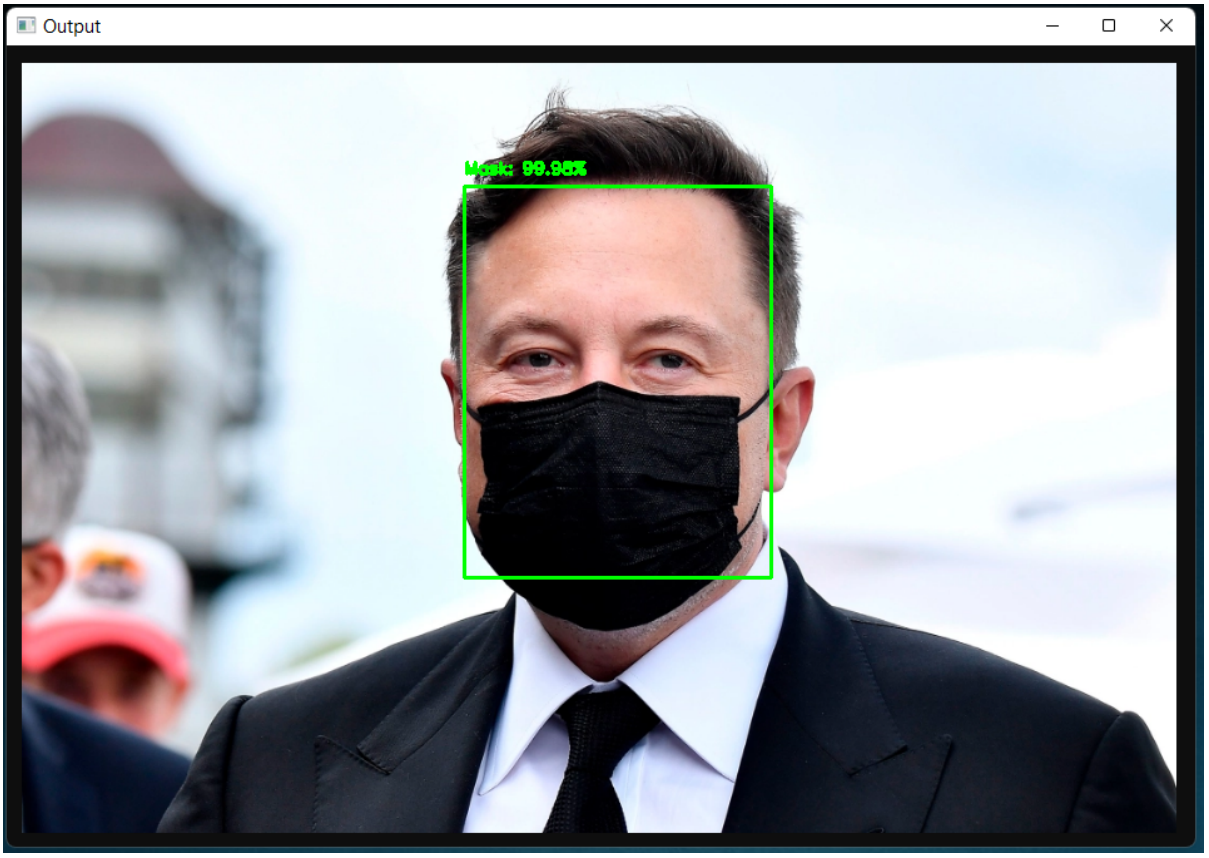
Code run in terminal-



Without mask



With Mask->



- ***App deployment (app.py)***

```
import streamlit as st
from PIL import Image, ImageEnhance
import numpy as np
import cv2
import os
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 detect_mask_image

# Setting custom Page Title and Icon with changed layout and sidebar
state
st.set_page_config(page_title='Face Mask Detector', page_icon='😷',
layout='centered', initial_sidebar_state='expanded')



def local_css(file_name):
    """ Method for reading styles.css and applying necessary changes to
HTML"""
    with open(file_name) as f:
        st.markdown(f'<style>{f.read()}</style>', unsafe_allow_html=True)
def mask_image():
    global RGB_img
    # load our serialized face detector model from disk
    print("[INFO] loading face detector model...")
    prototxtPath = os.path.sep.join(["face_detector", "deploy.prototxt"])
    weightsPath = os.path.sep.join(["face_detector",
                                    "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("mask_detector.model")

# load the input image from disk and grab the image spatial
# dimensions
image = cv2.imread("./images/out.jpg")
(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 > 0.5:
        # compute the (x, y)-coordinates of the bounding box for
```

```
            # the object
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

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

            # extract the face ROI, convert it from BGR to RGB channel
            # ordering, resize it to 224x224, and preprocess it
            face = 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)
```

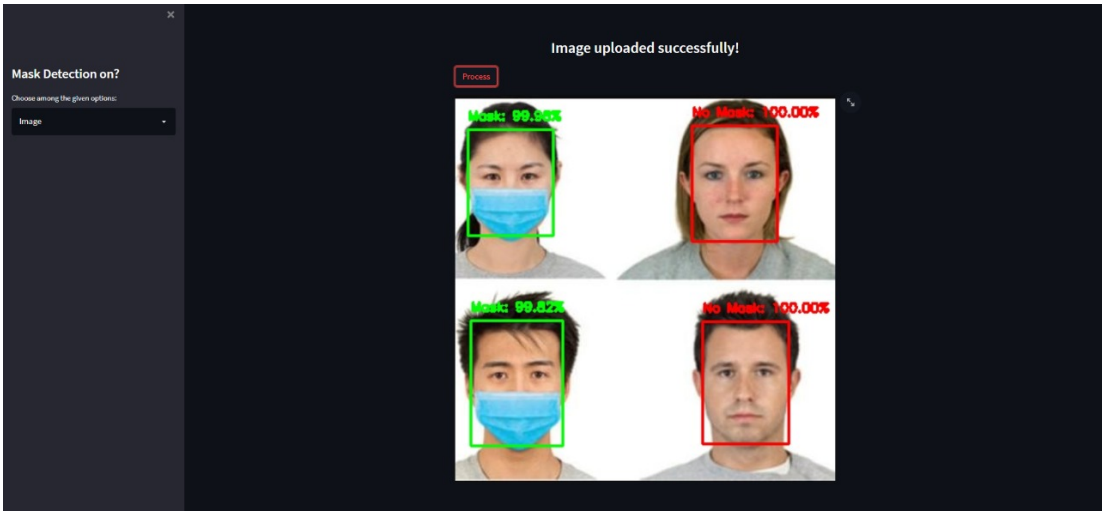
```
            # display the label and bounding box rectangle on the output
            # frame
            cv2.putText(image, label, (startX, startY - 10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)
            RGB_img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
mask_image()

def mask_detection():
    local_css("css/styles.css")
    st.markdown('<h1 align="center">😷 Face Mask Detection</h1>',
unsafe_allow_html=True)
    activities = ["Image", "Webcam"]
    st.set_option('deprecation.showfileUploaderEncoding', False)
    st.sidebar.markdown("# Mask Detection on?")
    choice = st.sidebar.selectbox("Choose among the given options:",
activities)

    if choice == 'Image':
        st.markdown('<h2 align="center">Detection on Image</h2>',
unsafe_allow_html=True)
        st.markdown("### Upload your image here ⬇")
        image_file = st.file_uploader("", type=['jpg'])  # upload image
        if image_file is not None:
            our_image = Image.open(image_file)  # making compatible to
PIL
            im = our_image.save('./images/out.jpg')
            saved_image = st.image(image_file, caption='',
use_column_width=True)
```

```
            st.markdown('<h3 align="center">Image uploaded
successfully!</h3>', unsafe_allow_html=True)
            if st.button('Process'):
                st.image(RGB_img, use_column_width=True)

    if choice == 'Webcam':
        st.markdown('<h2 align="center">Detection on Webcam</h2>',
unsafe_allow_html=True)
        st.markdown('<h3 align="center">This feature will be available
soon!</h3>', unsafe_allow_html=True)
mask_detection()
```

OUTPUT:

Before processing-

After processing-



CHAPTER 5
EXPERIMENTAL RESULT

All the experimental trials have been conducted on a laptop equipped by a i5 8th generation, 8 GB of RAM with 2 GB of VRAM. The Jupyter Notebook software equipped with Python 3.8 kernel was selected in this research for the development and implementation of the different experimental trails. The metrics selected for evaluation of SSDMNV2 model is explained below.

Accuracy = Tp + Tn / (Tp + Fp + Fn + Tn) (8)

Precision = Tp / (Tp + Fn) (9)

Recall = Tp / (Tp + Fn) (10)

f 1 score = 2 ∗ Recall ∗ Precision / (Recall + Precision) (11)

Where Tp = True positive,

Tn = True negative,

In above formulas, True positive values refer to images which were labelled true and after prediction by model gave true result. Likewise, for True negative refers to images which were labelled true but after prediction resulted in false result. False positive refers to images which were labelled false and after prediction resulted in false hence false positives. False negative refers to images which were labelled false and after prediction resulted in true hence false negatives. The accuracy was a good measure to start with, because the classes were balanced. Precision gave the measure of positive predicted values. Recall gave the ability to a classifier to find all positive samples and f1 score gave the measure of test accuracy. These evaluation metrics were chosen because of their ability to give best results in a balanced dataset.

5.1 Pre-processing results

This part explains the results of pre-processing the data and then training on data. First, a function named sorted_ alphanumerically is used to sort the list in lexicographical order. A function pre-processing is defined, which takes the folder to the dataset as

input, then loads all the files from the folder and resizes the images according to the model. Then after the list is sorted, the images are converted into tensors. Then the list is converted to a NumPy array for faster calculation. After applying pre-processing, the accuracy of our model increased significantly. After this, the process of data augmentation is applied to increase the accuracy after training the model.

5.2 Data augmentation results

The results of the SSDMNV2 model before data augmentation are shown in Fig. 5.1, Fig. 5.2. The model without data augmentation showed significant decrease in its growth as when trained on 100 epochs after data augmentation. In Fig. 5.1 depicting training accuracy without data augmentation, the model struggled to learn features till 60 epochs in validation accuracy and became stable afterwards. After 100 epochs the training accuracy came out to be 87.51 % as compared to 92.64 % training accuracy when data augmentation is applied. In Figure 8depicting training loss on a train set without data augmentation shows the training loss of the model till 10 epochs and started with approximately at 0.5 loss on validation loss maximum validation loss at a peak of 1.6 even at 80 epochs. Fig. 5.3, Fig. 5.4 shows improvement after applying data augmentation which could not be achieved before data augmentation.

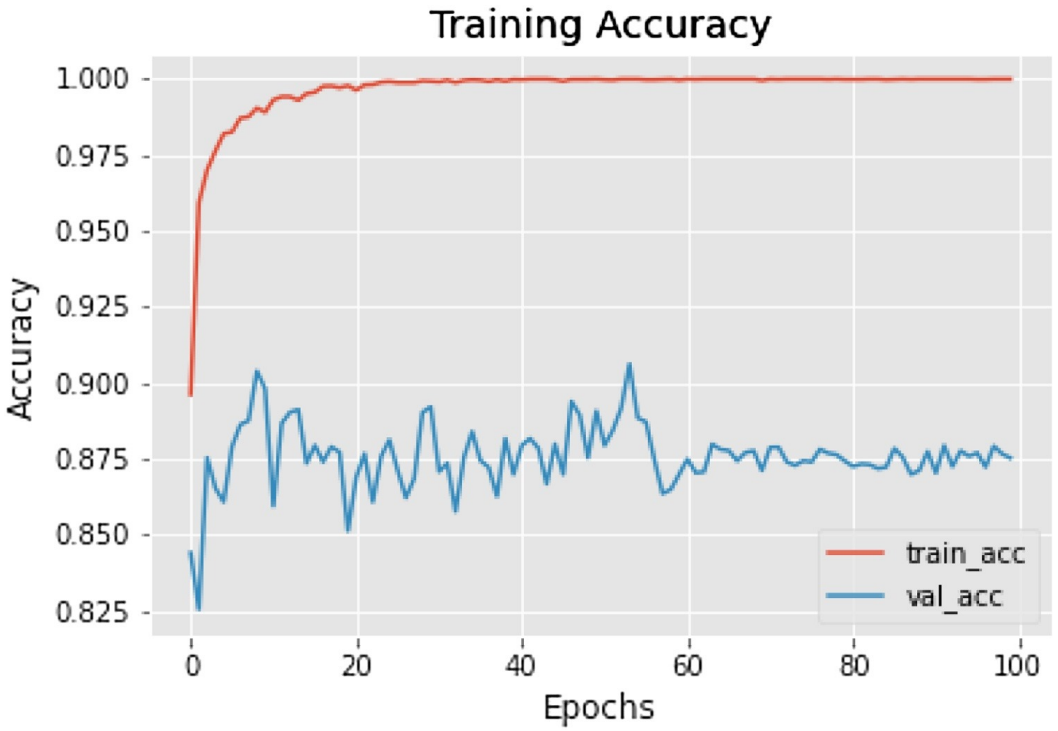
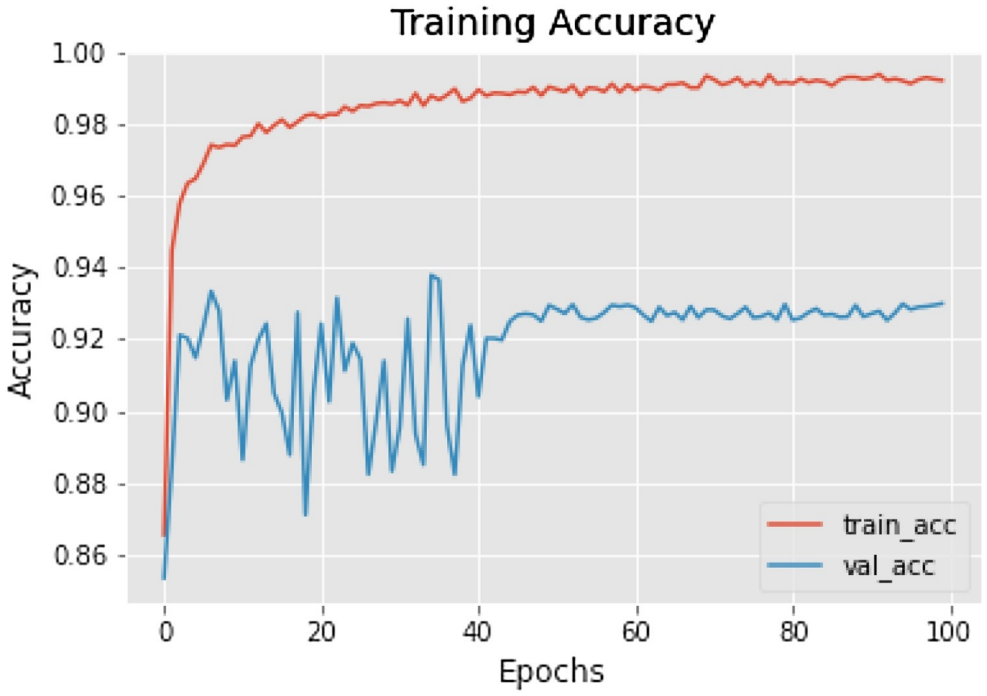
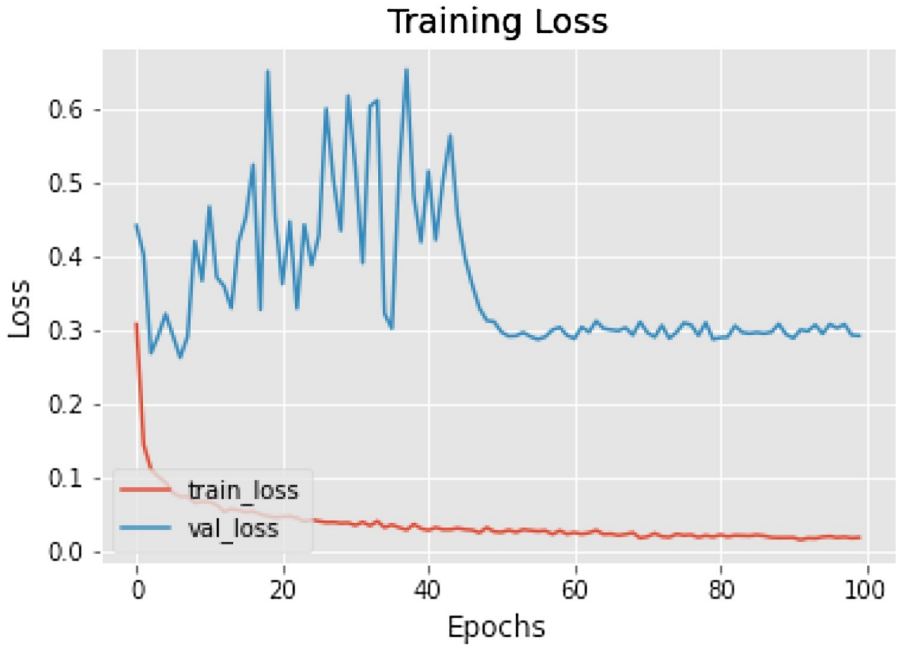


Fig-(5.1)Training accuracy curve (Without data augmentation).

Fig(5.2) Training loss curve (Without data augmentation).



Fig(5.3) Training accuracy curve.

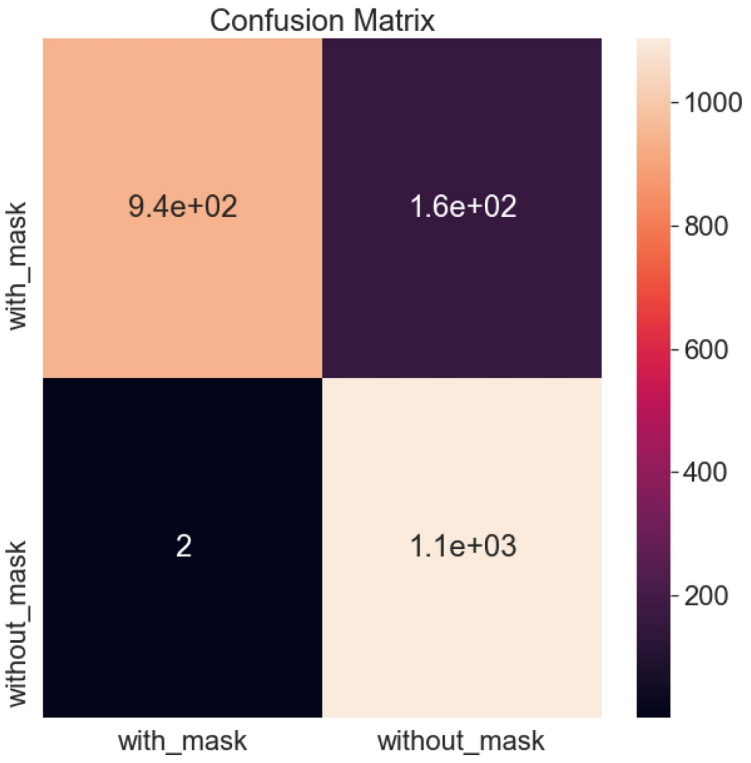


Fig(5.4) Training loss curve on the train.

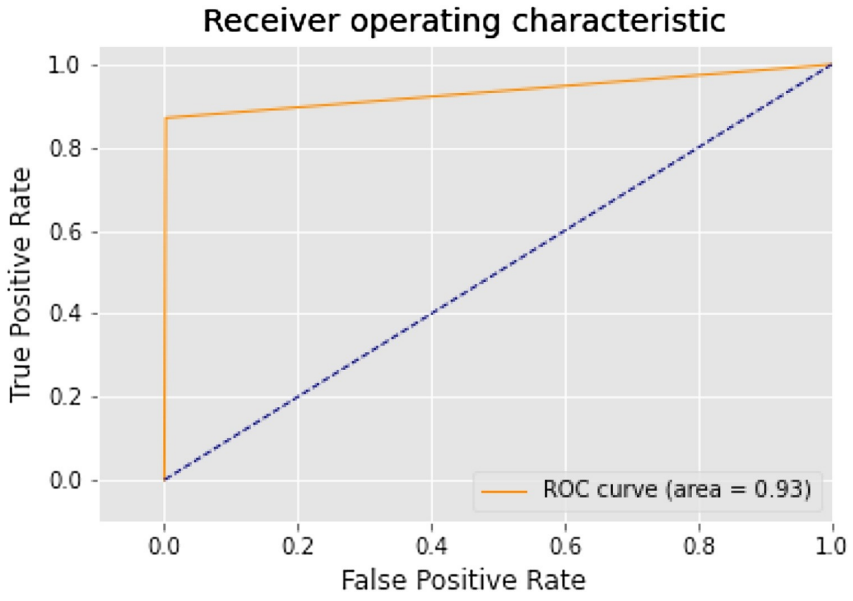
5.3 Evaluation of SSDMNV2 model

To solve the binary classification, a deep learning model problem is used in this paper. Keras is used to make a classification model in this paper, which is an advanced-level artificial neural networks API. Datasets for the (training, and testing) are split up to (80 % for training, 20 % for testing phase).The evaluation metrics used in this paper are accuracy, the area under the Receiver Operating Characteristics (ROC) curve, classification report, confusion matrix, and comparison of models. The plots are based on model accuracy; the pyplot command style function makes matplotlib work like MATLAB. The accuracy gives us the level of correct prediction of the masked person identified by the machine by the proposed model. In Fig. 5.3 depicting training accuracy with data augmentation, the model struggled to learn features till 42 epochs in validation accuracy and became stable afterwards. After 100 epochs the training accuracy came out to be 92.64 %. The average accuracy of our model is ‘93 %’ for predicting if a person is wearing a mask or not on a validation dataset, as shown in Fig. 5.3. The red curve shows the training accuracy, which is nearly equal to 99 %, whereas the blue line represents the accuracy of the validation dataset. The training loss curve corresponding to training andvalidation is shown in Fig. 5.4. Here, the red line shows the loss in training dataset less than 0.1, whereas the blue line depicts the training loss on the validation dataset.

The confusion matrix shown in Fig. 5.5 depicts a form to compare the labels, model prediction, and actual labels it was supposed to predict. It is showing where the model is getting confused. The confusion matrix is plotted with the help of heatmap showing a 2D matrix data in graphical format. It has successfully identified 941 true positives, 163 false negatives, 2 false-positive, and 1103 true negative out of 2209 images used for validation.The Roc curve compares a model’s ‘true positive rate’ (tpr) from a model’s with ‘false positive rate’ (fpr), as shown in Fig. 5.5. The model's roc accuracy score is close to the ideal roc curve but not the same. The roc accuracy score showing 93 % predicts the correctness of predicting the model values.



fig(5.5)Confusion matrix.



fig(5.6)Roc curve

CHAPTER 6

Conclusion

In the proposed face mask detection model named SSDMNV2, both the training and development of the image dataset, which was divided into categories of people having masks and people not having masks have been done successfully. The technique of OpenCV deep neural networks used in this model generated fruitful results. Classification of images was done accurately using the MobilenetV2 image classifier, which is one of the uniqueness of the proposed approach.

Many existing researches faced problematic results, while some were able to generate better accuracy with their dataset. The problem of various wrong predictions have been successfully removed from the model as the dataset used was collected from various other sources and images used in the dataset was cleaned manually to increase the accuracy of the results. Real-world applications are a much more challenging issue for the upcoming future. The SSDMNV2 model should hopefully help the concerned authorities in this great pandemic situation which had largely gained roots in most of the world; other researchers can use the dataset provided in this paper for further advanced models such as those of face recognition, facial landmarks, and facial part detection process.

In scenarios such as community access, campus governance, and enterprise resumption, the algorithm will provide contactless facial authentication. Our research has strengthened the scientific and technological capabilities of the planet.

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. Chenet al., "Characterization of a novel coronavirus associated with severe acute respiratorysyndrome,"science, vol. 300, no. 5624, pp. 1394–1399, 2003.

[2] Z. A. Memish, A. I. Zumla, R. F. Al-Hakeem, A. A. AlRabeeah, and G. M. Stephens, "Family cluster of middle east respiratory syndrome coronavirus infections,"New England Journal of Medicine, vol. 368, no. 26, pp.2487–2494, 2013.

[3] Y. Liu, A. A. Gayle, A. Wilder-Smith, and J. Rocklöv, "The reproductive number of covid-19 is higher comparedto sars coronavirus,"Journal of travel medicine, 2020.

[4] Y. Fang, Y. Nie, and M. Penny, "Transmission dynamics of the covid-19 outbreak and effectiveness of governmentinterventions: A data-driven analysis,"Journal of medical virology, vol. 92, no. 6, pp. 645–659, 2020.

[5] N. H. Leung, D. K. Chu, E. Y. Shiu, K.-H. Chan, J. J. McDevitt, B. J. Hau, H.-L. Yen, Y. Li, D. KM, J. Ipet al.,"Respiratory virus shedding in exhaled breath and efficacy of face masks."

[6] S. Feng, C. Shen, N. Xia, W. Song, M. Fan, and B. J. Cowling, "Rational use of face masks in the covid19pandemic,"The Lancet Respiratory Medicine, 2020. [7] Z. Wang, G. Wang, B. Huang, Z. Xiong, Q. Hong, H. Wu, P. Yi, K. Jiang, N. Wang, Y.

Peiet al., “Masked facerecognition dataset and application,”arXiv preprint arXiv:2003.09093, 2020.[10]Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, “Object detection with deep learning: A review,”IEEE transactions on neural networks and learning systems, vol. 30, no. 11, pp. 3212–3232, 2019.

[8] A. Kumar, A. Kaur, and M. Kumar, “Face detection techniques: a review,”Artificial Intelligence Review, vol. 52,no. 2, pp. 927–948, 2019.D.-H. Lee, K.-L. Chen, K.-H. Liou, C.-L. Liu, and J.-L. Liu, “Deep learning and control algorithms of direct perception for autonomous driving,2019