**FACE MASK DETECTION SYSTEM USING CONVOLUTIONAL NEURAL NETWORK (CNN)**

*A Major Project report submitted in partial fulfillment of the requirements for the degree of*

## BACHELOR OF TECHNOLOGY

IN

## ELECTRONICS AND TELEMATICS ENGINEERING

SUBMITTED BY

## D. Sravani (17251A1705)

**Vempati Divya (17251A1730)**

**D. Rachitha (17251A1734)**

**Shivangee Dixit (17251A1753)**

Under the guidance of

## Dr. M. Vijaya Lakshmi,

## Asst. Prof, ETM Department

**Department of Electronics & Telematics Engineering**

**G. Narayanamma Institute of Technology & Science (For Women)**

### Autonomous

**(NAAC & NBA Accredited Institution)**

**(**Affiliated to Jawaharlal Nehru Technological University, Hyderabad, Telangana.)

Shaikpet, Hyderabad-500104, Telangana, INDIA

May, 2021

**Department of Electronics and Telematics Engineering**

# CERTIFICATE

Date: 29th May, 2021

This is to certify that the project report entitled “**Face Detection System using Convolutional Neural Network”,** is a bonafide work carried out by

## D.Sravani

**Vempati Divya D. Rachitha Shivangee Dixit**

**(17251A1705) (17251A1730) (17251A1734) (17251A1753)**

in partial fulfillment of the requirement for the degree of **BACHELOR OF TECHNOLOGY** in **ELECTRONICS & TELEMATICS** by the Jawaharlal Nehru Technological University, Hyderabad during the academic year 2017-21.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

### Internal Guide Head of Department

Dr. M Vijaya Lakshmi Dr. K. Rama Linga Reddy

Assistant Professor Professor and HOD

Department of ETE Department of ETE

**EXTERNAL EXAMINER**

# ACKNOWLEDGEMENT

We are extremely grateful to **Dr. K. Rama Linga Reddy**, Head of Department, Electronics and Telematics Engineering (ETM) for help through thought provoking discussions and suggestions that were extended to us with immense care and zeal throughout our work

The satisfaction and knowledge that we have acquired during the course of this project owes its success to many distinguished personnel.

It is our great pleasure to acknowledge our profound sense of gratitude to our project guide **Dr. M. Vijaya Lakshmi** for her invaluable and inspiring guidance comments, invigorating suggestions and encouragement throughout the course of this project. We offer our sincere thanks to her for readily responding to our request to do the project under her guidance.

We sincerely thank **Mr. G. Krishna Reddy** for helping throughout the project period. We wish our deep sense of gratitude to our principal **Dr. K. Ramesh Reddy** for his efforts in helping us in every possible way. We would like to thank our Staff for having guided us throughout our academic course, the result of which manifested in the project work undertaken by us. We would like to thank our family and friends for being supportive all the time and we are very much obliged towards them.

This satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose encouragement and guidance has been a source of inspiration through the course of the project. With immense pleasure we take this opportunity to thank one and all who have helped for making this project possible.

D.Sravani

Vempati Divya D.Rachitha Shivangee Dixit

(17251A1705) (17251A1730) (17251A1734) (17251A1753)

### ABSTRACT

Rapid worldwide spread of several viruses has resulted in a global pandemic. Correct facemask wearing is valuable in infectious disease control, but the effectiveness of facemasks has been diminished mostly due to improper wearing.

The major transmission of the virus is either through droplets of saliva or the discharge from the nose when the host (infected person) coughs or sneezes. Social distancing, use of hand gloves, face shield, use of sanitizer and using face masks is an effective way to stop the transmission of the virus from one person to the other.

Technology holds the key here. We introduce a Deep Learning based system that can detect instances where face masks are not used properly. Our system consists of a dual stage Convolutional Neural Network (CNN) architecture capable of detecting masked and unmasked faces and can be integrated with pre-installed CCTV cameras. This will help track safety violations, promote the use of face masks, and ensure a safe working environment.

This study involves creating a Computer Vision based face mask detector using TensorFlow, Keras and Deep Neural Networks which can easily identify whether a person is wearing mask or not. This can be helpful for many organizations like Hospitals, Airport, Offices, Industries, Shopping malls etc. Our method was trained with 1376 images with 686 images of no facemask wearing, and 690 images of correct facemask wearing.

## INDEX

GNITS Certificate i

[Acknowledgement ii](#_TOC_250003)

[Abstract iii](#_TOC_250002)

[List of Figures vii](#_TOC_250001)

[List of Acronyms ix](#_TOC_250000)

CONTENTS

**CHAPTER 1: INTRODUCTION 2**

**CHAPTER 2: SOFTWARE REQUIREMENTS 4**

* 1. Python ……………………………………………………………4
  2. PyCharm… 5
  3. Sublime Text 5

**CHAPTER 3: ALGORITHMS AND LIBRARIES 7**

* 1. Machine Learning 7
  2. Deep Learning………………………………….................................................8
  3. Computer Vision 9
  4. Tensorflow 10
  5. OpenCV 11
  6. Keras 13
  7. PyTorch 14

**CHAPTER 4: PROPOSED METHODOLOGY 15**

* 1. Approach 15
  2. Data Pre-Processing 16
  3. Frameworks And Architectures 16

4.3.1 Deep Learning Frameworks 16

4.3.2 MobileNetV2 17

**CHAPTER 5: LIBRARIES 18**

5.1 Tensorflow … 18

5.2 Keras … 18

# 5.3 MobileNetv2 ……………………………..………………………………… 19

# 5.4 NumPy ……..……………………………..…………………………………20

5.5 Argparse … 20

5.6 OpenCV … 21

5.7 Flask 21

# 5.8 Flask-Request………………………………………………………………..22

# 5.9 OS Module.…………………………………………………………………..22

5.10 Base64… 23

5.11 Python Imaging Library … 23

5.12 IO Module … 23

5.13 Flask-CORS … 24

**CHAPTER 6: PROJECT IMPLEMENTATION 25**

* 1. Collecting or Loading Dataset … 25

6.1.1 Prepare own dataset…………………………..………………………..26

# 6.6.2 Download Dataset...…………………………..….…………………… 28

* 1. Object Detection … ..32

# 6.2.1 Traditional Object Detection…………………………………………..32

# 6.2.2 Convolutional Neural Networks………………………..……………...32

# 6.2.3 Modern Object Detection Algorithm...………………………………...32

* 1. Training Mask Detector … 33
  2. Face Mask Detector for Images … 42
  3. Face Mask Detection in Webcam Stream…………………………………….50

# 6.5.1 Two-Fold Process to Identify ………………………………………....50

# 6.5.1.1 Identify the face in the webcam………...................………..50

# 6.5.1.2 Classify faces based on the mask………………………… .50

# 6.5.2 Use Cases……………………. ………………………………..………51

# 6.5.3 Code ………………………………….…………………………...…...53

* 1. Web Application of Mask Detector 60

# 6.6.1 Python Code……………………………………………………………60

# 6.6.2 HTML Code……………………….…………………………………...64

**CHAPTER 7:CONCLUSION 69**

7.1 Future Scope …………………………………………………………..…….69

**REFERENCES 71**

# LIST OF FIGURES

**Figure No. Figure Name Page No.**

3.1 Machine Learning 7

3.2 Deep Learning 8

3.3 Computer Vision 10

3.4 Tensorflow Architecture 11

3.5 OpenCV 12

4.1 Data at source 15

6.1 Design Process 25

6.2 Without Masks 28

6.3 Image Without Masks 29

6.4 Applying Face Detection 30

6.5 Extract Face ROI 30

6.6 Detect Facial Landmarks 31

6.7 With Mask 31

6.8 Plot 42

6.9 Output – With Mask 47

6.10 Output – Without Mask 48

6.11 Output – With and Without Mask 49

6.12 Face Mask Detection Flow for Webcam 51

6.13 Use Case in Airports 51

6.14 Use Case in Hospitals 52

6.15 Use Case in Offices and Working Spaces 52

6.16 Use Case in Government Sector 53

6.17 Home Page 65

6.18 Click on choose file to select an image 65

6.19 Choose an image from PC 66

# LIST OF FIGURES

**Figure No. Figure Name Page No.**

6.20 The image name will be shown on the home page 66

6.21 Click on submit 67

6.22 Image will be displayed 67

6.23 Result 68

# CHAPTER 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. Here we introduce a mask face detection model that is based on computer vision and deep learning.

The proposed model can be integrated with surveillance cameras to impede the COVID-19 transmission by allowing the detection of people who are wearing masks not wearing face masks. The model is integration between deep learning and classical machine learning techniques with OpenCV, tensor flow and Keras. We have used deep transfer learning for feature extractions and combined it with three classical machine learning algorithms. We introduced a comparison between them to find the most suitable algorithm that achieved the highest accuracy and consumed the least time in the process of training and detection.

# CHAPTER 2

# SOFTWARE REQUIREMENTS

# Python

**Python** is an object-oriented programming language created by Guido Rossum in 1989. Since then, Python has been gaining popularity and is considered as one of the most popular and flexible server-side programming languages.

It is very user-friendly and is most popular for its easy to use syntax and readable code.

## Features of Python

Enlisted below are the various features of Python:

* + 1. **Simple and Easy to Learn**: Python is simple and easy to learn, read and write.
    2. **Free and open source:** Python is a free and open source software which means that a user can edit, modify or reuse the software’s source code. This gives the programmers an opportunity to improve the program functionality by modifying it.
    3. **Interpreted Language:** Python is an interpreted language, which means when we execute a python program, the interpreter executes the code line by line at a time. This makes debugging easy and thus is suitable for beginners.
    4. **Python is Interactive:** Interactive mode is a command line shell which gives immediate response for each statement.
    5. **Portable:** Python supports many platforms like Linux, Windows, MacOS, and Solaris.
    6. **Object-Oriented:** Python supports Object-Oriented technique of programming which provides a means of structuring programs so that properties and behaviors are bundled into individual objects.
    7. **Supports different programming models:** Python supports procedure- oriented programming as well as object-oriented programming.
    8. **Flexible:** Python code can invoke C and C++ libraries and can be called from and C++ programs, and can integrate with Java and .NET components.

## Prerequisites

* A system running windows 10 with admin privileges
* Command Prompt (comes with windows by default)
* A Remote Desktop Connection app (use if you are installing Python on a remote Windows server)

# PyCharm

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive [Python](https://www.jetbrains.com/help/pycharm/python.html), [web](https://www.jetbrains.com/help/pycharm/web-frameworks.html), and data science development.

PyCharm is available in three editions:

* **Community** (free and [open-sourced](https://github.com/JetBrains/intellij-community/blob/master/LICENSE.txt)): for smart and intelligent Python development, including code assistance, refactorings, visual debugging, and version control integration.
* **Professional** ([paid](https://www.jetbrains.com/pycharm/buy/%23commercial?billing=yearly)) : for professional Python, web, and data science development, including code assistance, refactorings, visual debugging, version control integration, remote configurations, deployment, support for popular web frameworks, such as Django and Flask, database support, scientific tools (including Jupyter notebook support), big data tools.
* **Edu** (free and [open-sourced](https://github.com/JetBrains/intellij-community/blob/master/LICENSE.txt)): for learning programming languages and related technologies with integrated educational tools.

PyCharm supports the following versions of Python:

* **Python 2:** version 2.7
* **Python 3:** from the version 3.6 up to the version 3.10

# Sublime Text

[Sublime Text 3](http://www.sublimetext.com/3) (ST3) is a lightweight, cross-platform code editor known for its speed, ease of use, and strong community support. It’s an incredible editor right out of the box, but the real power comes from the ability to enhance its functionality using Package Control and creating custom settings.

## Features

Let’s start by looking at a few of the default features of Sublime Text 3:

* + 1. Split Layouts
    2. Chrome-like Tabs Automatic loading of the last session
    3. Code Snippets
    4. Performance
    5. Vintage Mode

Sublime Text editor is supported by the following major operating systems −

* Windows
* Linux and its distributions
* OS X

You can download Sublime Text from its official website − [www.sublimetext.com](https://www.sublimetext.com/)

# CHAPTER 3

# ALGORITHMS AND LIBRARIES

# 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.

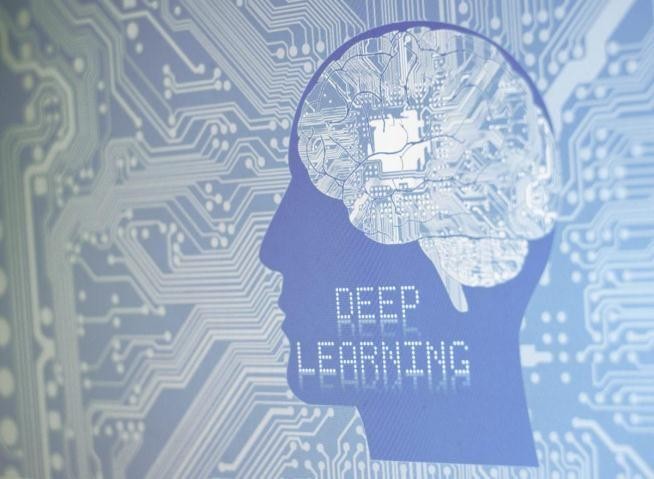


**Figure 3.1** Machine learning

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

# Deep Learning

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



**Figure 3.2**  Deep Learning

The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning. Deep learning excels on problem domains where the inputs (and even output) are analog. Meaning, they are not a few quantities in a tabular format but instead are images of pixel data, documents of text data or files of audio data. Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

# Computer Vision

Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do. Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that make sense to thought processes and can elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

The scientific discipline of computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, multidimensional data from a 3D scanner or medical scanning device. The technological discipline of computer vision seeks to apply its theories and models to the construction of computer vision systems. Computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do 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.



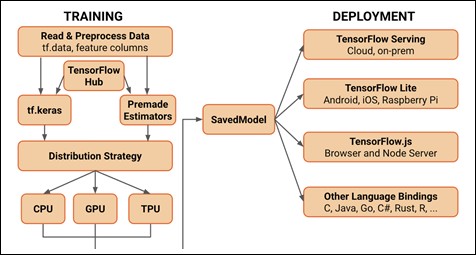
**Figure 3.3** Computer Vision

The image data can take many forms, such as video sequences, views from multiple cameras, or multidimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models for the construction of computer vision systems.

# TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google, TensorFlow is Google Brain's second- generation system. Version 1.0.0 was released on February 11, while the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). Tensor Flow is available on 64-bit Linux, macOS, Windows and mobile computing platforms including Android and iOS. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

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

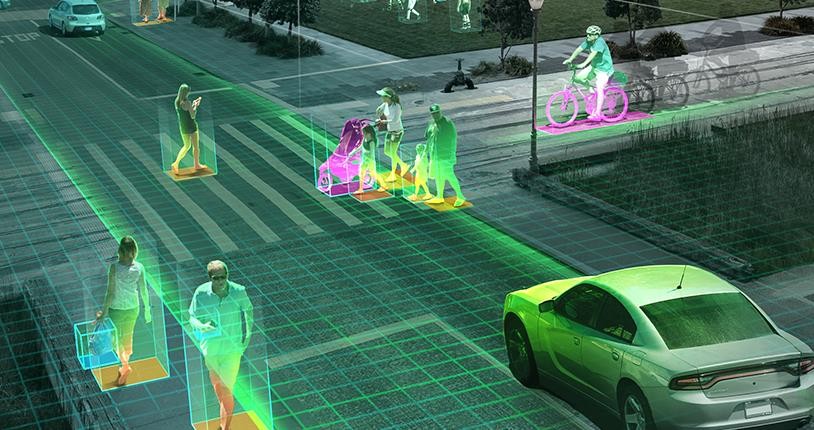


**Figure 3.4** Tensor Flow architecture

It can run on single CPU systems, GPUs as well as mobile devices and large-scale distributed systems of hundreds of machines.

# OpenCV

OpenCV (Open-Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.



**Figure 3.5** OpenCV

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high-resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery, and establish markers to overlay it with augmented reality, etc.

OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies. Along with well- established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many start-ups such as Applied Minds, Video Surf and Zeitera, that make extensive use of OpenCV. OpenCV’s deployed uses span the range from stitching street view images together detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan [9].

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

# 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.

It runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license. Keras was developed and maintained by François Chollet, a Google engineer using four guiding principles:

* + 1. **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.
    2. **Minimalism:** The library provides just enough to achieve an outcome, no frills and maximizing readability.
    3. **Extensibility:** New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.
    4. **Python:** No separate model files with custom file formats. Everything is native Python. Keras is designed for minimalism and modularity allowing you to define deep learning models and run them on top of a Theano or TensorFlow backend very quickly.

# PyTorch

PyTorch is an open-source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Face book’s AI Research lab (FAIR). It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface. Tensor computing (like NumPy) with strong acceleration via graphics processing units (GPU).

Deep neural networks built on a tape-based automatic differentiation system PyTorch defines a class called Tensor (torch Tensor) to store and operate on homogeneous multidimensional rectangular arrays of numbers. PyTorch Tensors are similar to NumPy Arrays but can also be operated on a CUDA-capable Nvidia GPU. PyTorch supports various sub-types of Tensors.

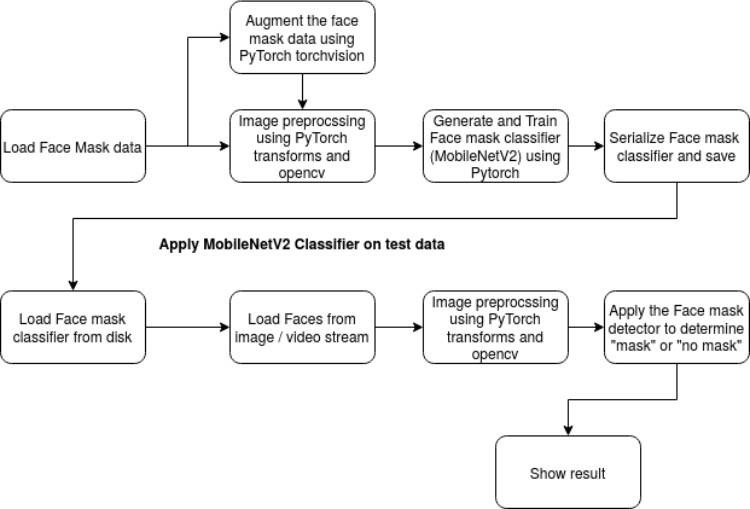
# CHAPTER 4

# PROPOSED METHODOLOGY

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

# Approach

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



**Figure 4.1** Data at Source

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.

# Data Pre-processing

Pre-processing steps as mentioned below was applied to all the raw input images to convert them into clean versions, which could be fed to a neural network machine learning model.

* + 1. Resizing the input image (256 x 256).
    2. Applying the color filtering (RGB) over the channels (Our model MobileNetV2 supports 2D 3 channel image).
    3. Scaling / Normalizing images using the standard mean of PyTorch build in weights.
    4. Center cropping the image with the pixel value of 224x224x3.
    5. Finally Converting them into tensors (Similar to NumPy array).

# Frameworks and Architectures

### Deep Learning Frameworks

To implement this deep learning network we have the following options.

* + - 1. Tensorflow
      2. Keras
      3. PyTorch
      4. Caffee
      5. MxNet
      6. Microsoft Cognitive Tool Kit

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

* + - * 1. Data Parallelism
        2. It looks like a Framework

### MobileNetV2

MobileNetV2 builds upon the ideas from MobileNetV1, using depth wise separable convolution as efficient building blocks. However, V2 introduces two new features to the architecture:

* + - 1. Linear bottlenecks between the layers and
      2. Shortcut connections between the bottlenecks.

The typical MobilenetV2 architecture has as many layers listed below. In PyTorch we can use the models library in Torch Vision to create the MobileNetV2 model instead of defining/building our own model [12]. The weights of each layer in the model are predefined based on the ImageNet dataset. The weights indicate the padding, strides, kernel size, input channels and output channels. MobileNetV2 was chosen as an algorithm to build a model that could be deployed on a mobile device. A customized fully connected layer which contains four sequential layers on top of the MobileNetV2 model was developed. The layers are

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

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

# CHAPTER 5

# LIBRARIES

These are the following important libraries which need to be installed at the time of running the code:

# Tensorflow

Tensorflow is a [free](https://en.wikipedia.org/wiki/Free_software) and [open-source](https://en.wikipedia.org/wiki/Open-source_software) [software library](https://en.wikipedia.org/wiki/Library_(computing)) for [machine learning](https://en.wikipedia.org/wiki/Machine_learning) and high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

It can be used across a range of tasks but has a particular focus on tra[ining](https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks#Training) and [inference](https://en.wikipedia.org/wiki/Statistical_inference) of [deep neural networks](https://en.wikipedia.org/wiki/Deep_neural_networks). Tensorflow is a symbolic math library based on [dataflow](https://en.wikipedia.org/wiki/Dataflow_programming) and [differentiable programming](https://en.wikipedia.org/wiki/Differentiable_programming). It is used for both research and production at [Google](https://en.wikipedia.org/wiki/Google).

## Installing

Install and update using pip:

pip install --upgrade tensorflow

# Keras

Keras is an [open-source](https://en.wikipedia.org/wiki/Open-source_software) [software](https://en.wikipedia.org/wiki/AI_software) library that provides a [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) interface for [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network). Keras acts as an interface for the [TensorFlow](https://en.wikipedia.org/wiki/TensorFlow) library.

Keras allows users to productize deep models on smartphones ([iOS](https://en.wikipedia.org/wiki/IOS) and [Android](https://en.wikipedia.org/wiki/Android_(operating_system))), on the web, or on the [Java Virtual Machine](https://en.wikipedia.org/wiki/Java_Virtual_Machine). It also allows use of distributed training of deep-learning models on clusters of [Graphics processing units (GPU)](https://en.wikipedia.org/wiki/Graphics_processing_unit) and [tensor](https://en.wikipedia.org/wiki/Tensor_processing_unit) [processing units (TPU)](https://en.wikipedia.org/wiki/Tensor_processing_unit).

## Installing

Install using pip:

pip install keras

# MobileNetV2

MobileNetV2 is a general architecture and can be used for multiple use cases. Depending on the use case, it can use different input layer size and different width factors. This allows different width models to reduce the number of multiply-adds and thereby reduce inference cost on mobile devices.

The number of parameters and number of multiply-adds can be modified by using the alpha parameter, which increases/decreases the number of filters in each layer. By altering the image size and alpha parameter, all 22 models from the paper can be built, with ImageNet weights provided.

## Functions

**MobileNetV2(…):** Instantiates the MobileNetV2 architecture.

**decode\_predictions (…)**: Decodes the prediction of an ImageNet model.

**Preprocess\_input (…)**: Preprocesses a tensor or Numpy array encoding a batch of images.

## Installing

Install using pip:

pip install image-classifiers

# Numpy

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

## Installing

Install using pip:

pip install numpy

# Argparse

The [argparse](https://docs.python.org/3/library/argparse.html#module-argparse) module makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and [argparse](https://docs.python.org/3/library/argparse.html#module-argparse) will figure out how to parse those out of [sys.argv](https://docs.python.org/3/library/sys.html#sys.argv). The [argparse](https://docs.python.org/3/library/argparse.html#module-argparse) module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

## Installing

Install using pip:

pip install argparse

# OpenCV

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

## Installing

Install using pip:

pip install opencv-python

# Flask

Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications. Flask is a lightweight [WSGI](https://wsgi.readthedocs.io/) web application framework. (WSGI is the Web Server Gateway Interface. It is a specification that describes how a web server communicates with web applications, and how web applications can be chained together to process one request.

WSGI is a Python standard described in detail in [**PEP 3333**](https://www.python.org/dev/peps/pep-3333).)

Flask is designed to make getting started quick and easy, with the ability to scale up to complex applications.It began as a simple wrapper around [Werkzeug](https://www.palletsprojects.com/p/werkzeug/) and [Jinja](https://www.palletsprojects.com/p/jinja/) and has become one of the most popular Python web application frameworks.

## Installing

Install and update using pip:

pip install –U Flask

# Flask – Request

[Flask](https://flask.palletsprojects.com/en/1.1.x/api/#flask.Flask) application handles a request, it creates a [Request](https://flask.palletsprojects.com/en/1.1.x/api/#flask.Request) object based on the environment it received from the WSGI server. Because a worker (thread, process depending on the server) handles only one request at a time, the request data can be considered global to that worker during that request. Flask uses the term context local for this.

Flask automatically pushes a request context when handling a request. View functions, error handlers, and other functions that run during a request will have access to the [request](https://flask.palletsprojects.com/en/1.1.x/api/#flask.request) proxy, which points to the request object for the current request.

## Installing

Install using pip:

pip install flask-requests

# OS Module

The OS module in python provides functions for interacting with the operating system. OS, comes under Python’s standard utility modules. This module provides a portable way of using operating system dependent functionality. The \*os\* and

\*os.path\* modules include many functions to interact with the file system.

### Installing

**4.10 Base64**

Install using pip:

**4.10 Base64**

pip install os-sys

# Base64

Base64 encoding allows us to convert bytes containing binary or text data to ASCII characters. By encoding our data, we improve the chances of it being processed correctly by various systems.

### Installing

Install using pip:

pip install pybase64

# Python Imaging Library

Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) additional [library](https://en.wikipedia.org/wiki/Library_(computing)) for the [Python programming](https://en.wikipedia.org/wiki/Python_(programming_language)) [language](https://en.wikipedia.org/wiki/Python_(programming_language)) that adds support for opening, [manipulating](https://en.wikipedia.org/wiki/Image_editing), and saving many different [image file formats](https://en.wikipedia.org/wiki/Image_file_formats).

## Installing

Install using pip:

pip install Pillow

# 5.12 IO Module

Python io module allows us to manage the file-related input and output operations. The advantage of using the IO module is that the classes and functions available allows us to extend the functionality to enable writing to the Unicode data.

## Installing

Install using pip:

pip install simpy.io

# Flask-CORS

CORS (Cross Origin Resource Sharing) allows access to resources from other domains, a good use case for this is a web app trying to fetch data from an API on another domain. CORS is only required when trying to fetch data from a browser, as browsers by default will block requests to different origins (domains).

CORS uses HTTP headers to tell the browser if it has permission to fetch the resources.

## Installing

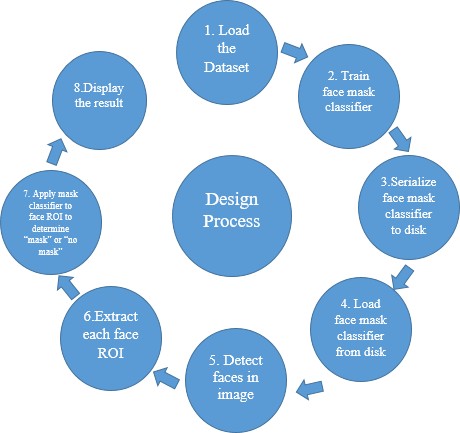
Install using pip:

pip install Flask-Cors

# CHAPTER 6

# PROJECT IMPLEMENTATION

Project implementation has various steps. These include:



**Figure 6.1** Design Process

# Collecting or Loading Dataset

There are two ways to collect the dataset.

* + 1. Prepare your own dataset.
    2. Download already existing dataset

For training the project we need large number of images.

* With mask
* Without mask

## Prepare own dataset

Here is the code through which one can prepare their own dataset.

### Code for Collecting Dataset With Mask:

import cv2

cam = cv2.VideoCapture(0) currentframe=0

while (True):

ret,frame = cam.read()

name ='./images/withMask/mimage ' + str(currentframe) + '.jpg' print('Creating...' +name)

cv2.imwrite(name,frame) # currentframe += 1

if currentframe>500: break

cv2.imshow('img',frame)

k = cv2.waitKey(30) & 0xff if k == 27:

break

cam.release() cv2.destroyAllWindows()

### Explanation:

Creates a folder ‘withMask’ and save the image name starting with ‘mimage’

e.g. mimage 0.jpg, mimage 1.jpg and so on. Here we are collecting 500 images of with mask.

cv2.VideoCapture(0) – means we are using the in-built camera to capture images.

### Code for Collecting Dataset Without Mask:

import cv2

cam = cv2.VideoCapture(0) currentframe=0

while(True):

ret,frame = cam.read()

name ='./images/withoutMask/wmimage ' + str(currentframe) + '.jpg' print('Creating...' +name)

cv2.imwrite(name,frame) # currentframe += 1

if currentframe>500: break

cv2.imshow('img',frame)

k = cv2.waitKey(30) & 0xff if k == 27:

break

cam.release() cv2.destroyAllWindows()

### Explanation:

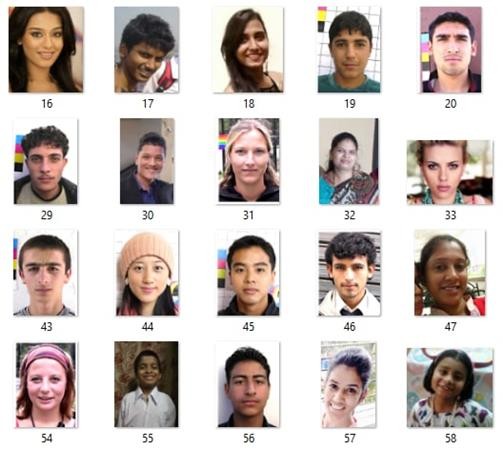
Creates a folder ‘withoutMask’ and save the image name starting with ‘wmimage’

e.g. wmimage 0.jpg, wmimage 1.jpg and so on. Here we are collecting 500 images of without mask.

cv2.VideoCapture(0) – means we are using the in-built camera to capture images.

## Download Dataset

There are many sources available in google. First we download the dataset consists of 686 images without mask.



**Figure 6.2** Without Masks

Then creating a custom computer vision Python code to add face mask to them, thereby creating an artificial (but still real-world applicable) dataset.

This method is actually a lot easier than it sounds once you [apply facial landmarks](https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/) [to the problem](https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/).

Facial landmarks allow us to automatically infer the location of facial structures, including:

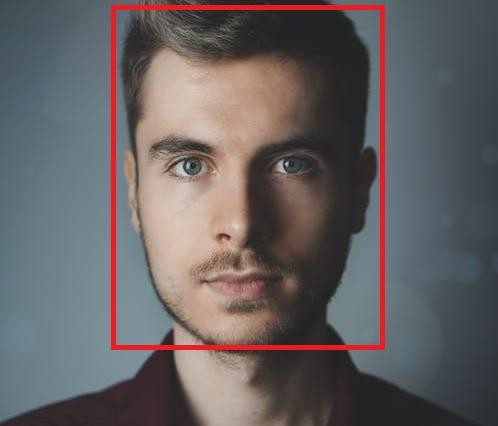
* + - * Eyes
      * Eyebrows
      * Nose
      * Mouth
      * Jawline

To use facial landmarks to build a dataset of faces wearing face masks, we need to first start with an image of a person not wearing a face mask:



**Figure 6.3** Image Without Masks

From there, we apply face detection to compute the bounding box location of the face in the image:



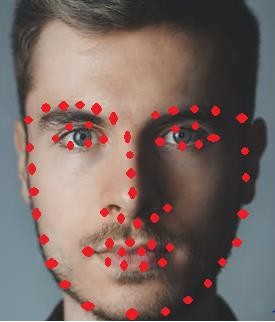
**Figure 6.4** Applying Face Detection

Once we know where in the image the face is, we can extract the face Region of Interest (ROI):



**Figure 6.5** Extract face ROI

And from there, we apply facial landmarks, allowing us to localize the eyes, nose, mouth, etc.:



**Figure 6.6** Detect Facial Landmarks

Next, we apply mask to all the images in the dataset.



**Figure 6.7** With Mask

# Object Detection

## Traditional Object Detection

The problem of detecting multiple masked and unmasked faces in images can be solved by a traditional object detection model. The process of object detection mainly involves localizing the objects in images and classifying them (in case of multiple objects). Traditional algorithms like Haar Cascade (Viola and Jones, 2001) and HOG (Dalal and Triggs, 2005) have proved to be effective for such tasks, but these algorithms are heavily based on Feature Engineering. In the era of Deep learning, it is possible to train Neural Networks that outperform these algorithms, and do not need any extra Feature Engineering.

## Convolutional Neural Networks

Convolutional Neural Networks (CNNs) (LeCun et al.,1998) is a key aspect in modern Computer Vision tasks like pattern object detection, image classification, pattern recognition tasks, etc.

A CNN uses convolution kernels to convolve with the original images or feature maps to extract higher-level features, thus resulting in a very powerful tool for Computer Vision tasks.

## Modern Object Detection Algorithm

CNN based object detection algorithms can be classified into 2 categories:

* + - 1. Multi-Stage Detectors
      2. Single-Stage Detectors.

### Multi-Stage Detectors

In a multi-stage detector, the process of detection is split into multiple steps. A two- stage detector like RCNN (Girshick et al., 2014) first estimates and proposes a set of regions of interest using selective search. The CNN feature vectors are then extracted from each region independently. Multiple algorithms based on Regional Proposal Network like Fast RCNN (Girshick, 2015) and Faster RCNN (Ren et al., 2015) have achieved higher accuracy and better results than most single stage detectors.

### Single-Stage Detectors

A single-stage detector performs detections in one step, directly over a dense sampling of possible locations. These algorithms skip the region proposal stage used in multi-stage detectors and are thus considered to be generally faster, at the cost of some loss of accuracy. One of the most popular single- stage algorithms, You Only Look once (YOLO) (Redmon et al., 2016), was introduced in 2015 and achieved close to real- time performance. Single Shot Detector (SSD) (Liu et al., 2016) is another popular algorithm used for object detection, which gives excellent results. RetinaNet (Lin et al., 2017b), one of the best detectors, is based on Feature Pyramid Networks (Lin et al., 2017a), and uses focal loss.

# Training Mask Detector

Here we load our face mask detection dataset from disk then we train a model (using Keras/Tensorflow) on this dataset and then serializing the face mask detector to disk.

### CODE:

# 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

The set of **tensorflow.keras** import allow for:

* Data augmentation
* Loading the MobileNetV2 classifier (Fine-tune this model with pre-trained ImageNet weights)
* Building a new fully-connected (FC) head
* Pre-processing
* Loading image data

**Sklearn** is used for binarizing class labels, segmenting our dataset, and printing a classification report.

[**imutils**](https://github.com/jrosebr1/imutils/) **paths**implementation will help us to find and list images in our dataset.

**matplotlib** to plot our training curves.

# 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())

**--dataset :** The path to the input dataset of faces and faces with masks

**--plot :** The path to your output training history plot, which will be generated using matplotlib

**--model :** The path to the resulting serialized face mask classification model

# initialize the initial learning rate, number of epochs to train for and batch size INIT\_LR = 1e-4

EPOCHS = 20

BS = 32

# 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)

* Grabbing all of the **imagePaths** in the dataset
* Initializing **data** and **labels** lists
* Looping over the **imagePaths** and loading pre-processing images. Pre- processing steps include resizing to *224×224* pixels, conversion to array format, and scaling the pixel intensities in the input image to the range *[-1, 1]* (via the **preprocess\_input** convenience function)
* Appending the pre-processed **image** and associated **label** to the data and labels lists, respectively
* Ensuring our training data is in **NumPy** array format # perform one-hot encoding on the labels

lb = LabelBinarizer()

labels = lb.fit\_transform(labels) labels = to\_categorical(labels)

# partition the data into training and testing splits using 75% of # the data for training and the remaining 25% for testing (trainX, testX, trainY, testY) = train\_test\_split(data, labels,

test\_size=0.20, stratify=labels, random\_state=42)

# construct the training image generator for data augmentation aug = ImageDataGenerator(

rotation\_range=20, zoom\_range=0.15, width\_shift\_range=0.2, height\_shift\_range=0.2, shear\_range=0.15,

horizontal\_flip=True, fill\_mode="nearest")

# 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

* Load **MobileNetV2** with pre-trained **ImageNet** weights, leaving off head of network
* Construct a new FC head, and append it to the base in place of the old
* Freeze the base layers of the network. The weights of these base layers will not be updated during the process of backpropagation, whereas the head layer weights will be tuned.

# 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"])

Now it’s time to execute the program. Open the terminal and type

### python train\_mask\_detector.py –dataset dataset

[INFO] loading images… [INFO] compiling model... [INFO] training head…

Train for 34 steps, validate on 276 samples Epoch 1/20

34/34 [==============================] - 30s 885ms/step - loss: 0.6431 -

accuracy: 0.6676 - val\_loss: 0.3696 - val\_accuracy: 0.8242 Epoch 2/20

34/34 [==============================] - 29s 853ms/step - loss: 0.3507 -

accuracy: 0.8567 - val\_loss: 0.1964 - val\_accuracy: 0.9375 Epoch 3/20

34/34 [==============================] - 27s 800ms/step - loss: 0.2792 -

accuracy: 0.8820 - val\_loss: 0.1383 - val\_accuracy: 0.9531 Epoch 4/20

34/34 [==============================] - 28s 814ms/step - loss: 0.2196 -

accuracy: 0.9148 - val\_loss: 0.1306 - val\_accuracy: 0.9492 Epoch 5/20

34/34 [==============================] - 27s 792ms/step - loss: 0.2006 -

accuracy: 0.9213 - val\_loss: 0.0863 - val\_accuracy: 0.9688 Epoch 16/20

34/34 [==============================] - 27s 801ms/step - loss: 0.0767 -

accuracy: 0.9766 - val\_loss: 0.0291 - val\_accuracy: 0.9922 Epoch 17/20

34/34 [[==============================] - 27s 795ms/step - loss: 0.1042

- accuracy: 0.9616 - val\_loss: 0.0243 - val\_accuracy: 1.0000] Epoch 18/20

34/34 [==============================] - 27s 796ms/step - loss: 0.0804 -

accuracy: 0.9672 - val\_loss: 0.0244 - val\_accuracy: 0.9961 Epoch 19/20

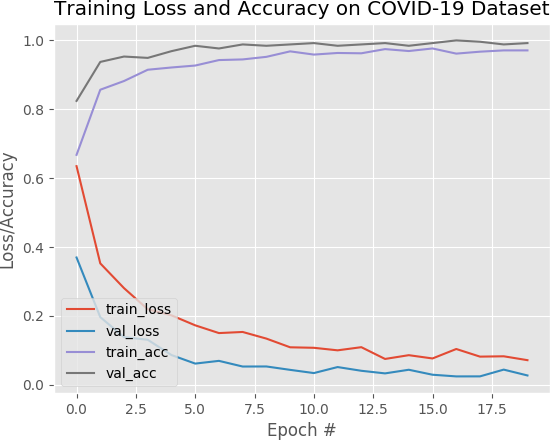
34/34 [==============================] - 27s 793ms/step - loss: 0.0836 -

accuracy: 0.9710 - val\_loss: 0.0440 - val\_accuracy: 0.9883 Epoch 20/20

34/34 [==============================] - 28s 838ms/step - loss: 0.0717 -

accuracy: 0.9710 - val\_loss: 0.0270 - val\_accuracy: 0.9922 [INFO] evaluating network...

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | recall | f1-score | support |
| With\_mask | 0.99 | 1.00 | 0.99 | 138 |
| without\_mask | 1.00 | 0.99 | 0.99 | 138 |
| Accuracy |  |  | 0.99 | 276 |
| macro avg | 0.99 | 0.99 | 0.99 | 276 |
| weighted avg | 0.99 | 0.99 | 0.99 | 276 |



**Figure 6.8** Plot

# Face Mask Detector for Images

Once the face mask detector is trained, we can

* Load an input image from disk
* Detect faces in the image
* Apply our face mask detector to classify the face as either with\_mask or without\_mask

## CODE

# 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

**OpenCV** is required for display and image manipulations.

# 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())

command line arguments include:

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

# load our serialized face detector model from disk print("[INFO] loading face detector model...")

prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"]) weightsPath = os.path.sep.join([args["face"],

"res10\_300x300\_ssd\_iter\_140000.caffemodel"]) net = cv2.dnn.readNet(prototxtPath, weightsPath)

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

model = load\_model(args["model"])

# 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()

* Upon loading our **–image** from disk we make a copy and grab frame dimensions for future scaling and display purposes.
* Pre-processing is handled by [**OpenCV’s blobFromImage function**](https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/). As shown in the parameters, we resize to *300×300* pixels and perform mean subtraction.
* Once we know where each face is predicted to be, we’ll ensure they meet the -

**-confidence** threshold before we extract the faceROIs.

# 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]

In this block, we:

* Extract the face ROI via NumPy slicing
* Pre-process the ROI the same way we did during training
* Perform mask detection to predict with\_mask or without\_mask

# determine the class label and color we'll use to draw # the bounding box and text

label = "Mask" if mask > withoutMask else "No Mask" color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

# include the probability in the label

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

# display the label and bounding box rectangle on the output # frame

cv2.putText(image, label, (startX, startY - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, color, 2)

cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)

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

* + First, we determine the class label based on probabilities returned by the mask detector model and assign an associated color for the annotation. The color will be “green” for with\_mask and red for without\_mask.
  + We then draw the label text (including class and probability), as well as a bounding box rectangle for the face, using OpenCV drawing functions.
  + Once all detections have been processed, we display the output image.

Now open up a terminal and execute the following command:

python detect\_mask\_image.py --image examples/example\_01.png [INFO] loading face detector model...

[INFO] loading face mask detector model... [INFO] computing face detections…



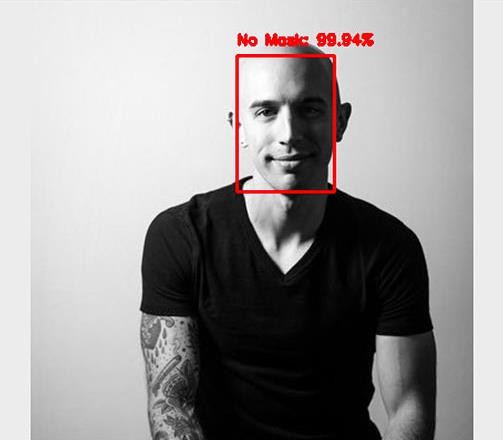
**Figure 6.9** Output-With Mask

Our face mask detector correctly labelled this image as **Mask**.

Let’s try another image, this one of a person not wearing a face mask:

python detect\_mask\_image.py --image examples/example\_02.png [INFO] loading face detector model...

[INFO] loading face mask detector model... [INFO] computing face detections…



**Figure 6.10** Output Without Mask

Our face mask detector has correctly predicted **No Mask.**

Let’s try one final image:

python detect\_mask\_image.py --image examples/example\_03.png

[INFO] loading face detector model... [INFO] loading face mask detector model... [INFO] computing face detections…



**Figure 6.11** Output – With and Without Mask

The reason we cannot detect the face in the foreground is because:

* + 1. It’s too obscured by the mask.
    2. The dataset used to train the *face detector* did not contain example images of people wearing face masks.

Therefore, if a large portion of the face is occluded, our face detector will likely fail to detect the face.

# Face Mask Detection in Webcam Stream

## Two-Fold Process to Identify

The flow to identify the person in the webcam wearing the face mask or not. The process is two-fold.

* + - 1. To identify the faces in the webcam.
      2. Classify the faces based on the mask.

### Identify the face in the Webcam

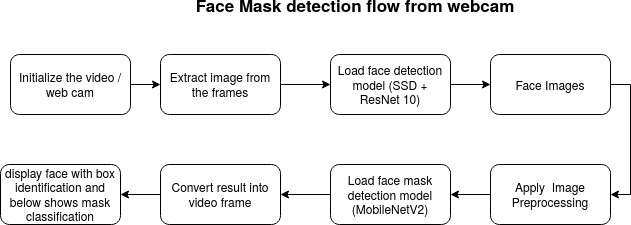
To identify the faces a pre-trained model provided by the OpenCV framework was used. The model was trained using web images. OpenCV provides 2 models for this face detector:

* + - * 1. Floating-point 16 version of the original Caffe implementation.
        2. 8-bit quantized version using Tensorflow.

The Caffe model in this face mask detector. There has been a lot of discussion around deep learning-based approaches for person detection. This encouraged us to come up with our own algorithm to solve this problem. Our work on face mask detection comprises of data collection to tackle the variance in kinds of face masks worn by the workers. The face mask detection model is a combination of face detection model to identify the existing faces from camera feeds and then running those faces through a mask detection model.

### Classify faces based on the mask

Subsequent to face detection, the mask detector model was used to identify the frame face covered by mask or not. The flow of this process described in the image below.



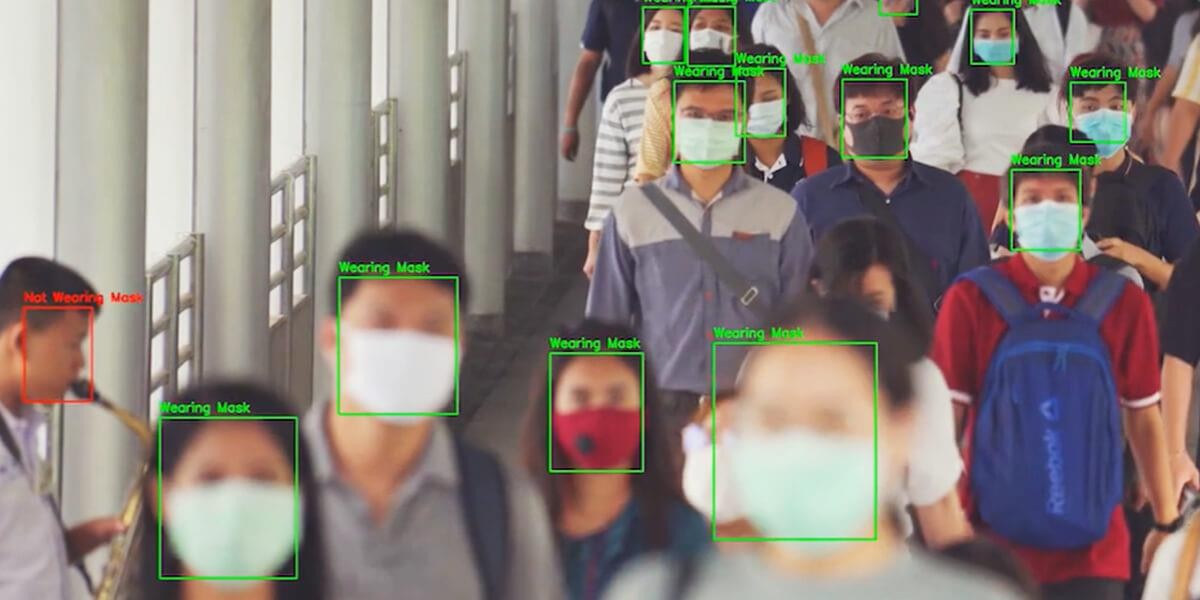
**Figure 6.12** Face mask Detection Flow from Webcam

## Use Cases

Here are a few use cases where this mask detection technology could be leveraged.

### Airports

The Face Mask Detection System could be used at airports to detect travelers without masks. Face data of travelers can be captured in the system at the entrance. If a traveler is found to be without a face mask, their picture is sent to the airport authorities so that they could take quick action.



**Figure 6.13** Use Case in Airports

### Hospitals

Using Face Mask Detector System, Hospitals can monitor if quarantined people required to wear a mask are doing so or not. The same holds good for monitoring staff on duty too.



**Figure 6.14** Use Case in Hospitals

### Offices & Working Spaces

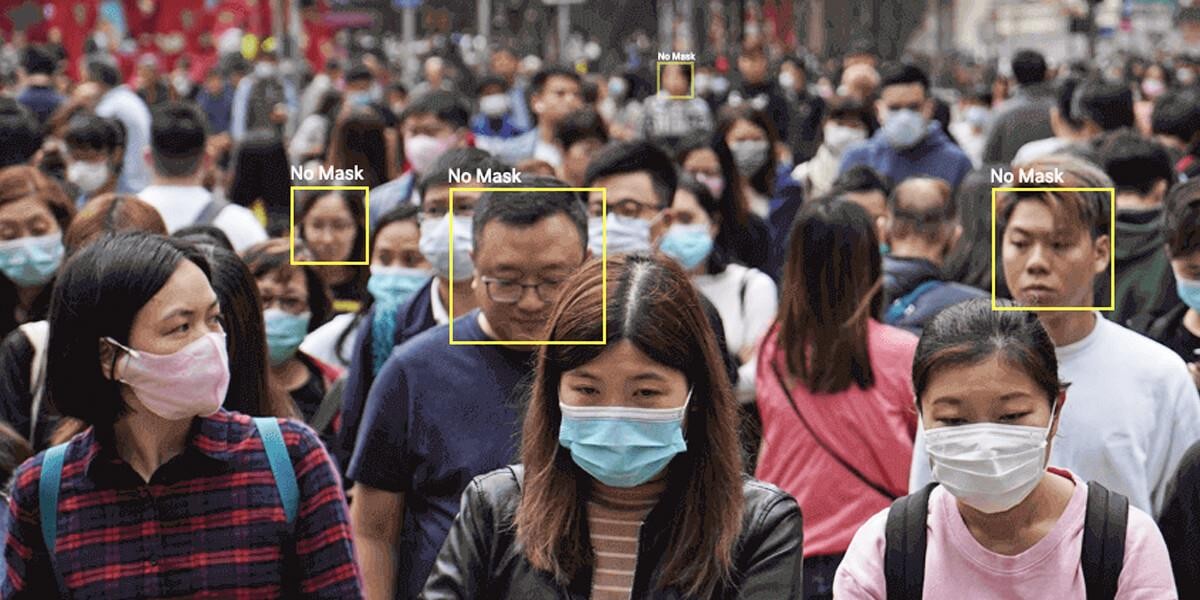
The Face Mask Detection System can be used at office premises to ascertain if employees are maintaining safety standards at work. It monitors employees without masks and sends them a reminder to wear a mask.



**Figure 6.15** Use Case in Offices & Working Spaces

### Government

To limit the spread of coronavirus, the police could deploy the face mask detector on its fleet of surveillance cameras to enforce the compulsory wearing of face masks in public places.



**Figure 6.16** Use Case in Government

# Code

# 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 = []

Our **detect\_and\_predict\_mask** function accepts three parameters:

* **frame :** A frame from our stream
* **faceNet :** The model used to detect where in the image faces are
* **maskNet :** Our COVID-19 face mask classifier model

Inside, we construct a **blob**, detect faces, and initialize lists, two of which the function is set to return. These lists include our **faces** (i.e., ROIs), **locs** (the face locations), and **preds** (the list of mask/no mask predictions).

From here, we’ll loop over the face **detections**:

# 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))

Inside the loop, we filter out weak detections (**Lines 34-38**) and extract bounding boxes while ensuring bounding box coordinates do not fall outside the bounds of the image.

Next, we’ll add face ROIs to two of our corresponding lists:

# 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))

After extracting face ROIs and pre-processing, we append the face ROIs and bounding boxes to their respective lists.

We’re now ready to run our **faces** through our mask predictor:

# 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)

The logic here is built for speed. First we ensure at least one face was detected if not, we’ll **return** empty **preds**. Secondly, we are performing inference on our entire batch of **faces** in the frame so that our pipeline is faster.

# 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()

Our command line arguments include:

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

# load our serialized face detector model from disk print("[INFO] loading face detector model...")

prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"]) weightsPath = os.path.sep.join([args["face"],

"res10\_300x300\_ssd\_iter\_140000.caffemodel"]) 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)

Here we have initialized our:

* Face detector
* COVID-19 face mask detector
* Webcam video stream

Let’s proceed to loop over frames in the stream:

# loop over the frames from the video stream while True:

# grab the frame from the threaded video stream and resize it # to have a maximum width of 400 pixels

frame = vs.read()

frame = imutils.resize(frame, width=400)

# detect faces in the frame and determine if they are wearing a # face mask or not

(locs, preds) = detect\_and\_predict\_mask(frame, faceNet, maskNet)

Inside, we grab a frame from the stream and resize it. From there, we put our convenience utility to use; detects and predicts whether people are wearing their masks or not.

# 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)

Inside our loop over the prediction results we:

* Unpack a face bounding box and mask/not mask prediction
* Determine the label and color
* Annotate the label and face bounding box Finally, we display the results and perform cleanup:

# 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()

After the **frame** is displayed, we capture **key** presses. If the user presses **q** (quit), we **break** out of the loop and perform housekeeping.

# Web Application of Mask Detector

## Python Code

from flask import Flask, request, flash

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 import base64

from PIL import Image import io

from flask\_cors import CORS

from werkzeug.utils import secure\_filename ALLOWED\_EXTENSIONS = ['.png', '.jpeg']

def allowed\_file(filename): return '.' in filename and \

filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

app = Flask( name ) CORS(app)

print("[INFO] loading face mask detector model...") model = load\_model('mask\_detector.model')

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)

@app.route('/upload', methods=['GET', 'POST']) def upload\_file():

if request.method == 'POST':

# check if the post request has the file part print(request.files)

file = request.files['file'] img = file.stream.read()

npimg = np.fromstring(img, dtype=np.uint8) image = cv2.imdecode(npimg, 1)

# image = cv2.imread('./examples/example\_02.png') # print(type(image))

orig = image.copy()

(h, w) = image.shape[:2]

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]

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]

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) print(type(image))

im = Image.fromarray(image.astype('uint8'))

rawBytes = io.BytesIO()

im.save(rawBytes, "PNG")

rawBytes.seek(0) # return to the start of the file

# return 'data:image/png;base64,' + str(base64.b64encode(rawBytes.read(

))).split("'")[1]

return ' ' '

<!doctype html>

<title> Result </title>

<h3> Result <h3>

<img src="'''+"data:image/png;base64," + str(base64.b64encode(rawByte s.read())).split("'")[1]+'''">

' ' '

## HTML Code

<!DOCTYPE html>

<html>

<body>

<h1>Show File-select Fields</h1>

<h3>Show a file-select field which allows only one file to be chosen:</h3>

<form action="http://127.0.0.1:5000/upload" method="POST" enctype="multipart/form-data">

<label for="file">Select a file:</label>

<input type="file" id="file" key="file" name="file"><br><br>

<input type="submit">

</form>

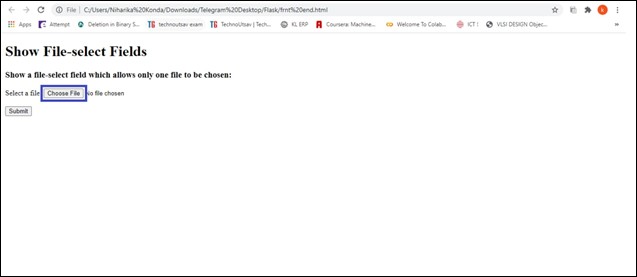
</body>

</html>

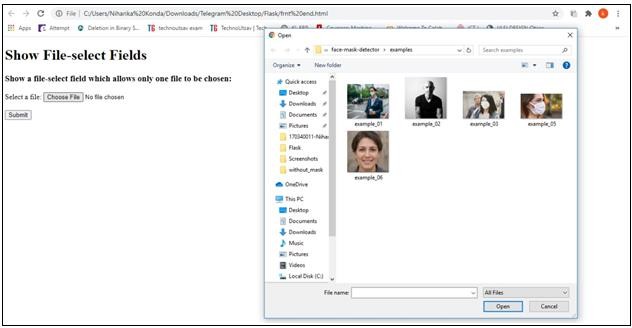
Before running the code, open command prompt and go to the folder where codes where saved and type:

### set FLASK\_APP=mask\_image\_detection\_new.py flask run

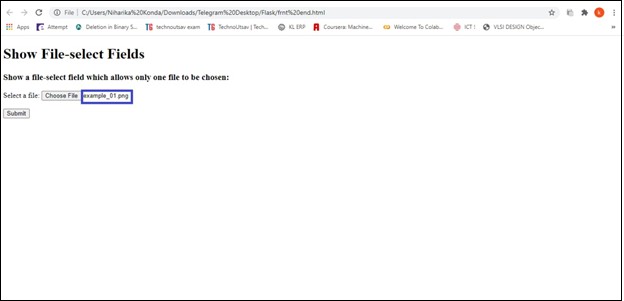
Home page



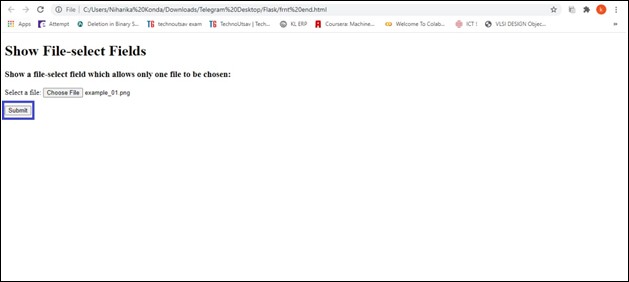
Click on choose file to select an image



choose an image from pc



The image name will be shown on the home page



Click on submit

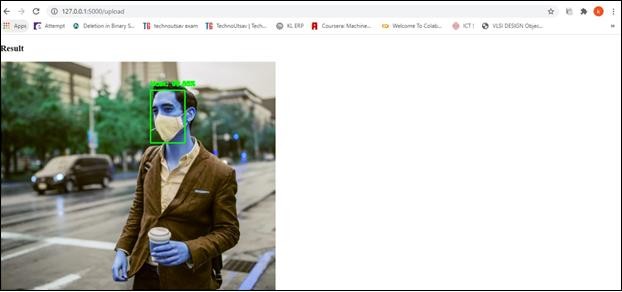
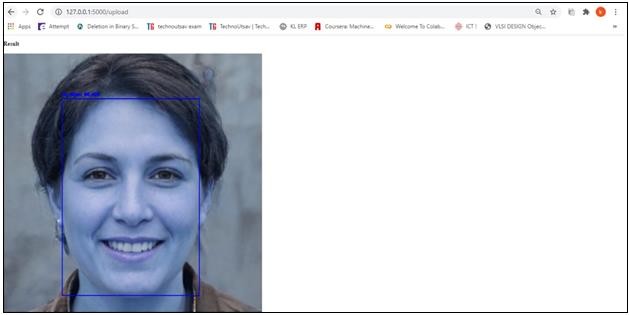


Image will be displayed



Result

# CHAPTER 7

# CONCLUSIONS

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

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

# 7.1 Future Scope

The current system is evaluated with different classifiers. The best system may be implemented along with interfacing with alarm and alerting systems in the near future. This system may be integrated with a system which can integrate with a system implementing social distancing which can make it a wholesome system which can bring dramatic impact on the spread. We're headed for a faceless future as masks become the new normal. That can be a big security concern. Though using

face masks are proved to be the best solution to mitigate the spread of air borne viruses like Corona; it poses a big security challenge to the nation ahead as it could create opportunities for people who cover their faces for nefarious reasons. Experts say mass mask-wearing could complicate crime investigations in the coming days, as facial recognition becomes an important part in tracking criminals. Human beings have proved out to be very good at recognizing familiar faces and facial recognition algorithms are getting better in identifying patterns. These masks throw new challenges into the mix. This challenge may create a scope to new face detection algorithms which can identify faces which are covered with greater accuracies.

# REFERENCES

1. W. H. Organization et al., “Coronavirus disease 2019 (covid-19): situation report, 96,” 2020.
2. P. A. Rota, M. S. Oberste, S. S. Monroe, W. A. Nix, R. Campagnoli, J. P. Icenogle, S. Penaranda, B. Bankamp, K. Maher, M.-h. Chen et al., “Characterization of a novel coronavirus associated with severe acute respiratory syndrome,” science, vol. 300, no. 5624, pp. 1394–1399, 2003.
3. Z. A. Memish, A. I. Zumla, R. F. Al-Hakeem, A. A. Al-Rabeeah, and G. M. Stephens, “Family cluster of middle east respiratory syndrome.
4. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018, pp. 4510- 4520, doi: 10.1109/CVPR.2018.00474.
5. Xin, M., Wang, Y. Research on image classification model based on deep convolution neural network. J Image Video Proc. 2019, 40 (2019).
6. Sultana, F., A. Sufian, and P. Dutta. "A review of object detection models based on convolutional neural network." arXiv preprint arXiv:1905.01614 (2019).
7. Image Classification Using Convolutional Neural Networks Deepika Jaswal,

Sowmya.V, K.P.Soman

1. [https://edition.cnn.com/2020/05/10/world/face-masks-security-intl-](https://edition.cnn.com/2020/05/10/world/face-masks-security-intl-gbr/index.html) [gbr/index.html](https://edition.cnn.com/2020/05/10/world/face-masks-security-intl-gbr/index.html)
2. [https://www.theverge.com/2020/12/1/21755021/nist-facial-recognition-mask-](https://www.theverge.com/2020/12/1/21755021/nist-facial-recognition-mask-error-rate-data-frvt) [error-rate-data-frvt](https://www.theverge.com/2020/12/1/21755021/nist-facial-recognition-mask-error-rate-data-frvt)
3. [https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-](https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/) [with-opencv-keras-tensorflow-and-deep-learning/](https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/)
4. [https://www.sfcdcp.org/communicable-disease/healthy-habits/how-to-put-on-](https://www.sfcdcp.org/communicable-disease/healthy-habits/how-to-put-on-and-remove-a-face-mask/) [and-remove-a-face-mask/](https://www.sfcdcp.org/communicable-disease/healthy-habits/how-to-put-on-and-remove-a-face-mask/)
5. <https://www.leewayhertz.com/face-mask-detection-system/>
6. <https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html>
7. [https://theconversation.com/facial-recognition-ten-reasons-you-should-be-](https://theconversation.com/facial-recognition-ten-reasons-you-should-be-worried-about-the-technology-122137) [worried-about-the-technology-122137](https://theconversation.com/facial-recognition-ten-reasons-you-should-be-worried-about-the-technology-122137)
8. [https://towardsdatascience.com/covid-19-face-mask-detection-using-](https://towardsdatascience.com/covid-19-face-mask-detection-using-tensorflow-and-opencv-702dd833515b) [tensorflow-and-opencv-702dd833515b](https://towardsdatascience.com/covid-19-face-mask-detection-using-tensorflow-and-opencv-702dd833515b)
9. <https://www.wired.com/story/algorithms-recognize-masked-face/>