

# **FACE MASK DETECTION BASED ON DEEP LEARNING MODEL FOR COVID – 19 PANDEMIC**

*Submitted in partial fulfillment for the award of the degree of*

## **Bachelor of Computer Application**

*by*

**O. SHYAMKUMAR (18BCA0110)**

**K.R. KISHORKUMAR (18BCA0104)**

**R. ANISHKUMAR (18BCA0118)**

**Under the guidance of**

**Prof. Jabanjalin Hilda**

**School of Information Technology and Engineering**

**VIT, Vellore**



**VIT<sup>®</sup>**

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

**June, 2021**

## **DECLARATION**

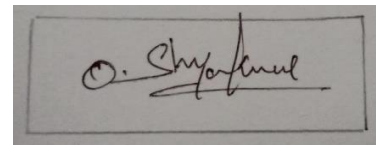
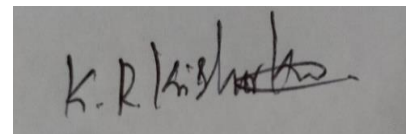
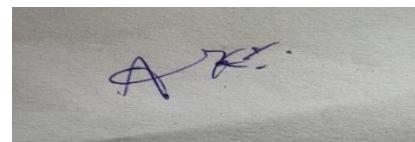
We hereby declare that the project entitled "Face Mask Detection Based on Deep Learning Model for COVID-19 Pandemic" submitted by us for the award of the degree of *Bachelor of Computer Application* to VIT is a record of bonafide work carried out by us under the supervision of Prof. Jabanjalin Hilda.

We further declare that the work reported has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 02-06-2021

### **Signature of the Candidates**

A rectangular box containing a handwritten signature in black ink that reads "O. Srinivas".A rectangular box containing a handwritten signature in black ink that reads "K. R. Kishore".A rectangular box containing a handwritten signature in blue ink that reads "A. R.". The signature is stylized and appears to be the initials of the candidate.

## **CERTIFICATE**

This is to certify that the project entitled “Face Mask Detection Based on Deep Learning Model for COVID-19 Pandemic” submitted by us, School of Information Technology and Engineering, VIT Vellore, for the award of the degree of *Bachelor of Computer Application*, is a record of bonafide work carried out by us, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore

Date: 02-06-2021



**Signature of the Guide**

**Internal Examiner**

**External Examiner**

**Approved by**

-----  
**Head of the Department**  
**Bachelor of Computer Application**

## **ABSTRACT**

Corona virus disease has been rapidly increasing amid time in various countries. The wide spread can be controlled only by wearing mask and this is the major protection to safe guard ourselves. However, there are only a few research studies about face mask detection based on image analysis. In this project, we propose an approach for automatic detection of face mask in real-time using trained models and datasets that would be useful for the public health care system. The MobileNetV2, which attain good-accuracy and efficient for face mask detector. The proposed MobileNetV2 model is a light-weighted neural network which can implemented to embedded or mobile devices. The proposed approach first detects the face from camera using face detector caffe model. Then it determines whether there wearing face mask or not using visual features and binary classifier. The experimental results show detection accuracy of 98.02 % in real-time.

## **ACKNOWLEDGEMENTS**

We would like to express our gratitude to our Chancellor Dr. G. Viswanathan for giving us a chance to pursue our Higher education without any fees. We express our sincere thanks to Vice Chancellor Dr. Rambabu Kodali, Pro Vice Chancellor Dr. S. Narayanan, and Dr. Balakrushna Tripathy, Dean, School of Information Technology and Engineering, for their inspiration and guidance during the tenure of the course. With pleasure, we express our special thanks to Dr. Malaserene I, Head of Department, all teaching staff and members working as limbs of our university for their selfless enthusiasm coupled with timely encouragements showered on us, which prompted the acquirement of the requisite knowledge to finalize our course of study successfully.

It is our pleasure to express with a deep sense of gratitude to Prof. Jabanjalin Hilda J, Assistant Professor (Senior), School of Information Technology and Engineering, Vellore Institute of Technology, for her constant guidance, continuous encouragement throughout our academic career and project work in VIT. It is a great opportunity on our part to work with an intellectual and expert in the field of technology.

### **Student name**

O. Shyamkumar

K.R. Kishorkumar

R. Anishkumar

## **Executive Summary**

The main aim of our project is to apply the concepts of deep learning algorithms for creating an automated system for Face mask detection. Deep Learning is the subset of Artificial Intelligence that deals with the extraction of patterns from datasets. Deep learning is the act of utilizing algorithms to parse information, gain from it, and afterward make a prediction or expectation about something in the world. In our new system, we apply the techniques of deep learning which is a subset of machine learning algorithm that utilizes complex MobileNet. One of the main areas of research for our proposed innovation includes the usage of pre-trained real time object detection models and image classification models using convolutional neural networks (CNNs). Image classification using CNNs aims to classify an input image based on visual content. Many models such as Caffe model (Face detection), MobileNetV2 etc. have been developed over the years and some of them are quite popular due to its high performance. MobileNetV2 is a very popular model which is known to show almost 90% accuracy on the ImageNet dataset. The model is a combination of ideas developed by many researchers. In our system, we use the MobileNetV2 model for transfer learning. By using such Deep learning models, we can automate the process of face mask detection with the use of a real time surveillance system.

<b>CONTENTS</b>	<b>Page no.</b>
<b>Abstract .....</b>	<b>4</b>
<b>Acknowledgement .....</b>	<b>5</b>
<b>Executive Summary.....</b>	<b>6</b>
<b>Table of Content.....</b>	<b>7</b>
<b>List of Figures.....</b>	<b>9</b>
<b>List of Tables.....</b>	<b>9</b>
<b>Abbreviations .....</b>	<b>9</b>
<b>1. CHAPTER – 1</b>	
<b>INTRODUCTION</b>	
1.1 PROJECT STATEMENT .....	10
1.2 LITERATURE SURVEY .....	13
1.3 OBJECTIVES .....	15
1.4 SCOPE OF THE PROJECT .....	15
1.5 ARCHITECUTRE DIAGRAM .....	19
<b>2. CHAPTER – 2</b>	
<b>LIBRARIES</b>	
2.1 KERAS .....	19
2.2 TENSORFLOW .....	22
2.3 MOBILENET .....	25
2.4 OPENCV .....	27
2.5 MATPLOTLIB .....	29
2.6 NUMPY .....	32

2.7 IMUTILS .....	33
<b>3. CHAPTER – 3</b>	
3.1 DATA COLLECTION .....	36
3.2 DATA VISUALIZATION .....	37
3.2 DATA AUGMENTATION .....	37
3.4 SPLITTING THE DATA .....	38
3.5 BUILDING THE MODEL .....	38
3.6 TRAINING THE MOBILENET .....	39
3.7 RESULT .....	39
<b>4. CHAPTER – 4</b>	
4.1 FACE DETECTOR.....	40
4.2 DETECTING FACES WITH OR WITHOUT MASK.....	42
<b>5. CONCLUSION AND FUTURE WORK .....</b>	<b>43</b>
<b>6. SUMMARY .....</b>	<b>45</b>
<b>7. APPENDICES.....</b>	<b>46</b>
<b>8. REFERENCES .....</b>	<b>57</b>



### LIST OF FIGURES AND TABLES:

Figure No.	Title	Page No.
1.1	Architecture Diagram	19
1.2	Numpy Process	32
1.3	Object Detection on FPS	34
1.4	Convolutional Architecture Diagram	38
1.5	Loss and Accuracy Graph	38
1.6	Experimental Output	43
	<b>TABLES</b>	
2.1	Splitting the Dataset	38
2.2	Validation	40

### LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
DNN	Deep Neural Network
Open-CV	Open-Source Computer Vision Library
DL	Deep Learning
WHO	World Health Organization
MERS - CoV	Middel East Respiratory Syndrome Coronavirus

## Chapter 1

### Introduction

#### 1.1 FACE MASK DETECTION

According to the World Health Organization (WHO)'s coronavirus disease 2019 (COVID-19) has globally infected over 20 million people causing over 0.7million deaths. Respiratory problems like shortness of breath or difficulty in breathing is one of them. Some common human coronaviruses that infect public around the world. People having respiratory problems can expose anyone (who is in close contact with them) to infective beads. COVID-19, wearing a clinical mask is very necessary. The public should be aware of whether to put on the mask for source control or aversion of COVID-19. Therefore, face mask detection has become a crucial task in present global society. Face mask detection involves in detecting the location of the face and then determining whether it has a mask on it or not. It has numerous applications, such as autonomous driving, education, surveillance, and so on. The coronavirus pandemic is responsible for producing an atmosphere of terror as this disease can transmit through the respiratory system. This virus has killed more than a million people around the globe, and it is expected to kill close to 400,000 people by February 1st, 2021 in US. Currently, there is not any specific single medicine or vaccine in hand to fight against this virus. Therefore, the only option left is to take the utmost care from our side to stay away from the disease. For example, you can maintain the social distancing, wash your hand regularly, and wear a mask. To take part in the protection against the pandemic, my aim is to design a Face Mask Detection program using the famous Deep Learning technique. This technique is useful to find out who is not wearing the facial mask and not deploying the trained model. The WHO report points out that there are two ways of coronavirus spread i.e. the respiratory droplets and any type of physical contact. The droplets are produced through the respiratory system in case an infected person is coughing or sneezing. If a human is present closer than 4 feet, there are high chances he can inhale these infection-causing droplets. These droplets can stick on those surfaces where the virus can live for days. This way the infected person's surroundings can become a big reason for virus spread. To

prevent the virus from the spread, medical masks are the best bet. In the research, medical masks mean surgical as well as the procedure masks and maybe look like a cup shape or folded. These masks can be attached to the head with cords. They are examined well to control the filtration, 6 easy breathing, and some time for water resistivity. The research examines the collection of video as well as the images to find out those persons who are wearing those medical masks that are according to the govt. guidelines. This way, it can greatly assist the govt. to do action against those people who are not wearing the right type of masks. Using the mask in public has been a normal thing in countries like China and other Asian countries with the very start of the pandemic. Currently, the USA is under the grip and severe pandemic outbreak and cases are increasing day by day along with the confirmed deaths. The CDC (Centers for Disease Control and Prevention) has cautioned the people too must wear protective equipment like the masks. The studies have revealed that many people, particularly the young ones who have the virus but without any symptoms and can spread the virus to many other persons unknowingly, the same is true for those people who eventually develop the symptoms, but spread the disease before being tested positive. Seeing this, CDC has issue advisory to wear masks in a public gathering where the social distancing is impossible to spread the community-based virus spread. This advisory of the CDC is backed with various studies including the one published in the New England Journal of Medicine which show. Wearing a mask while going outdoors during pandemic has been a great helping hand in controlling the spread of coronavirus. It is a symbol of being a sensible citizen of a country. Several countries like China and Korea successfully controlled the Covid-19 in a short time due to the habit of using masks regularly. It was recommended to use masks, no matter what type of mask is available to you, just use it and become safe. The mask acts as a physical barrier to prevent the entry of the virus. Many individuals unknowingly infected many other people with the Covid-19. Mask is necessary because of two benefits. It does not let the virus enter your mouth or nose directly from the infected person's sneeze or cough. The irresponsibility of a few people has to lead to the death of many others due to the spread of the virus. Secondly, if you touch a virus-contaminated surface and then your mouth or nose, the mask will stop the transmission of the virus. Many governments made it compulsory to use masks and people were compelled and monitor to act upon the order.

The paper understudy will make deep learning about using masks to prevent the spread of deadly coronavirus. It will also learn the facemask detection by deep learning strategy. This is a useful technique of learning being used these days. Deep learning is a subfield of machine learning. We study a hierarchy of features and functions of the subject under study with the help of input data. The researchers are found to use this technique intensively for their research work. They used it in their research related to image classification, speech recognition, signal processing, and natural language processing. Deep learning is similar to machine learning. It builds a hierarchy of features from top to bottom. The systematic and arranged features are easy to understand and explain to others as well. This method of learning can learn the features at any level stigmatically. The help of human-made techniques is not required. The best thing about deep learning is that all the models have deep architectures. Being the best opposite of shallow architecture which has few hidden layers, deep architecture has several layers. In this project, we'll discuss our two-phase COVID-19 face mask detector, detailing how our computer vision/deep learning pipeline will be implemented. From there, we'll review the dataset we'll be using to train our custom face mask detector. I'll then show you how to implement a Python script to train a face mask detector on our dataset using Keras and TensorFlow. We'll use this Python script to train a face mask detector and review the results.

## 1.2 PROBLEM STATEMENT

To build a system that can detect faces in real-world images and identify if the detected faces are wearing masks or not. If you look at the people in videos captured by CCTV cameras, you can see that the faces are small, blurry, and low resolution. People are not looking straight to the camera, and the face angles vary from time to time. These real-world videos are entirely different from the videos captured by webcams or selfie cameras, making the face mask detection problem much more difficult in practice.

### 1.3 LITERATURE SURVEY

No	Title and Year	Algorithm	Advantages	Disadvantages
1.	A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the COVID-19 pandemic.[1] (2021)	<ul style="list-style-type: none"> <li><b>Classification process</b> -decision trees, Support Vector Machine (SVM), and ensemble algorithm.</li> </ul>	<ul style="list-style-type: none"> <li>The SVM classifier achieved 99.64% testing accuracy in RMFD. In SMFD, it achieved 99.49%,</li> </ul>	<ul style="list-style-type: none"> <li>The major drawback is not tray most of classical machine learning methods to get lowest consume time and highest accuracy.</li> </ul>
2.	Real Time Face Detection and Tracking using OpenCV [2] (2014)	<ul style="list-style-type: none"> <li>Haar cascade, adaboost, template matching</li> </ul>	<ul style="list-style-type: none"> <li>It detects facial features and ignores anything else, such as buildings, trees and bodies.</li> </ul>	<ul style="list-style-type: none"> <li>Ada boost algorithm is main disadvantage, take training data and define weak classifier function for each sample of training data .</li> </ul>
3.	Real-Time Facemask Recognition with Alarm System using Deep Learning [3] (2020)	<ul style="list-style-type: none"> <li>Convolutional Neural Network (CNN) Deep neural networks(DNN)</li> </ul>	<ul style="list-style-type: none"> <li>The dataset collected contains 25,000 images using 224x224 pixel resolution and achieved an accuracy rate of 96% as to the performance of the trained model.</li> </ul>	<ul style="list-style-type: none"> <li>measures the distance between each person and creates an alarm if the physical distancing does not observe properly.</li> </ul>
4.	Face mask detection using MobileNet and Global Pooling Block [4] (2020)	<ul style="list-style-type: none"> <li>Deep convolution neural networks (CNN)</li> </ul>	<ul style="list-style-type: none"> <li>The pre-trained MobileNet takes a color image and generates a multi-dimensional feature map. Our proposed model has achieved 99% and 100% accuracy on DS1 and DS2 respectively.</li> </ul>	<ul style="list-style-type: none"> <li>Not present work focuses on face mask detection over multi-face images.</li> </ul>

5.	Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV [5] (2020)	<ul style="list-style-type: none"> <li>Sequential Convolutional Neural Network mode (CNN),locally linear embedding (LLE) algorithm</li> </ul>	<ul style="list-style-type: none"> <li>The method attains accuracy up to 95.77% and 94.58%respectivel y on two different datasets.</li> </ul>	<ul style="list-style-type: none"> <li>It can be extended to detect if a person is wearing the mask properly or not. The model can be further improved to detect if the mask is virus prone or not i.e. the type of the mask is surgical, N95 or not.</li> </ul>
6	Performance Evaluation and Comparison of Software for Face Recognition, based on Dlib and Opencv Library. [6] (2018)	<ul style="list-style-type: none"> <li>HOG (Histogram of oriented gradients)</li> <li>DCNN (Convolutional neutral network)</li> <li>SVM (Support vector machine).</li> </ul>	<ul style="list-style-type: none"> <li>Has better performance for face detection and detection.</li> <li>It automatically detects the important features without any human supervision.</li> <li>Works well with even unstructured and semi structured data like text,images and trees.</li> </ul>	<ul style="list-style-type: none"> <li>HOG is very sensitive to image rotation.</li> <li>High computational cost and need a lot of training data.</li> <li>Long training time for large datasets.</li> </ul>
7	Study of the Performance of Machine Learning Algorithms for Face Mask Detection [7] (2020)  2020	<ul style="list-style-type: none"> <li>KNN.</li> <li>SVM.</li> </ul>	<ul style="list-style-type: none"> <li>Very simple implementation.</li> <li>Works well with even unstructured and semi structured data like text,images and trees.</li> <li>MobileNet is a</li> </ul>	<ul style="list-style-type: none"> <li>Require high memory-need to store all of the training data.</li> <li>Long training time for large datasets.</li> <li>Gradually drops</li> </ul>

		<ul style="list-style-type: none"> <li>Mobile Net.</li> </ul>	<p>lightweight model which rapidly works, low latency time and consumption for a few resources, so it particularly useful for mobile and embedded vision application</p>	<p>when taking real-time due to factors such as camera quality, illumination.</p>
8	RETINA FACE MASK: A FACE MASK DETECTOR [8] (2020)	<ul style="list-style-type: none"> <li>object removal cross-class algorithm.(ORCC)</li> <li>Mobile Net.</li> <li>ResNet.</li> </ul>	<ul style="list-style-type: none"> <li>Reject predictions with low confidences and the high intersection of union.</li> <li>Light-weighted neural network MobileNet for embedded or mobile devices.</li> <li>While increasing network depth, it avoids negative outcomes. So we can increase the depth but we have fast training and higher accuracy.</li> </ul>	<ul style="list-style-type: none"> <li>Doesn't helps to remove objects with higher confidences and low IoU.</li> <li>It is less accurate than other state-of-the-art networks.</li> <li>Deeper network usually requires weeks for training, making it practically infeasible in real-world applications.</li> </ul>
9	Object Detection Using Image Processing [9] (2016)	<ul style="list-style-type: none"> <li>Haar Cascade algorithm.</li> <li>Voila-jones algorithm.</li> </ul>	<ul style="list-style-type: none"> <li>The key <b>advantage</b> of a <b>Haar</b>-like feature over most other features is its calculation speed.</li> <li>Applicable in real-time and Low false <b>positives</b>, higher true <b>positives</b>.</li> </ul>	<ul style="list-style-type: none"> <li>Low percentage of classification and high mean square error.</li> <li>It is sensitive to lighting conditions and there could possibly be different detections of the exact face.</li> </ul>

10	Automatic Detection of Bike-riders without Helmet using Surveillance Videos in Real-time [10] (2016)	<ul style="list-style-type: none"> <li>• Histogram of oriented gradients (HOG).</li> <li>• Scale-invariant feature transform (SIFT).</li> <li>• Local binary patterns (LBP).</li> </ul>	<ul style="list-style-type: none"> <li>• Shows invariance to geometric and photometric changes.</li> <li>• Many <b>features</b> can be generated for even small objects.</li> <li>• High discriminative power and Computational simplicity.</li> </ul>	<ul style="list-style-type: none"> <li>• It is very sensitive to image rotation.</li> <li>• It is still quite slow, costs long time, and is not effective for low powered devices.</li> </ul> <p>They produce rather long histograms, which slow down the recognition speed especially on large-scale face database.</p>
11	Face Mask Detection using YOLOv5 for COVID-19 [11] (2020)	<ul style="list-style-type: none"> <li>• YOLOv5</li> </ul>	<ul style="list-style-type: none"> <li>• <b>YOLOv5</b> is also much faster than all the previous versions of YOLO.</li> </ul>	<ul style="list-style-type: none"> <li>• It can't detect the objects properly when the objects are small and it also can't generalize the objects if the image is of different dimensions.</li> </ul>
12	Fighting against COVID-19: A novel deep learning model based on	Region convolutional neural network(RCNN) You Only Look	The achieved results concluded that the adam optimizer achieved the highest average precision <ul style="list-style-type: none"> <li>• percentage of 81%</li> </ul>	To detect a kind of masked face in image and video-based on deep learning models.



	YOLOv2 with ResNet-50 for medical face mask detection. Year :6-Jun-2020	Once (YOLO v1) Single Shot Multi-Box Detector (SSD)	as a detector. The proposed detector achieved higher accuracy and precision than the related work.	
13	A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2	<b>YOLOv3 algorithm</b>	An accuracy of over 90 % was obtained from Minivision Technology, This method achieved 93.9 % accuracy.	Real-world applications are a much more challenging issue for the upcoming future. 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.
14	Deep Learning based Safe Social Distancing and Face Mask Detection in Public Areas for COVID-19 Safety	Convolutional Neural Networks (CNNs), Single Shot Detector.	It allows automating the solution and enforces the wearing of the mask and follows the guidelines of social distancing. This model was created to run on raspberry pi4 and the	The above mentioned use cases are only some of the many features that were incorporated as part of this solution. We assume there are several other cases of

	Guidelines Adherence		accuracy obtained was between 85% and 95%.	usage that can be included in this solution to offer a more detailed sense of safety.
15	Facial Mask Detection using Semantic Segmentation	Fully Convolutional Neural Network (CNN)	Experiments were performed on Multi Parsing Human Dataset obtaining mean pixel level accuracy of 93.884 % for the segmented face masks.	In future it can be extended to detect if a person is wearing the mask properly or not. The model can be further improved to detect if the mask is virus prone or not i.e. the type of the mask is surgical, N95 or not.

## 1.6 ARCHITECTURE DIAGRAM

In order to use facial marks to construct a data set of facial masks, we need to begin with an image of a person who does not wear a facial mask. We apply face detection from there to calculate the location of the bounding box in the image. Once we know where in the image the face is, we can extract the face Region of Interest (ROI), and from there, we apply facial landmarks, allowing us to localize mouth, face, and eyes. In order to apply masks, we need an image of a mask (with a transparent and high definition image). Add the mask to the detected face and then resize and rotate, placing it on the face. Repeat this process for all input images. Train the mask and without mask images with an appropriate algorithm. Deployment: Once the models are trained, then move on to the loading mask detector, perform face detection, then classify each face. Once an image has been uploaded, the classification happens automatically.

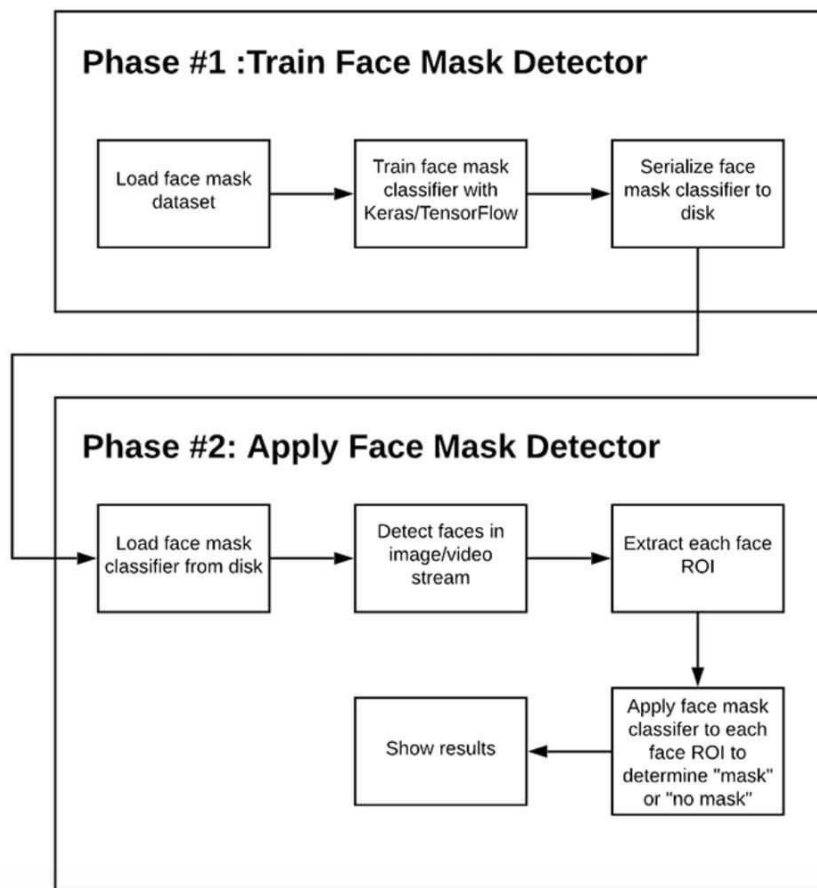


Fig 1.1 Architecture Diagram

## Chapter 2

### 2.1 KERAS

While deep neural networks are all the rage, the complexity of the major frameworks has been a barrier to their use for developers new to machine learning. There have been several proposals for improved and simplified high-level APIs for building neural network models, all of which tend to look similar from a distance but show differences on closer examination.

Keras is one of the leading high-level neural networks APIs. It is written in Python and supports multiple back-end neural network computation engines.

Keras was created to be user friendly, modular, easy to extend, and to work with Python. The API was “designed for human beings, not machines,” and “follows best practices for reducing cognitive load.”

Neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that you can combine to create new models. New modules are simple to add, as new classes and functions. Models are defined in Python code, not separate model configuration files.

The biggest reasons to use Keras stem from its guiding principles, primarily the one about being user friendly. Beyond ease of learning and ease of model building, Keras offers the advantages of broad adoption, support for a wide range of production deployment options, integration with at least five back-end engines (TensorFlow, CNTK, Theano, MXNet, and PlaidML), and strong support for multiple GPUs and distributed training. Plus, Keras is backed by Google, Microsoft, Amazon, Apple, Nvidia, Uber, and others.

Keras proper does not do its own low-level operations, such as tensor products and convolutions; it relies on a back-end engine for that. Even though Keras supports multiple back-end engines, its primary (and default) back end is TensorFlow, and its primary supporter is Google. The Keras API comes packaged in TensorFlow as `tf.keras`, which as mentioned earlier will become the primary TensorFlow API as of TensorFlow 2.0.

To change back ends, simply edit your `$HOME/.keras/keras.json` file and specify a different back-end name, such as `theano` or `CNTK`. Alternatively, you can override the configured back end by defining the environment

variable `KERAS_BACKEND`, either in your shell or in your Python code using the `os.environ["KERAS_BACKEND"]` property.

The **Model** is the core Keras data structure. There are two *main* types of models available in Keras: the Sequential model, and the Model class used with the functional API.

Keras also supplies ten well-known models, called Keras Applications, pretrained against ImageNet: Xception, VGG16, VGG19, ResNet50, InceptionV3, InceptionResNetV2, MobileNet, DenseNet, NASNet, MobileNetV2TK. You can use these to predict the classification of images, extract features from them, and fine-tune the models on a different set of classes.

By the way, fine-tuning existing models is a good way to speed up training. For example, you can add layers as you wish, freeze the base layers to train the new layers, then unfreeze some of the base layers to fine-tune the training. You can freeze a layer with by setting `layer.trainable = False`.

The Keras examples repository contains more than 40 sample models. They cover vision models, text and sequences, and generative models

Keras models can be deployed across a vast range of platforms, perhaps more than any other deep learning framework. That includes iOS, via CoreML (supported by Apple); Android, via the TensorFlow Android runtime; in a browser, via Keras.js and WebDNN; on Google Cloud, via TensorFlow-Serving; in a Python webapp back end; on the JVM, via DL4J model import; and on Raspberry Pi.

## 2.2 TENSORFLOW

Machine learning is a complex discipline. But implementing machine learning models is far less daunting and difficult than it used to be, thanks to machine learning frameworks—such as Google’s TensorFlow—that ease the process of acquiring data, training models, serving predictions, and refining future results.

Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.

TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. Best of all, TensorFlow supports production prediction at scale, with the same models used for training.

TensorFlow allows developers to create *dataflow graphs*—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or *tensor*.

TensorFlow provides all of this for the programmer by way of the Python language. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions can be coupled together. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together. TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

TensorFlow 2.0, released in October 2019, revamped the framework in many ways based on user feedback, to make it easier to work with (e.g., by using the relatively simple Keras API for model training) and more performant. Distributed training is easier to run thanks to a new API, and support for TensorFlow Lite makes it possible to deploy models on a greater variety of platforms. However, code written for earlier versions of TensorFlow must be rewritten—sometimes only slightly, sometimes significantly—to take maximum advantage of new TensorFlow 2.0 features.

The single biggest benefit TensorFlow provides for machine learning development is *abstraction*. Instead of dealing with the nitty-gritty details of implementing algorithms, or figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application. TensorFlow takes care of the details behind the scenes.

TensorFlow offers additional conveniences for developers who need to debug and gain introspection into TensorFlow apps. The eager execution mode lets you evaluate and modify each graph operation separately and transparently, instead of constructing the entire graph as a single opaque object and evaluating it all at

once. The TensorBoard visualization suite lets you inspect and profile the way graphs run by way of an interactive, web-based dashboard.

TensorFlow also gains many advantages from the backing of an A-list commercial outfit in Google. Google has not only fueled the rapid pace of development behind the project, but created many significant offerings around TensorFlow that make it easier to deploy and easier to use: the above-mentioned TPU silicon for accelerated performance in Google's cloud; an online hub for sharing models created with the framework; in-browser and mobile-friendly incarnations of the framework; and much more.

One caveat: Some details of TensorFlow's implementation make it hard to obtain totally deterministic model-training results for some training jobs. Sometimes a model trained on one system will vary slightly from a model trained on another, even when they are fed the exact same data. The reasons for this are slippery—e.g., how random numbers are seeded and where, or certain non-deterministic behaviors when using GPUs). That said, it is possible to work around those issues, and TensorFlow's team is considering more controls to affect determinism in a workflow.

### **TensorFlow vs. the competition**

TensorFlow competes with a slew of other machine learning frameworks. PyTorch, CNTK, and MXNet are three major frameworks that address many of the same needs. Below I've noted where they stand out and come up short against TensorFlow.

- **PyTorch**, in addition to being built with Python, and has many other similarities to TensorFlow: hardware-accelerated components under the hood, a highly interactive development model that allows for design-as-you-go work, and many useful components already included. PyTorch is generally a better choice for fast



development of projects that need to be up and running in a short time, but TensorFlow wins out for larger projects and more complex workflows.

- **CNTK**, the Microsoft Cognitive Toolkit, like TensorFlow uses a graph structure to describe dataflow, but focuses most on creating deep learning neural networks. CNTK handles many neural network jobs faster, and has a broader set of APIs (Python, C++, C#, Java). But CNTK isn't currently as easy to learn or deploy as TensorFlow.
- **Apache MXNet**, adopted by Amazon as the premier deep learning framework on AWS, can scale almost linearly across multiple GPUs and multiple machines. It also supports a broad range of language APIs—Python, C++, Scala, R, JavaScript, Julia, Perl, Go—although its native APIs aren't as pleasant to work with as TensorFlow's.

### 2.3 MOBILENET

MobileNet is a CNN architecture model for Image Classification and Mobile Vision. There are other models as well but what makes MobileNet special that it very less computation power to run or apply transfer learning to. This makes it a perfect fit for Mobile devices, embedded systems and computers without GPU or low computational efficiency with compromising significantly with the accuracy of the results. It is also best suited for web browsers as browsers have limitation over computation, graphic processing and storage. As a lightweight deep neural network, MobileNet has fewer parameters and higher classification accuracy. In order to further reduce the number of network parameters and improve the classification accuracy, dense blocks that are proposed in Dense Nets are introduced into MobileNet. In Dense-MobileNet models, convolution layers with the same size of input feature maps in MobileNet models are taken as dense blocks, and dense connections are carried out within the dense blocks. The new network structure can make full use of the output feature maps generated by the

previous convolution layers in dense blocks, so as to generate a large number of feature maps with fewer convolution cores and repeatedly use the features. By setting a small growth rate, the network further reduces the parameters and the computation cost. Two Dense-MobileNet models, Dense1-MobileNet and Dense2-MobileNet, are designed. Experiments show that Dense2-MobileNet can achieve higher recognition accuracy than MobileNet, while only with fewer parameters and computation cost.

Compared with VGG-16 network, MobileNet is a lightweight network, which uses depth wise separable convolution to deepen the network, and reduce parameters and computation. At the same time, the classification accuracy of MobileNet on ImageNet data set only reduces by 1%. However, in order to be better applied to mobile devices with limited memory, the parameters and computational complexity of the MobileNet model need to be further reduced. Therefore, we use dense blocks as the basic unit in the network layer of MobileNet. By setting a small growth rate, the model has fewer parameters and lower computational cost. The new models, namely Dense-MobileNets, can also achieve high classification accuracy. MobileNet is a streamlined architecture that uses depth wise separable convolutions to construct lightweight deep convolutional neural networks and provides an efficient model for mobile and embedded vision applications Depth wise Separable Convolution.

This convolution originated from the idea that a filter's depth and spatial dimension can be separated- thus, the name separable. Let us take the example of Sobel filter, used in image processing to detect edges. You can separate the height and width dimensions of these filters. Gx filter can be viewed as a matrix product of  $\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$  transpose with  $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ .

We notice that the filter had disguised itself. It shows it had nine parameters, but it has 6. This has been possible because of the separation of its height and width dimensions.

The same idea applied to separate depth dimension from horizontal (width\*height) gives us depth-wise separable convolution where we perform depth-wise convolution. After that, we use a  $1 \times 1$  filter to cover the depth dimension.

One thing to notice is how much parameters are reduced by this convolution to output the same no. of channels. To produce one channel, we need  $3 \times 3 \times 3$  parameters to perform depth-wise convolution and  $1 \times 3$  parameters to perform further convolution in-depth dimension.

But If we need three output channels, we only need  $3 \times 3$  depth filter, giving us a total of 36 ( = 27 +9) parameters while for the same no. of output channels in regular convolution, we need  $3 \times 3 \times 3$  filters giving us a total of 81 parameters.

## 2.4 OPENCV

OpenCV is a library of programming functions mainly aimed at real-time computer vision. The library has more than 2500 optimized algorithms. And it will help us to load images in python and convert them into array. Each index of array represents (red, green, blue) color pixel which ranges from 0 to 255. Computer vision is a process by which we can understand the images and videos how they are stored and how we can manipulate and retrieve data from them. Computer Vision is the base or mostly used for Artificial Intelligence. Computer-Vision is playing a major role in self-driving cars, robotics as well as in photo correction apps.

OpenCV is the huge open-source library for computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human. When it integrated with various libraries, such as Numpy, python is capable of processing the OpenCV array structure for analysis. To Identify image patterns and its various features we use vector space and perform mathematical operations on these features.

The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both **academic** and **commercial** use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When OpenCV was designed the main focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

### **Image-Processing**

Image processing is a method to perform some operations on an image, in order to get an enhanced image and or to extract some useful information from it. If we talk about the basic definition of image processing. An image may be defined as a two-dimensional function  $f(x, y)$ , where  $x$  and  $y$  are spatial(plane) coordinates, and the amplitude of fat any pair of coordinates  $(x, y)$  is called the intensity or grey level of the image at that point. In another word An image is nothing more than a two-dimensional matrix (3-D in case of colored images) which is defined by the mathematical function  $f(x, y)$  at any point is giving the pixel value at that point of an image, the pixel value describes how bright that pixel is, and what color it should be. Image processing is basically signal processing in which input is an image and output is image or characteristics according to requirement associated with that

image.

**Image processing basically includes the following three steps:**

1. Importing the image
2. Analyzing and manipulating the image
3. Output in which result can be altered image or report that is based on image analysis

We are humans we can easily make it out that is the image of a person who is me. But if we ask computer “is it my photo?”. The computer can’t say anything because the computer is not figuring out it all on its own. The computer reads any image as a range of values between 0 and 255. For any color image, there are 3 primary channels -red, green and blue.

### **Matplotlib:**

Matplotlib is an open-source plotting library in Python introduced in the year 2003. It is a very comprehensive library and designed in such a way that most of the functions for plotting in MATLAB can be used in Python.

It consists of several plots like the Line Plot, Bar Plot, Scatter Plot, Histogram etc., through which we can visualise various types of data.

### **IMAGE DATA GENERATOR:**

When working with deep learning models, I have often found myself in a peculiar situation when there is not much data to train my model. It was in times like these when I came across the concept of image augmentation. The image augmentation technique is a great way to expand the size of your dataset. You can come up with new transformed images from your original dataset. But many people use the conservative way of augmenting the images i.e. augmenting images and storing them in a numpy array or in a folder. I have got to admit, I used to do

this until I stumbled upon the ImageDataGenerator class. Augmentation techniques with Keras ImageDataGenerator class

### 1. Random Rotations

Image rotation is one of the widely used augmentation techniques and allows the model to become invariant to the orientation of the object. ImageDataGenerator class allows you to randomly rotate images through any degree between 0 and 360 by providing an integer value in the rotation\_range argument. When the image is rotated, some pixels will move outside the image and leave an empty area that needs to be filled in. You can fill this in different ways like a constant value or nearest pixel values, etc. This is specified in the fill\_mode argument and the default value is “nearest” which simply replaces the empty area with the nearest pixel values.

### 2. Random Shifts

It may happen that the object may not always be in the center of the image. To overcome this problem we can shift the pixels of the image either horizontally or vertically; this is done by adding a certain constant value to all the pixels. ImageDataGenerator class has the argument height\_shift\_range for a vertical shift of image and width\_shift\_range for a horizontal shift of image. If the value is a float number, that would indicate the percentage of width or height of the image to shift. Otherwise, if it is an integer value then simply the width or height are shifted by those many pixel values.

### 3. Random Flips

Flipping images is also a great augmentation technique and it makes sense to use it with a lot of different objects. ImageDataGenerator class has parameters horizontal\_flip and vertical\_flip for flipping along the vertical or the

horizontal axis. However, this technique should be according to the object in the image. For example, vertical flipping of a car would not be a sensible thing compared to doing it for a symmetrical object like football or something else. Having said that, I am going to flip my image in both ways just to demonstrate the effect of the augmentation.

#### 4. Random Brightness

It randomly changes the brightness of the image. It is also a very useful augmentation technique because most of the time our object will not be under perfect lighting condition. So, it becomes imperative to train our model on images under different lighting conditions. Brightness can be controlled in the ImageDataGenerator class through the `brightness_range` argument. It accepts a list of two float values and picks a brightness shift value from that range. Values less than 1.0 darkens the image, whereas values above 1.0 brighten the image.

#### 5. Random Zoom

The zoom augmentation either randomly zooms in on the image or zooms out of the image. ImageDataGenerator class takes in a float value for zooming in the `zoom_range` argument. You could provide a list with two values specifying the lower and the upper limit. Else, if you specify a float value, then zoom will be done in the range  $[1-\text{zoom\_range}, 1+\text{zoom\_range}]$ . Any value smaller than 1 will zoom in on the image. Whereas any value greater than 1 will zoom out on the image.

#### ADAM OPTIMIZER:

The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent

procedure to update network weights iterative based in training data. Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. Adam is relatively easy to configure where the default configuration parameters do well on most problems.

### NumPy:

NumPy is a free Python library equipped with a collection of complex mathematical operations suitable for processing statistical data. **Data manipulation** is a core component in data sciences and machine learning. In the case of machine learning, the system takes in a huge amount of data and trains itself to make accurate predictions. Naturally, it would need to perform arithmetic operations on numerical data to produce something meaningful. This is where NumPy comes into play. The main data structure in NumPy is the NumPy array. All the data is stored in arrays. NumPy provides a vast repertoire of mathematical and statistical operations which can be performed on these arrays.

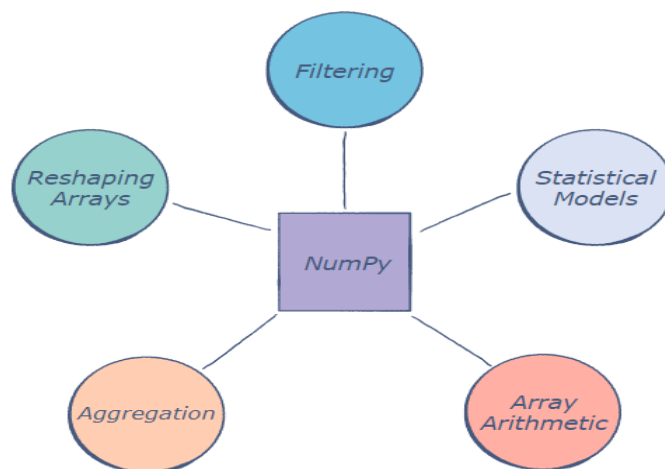


Fig.1.2 Numpy process



## IMUTILS:

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

## OPENCV DNN MODULES:

The field of computer vision has existed since the late 1960s. Image classification and object detection are some of the oldest problems in the field of computer vision that researchers have tried to solve for many decades. Using neural networks and deep learning, we have reached a stage where computers can start to actually understand and recognize an object with high accuracy, even surpassing humans in many cases. And to learn about neural networks and deep learning with computer vision, the OpenCV's DNN module is a great place to start. With its highly optimized CPU performance, beginners can also get started easily even if they do not have a very powerful GPU enabled system.

We all know OpenCV as one of the best computer vision libraries out there. Additionally, it also has functionalities for running deep learning inference as well. The best part is supporting the loading of different models from different frameworks using which we can carry out several deep learning functionalities. The feature of supporting models from different frameworks has been a part of OpenCV since version 3.3. Still, many newcomers into the field are not aware of this great feature of OpenCV. Therefore, they tend to miss out on many fun and good learning opportunities. The OpenCV DNN module only supports deep learning inference on images and videos. It does not support fine-tuning and

training. Still, the OpenCV DNN module can act as a perfect starting point for any beginner to get into the field of deep-learning based computer vision and play around.

One of the OpenCV DNN module's best things is that it is highly optimized for Intel processors. We can get good FPS when running inference on real-time videos for object detection and image segmentation applications. We often get higher FPS with the DNN module when using a model pre-trained using a specific framework. For example, let us take a look at image classification inference speed for different frameworks.

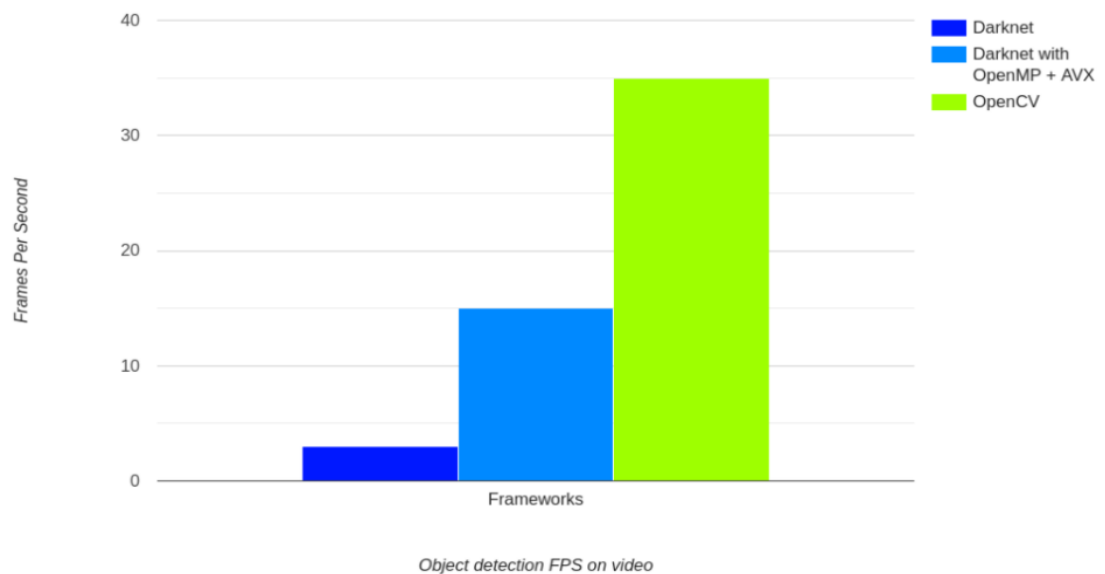


Fig. 1.3 Object detection FPS

#### **Object detection speed comparison on CPU for different frameworks:**

The above plot shows the results for FPS on video with Tiny YOLOv4 on the original Darknet framework and OpenCV. The benchmark was done on an Intel i7 8th Gen laptop CPU with 2.6Ghz clock speed. We can see that on the same video, the *OpenCV's DNN module is running at 35 FPS* whereas *Darknet compiled with*

*OpenMP and AVX is running at 15 FPS. And Darknet (without OpenMP or AVX) Tiny YOLOv4 is the slowest, running at only 3 FPS.* This is a huge difference considering we are using the original Darknet Tiny YOLOv4 models in both cases. The above graphs show the actual usefulness and power of the OpenCV DNN module when working with CPUs.

### **Different Frameworks that OpenCV DNN Module Supports:**

OpenCV DNN module supports many popular deep learning frameworks. The following are the deep learning frameworks that the OpenCV DNN module supports.

#### **Caffe**

To use a pre-trained Caffe model with OpenCV DNN, we need two things. One is the model. Caffe model file that contains the pre-trained weights. The other one is the model architecture file which has a .prototxt extension. Caffe is a deep learning framework made with expression, speed, and modularity in mind. Deep networks are compositional models that are naturally represented as a collection of inter-connected layers that work on chunks of data. Caffe defines a net layer-by-layer in its own model schema. The network defines the entire model bottom-to-top from input data to loss.

- 1 .prototxt file which defines the model architecture
- 2 .caffemodel file which contains the weights for the actual layers

## CHAPTER - 3

### 3.1 DATA COLLECTION

Data collection is the first step in the face mask identification model development project. Accuracy of the training data impacts on the final overall accuracy of the model. In our case, we have to train the model to find if the person is wearing facemask or not. So, we have downloaded a large volume of images of peoples who wear facemask and people who don't wear the facemasks and one who did not properly wear the facemask. System should be able to classify whether a person is wearing a facemask or not

### 3.2 DATA VISUALIZATION

In the first step, let us visualize the total number of images in our dataset in both categories. We can see that there are **1915** images in the 'yes' class and **1918** images in the 'no' class.

The number of images with facemask labelled 'Yes' : 1915. The number of images with facemask labelled 'No' : 1918. Dependencies are all of the software components required by your project in order for it to work as intended and avoid runtime errors.

- TensorFlow Datasets is a collection of datasets ready to use, with TensorFlow or other Python ML frameworks, such as Jax. All datasets are exposed as tensorflow data. Datasets, enabling easy-to-use and high-performance input pipelines. To get started see the guide and our list of datasets. Convert the image pixels to float datatype. Normalize the image to have pixel values scaled down between 0 and 1 from 0 to 255. **Image data for Deep Learning models should be either a numpy to array or a tensor object.**

### 3.3 DATA AUGMENTATION

- In the next step, we *augment* our dataset to include more number of images for our training. In this step of *data augmentation*, we *rotate* and *flip* each of the images in our dataset. We see that, after data augmentation, we have a total of **3833** images with **1532** images in the 'yes' class and **1535** images in the '*no*' class.

### 3.4 SPLITTING THE DATA

In this step, we *split* our data into the *training set* which will contain the images on which the CNN model will be trained and the *test set* with the images on which our model will be tested. In this, we take *split\_size* = 0.8, which means that 80% of the total images will go to the *training set* and the remaining 20% of the images will go to the *test set*.

- The number of images with facemask in the training set labelled 'yes': 1332
- The number of images with facemask in the test set labelled 'yes': 276
- The number of images without facemask in the training set labelled 'no': 1335
- The number of images without facemask in the test set labelled 'no': 275

DATASET SPLITTING				
	WITH MASK	WITHOUT MASK	TOTAL	PERCENTAGE
TRAIN	1530	1536	3066	80%
VALIDATE	384	382	766	20%
			3832	

TABLE 2.1 SPLTTING THE DATASET

### 3.5 BUILDING THE MODEL

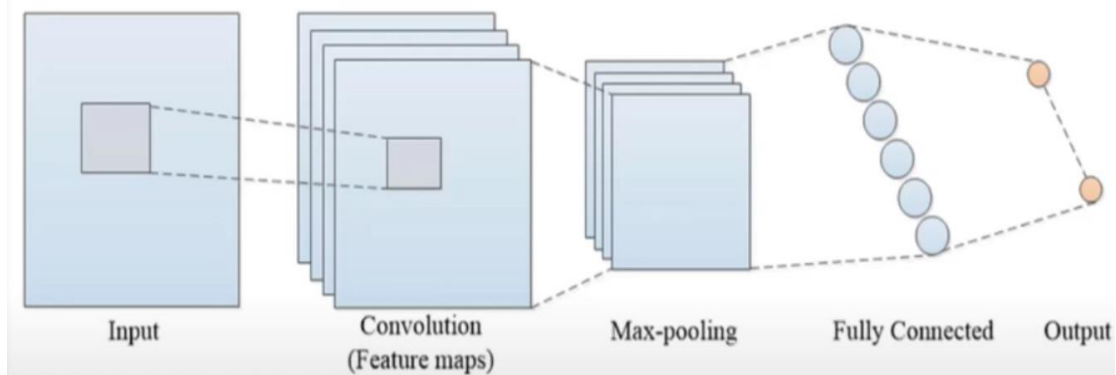


Fig 1.4 Convolution Architecture Diagram

The next step is to load the pre-trained model and customize it according to our problem. So we just remove the top layers of this pre-trained model and add few layers of our own. As you can see the last layer has two nodes as we have only two outputs. This is called transfer learning. In the next step, we build our *Sequential CNN model* with various layers such as *Conv2D*, *MaxPooling2D*, *Flatten*, *Dropout* and *Dense*. In the last Dense layer, we use the ‘*softmax*’ function

to output a vector that gives the *probability* of each of the two classes. Here, we use the ‘adam’ optimizer and ‘binary\_crossentropy’ as our loss function as there are only two classes. Additionally, you can even use the *MobileNetV2* for better accuracy.

### 3.5 TRAINING THE MOBILENET MODEL

After building our model, let us create the ‘train\_generator’ and ‘validation\_generator’ to fit them to our model in the next step. We see that there are a total of 2200 images in the training set and 551 images in the test set. We see that after the 2nd try, our model has an accuracy of 92% with the training set and an accuracy of 98% with the test set. This implies that it is well trained without any over-fitting.

### 3.6 Results

Analyzing the conducted experiments, we obtain the following results:

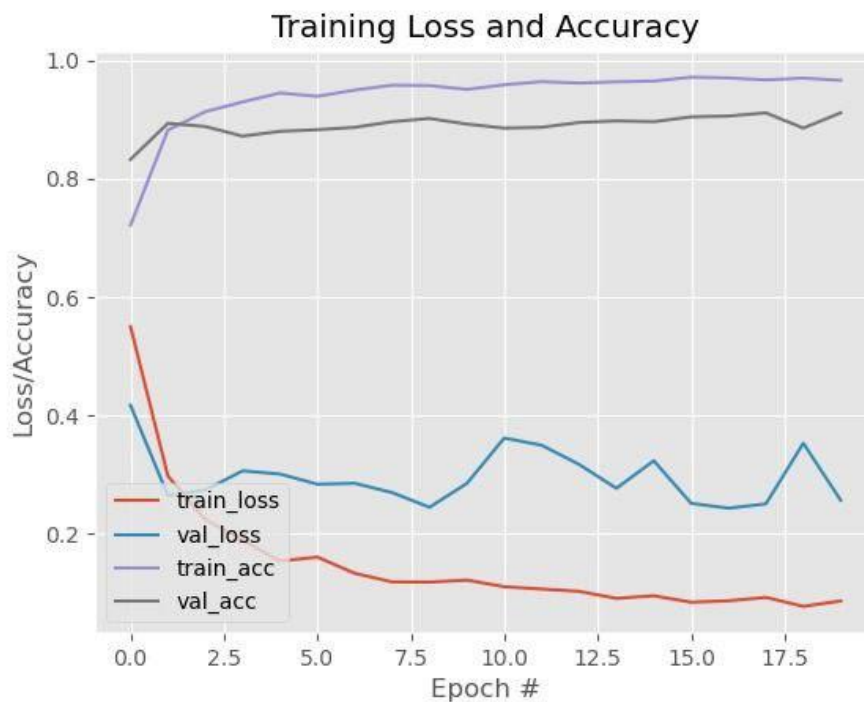


Fig. 1.5 Loss and Accuracy Graph

Using MobileNetV2 model, we got an accuracy of 92% and got a good classification report. The precision we got for with mask and without mask are 0.98 and 0.97 respectively. And recall for with mask and without mask are 0.97 and 0.98 respectively. And f1-score for both classes is of 0.98. And obtained a validation accuracy of 98% as shown in Fig.1.3

	Precision	Recall	F1-score	support
With mask	0.98	0.97	0.98	383
without mask	0.97	0.98	0.98	384
Accuracy			0.98	787
Macro avg	0.98	0.98	0.98	787
Weighted avg	0.98	0.98	0.98	787

Table 2.2 Validation

## CHAPTER - 4

### 4.1 Face Detector

At the starting stage, camera provides an object detection model for the real-time video. The person face along with their masks are detected by the object detection model. The person's overlapped area and depending on the position, the bounding boxes are created around the person's face and were detected. This new method is used to crop the frame, the current cropped frame and person's face combination is passed through the image classification model. This image classification model can classify the cropped image into mask or no mask class.



In the last step, we use the OpenCV library to run an infinite loop to use our web camera in which we detect the face using the Cascade Classifier. The code `webcam = cv2.VideoCapture(0)` denotes the usage of webcam. The model will predict the possibility of each of the two classes ([without\_mask, with\_mask]). Based on which probability is higher, the label will be chosen and displayed around our faces. Additionally, you can download the [DroidCam](#) application for both Mobile and PC to use your mobile's camera and change the value from 0 to 1 in `webcam = cv2.VideoCapture(1)`.

We apply face detection from there to calculate the location of the bounding box in the image. Once we know where in the image the face is, we can extract the face Region of Interest (ROI), and from there, we apply facial landmarks, allowing us to localize mouth, face, and eyes. In order to apply masks, we need an image of a mask (with a transparent and high definition image). Add the mask to the detected face and then resize and rotate, placing it on the face. Repeat this process for all input images. With `vs.read()` we catch our videostream from our camera and then resize frame to fixed width and height of 1000 pixels. We need to extract height and weight dimensions for a stream with `frame.shape[:2]` where we extract first and second dimension. Now we are creating a "blob" and passing through Neural Network with `cv2.dnn.blobFromImage()` function. With `setInput()` we are setting new input value for the network which is our "blob" and with `forward()` we are running forward pass to compute the output of layer. Now we will loop over detections, compare detections with our confidence threshold and draw a box around detection and output prediction value. We are extracting confidence and compare it to the confidence threshold so we can filter out detection that are weak. If confidence is at least at minimum threshold we proceed to draw a rectangle with the probability of the detection. First, we calculate x,y coordinates of the bounding box and build confidence with "text" string which is holding probability of the detection.

## 4.2 DETECTING FACES WITH OR WITHOUT MASK

We conducted the experiments using laptop camera with aspect ratio 4:3 to shoot the real-time video, the shot video is of the front view. From the threaded video stream grab the frame and resize it to have a maximum width of 400 pixels. Now we draw bounding box and text prediction with `cv2.rectangle()` and `cv2.putText()` on our frame. And we are repeating loop process to check if there are any other detection on the image and if there isn't we are output image on the screen. Until we press 'q' on our keyboard stream with face detection will run. And after pressing 'q' stream is stopping and we are closing the window with video stream.



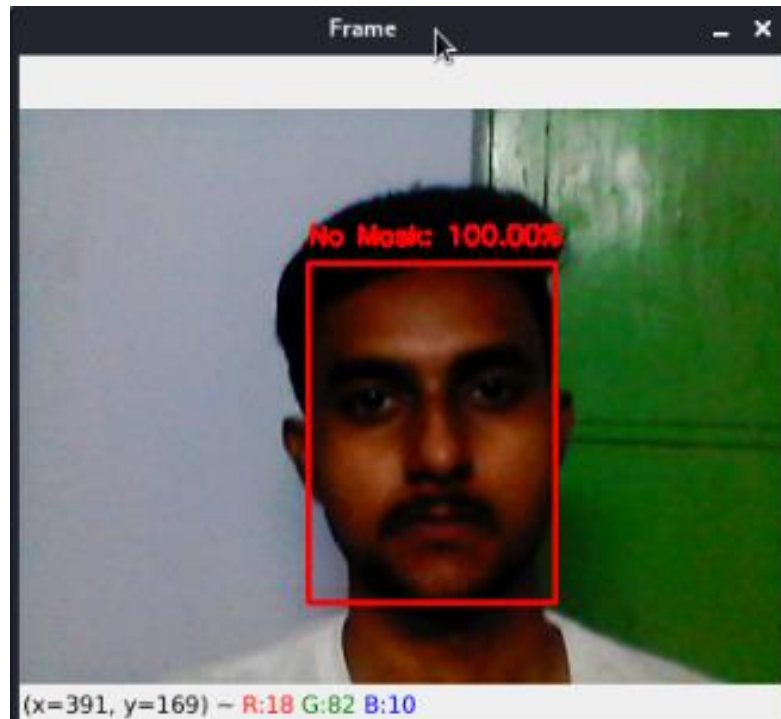


Fig. 1.6 Experimental Output

### Haar Cascade with Face Detection.

- In this project we just compare with Haar Cascade also (Object Detection Algorithm)
- It is an Object Detection Algorithm used to identify faces in an image or a real time video.
- The algorithm uses edge or line detection features proposed by Viola and Jones in their research paper “Rapid Object Detection using a Boosted Cascade of Simple Features”
- **Haar Cascade** is a machine learning-based approach where a lot of positive and negative images are used to train the **classifier**.
- Positive images – These images contain the images which we want our **classifier** to identify.

- Negative Images – Images of everything else, which do not contain the object we want to detect.
- **Haar-like features** are digital image **features** used in object recognition.
- A **Haar-like feature** considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums.
- But, why use Caffe Model.
- For computer vision problems, OpenCV's Caffe **model** of the DNN module is the **best**. It works well with occlusion, quick head movements, and can identify side faces as well. Moreover, it also gave the quickest fps among all.

## Conclusion & Future Work

We have developed a system which monitor our face through the ordinary laptop front camera in real-time. The system can check whether the people wearing masks or not. So, it helps to stop the widespread of corona virus. It can be used for a variety of applications by contribute immensely to the public health care system. Because not wearing masks increases the COVID-19 virus into wider community spread. We extract the images in real-time from camera and capturing it frame by frame and predict using the model. For face detection caffe model is used. The datasets used for training the model consists of two classes which are people wearing with mask and without mask. The model used for detection is MobileNetV2, the validation accuracy score is 98.2%. We worked with a medium scale of dataset this model gives a good result; we are still trying to enhance the model so that we have a better score. We will use different kind of models to analyze the accuracy score and choose the best model.

**Future scope:**

We can further improve the model with better training for face mask detection. The accuracy obtained is limited, due to lack of resources. This model can also be used in public places like supermarkets, airports, offices, etc. As the door can be opened only when people wearing masks and does not open the door unless people wearing mask. This is one of the future developments that can be implemented to this model. The system can be converted to automation as someone enters the shops without mask an alarm sounds loud there for ensuring the safety.

## SUMMARY

Deep Learning is the subset of Artificial Intelligence that deals with the extraction of patterns from datasets. Deep learning is the act of utilizing algorithms to parse information, gain from it, and afterward make a prediction or expectation about something in the world. In our new system, we apply the techniques of deep learning which is a subset of machine learning algorithm that utilizes complex MobileNet. One of the main areas of research for our proposed innovation includes the usage of pre-trained real time object detection models and image classification models using convolutional neural networks (CNNs). Image classification using CNNs aims to classify an input image based on visual content. Many models such as Caffe model (Face detection), MobileNetV2 etc. have been developed over the years and some of them are quite popular due to its high performance. MobileNetV2 is a very popular model which is known to show almost 90% accuracy on the ImageNet dataset. The model is a combination of ideas developed by many researchers. In our system, we use the MobileNetV2 model for transfer learning. By using such Deep learning models, we can automate the process of face mask detection with the use of a real time surveillance system. Being the best opposite of shallow architecture which has few hidden layers, deep architecture has several layers. In this project, we'll discuss our two-phase COVID-19 face mask detector, detailing how our computer vision/deep learning pipeline will be implemented. From there, we'll review the dataset we'll be using to train our custom face mask detector. I'll then show you how to implement a Python script to train a face mask detector on our dataset using Keras and TensorFlow. We'll use this Python script to train a face mask detector and review the results.

## **Appendices**

### **1. Load dataset:**

```
DIRECTORY = r"D:\capstone\face-mask-detector\dataset"
```

```
CATEGORIES = ["with_mask", "without_mask"]
```

```
print("[INFO] loading images...")
```

```
data = []
```

```
labels = []
```

```
for category in CATEGORIES:
```

```
    path = os.path.join(DIRECTORY, category)
```

```
    for img in os.listdir(path):
```

```
img_path = os.path.join(path, img)
```

```
    image = load_img(img_path, target_size=(224, 224))
```

```
    image = img_to_array(image)
```

```
    image = preprocess_input(image)
```

```
data.append(image)
```

```
labels.append(category)
```

### **2. Perform one-hot encoding on the labels:**

```
lb = LabelBinarizer()
```

```
labels = lb.fit_transform(labels)
```

```
labels = to_categorical(labels)
```

```
data = np.array(data, dtype="float32")
```

```
labels = np.array(labels)
```

### **3. Splitting the data:**

```
(trainX, testX, trainY, testY) = train_test_split(data, labels,  
test_size=0.20, stratify=labels, random_state=42)
```

### **4. 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")
```

### **5. Building the model:**

```
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

model = Model(inputs=baseModel.input, outputs=headModel)

for layer in baseModel.layers:

    layer.trainable = False

```

## 6. Pre-training:

```

opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)

model.compile(loss="binary_crossentropy", optimizer=opt,

    metrics=["accuracy"])

```

## 7. Training:

```

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)
```

#### **8. Predictions on the testing set:**

```
print("[INFO] evaluating network...")  
  
predIdxs = model.predict(testX, batch_size=BS)  
  
predIdxs = np.argmax(predIdxs, axis=1)  
  
# show a 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("mask_detector.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("plot.png")
```

## **9. Face Detector**

```
# 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 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, (224, 224),
                              (104.0, 177.0, 123.0))

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

detections = faceNet.forward()

print(detections.shape)

# 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 > 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 = frame[startY:endY, startX:endX]
    face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
    face = cv2.resize(face, (224, 224))
    face = img_to_array(face)
    face = preprocess_input(face)

```

```

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

        faces.append(face)

        locs.append((startX, startY, endX, endY))

# only make a predictions if at least one face was detected
if len(faces) > 0:

    # for faster inference we'll make batch predictions on *all*
    # faces at the same time rather than one-by-one predictions

    # in the above `for` loop

    faces = np.array(faces, dtype="float32")

    preds = maskNet.predict(faces, batch_size=32)

# return a 2-tuple of the face locations and their corresponding
# locations

return (locs, preds)

# load our serialized face detector model from disk

prototxtPath = r"face_detector\deploy.prototxt"

weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"

faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

```

```
# load the face mask detector model from disk

maskNet = load_model("mask_detector.model")


# initialize the video stream

print("[INFO] starting video stream...")

vs = VideoStream(src=0).start()


# loop over the frames from the video stream

while True:

    # grab the frame from the threaded video stream and resize it

    # to have a maximum width of 400 pixels

    frame = vs.read()

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


    # detect faces in the frame and determine if they are wearing a

    # face mask or not

    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)


    # loop over the detected face locations and their corresponding

    # locations
```

```

for (box, pred) in zip(locs, preds):

    # unpack the bounding box and predictions

    (startX, startY, endX, endY) = box

    (mask, withoutMask) = pred


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

    # the bounding box and text

    label = "Mask" if mask > withoutMask else "No Mask"

    color = (0, 255, 0) if label == "Mask" else (0, 0, 255)


    # include the probability in the label

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


    # display the label and bounding box rectangle on the output

    # frame

    cv2.putText(frame, label, (startX, startY - 10),

                cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)

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


    # show the output frame

    cv2.imshow("Frame", frame)

```



```
key = cv2.waitKey(1) & 0xFF
```

```
# if the `q` key was pressed, break from the loop
```

```
if key == ord("q"):
```

```
    break
```

```
# do a bit of cleanup
```

```
cv2.destroyAllWindows()
```

```
vs.stop()
```

## REFERENCES

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Ghemawat, S. (2018). "Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV.(2020)"
2. Boyko, Nataliya, Oleg Basystiuk, and Nataliya Shakhovska. "Performance evaluation and comparison of software for face recognition, based on dlib and opencv library." 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP). IEEE, 2018.
3. Dahiya, Kunal, Dinesh Singh, and C. Krishna Mohan. "Automatic detection of bike-riders without helmet using surveillance videos in real-time." 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017.
4. Das, Arjya, Mohammad Wasif Ansari, and Rohini Basak. "Covid-19 Face Mask Detection Using TensorFlow, Keras and OpenCV.(2020)"
5. Jalled, Fares, and Ilia Voronkov. "Object detection using image processing." arXiv preprint arXiv:1611.07791 (2019).
6. Jiang, Mingjie, and Xinqi Fan. "RetinaMask: a face mask detector." arXiv preprint arXiv:2005.03950 (2020).
7. Kalas, Mamata S. "Real time face detection and tracking using opencv." international journal of soft computing and Artificial Intelligence 2.1 (2018): 41-44.
8. Loey, Mohamed, et al. "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the COVID-19 pandemic." Measurement 167 (2020): 108288.
9. Militante, Sammy V., and Nanette V. Dionisio. "Real-Time Facemask Recognition with Alarm System using Deep Learning." 2020 11th IEEE Control and System Graduate Research Colloquium (ICSGRC). IEEE, 2020.

10. Redmon, J., &Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),
11. Rohith, C. A., et al. "An efficient helmet detection for MVD using deep learning." 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI). IEEE, 2019.
12. Sharma, Vinay. "Face Mask Detection using YOLOv5 for COVID-19." (2020).
13. S. Wang Chen, Horby Peter W, Hayden Frederick G, Gao George F. A novel coronavirus epidemic of global concern for health. It's the Lancet.
14. T.-H. Kim, D.-C. Park, D.-M.Woo, T. Jeong, and S.-Y. Min, "Multi-class classifier-based adaboost algorithm," in Proceedings of the Second Sinoforeign-interchange Conference on Intelligent Science and Intelligent Data Engineering, ser. IScIDE'11.
15. Venkateswarlu, Isunuri B., JagadeeshKakarla, and Shree Prakash. "Face mask detection using MobileNet and Global Pooling Block." 2020 IEEE 4th Conference on Information & Communication Technology (CICT). IEEE, 2020.

