**A PROJECT REPORT(IBM19-G20)**

**ON**

**MASK DETECTION ON HUMAN FACE**

*A report submitted in partial fulfilment of the requirement for the award of*

*The degree of*

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE AND ENGINEERING/INFORMATION TECHNOLOGY**

A picture containing drawing

Description automatically generated

**Submitted to:**

Guide Name- DR. SRABANTI MAJI

Designation- Assistant Professor

**Submitted by:**

KUSHAGRA SINHA - 190111031 - BTECH CSE CSF

ALOK RANJAN TIWARI - 190111032- BTECH CSE CSF

RISHI RAJ SINGH CHAUHAN - 190111033- BTECH CSE CSF

SOURAV KUMAR MAHTO - 190111034 - BTECH CSE CSF

**SCHOOL OF COMPUTING**

**DIT UNIVERSITY, DEHRADUN**

(State Private University through State Legislature Act No. 10 of 2013 of Uttarakhand and approved by UGC)

**ACKNOWLEDGEMENT**

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

We would like to thank our Project Co-ordinator Dr. Srabanti Maji for his valuable guidelines and for always encouraging us. We are wholeheartedly thankful to his for giving us his valuable time & attention & for providing us a systematic way for progressing our project in time.

Also, not forgetting our parents and friends who have been supporting us and helping us in all the possible ways they can, despite the miserable situation of Covid-19. This project is a small contribution towards the betterment of the ongoing negative situation.

Thank you everyone.

# TABLE OF CONTENTS

**S. No. Chapter Contents**

**1 Abstract**

**2 Chapter 1**  **Introduction**

**3 Chapter 1** **Aim**

**4 Chapter 1** **Objective**

**5 Chapter 1 Problem Definition**

**6 Chapter 1** **Related Work**

**7 Chapter 1** **Expected Output**

**8 Chapter 2**  **Requirement Specification and Analysis**

**9 Chapter 2**  **Functional Requirements**

**10 Chapter 2** **Non-Functional Requirements**

**11 Chapter 2** Fundamental Steps to Perform For Mask Detection

**12 Chapter 2** **Data Collection**

**13 Chapter 2**  **Sequence Diagram**

**14 Chapter 2** **Use Case Diagram**

**15 Chapter 2** **Activity Diagram**

**16 Chapter 2** **Class Diagram**

**17 Chapter 3**  **Design, Implementation and Result**

**18 Chapter 3**  **Hardware and Software Requirements**

**19 Chapter 3 Face Mask Detection Dataset Pre-Processing**

**20 Chapter 3** **Algorithms**

**21 Chapter 3** **Methodologies**

**22 Chapter 3** **Model Summary**

**23 Chapter 3** **Implementation, Explanation and Output**

**24 Chapter 3** **Conclusion**

**25 Chapter 3** **Scope for Improvement And Future Work**

**26 Chapter 3** **References**

# ABSTRACT

This project intends to develop a Face Mask Detection system using OpenCV,

Keras/TensorFlow and Deep Learning. The system can be easily integrated to various embedded devices with limited computational capacity as it uses MobileNetV2 architecture. It will detect face masks in images as well as in real-time videos.

In recent times, where Covid-19 has impacted a domino effect on manufacturing, travel, tourism, hospitality, crippling the global economy. In addition to it, is the growing curve of human deaths across the globe due to the pandemic, this project which relies on computer vision and deep learning, intends to make an impact, and solve the real-world problem of safety measures at some significant level.

This project can be used at airports, offices, hospitals and many more public places to ensure that the safety standards are maintained, and people are abiding by the rules and regulations to wear

protective masks at public places. If the detection system classifies as ‘No Mask’, reminders can be given as well as actions can be taken against such individuals.

## CHAPTER-1

**INTRODUCTION**

**AIM:**

To develop a Face Mask Detection system with OpenCV, Keras/TensorFlow and Deep Learning in order to detect face masks in static images as well as in real-time video streams.

**OBJECTIVE:**

* To train a custom deep learning model to detect whether a person is wearing a mask or not.
* To develop a custom dataset of with mask images with the help of Bing Search API, few existing Kaggle datasets and RMFD dataset.
* Datasets will be divided into two classes: ‘with mask’ and ‘without mask’.
* To train the face mask detector on the custom dataset using Keras and TensorFlow.
* To implement the trained model to detect masks in static input images and also in real-time videos.
* The model is expected to give an accuracy of 90% and above.

**PROBLEM DEFINITION:**

In the present scenario due to Covid-19, there is no efficient face mask detection applications, which are now in high demand for transportation means, densely populated areas, residential districts, large-scale manufacturers, and other enterprises to ensure that the safety guidelines are strictly followed. Also, the absence of large datasets of ‘*with mask*’ images has made this task more cumbersome and challenging.

Therefore, the need of the hour is to generate a huge custom dataset of ‘with mask’ images with the help of existing datasets and search APIs followed by developing a face mask detection system. This system is need of the hour as India tries to battle the novel coronavirus that has infected more than 40,68,218 and has caused more than 4,52,162 deaths with the figures still increasing at a rapid pace.

**RELATED WORK:**

* FebriEye, a thermal camera, comes with additional analytics such as face mask and social distancing monitoring system which generates an alarm in case of any violations. It is being developed by Vehant Technologies which is to be implemented by the Telangana government. [13][10]

* Uber has confirmed to CNN Business that it’s requiring face masks or similar coverings for both drivers and passengers in countries like the US, and is developing technology to detect whether or not drivers are abiding by those rules. [11]

* Face Mask Alert app which is in development process by LeewayHertz software solutions. It sends an alert to the users enforcing them to wear masks. [12]

* AIZOOTech face mask detection system which uses dataset composed of WIDER Face and MAFA, but lacks landmark net for the purpose of face alignment.

**EXPECTED OUTPUT:**

* A two-step system to first **"detect"** faces on an image/live video stream, using a trained face detector from OpenCV and after that to pass the found faces on the **"mask predictor"** that returns if the face is wearing a mask or not.

* A custom dataset of ‘with\_mask’ with 5000+ images for both ‘with\_mask’ and ‘without\_mask’.

* A face mask detector training script in Python, which accepts the input dataset, loads and pre-processes the images, and labels them using TensorFlow.keras and sklearn.

* It would also fine-tune the model with MobileNetV2 classifier using pre-trained ImageNet weights.

* Training history plot with accuracy and loss curves produced with the help of matplotlib.

* Perform face mask detection correctly in static images present in folders.

* Using your webcam, the system developed applies face mask detection correctly to every frame in the real-time video stream.

* Outputs the confidence probability of “Mask” or “No Mask” in the static images and realtime video streams examined.

* The system can be easily integrated to various embedded devices with limited computational capacity as it uses MobileNetV2 architecture.

## CHAPTER-2

### REQUIREMENT SPECIFICATION AND ANALYSIS

**FUNCTIONAL REQUIREMENTS:**

Functional Requirements of Face Mask Dataset:

**R1.** The system must have an unbiased ‘with\_mask’ dataset.

**R2.** The dataset must have over 1500+ images in both ‘with\_mask’ and ‘without\_mask’ classes.

**R3.** The dataset must not re-use the same images in training and testing phases.

Functional Requirements of Face Mask Detector:

**R1.** The system must be correctly able to load the face mask classifier model.

**R2.** The system must be able to detect faces in images or video stream.

**R3.** The system must be able to extract each face’s Region of Interest (ROI).

**R4.** There must not be any object between the system and the face of the user for a successful face detection and hence the face mask detection.

**R5.** The end position of the face must be fit inside the webcam frame and must be closer to the camera.

**R6.** Correctly able to detect masks in ‘*png*’, ‘*jpg*’, ‘*jpeg*’, and ‘*gif*’ format images.

**R7.** The system must be able to detect face masks on human faces on every frame in a live video.

**R8.** The results must be viewed by showing the probability along with the output of ‘Mask’ or ‘No Mask’.

**NON-FUNCTIONAL REQUIREMENTS:**

Product Operation:

**R1.** The face should be localized by detecting the facial landmarks and the background must be ignored.

**R2.** The system will be implemented in Python script with an accuracy of the model of over 90%.

**R3.** The user must not move his/her face out of camera’s sight in order to get correct results.

**R4.** The background must not be too bright or too dark while detecting the face mask.

Product revision:

**R5.** The system must be portable and can be applied to embedded devices with limited computational capacity (ex., Raspberry Pi, Google Coral, NVIDIA Jetson Nano, etc.).

**R6.** The output response operation must be fast and under 5 seconds per person.

**R7.** The system must be able to correctly detect more than one face if present, and hence the presence of mask in the frame.

Product transition:

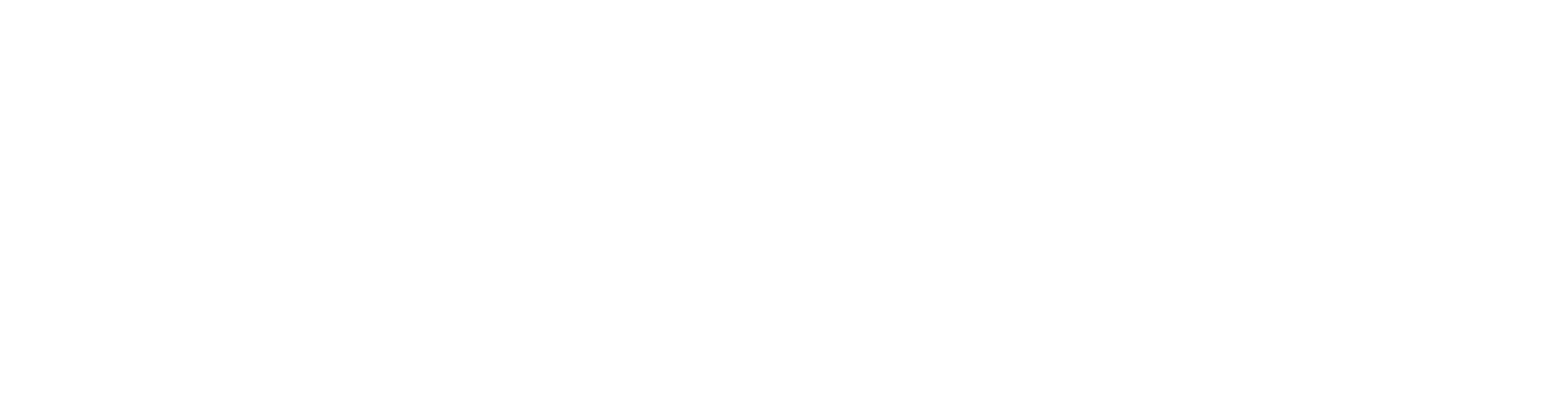
**R8.** The system should be easy for usability and self-descriptive for maintenance purposes.

**R9.** The system must be platform independent and flexible for updates.

**R10.** The security of the system developed will be under MIT License (GitHub Repository Link: https://github.com/chandrikadeb7/Face-Mask-Detection).

### FUNDAMENTAL STEPS TO PERFORM FOR FACE MASK DETECTION

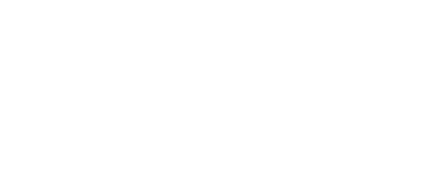
**ACTORS:**



**Phase #1**

**:**

**Train Face Mask Detector**

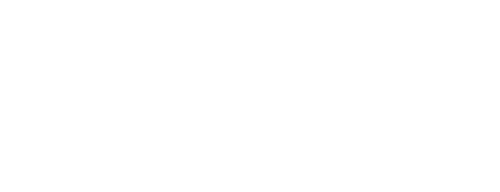


L

oad face mask

data

set

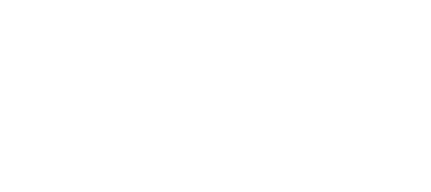


Train face mask

classifier with

Keras/

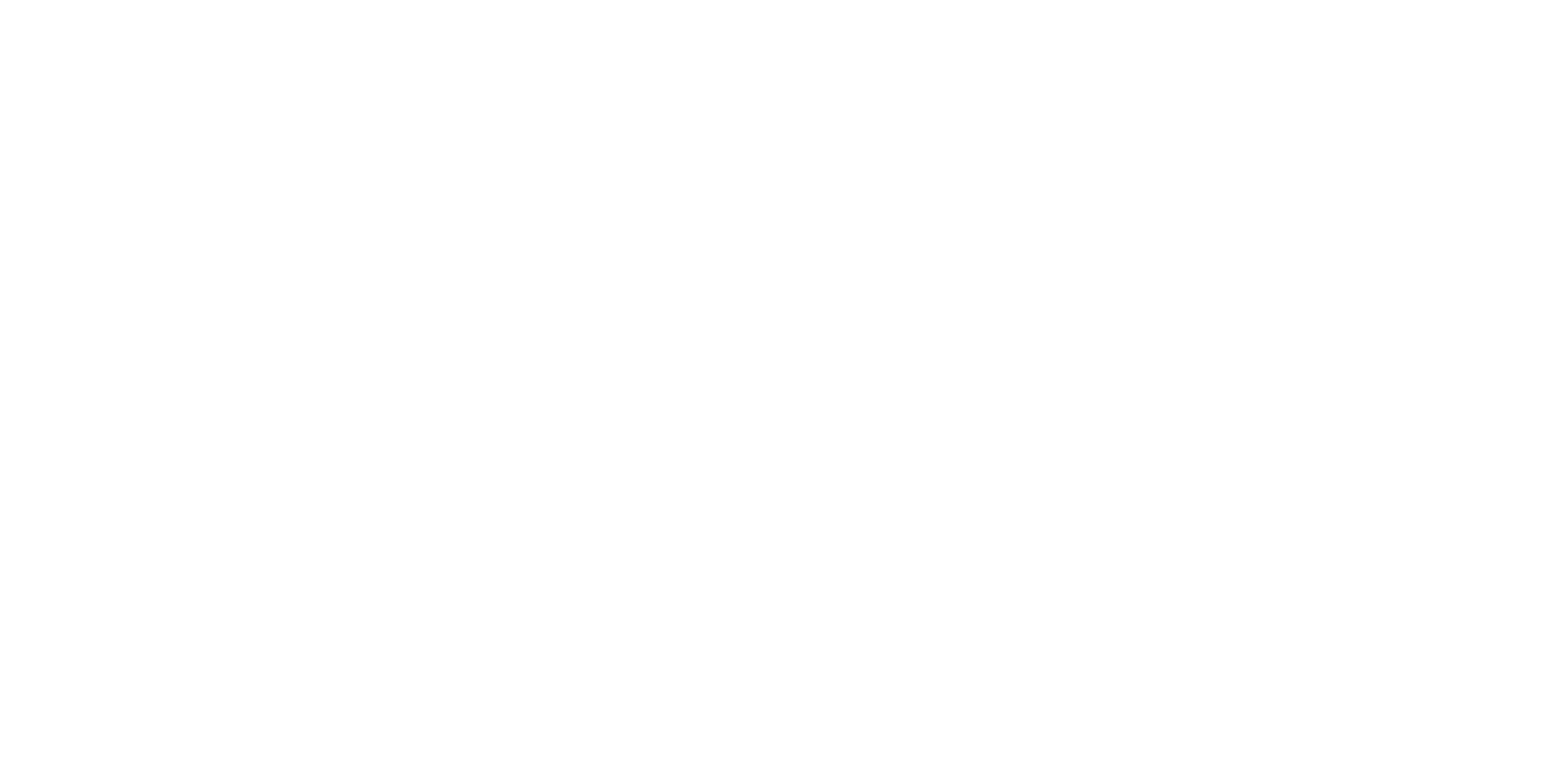
TensorFlow



Serialize face

mask classifier to

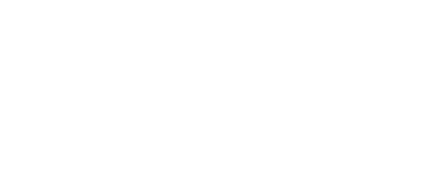
disk



**Phase #2**

**Apply Face Mask Detector**

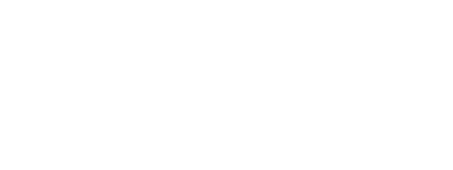
**:**



Load face mask

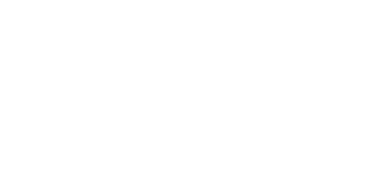
classifier from

disk



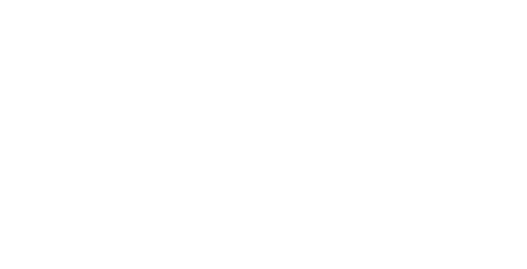
Detect faces in

image/video stream



Extract each

face ROI

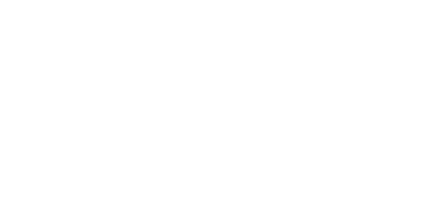


Apply face mask

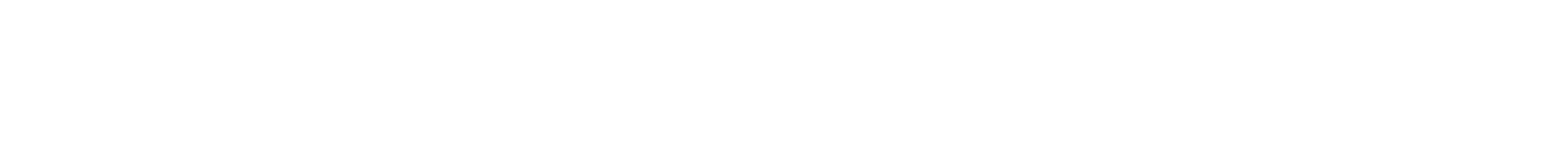
classifier to each face

ROI to determine

“mask” or “no mask”



Show results

**Figure 1:**  Phases and individual steps for building a COVID-19 face mask detector with computer vision and deep learning using Python, OpenCV, and TensorFlow/Keras.

1. **Training:** 
   1. Loading the face mask detection dataset from disk
   2. Training a model using Keras/TensorFlow on this loaded dataset
   3. Serializing the face mask detector back to disk

1. **Deployment:** 
   1. Loading the face mask detector
   2. Performing face detection
   3. Classifying each face as “Mask” or “No Mask”

**DATA COLLECTION**

The data will be collected from the below mentioned sources:

* Real-World Masked Face Dataset (RMFD)
* Kaggle Datasets
* Bing Search API

The aim is to collect more than 1800+ images in both “with\_mask” and “without\_mask” classes.

A Python script will be used for Bing Search API for finding images with multiple queries like “*covid mask*”, “*face mask*”, “*N95 mask*”.

**File: search.py**

|  |
| --- |
| from requests import exceptions import argparse import requests import cv2 import os ap = argparse.ArgumentParser() ap.add\_argument("-q", "--query", required=True, help="search query to search Bing Image API for") ap.add\_argument("-o", "--output", required=True, help="path to output directory of images") args = vars(ap.parse\_args())  API\_KEY = "d8982f9e69a4437fa6e10715d1ed691d"  MAX\_RESULTS = 500  GROUP\_SIZE = 50  URL = "https://api.cognitive.microsoft.com/bing/v7.0/images/search" EXCEPTIONS = set([IOError, FileNotFoundError, exceptions.RequestException, exceptions.HTTPError, |
| exceptions.ConnectionError, exceptions.Timeout]) term = args["query"] headers = {"Ocp-Apim-Subscription-Key" : API\_KEY} params = {"q": term, "offset": 0, "count": GROUP\_SIZE} print("[INFO] searching Bing API for '{}'".format(term)) search = requests.get(URL, headers=headers, params=params) search.raise\_for\_status() results = search.json() estNumResults = min(results["totalEstimatedMatches"], MAX\_RESULTS) print("[INFO] {} total results for '{}'".format(estNumResults, term)) total = 0 for offset in range(0, estNumResults, GROUP\_SIZE): print("[INFO] making request for group {}-{} of {}...".format( offset, offset + GROUP\_SIZE, estNumResults)) params["offset"] = offset search = requests.get(URL, headers=headers, params=params) search.raise\_for\_status() results = search.json() print("[INFO] saving images for group {}-{} of {}...".format( offset, offset + GROUP\_SIZE, estNumResults)) for v in results["value"]: try:  print("[INFO] fetching: {}".format(v["contentUrl"])) r = requests.get(v["contentUrl"], timeout=30) ext = v["contentUrl"][v["contentUrl"].rfind("."):] p = os.path.sep.join([args["output"], "{}{}".format( str(total).zfill(8), ext)]) f = open(p, "wb")  f.write(r.content)  f.close() except Exception as e:  if type(e) in EXCEPTIONS:  print("[INFO] skipping: {}".format(v["contentUrl"])) continue image = cv2.imread(p) if image is None: print("[INFO] deleting: {}".format(p)) os.remove(p) continue total += 1 |

**SEQUENCE DIAGRAM:**

Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when.

**Sequence Diagrams captures:**

* the interaction that takes place in a collaboration that either realizes a use case or an operation (instance diagrams or generic diagrams)
* high-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams)

**Purpose of Sequence Diagram**

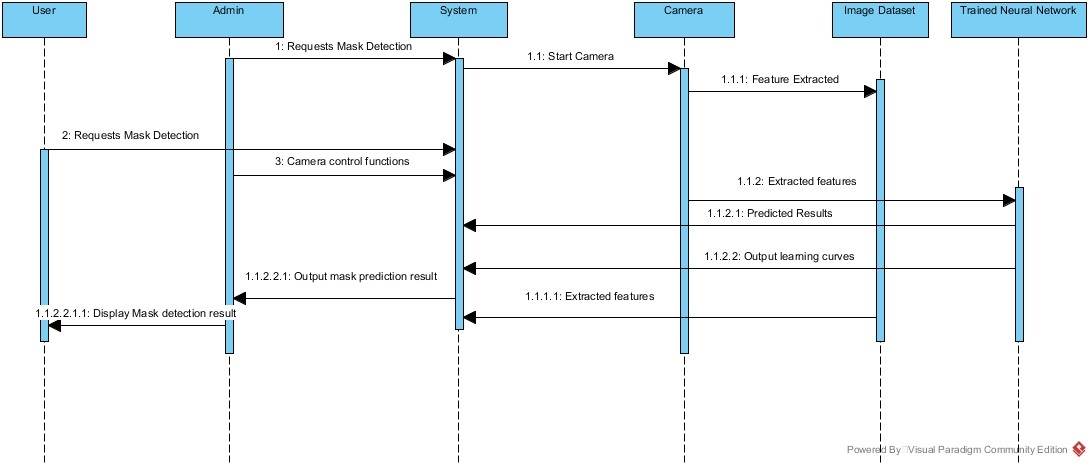
* Model high-level interaction between active objects in a system
* Model the interaction between object instances within a collaboration that realizes a use case
* Model the interaction between objects within a collaboration that realizes an operation
* Either model generic interactions (showing all possible paths through the interaction) or specific instances of a interaction (showing just one path through the interaction)

**Sequence Diagram at a Glance**

Sequence Diagrams show elements as they interact over time and they are organized according to object (horizontally) and time (vertically)

Like all other diagrams, sequence diagrams may also contain notes and constrains.

#### SEQUENCE DIAGRAM FOR FACE MASK DETECTION SYSTEM



**USE CASE DIAGRAM:**

A use case diagram is the primary form of system/software requirements for a new software program underdeveloped. Use cases specify the expected behavior (what), and not the exact method of making it happen (how).

A key concept of use case modelling is that it helps us design a system from the end user's perspective. It is an effective technique for communicating system behavior in the user's terms by specifying all externally visible system behavior.

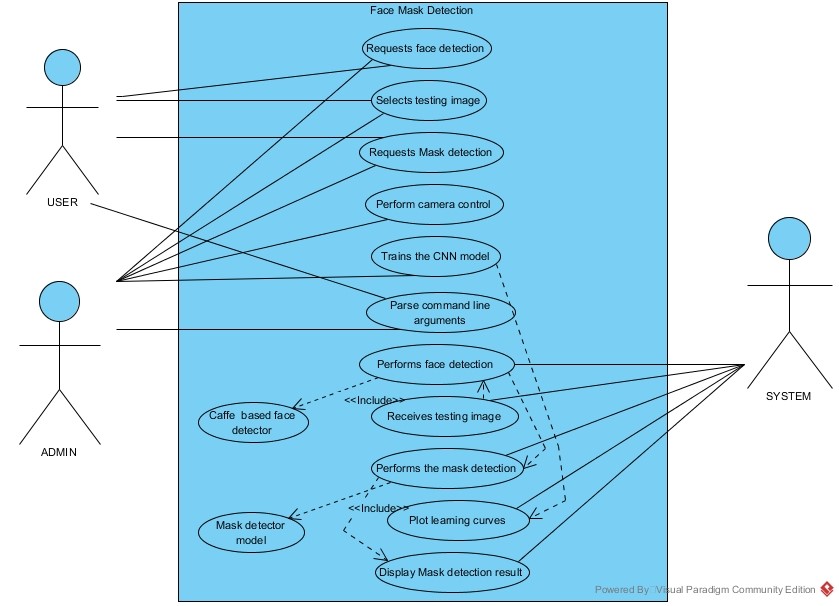
**Use Case Diagram captures:**

* It only summarizes some of the relationships between use cases, actors, and systems.
* It does not show the order in which steps are performed to achieve the goals of each use case.

**Purpose of Use Case Diagram:**

* Specify the context of a system
* Capture the requirements of a system
* Validate a systems architecture
* Drive implementation and generate test cases
* Developed by analysts together with domain experts

#### USE CASE DIAGRAM FOR FACE MASK DETECTION SYSTEM



**ACTIVITY DIAGRAM:**

Activity Diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another, in particular, use cases where activities may overlap and require coordination.

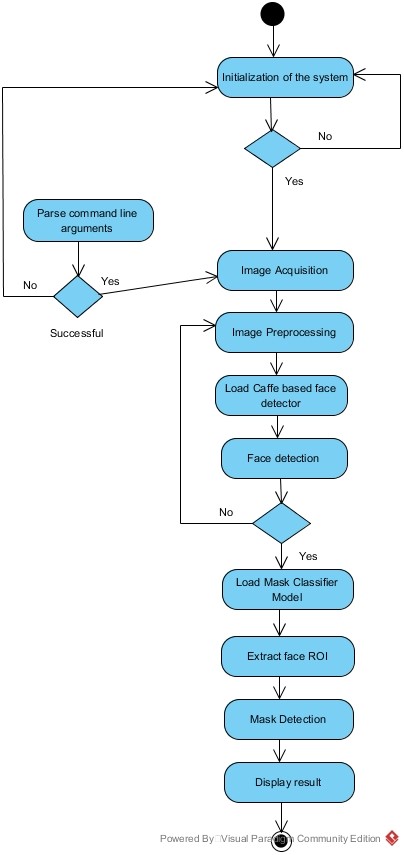
**Purpose of Activity Diagram:**

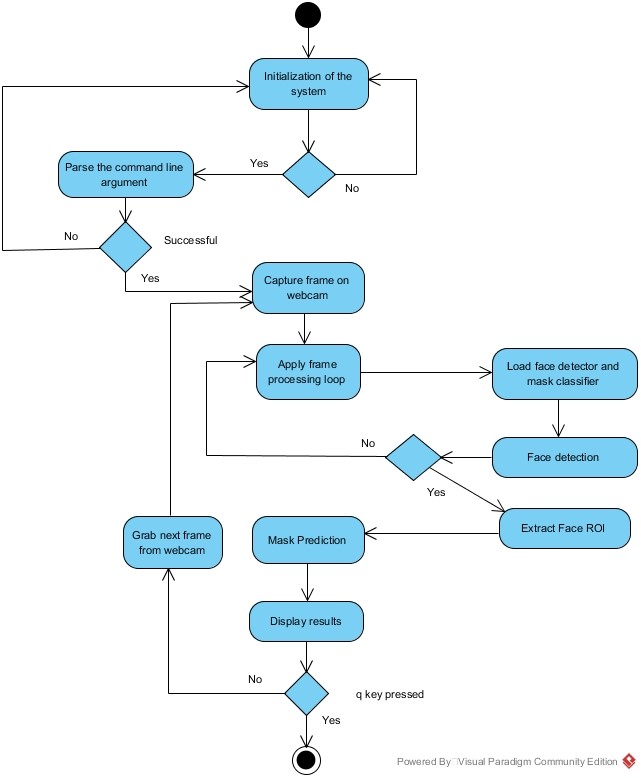
* Draw the activity flow of a system.
* Describe the sequence from one activity to another.
* Describe the parallel, branched and concurrent flow of the system.

**Activity diagram can be used for:**

* Modeling work flow by using activities.
* Modeling business requirements.
* High level understanding of the system's functionalities.
* Investigating business requirements at a later stage.

#### ACTIVITY DIAGRAM OF FACE MASK DETECTION SYSTEM





**CLASS DIAGRAM:**

Class diagrams are the main building blocks of every object-oriented methods. The class diagram can be used to show the classes, relationships, interface, association, and collaboration.

Class diagrams are a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

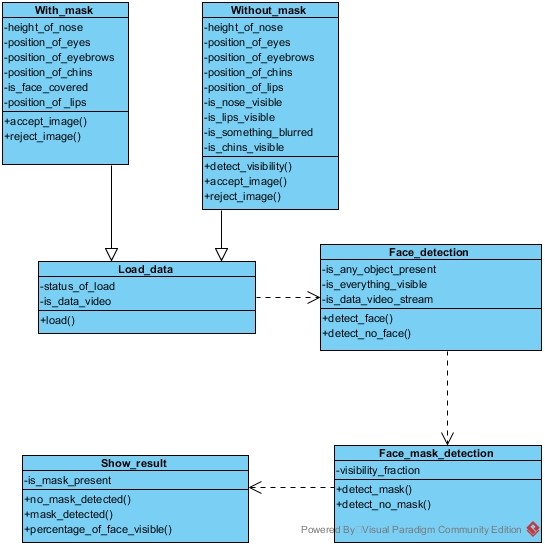
**Purpose of Class Diagram:**

* Shows static structure of classifiers in a system
* Diagram provides a basic notation for other structure diagrams prescribed by UML
* Helpful for developers and other team members too
* Business Analysts can use class diagrams to model systems from a business perspective

**A UML class diagram is made up of:**

* A set of classes and
* A set of relationships between classes

#### CLASS DIAGRAM OF FACE MASK DETECTION SYSTEM



## CHAPTER-3

**DESIGN, IMPLEMENTATION AND RESULT**

***Hardware Requirements:***

Processor: 1.6 GHz Intel Core i5/Pentium IV 2.4 GHz

RAM: at least 2 GB RAM

Speed: 500 MHz

Hard Disk: 80 GB minimum

Accessories: A high quality wireless/webcam camera, LCD/LED Monitor, Keyboard, Mouse

***Software Requirements:***

Operating System: • Windows 8 or later

* Mac OS 10.13.6 or later (preferable)
* Ubuntu 16.04 or later (64-bit) (preferable)
* PyCharm/ VSCode editor/ TensorFlow GPU (optional)

Programming Language:

* Python (3.7.6)

Open Libraries/Modules used:

* OpenCV (Intel's Computer Vision Open Source Library) (4.2.0)
* TensorFlow (1.14.0)
* Keras (2.3.1) (TensorFlow backend)
* sklearn (0.22.1)
* imutils (0.5.3)
* numpy (1.18.2)
* matplotlib (3.1.3)
* argparse (1.1)

**FACE MASK DETECTION DATASET PRE-PROCESSING:**

The custom dataset used in this project consists of real images of faces **with and without protective face masks**.

This dataset consists of **3835 images** belonging to two classes:

* ***with\_mask:* 1916 images**
* ***without\_mask:* 1919 images**

More than **5000 images** were collected out of which a handful were rejected for being blurred, morphed or found not fruitful. They were deleted and hence the process of **data pruning** was done.

The dataset will be split into 80% training and 20% testing data with the help of sklearn library. The training set has roughly **3068 images** and the testing set has roughly **767 images**.

**ALGORITHMS:**

1. **Feature Extraction using ConvNets**

Traditional machine learning approach uses feature extraction for images using Global feature descriptors such as Local Binary Patterns (LBP), Histogram of Oriented Gradients (HoG), Color Histograms etc. or Local descriptors such as SIFT, SURF, ORB etc. These are hand-crafted features that requires domain level expertise.

But here comes Convolutional Neural Networks (CNN). Instead of using hand-crafted features, Deep Neural Nets automatically learns these features from images in a hierarchical fashion. Lower layers learn low-level features such as Corners, Edges whereas middle layers learn color, shape etc. and higher layers learn high-level features representing the object in the image.

Instead of making a CNN as a model to classify images, we can use it as a Feature Extractor by taking the activations available before the last fully connected layer of the network

(i.e. *before* the final softmax classifier). These activations will be acting as the feature vector for a machine learning model (classifier) which further learns to classify it. This type of approach is well suited for Image Classification problems, where instead of training a CNN from scratch (which is time-consuming and tedious), a pre-trained CNN could be used as a Feature Extractor - **Transfer Learning**.

1. **Transfer Learning Algorithm**

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is the idea of overcoming the isolated learning paradigm and utilizing the knowledge acquired for one task to solve related ones.

Traditional learning is isolated and occurs purely based on specific tasks, datasets and training separate isolated models on them. No knowledge is retained which can be transferred from one model to another. In transfer learning, you can leverage knowledge (features, weights etc.) from previously trained models for training newer models and even tackle problems like having less data for the newer task!

Learning is not an easy process, not for humans and not for machines either. It is a heavy-duty, resource-consuming and time-consuming process and hence it was important to devise a method that would prevent a model from forgetting the learning curve that it attained from a specific dataset and also lets it learn more from new and different datasets.

Transfer learning is simply the process of using a pre-trained model that has been trained on a dataset for training and predicting on a new given dataset.

“A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task such as the **ImageNet**.”

**Applications of Transfer Learning:**

1. **Transfer learning for NLP:** Textual data presents all sorts of challenges when it comes to ML and deep learning. These are usually transformed or vectorized using different techniques. Embedding, such as Word2vec and FastText, have been prepared using different training datasets. These are utilized in different tasks, such as sentiment analysis and document classification, by transferring the knowledge from the source tasks.

1. **Transfer learning for Audio/Speech:** Similar to domains like NLP and Computer Vision, deep learning has been successfully used for tasks based on audio data. For instance, Automatic Speech Recognition (ASR) models developed for English have been successfully used to improve speech recognition performance for other languages, such as German.

1. **Transfer learning for Computer Vision:** Deep learning has been quite successfully utilized for various computer vision tasks, such as object recognition and identification, using different CNN architectures.

**iii. Transfer Learning with a pre-trained ConvNet**

We can have two ways to customize a pre-trained model:

1. **Feature Extraction:** Use the representations learned by a previous network to extract meaningful features from new samples. We simply add a new classifier, which will be trained from scratch, on top of the pre-trained model so that we can repurpose the feature maps learned previously for the dataset.

We do not need to (re)train the entire model. The base convolutional network already contains features that are generically useful for classifying pictures. However, the final, classification part of the pre-trained model is specific to the original classification task, and subsequently specific to the set of classes on which the model was trained.

1. **Fine-Tuning:** Unfreeze a few of the top layers of a frozen model base and jointly train both the newly-added classifier layers and the last layers of the base model. This allows us to "fine-tune" the higher-order feature representations in the base model in order to make them more relevant for the specific task.

We will create the base model from the **MobileNetV2** model developed at Google. This is pretrained on the **ImageNet dataset**, a large dataset consisting of 1.4M images and 1000 classes. ImageNet is a research training dataset with a wide variety of categories. This base of knowledge will help us classify “Mask” and “No Mask” from our specific dataset.

It is important to freeze the convolutional base before we compile and train the model. Freezing (*by setting layer.trainable = False*) prevents the weights in a given layer from being updated during training. MobileNetV2 has many layers, so setting the entire model's trainable flag to False will freeze all the layers.

**METHODOLOGIES:**

* **OpenCV’s “deep neural networks” (dnn) module:** OpenCV (3.3 or later) comprises of the highly efficient dnn module supported by a number of deep learning frameworks such as Caffe, TensorFlow, and Torch/PyTorch. This module has a more accurate Caffe-based face detector. In this project, we will be training our deep learning model using Caffe, hence we need the following files:

o The **.prototxt** file(s) which will define the *model architecture* (i.e., the layers) o The **.caffemodel** file which will contain the *weights* for the actual layers

* **OpenCV’s face detector based on the Single Shot Multibox Detector (SSD) framework** **combined with MobileNetV2 architecture:** In order to obtain the bounding box *(x, y)* coordinates for an object (mask in this case) in an image we need to apply object detection.
  + - SSDs, originally developed by Google, are a balance between R-CNNs and YOLO methods of object detection. The algorithm is more straightforward than Faster RCNNs.
    - Network architectures such as VGG or Resnet are unsuitable for resource constrained devices due to their sheer size and resulting number of computations. Instead, we use MobileNets. They are designed for resource constrained devices such as your smartphone.
    - Hence, if we combine both the MobileNet architecture and the Single Shot Detector (SSD) framework, we will have a fast, efficient deep learning-based method for object detection.

* **OpenCV’s blobFromImage and blobFromImages to facilitate image pre-processing:** cv2.dnn.blobFromImage and cv2.dnn.blobFromImages functions of OpenCV’s dnn module facilitates image pre-processing for deep learning classification. These two functions perform:
  + Mean subtraction - Mean subtraction is used to help tackle illumination changes in the input images in our dataset. For example, the mean values for the ImageNet training set are **R=103.93, G=116.77, and B=123.68** o Scaling - The scaling factor aids in normalization.
  + And optionally channel swapping

A ‘blob’ is just a collection of image(s) with the same spatial dimensions (width and height), same depth (number of channels), that have to be pre-processed in the same manner.

**[blobFromImage]** creates 4-dimensional blob from image. Optionally resizes and crops image from center, subtracts mean values, scales values by scale-factor, swaps Blue and Red channels.

*blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size, mean, swapRB=True)*

Each parameter is described below:

* **image:** This is the input image we want to pre-process before passing it through our deep neural network for classification.

* **scalefactor:** After we perform mean subtraction we can optionally scale our images by some factor. This value defaults to `1.0` (i.e., no scaling) but we can supply another value as well. It’s also important to note that scalefactor should be  as we’re actually multiplying the input channels (after mean subtraction) by scalefactor.

* **size:** Here we supply the spatial size that the Convolutional Neural Network expects. For most current state-of-the-art neural networks this is either *224×224*, *227×227*, or *299×299*. We will be using *224×224.*

* **mean:** These are our mean subtraction values. They can be a 3-tuple of the RGB means or they can be a single value in which case the supplied value is subtracted from every channel of the image.
* **swapRB:** OpenCV assumes images are in BGR channel order; however, the `mean` value assumes we are using RGB order. To resolve this discrepancy, we can swap the R and B channels in image by setting this value to `True`. By default, OpenCV performs this channel swapping for us.

The cv2.dnn.blobFromImage function returns a blob which is our input image after mean subtraction, normalizing, and channel swapping.

The **cv2.dnn.blobFromImages** function is exactly the same:

*blob = cv2.dnn.blobFromImages(images, scalefactor=1.0, size, mean,*

*swapRB=True)*

*The only exception is that we can pass in multiple images, enabling us to batch process a set of images.*

§ **Keras ImageDataGenerator:** Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches).

Keras ImageDataGenerator class works by:

* Accepting a batch of images used for training.
* Taking this batch and applying a series of random transformations to each image in the batch (including random rotation, resizing, shearing, etc.).
* **Replacing the original batch with the new, randomly transformed batch.**
* Training the CNN on this randomly transformed batch (i.e., the original data **itself is not** used for training).

### Model Summary

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

===============================================================

===================================

input\_1 (InputLayer) [(None, 224, 224, 3) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Conv1\_pad (ZeroPadding2D) (None, 225, 225, 3) 0 input\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Conv1 (Conv2D) (None, 112, 112, 32) 864 Conv1\_pad[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

bn\_Conv1 (BatchNormalization) (None, 112, 112, 32) 128 Conv1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Conv1\_relu (ReLU) (None, 112, 112, 32) 0 bn\_Conv1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

expanded\_conv\_depthwise (Depthw (None, 112, 112, 32) 288 Conv1\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

expanded\_conv\_depthwise\_BN (Bat (None, 112, 112, 32) 128 expanded\_conv\_depthwise[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

expanded\_conv\_depthwise\_relu (R (None, 112, 112, 32) 0 expanded\_conv\_depthwise\_BN[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

expanded\_conv\_project (Conv2D) (None, 112, 112, 16) 512 expanded\_conv\_depthwise\_relu[0][0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

expanded\_conv\_project\_BN (Batch (None, 112, 112, 16) 64 expanded\_conv\_project[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

block\_1\_expand (Conv2D) (None, 112, 112, 96) 1536 expanded\_conv\_project\_BN[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

block\_1\_expand\_BN (BatchNormali (None, 112, 112, 96) 384 block\_1\_expand[0][0]

.

.

.

.

.

block\_16\_expand\_relu (ReLU) (None, 7, 7, 960) 0 block\_16\_expand\_BN[0][0] \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

block\_16\_depthwise (DepthwiseCo (None, 7, 7, 960) 8640 block\_16\_expand\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

block\_16\_depthwise\_BN (BatchNor (None, 7, 7, 960) 3840 block\_16\_depthwise[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

block\_16\_depthwise\_relu (ReLU) (None, 7, 7, 960) 0 block\_16\_depthwise\_BN[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

block\_16\_project (Conv2D) (None, 7, 7, 320) 307200 block\_16\_depthwise\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

block\_16\_project\_BN (BatchNorma (None, 7, 7, 320) 1280 block\_16\_project[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Conv\_1 (Conv2D) (None, 7, 7, 1280) 409600 block\_16\_project\_BN[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Conv\_1\_bn (BatchNormalization) (None, 7, 7, 1280) 5120 Conv\_1[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

out\_relu (ReLU) (None, 7, 7, 1280) 0 Conv\_1\_bn[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

average\_pooling2d (AveragePooli (None, 1, 1, 1280) 0 out\_relu[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 1280) 0 average\_pooling2d[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 128) 163968 flatten[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (Dropout) (None, 128) 0 dense[0][0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 2) 258 dropout[0][0]

===============================================================

===================================

Total params: 2,422,210

Trainable params: 2,388,098

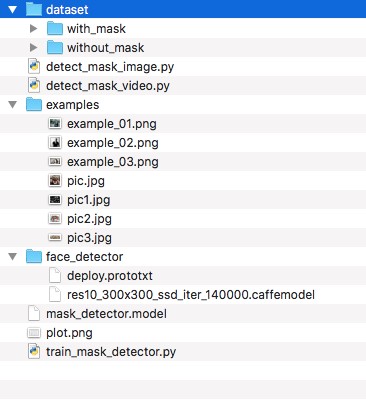
Non-trainable params: 34,112

This is the output of model.summary()

len(model.layers) = 160

**IMPLEMENTATION, EXPLANATION AND OUTPUT:**

**Directory structure**



**Step 1: Training the model with Keras and TensorFlow**

In this step, we will train our face mask detector model using Keras, TensorFlow, and Deep Learning.

v The following file accepts our input dataset and creates our custom mask detector model. The explanation of the code is also given partwise.

**File: train\_mask\_detector.py**

*# import the necessary packages*

**from** TensorFlow.keras.preprocessing.image **import** ImageDataGenerator **from** TensorFlow.keras.applications **import** MobileNetV2 **from** TensorFlow.keras.layers **import** AveragePooling2D **from** TensorFlow.keras.layers **import** Dropout **from** TensorFlow.keras.layers **import** Flatten

**from** TensorFlow.keras.layers **import** Dense **from** TensorFlow.keras.layers **import** Input **from** TensorFlow.keras.models **import** Model **from** TensorFlow.keras.optimizers **import** Adam

**from** TensorFlow.keras.applications.mobilenet\_v2 **import** preprocess\_input **from** TensorFlow.keras.preprocessing.image **import** img\_to\_array **from** TensorFlow.keras.preprocessing.image **import** load\_img **from** TensorFlow.keras.utils **import** to\_categorical **from** sklearn.preprocessing **import** LabelBinarizer **from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.metrics **import** classification\_report **from** imutils **import** paths **import** matplotlib.pyplot **as** plt **import** numpy **as** np **import** argparse **import** os

**Explanation:** We perform the following imports to allow the below mentioned tasks:

* Keras’ ImageDataGenerator class to perform data augmentation.
* Loading the MobileNetV2 classifier for fine-tuning this model with pre-trained ImageNet weights.
* Building a new fully-connected (FC) head
* Preprocessing and loading image data
* scikit-learn (sklearn) for binarizing class labels, segmenting our dataset, and printing a classification report.
* imutils paths implementation will help us to find and list images in our dataset.
* And we’ll use matplotlib to plot our training curves, argparse module to write user-friendly commandline interfaces.

--------------------------------------------------------------------------------------

*# 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())

**Explanation:** We parse the following command line arguments to launch our script via terminal. It includes:

* -d/--dataset: The path to the input dataset of faces with and without masks
* -p/--plot: The path to the output training history plot, which will be generated using matplotlib
* -m/--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

**Explanation:** The learning hyperparameter constants are declared including:

* Initial learning rate (INIT\_LR)
* Number of training epochs (EPOCHS)
* Batch Size (BS)

--------------------------------------------------------------------------------------

*# 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)

**Explanation:** In this section of code we do the following tasks:

* We get hold of all the ‘image path’ in the dataset.
* Initialize the data[] and labels[] list.
* We loop over the imagepaths to extract the class label, load and preprocess the input image into 224px\*224px, also convert it to array format and scale the pixel intensities in the range of [-1,1].
* We convert our training data into 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"**)

**Explanation:** In this section of code, we do the following tasks:

* ‘one-hot’ encoding of the class labels: One-hot encoding is essentially the representation of categorical variables as binary vectors. These categorical values are first mapped to integer values. Each integer value is then represented as a binary vector that is all 0s (except the index of the integer which is marked as 1).

* dataset partitioning: Using train\_test\_split() method of scikit-learn we partition our dataset into 80% for training and the rest 20% for testing purposes.

* prepare for data augmentation:Now, we apply on the fly data mutation methods to our image dataset in order to improve generalization. The Keras ImageDataGenerator class accepts the original data, randomly transforms it (via rotation, zoom, shear, shift and flip), and returns only the new, transformed data.

--------------------------------------------------------------------------------------

*# 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**

**Explanation:** In this code block, we will prepare the MobileNetV2 architecture for fine-tuning by a three-step procedure:

* Load MobileNet with pre-trained ImageNet weights, leaving off the head of network.
* Replace the fully connected old head (i.e., where the actual class label predictions are made) with new FC head.
* Freezing the base layers, so that they won’t be updated during the process of backpropagation, and only tuning the new head layer weights.

--------------------------------------------------------------------------------------

*# 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)

**Explanation:** In this section of code, we are ready to compile and train the face mask detector network.

* The model is compiled using Adam optimizer, time-based learning rate scheduler via ‘decay’ parameter and ‘binary cross-entropy’ (because 2 classes classifier).
* We then train the head of the network using the ‘aug’ object which provides batches of augmented data.

--------------------------------------------------------------------------------------

*# 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"**)

**Explanation:** In this code-section:

* We make predictions on the test set, grabbing the highest probability class label indices.
* Then, we print a classification report in the terminal for inspection.
* We then serialize our face mask classification model to disk.

--------------------------------------------------------------------------------------

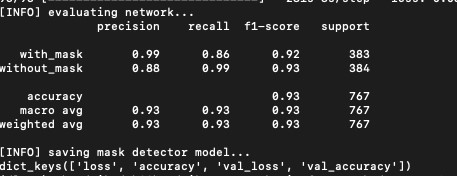
*# plot the training loss and accuracy* N = EPOCHS plt.style.use(**"ggplot"**) plt.figure()

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\_accuracy"**) plt.plot(np.arange(0, N), H.history[**"val\_accuracy"**], label=**"val\_accuracy"**) plt.title(**"Training Loss and Accuracy"**) plt.xlabel(**"Epoch #"**) plt.ylabel(**"Loss/Accuracy"**) plt.legend(loc=**"lower left"**) plt.savefig(args[**"plot"**])

**Explanation:** In this code block, we plot our learning curves of the training and validation accuracy/loss curves and save the plot to disk.

--------------------------------------------------------------------------------------

**Step 2: Execute the following command in Terminal after going into the project folder to train the model.** *$ python train\_mask\_detector.py --dataset dataset*



We now obtain our training history containing loss/accuracy curves as a separate ‘plot.png’. We can refer that the curves demonstrate high accuracy with little signs of overfitting.

We can now proceed to apply our trained model to detect face masks in static images and realtime video streams appropriately outside our input dataset.

**Step 3: Implementing our COVID-19 face mask detector model for static images with OpenCV**

v The following file loads our model and hence detects faces and classifies them as **‘Mask’** or **‘No Mask’** in static images given as an input.

**File: detect\_mask\_image.py**

**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

**Explanation:** In this code block, we require the TensorFlow/Keras imports for loading the model and pre-processing the input image. We import OpenCV for image display and modifications.

--------------------------------------------------------------------------------------

*# 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())

**Explanation:** We parse the following command line arguments to launch our script via terminal. It includes:

* --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
* --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"**])

**Explanation:** In this code block, we load both the face detector and face mask classifier models.

--------------------------------------------------------------------------------------

*# 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()

**Explanation:** In this code section, we load and pre-process an input image.

* We make a copy of the input image and grab the frame dimensions for scaling and display.
* Image pre-processing is done by OpenCV’s ‘blobFromImage’ method. We resize the image to 300px\*300px and mean subtraction is performed.
* Then we perform face detection to locate all the faces in the input image provided.

--------------------------------------------------------------------------------------

*# 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))

**Explanation:** In this section of code, we do the following:

* Loop over detections and extract the confidence parameter to measure against the confidence threshold.
* Compute bounding box value for a particular face and ensure that the box dimension falls within the boundary of the image.

--------------------------------------------------------------------------------------

*# 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]

**Explanation:** In this code block, we run the face ROI (Region of Interest) through our MaskNet model in the following ways:

* Extract the face ROI with the help of NumPy slicing.
* Pre-process the face ROI the same way we did during training.
* Perform prediction for “Mask” or “No 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)

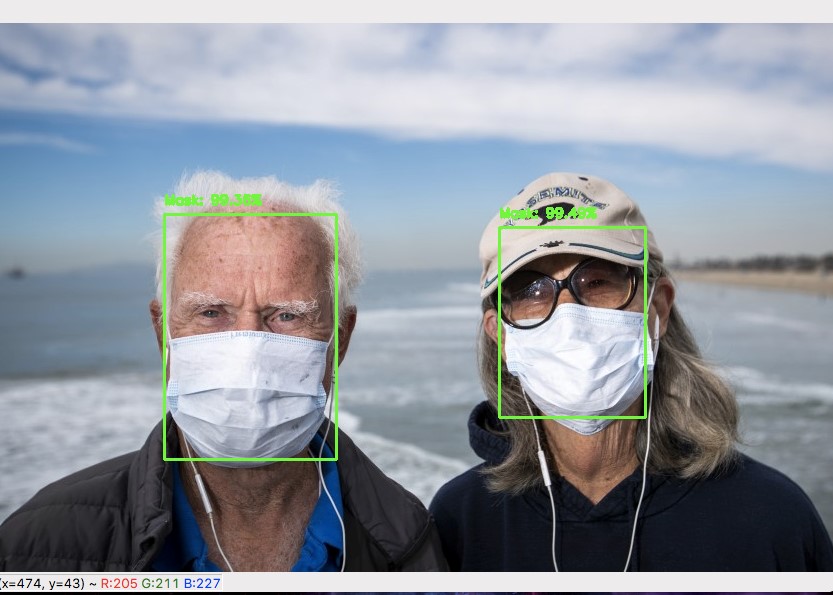
*# show the output image* cv2.imshow(**"Output"**, image) cv2.waitKey(0)

**Explanation:** In this block, we display our result.

* We determine the class label of “Mask” or “No Mask” based on the probabilities returned by the mask detector model.
* We then assign the color “GREEN” for “with\_mask” and “RED” for “without\_mask”.
* Then using OpenCV functions we draw a bounding box rectangle along and display the class as well as probability.
* Then finally image output is shown in an output window.

--------------------------------------------------------------------------------------

**Step 4: Execute the following command in Terminal to detect masks in static images present in ‘images’ folder.** *$ python detect\_mask\_image.py --image images/pic1.jpeg*



*$ python detect\_mask\_image.py --image images/pic2.jpg*



**Step 5: Implementing our COVID-19 face mask detector model in real-time video streams with OpenCV**

v The following file loads our model and hence detects faces and classifies them as **‘Mask’** or **‘No Mask’** in real-time video streams.

**File: detect\_mask\_video.py**

*# import the necessary packages*

**from** TensorFlow.keras.applications.mobilenet\_v2 **import** preprocess\_input **from** TensorFlow.keras.preprocessing.image **import** img\_to\_array **from** TensorFlow.keras.models **import** load\_model **from** imutils.video **import** VideoStream **import** numpy **as** np **import** argparse **import** imutils **import** time **import** cv2 **import** os

**Explanation:** In this code block, we allow processing of every frame of our webcam stream.

* So, we import the VideoStream class along with the time module.
* The imutils module helps for its aspect aware resizing method.

--------------------------------------------------------------------------------------

**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 = []

**Explanation:** In this code block, we apply our frame processing loop in the following way:

* The detect\_and\_predict\_mask function detects faces and then applies the face mask classifier to each face ROI. • This function accepts three parameters:
  + *frame*: A frame from our stream
  + *faceNet*: The model used to detect where in the image faces are o *maskNet*: Our COVID-19 face mask classifier model
* We then construct a blob, detect faces and initialize the lists including faces (i.e., ROIs), locs (the face locations), and preds (the list of mask/no mask predictions).

--------------------------------------------------------------------------------------

*# 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))

**Explanation:** In this code section, we do the following:

* we filter out the weak detections by extracting the confidence and measuring it against the confidence threshold.
* We also compute the bounding boxes and ensure the dimensions are within the frame.

--------------------------------------------------------------------------------------

*# extract the face ROI, convert it from BGR to RGB channel # ordering, resize it to 224x224, and preprocess it* face = frame[startY:endY, startX:endX] face = cv2.cvtColor(face, cv2.COLOR\_BGR2RGB) face = cv2.resize(face, (224, 224)) face = img\_to\_array(face) face = preprocess\_input(face) face = np.expand\_dims(face, axis=0)

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

*# lists* faces.append(face)

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

**Explanation:** In this code block, we do the following:

• extract the face ROI and pre-process it • append the face ROI and bounding boxes to their lists.

--------------------------------------------------------------------------------------

*# only make a predictions if at least one face was detected*

**if** len(faces) > 0:

*# for faster inference we'll make batch predictions on \*all\**

*# faces at the same time rather than one-by-one predictions*

*# in the above `for` loop* preds = maskNet.predict(faces)

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

*# locations* **return** (locs, preds)

**Explanation:** Here we’ll run the faces through the mask detector.

* We perform the inference on entire batch of faces in the frame so that our pipeline is faster.
* At last, we return the face bounding box locations and corresponding mask/non-mask predictions to the caller function.

--------------------------------------------------------------------------------------

*# 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())

**Explanation:** We parse the following command line arguments to launch our script via terminal. It includes:

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

**Explanation:** In this code block, we have initialized the following:

* Face detector
* Face Mask detector
* Webcam video 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)

**Explanation:** In this code block, we loop over the frames and resize it and apply our mask prediction function.

--------------------------------------------------------------------------------------

*# 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)

**Explanation:** In this code section, we post-process the face mask detection results:

* Unpack a face bounding box and with mask/without mask prediction
* Determine the label and color (GREEN and RED)
* Annotate the label and face bounding box

--------------------------------------------------------------------------------------

*# 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()

**Explanation:** Finally, we capture the key presses once the frame is displayed. If the user presses the key ‘q’, the system breaks out of the loop and stops.

--------------------------------------------------------------------------------------

**Step 6: Execute the following command in Terminal to detect masks in real-time video streams using webcam.**

*$ python detect\_mask\_video.py*

## CONCLUSION

An accurate and efficient face mask detection system has been developed which achieves comparable metrics with the existing state-of-the-art system. This project uses recent techniques in the field of computer vision and deep learning. Custom dataset was made from scratch using Bing Search API, Kaggle datasets and RMFD dataset, and the evaluation of the model on test dataset was found consistent. The system correctly detected the presence of face masks on human faces that it detected in static images as well as real-time video streams.

To create our face mask detector, we trained a two-class model with images of people ***wearing masks*** and ***not wearing masks****.*

We then fine-tuned our model using MobileNetV2 on our ***mask/no mask*** dataset and obtained an image classifier that was **93% accurate.**

We then took this face mask classifier and applied it to both ***images*** and ***real-time video streams*** by:

1. Detecting faces in the images/video

1. Extracting each individual face ROI

1. Applying our face mask classifier

Our face mask detector is accurate, and since we used the MobileNetV2 architecture, it’s also computationally efficient and thus making it easier to deploy the model to embedded systems (Raspberry Pi, Google Coral, etc.).

This system can therefore be used in real-time applications which require face-mask detection for safety purposes due to the outbreak of Covid-19. This project can be integrated with embedded systems for application in airports, railway stations, offices, schools, and public places to ensure that public safety guidelines are followed.

### SCOPE FOR IMPROVEMENT AND FUTURE WORK

Our current method of detecting whether a person is wearing a mask or not is a two-step process:

* **Step 1:** Perform face detection

* **Step 2:** Apply our face mask detector to each face

The problem with this approach is that a face mask, by definition, obscures part of the face. If enough of the face is obscured, the face cannot be detected, and therefore, the face mask detector will not be applied.

To circumvent that issue, we should train a two-class object detector that consists of a **‘with\_mask’** class and **‘without\_mask’** class. Combining an object detector with a dedicated **‘with\_mask’** class will allow improvement of the model in two respects.

First, the object detector will be able to naturally detect people wearing masks that otherwise would have been impossible for the face detector to detect due to too much of the face being obscured.

Secondly, this approach reduces our computer vision pipeline to a single step — rather than applying face detection and then our face mask detector model, all we need to do is apply the object detector to give us bounding boxes for people both **‘with\_mask’** and **‘without\_mask’** in a single forward pass of the network.

Not only is such a method more computationally efficient, it’s also more “elegant” and end-toend.

Also, we can develop an Android and/or web application for the same in future.

## REFERENCES

1. https://keras.io/api/applications/mobilenet/#mobilenetv2-function

1. https://arxiv.org/abs/1704.04861

1. https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/

1. https://www.pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-imagedataset/

1. https://gogul.dev/software/flower-recognition-deep-learning

1. https://www.tensorflow.org/tutorials/images/transfer\_learning

1. https://towardsdatascience.com/detecting-faces-with-python-and-opencv-face-detectionneural-network-f72890ae531c

1. https://www.indulgexpress.com/msociety/2020/may/04/hyderabad-company-launches-aibased-surveillance-tech-to-detect-face-mask-and-social-distancing-viol-24656.html

1. https://www.engadget.com/uber-face-mask-detection-technology-184137514.html

1. https://www.leewayhertz.com/face-mask-detection-system/

1. https://www.thehansindia.com/telangana/hyderabad-cameras-to-detect-face-masks-testtemp-soon-619505

1. https://www.kaggle.com/chandrikadeb/face-mask-dataset

1. https://www.pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-dataaugmentation/