A PROJECT REPORT

ON

Face Mask Detection

By

PATEL YASH K. (CE-127) (19CEUON068)

PATEL LUV P. (CE-124) (19CEUON069)

B.Tech CE Semester-VI

Subject: SYSTEM DESIGN PRACTICE

Guided by: Dr. Malay S. Bhatt

Associate Professor

Dept. of Comp. Engg.

Faculty of Technology

Department of Computer Engineering

Dharmsinh Desai University

Faculty of Technology Department of Computer Engineering

Dharmsinh Desai University

CERTIFICATE

This is to certify that the practical / term work carried out in the subject of

SYSTEM DESIGN PRACTICE and recorded in this journal is

the bonafide work of

PATEL YASH K. (CE-127) (19CEUON068)

PATEL LUV P. (CE-124) (19CEUON069)

of B.Tech semester VI in the branch of Computer Engineering

during the academic year 2021-2022.

Dr. Malay S. Bhatt	Dr. C. K. Bhensdadia,
Associate Professor,	Head,
Dept. of Computer Engg.,	Dept. of Computer Engg.,
Faculty of Technology	Faculty of Technology
Dharmsinh Desai University, Nadiad	Dharmsinh Desai University, Nadiad

Table of Contents

Introduction

Description

Face Mask detection refers to detect whether a person wearing a mask or not and what is the location of the face. The problem is closely related to general object detection to detect the classes of objects and face detection is to detect a particular class of objects. Applications of face detection can be found in many areas, such as surveillance Purpose etc.

The Basic Steps of the system are :

1. Preprocessing
2. Model Training
3. Object Detection

Scope

Video Captioning which aims to automatically detect face mask of a person and display result accordingly. This will help to prevent the spread of the virus in this pandemic and it will be used variety for public gathering areas.

Example : At the entrance of a shop.

Hardware Requirements

- Operating System : Windows 8 or Higher
- Processor : Intel i3 or higher
- Memory : 4GB or more
- Processor : 64 bit processor
- Working web cam

Software Requirement Specification

1. Functional Requirements

1.1 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.

1.2 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 capture.
- R8. The results must be viewed by showing the probability along with the output of 'Mask Detected' or 'No Mask Detected'.

2. Non-Functional Requirements

2.1 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.

2.2 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.

2.3 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.

Analysis and Design

1. Fundamental Steps to Perform for Face Mask Detection

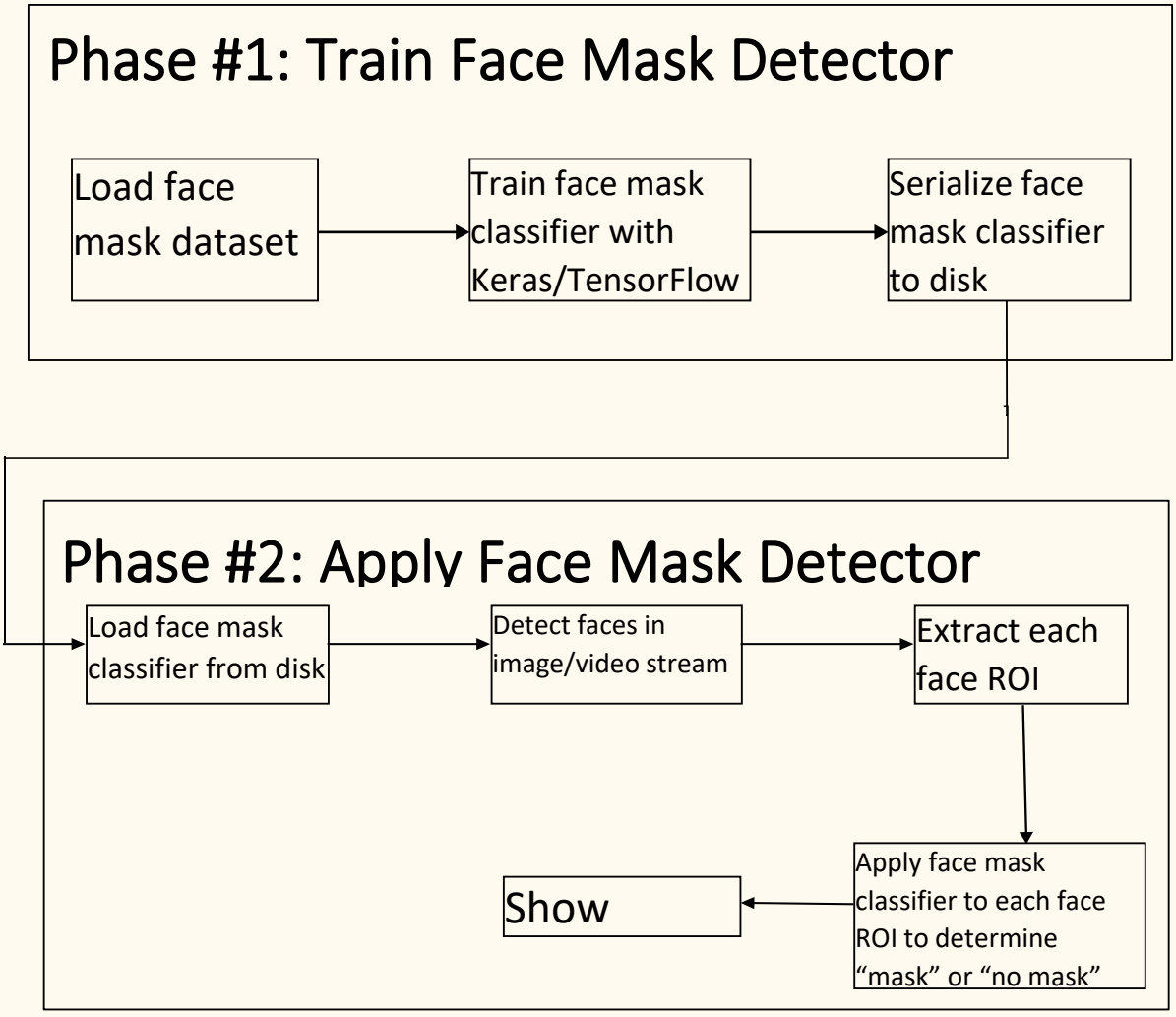


Figure 1: Phases and individual steps for building a COVID-19 face mask detector with computer vision and deep learning using Python, OpenCV, and TensorFlow/keras

1. Training:

- Loading the face mask detection dataset from disk
- Training a model using Keras/TensorFlow on this loaded dataset
- Serializing the face mask detector back to disk

2. Deployment:

- Loading the face mask detector
- Performing face detection
- Classifying each face as “Mask” or “No Mask”

2. Data Collection

The data will be collected from the below mentioned sources:

- Real-World Masked Face Dataset (RMFD)
- Kaggle Datasets

The aim is to collect more than 1800+ images in both “with_mask” and “without_mask” classes.

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.

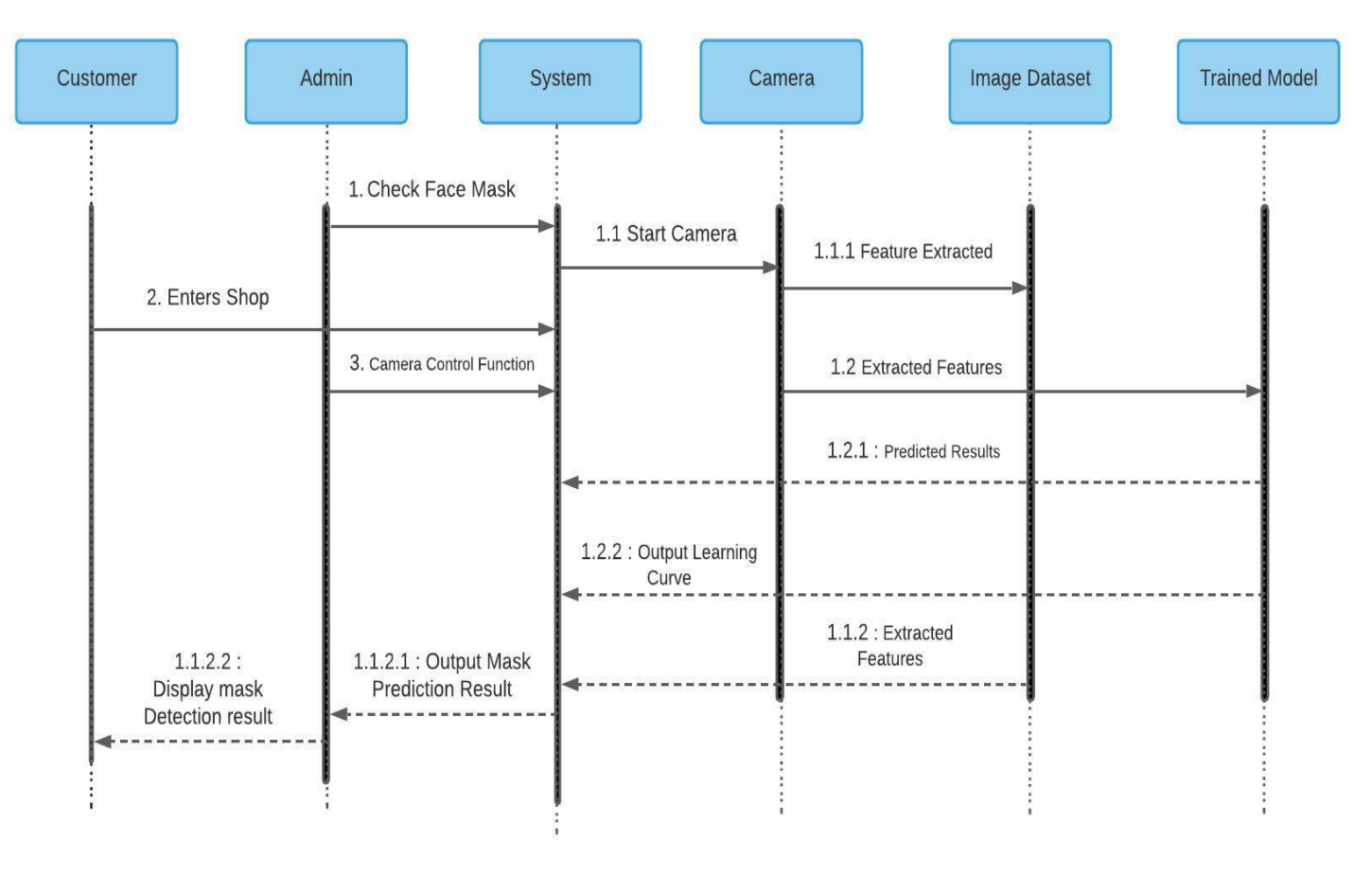
Purpose of Sequence Diagram

- 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.

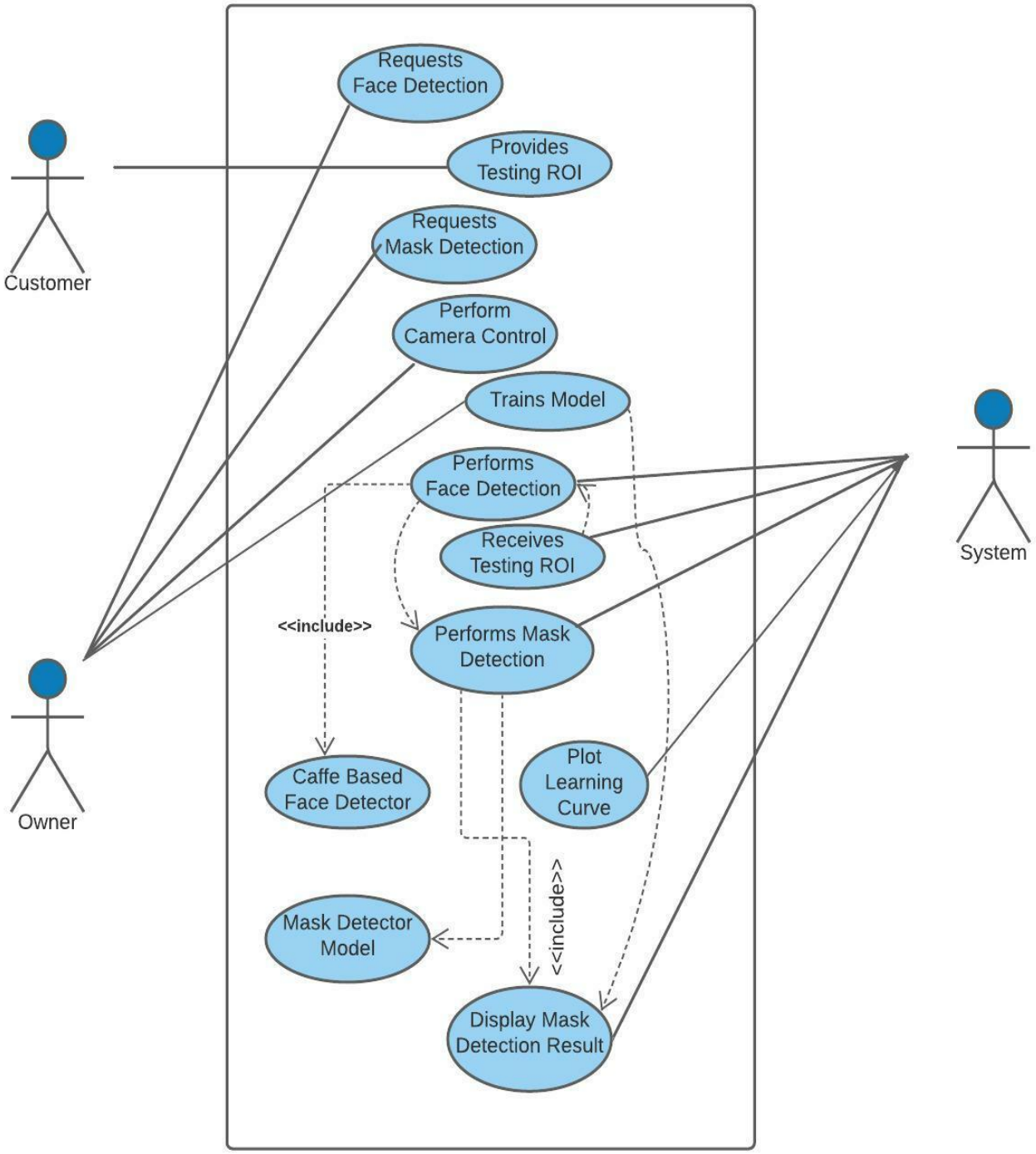
Use Case Diagram captures

- 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.

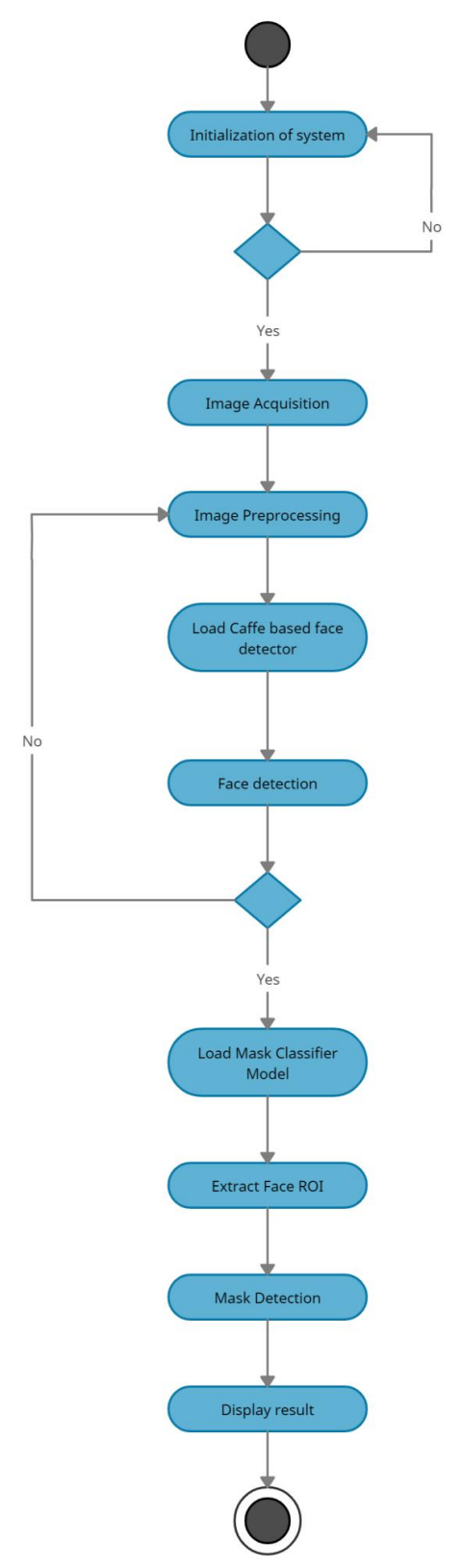
Purpose of Activity Diagram

- 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.

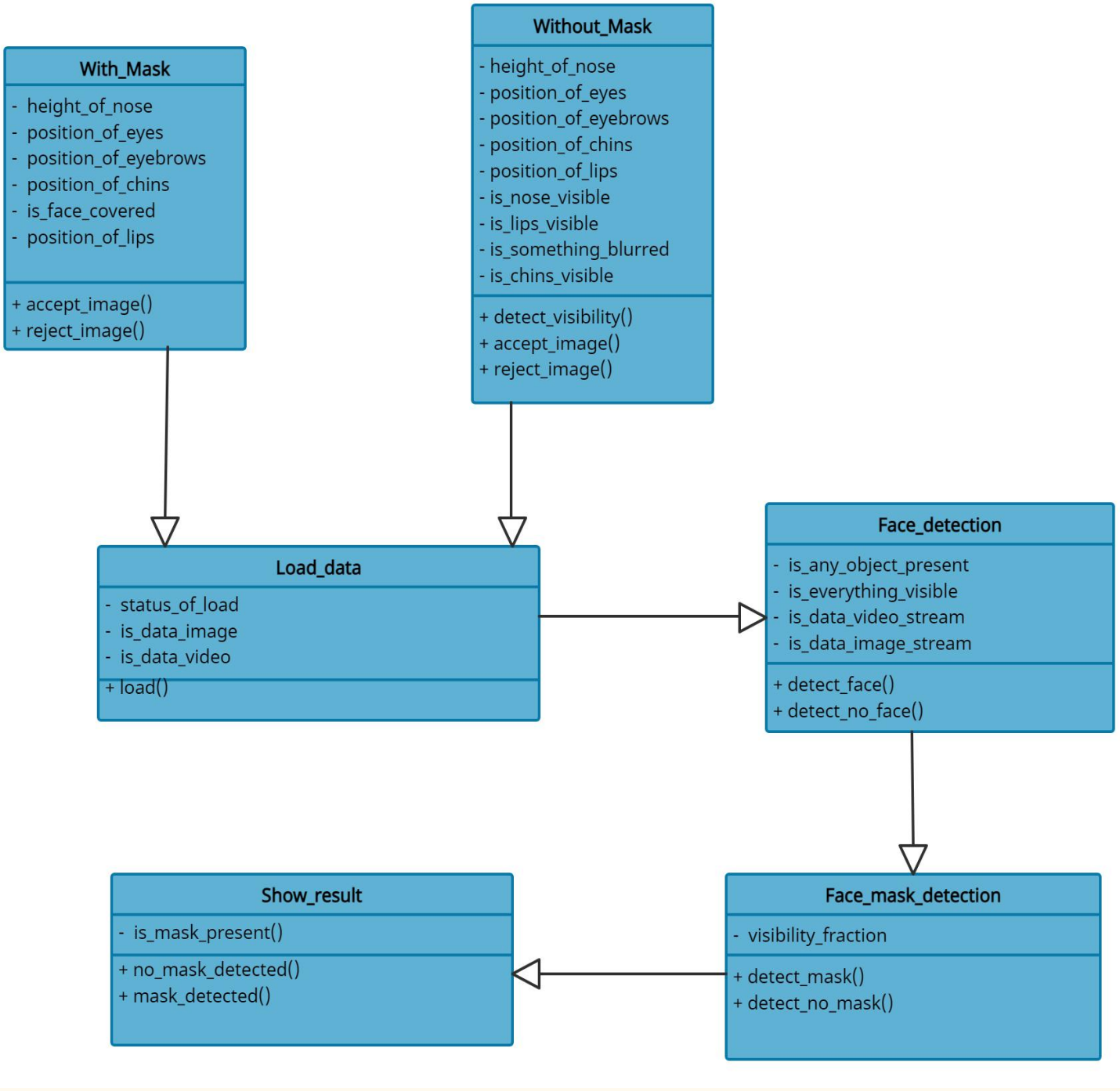
Purpose of Class Diagram

- 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



Implementation

Technology Details :

- Python

Libraries :

- OpenCV
- Tensorflow
- keras
- Numpy
- Sklearn
- Imutils

1. Libraries

1.1 OpenCV

OpenCV (Open Source Computer Vision Library), an open-source computer vision and ML software library, is utilized to differentiate and recognize faces, recognize objects, group movements in recordings, trace progressive modules, follow eye gesture, track camera actions, expel red eyes from pictures taken utilizing flash, find comparative pictures from an image database, perceive landscape and set up markers to overlay it with increased reality and so forth. The proposed method makes use of these features of OpenCV in resizing and color conversion of data images.

The .prototxt file(s) which will define the model architecture (i.e., the layers)

The .caffemodel file which will contain the weights for the actual layers

OpenCV's blobFromImage and blobFromImages to facilitate image pre-processing

cv2.dnn.blobFromImage and cv2.dnn.blobFromImages functions of OpenCV's module facilitates image pre-processing for deep learning classification. These two functions perform:

- [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.

- Scale factor: 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 scale factor.

- 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.

1.2 Keras Image Data Generator

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

Keras Image Data Generator 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.

2 Proposed Method

2.1 Data Processing

Data preprocessing involves conversion of data from a given format to much more user friendly, desired and meaningful format. It can be in any form like tables, images, videos, graphs, etc. These organized information fit in with an information model or composition and captures relationship between different entities . The proposed method deals with image and video data using Numpy and OpenCV.

2.1.1 Data Visualization

Data visualization is the process of transforming abstract data to meaningful representations using knowledge communication and insight discovery through encodings. It is helpful to study a particular pattern in the dataset.

The total number of images in the dataset is visualized in both categories – 'with mask' and 'without mask'. Then to find the number of labels, we need to distinguish those categories using labels=[i for i in range(len(categories))]. It sets the labels as: [0, 1]

2.1.2 Conversion of Image Size

Model require a fixed-size input image. Therefore we need a fixed common size for all the images in the dataset. Using cv2.resize() , the image is resized into 224 x 224.

2.2 Training The Model

2.2.1 Splitting the data and training the CNN model

After setting the blueprint to analyze the data, the model needs to be trained using a specific dataset and then to be tested against a different dataset. A proper model and optimized train_test_split help to produce accurate results while making a prediction. The test_size is set to 0.2 i.e. 80% data of the dataset undergoes training and the rest 20% goes for testing purposes. Next, the images in the training set and the test set are fitted to the Sequential model. Here, The model is trained for 40 epochs (iterations) which maintains a trade-off between accuracy and chances of overfitting.

3. Methods and techniques used :

```
data=[]
labels=[]

for i in imagePaths:
    label=i.split(os.path.sep)[-2]
    labels.append(label)
    image=load_img(i,target_size=(224,224))
    image=img_to_array(image)
    image=preprocess_input(image)
    data.append(image)
```

This loop iterates over the images taken In the dataset. Here the images iterated are and the label from the image path is taken and is then appended in the labels field. Then the image is passed on to perform the resizing of the image to 224 x 224 size. Further , the image pixels are converted in the form of an array which stores it in numeric form. Then, the numeric array is then passed to **preprocess_input()** function to perform further steps.

```
labels
```

```
Output exceeds the size limit. Open the full output data in a text editor
['without_mask',
 'without_mask',
 'without_mask',
 'without_mask',
 'without_mask',
 'without_mask',
 'without_mask',
 'without_mask',
 'without_mask',
 'without_mask',
```

This are the obtained **labels** based from the images form the dataset which was obtained from iterating over the images path as shown above step.

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

```
labels
```

```
array([[0., 1.],
       [0., 1.],
       [0., 1.],
       ...,
       [1., 0.],
       [1., 0.],
       [1., 0.]], dtype=float32)
```

Next , after obtaining the labels , **LabelBinarizer()** is used which convert the label obtained in above step into numeric value by assigning it a number corresponding to a particular label. Here there are two categories i. e “with mask” and “without mask”. So, only two numerics 0 and 1 are used.

```
train_X,test_X,train_Y,test_Y=train_test_split(data,labels,test_size=0.20,stratify=labels,random_state=10)
```

Here,We need to split a dataset into train and test sets to evaluate how well our machine learning model performs . Inorder to perform various operation on the dataset , we need to split the dataset into parts like training set, testing set. So , we used the **train_test_split()** function to perform the splitting of the array or metrices into random train and test subnets. random_state parameters controls the shuffling applied to the data before applying the split.

```
baseModel=MobileNetV2(weights='imagenet',include_top=False,input_tensor=Input(shape=(224,224,3)))
```

```
learning_rate=0.001
Epochs=40
BS=5

opt=Adam(lr=learning_rate,decay=learning_rate/Epochs)
model.compile(loss='binary_crossentropy',optimizer=opt,metrics=['accuracy'])

H=model.fit(
    aug.flow(train_X,train_Y,batch_size=BS),
    steps_per_epoch=len(train_X)//BS,
    validation_data=(test_X,test_Y),
    validation_steps=len(test_X)//BS,
    epochs=Epochs
)
```

Python

Here the number of epochs taken are 40. Also , the batch size taken is 5 so as to maintain the balance between the accuracy as well as the loss function.

```
#loop over the detections
for i in range(0,detections.shape[2]):
    confidence=detections[0,0,i,2]


    if confidence>0.5:
        #we need the X,Y coordinates
        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, resize it to 224,224 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)

        (mask,withoutMask)=model.predict(face)[0]

        #determine the class label and color we will 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 boxes
        cv2.putText(image,label,(startX,startY-10),cv2.FONT_HERSHEY_PLAIN,0.4,color,2)
        cv2.rectangle(image,(startX,startY),(endX,endY),color,2)
```

This loop performs the detection of the face from the input image.It defines the bounding on the face that is the rectangle which surrounds the face .It extracts the min X ,min Y as well as max X and max Y coordinates from the provided image. Thus, extracting the image with the provided dimensions and resizes it to 224 x 224 dimension. Then, this preprocessed image is passed onto the model for prediction from where it will return the result. If the result returned is labelled with “mask ” or “no mask” , then it will be displayed with the red box for “no mask ” or else with green box for “mask found” condition. Hence , this text will be put on the bounding box on the detected faces.

```
def detect_and_predict_mask(frame,faceNet,maskNet):
    #grab the dimensions of the frame and then construct a blob
    (h,w)=frame.shape[:2]
    blob=cv2.dnn.blobFromImage(frame,1.0,(300,300),(104.0,177.0,123.0))

    faceNet.setInput(blob)
    detections=faceNet.forward()

    #initialize our list of faces, their corresponding locations and list of predictions

    faces=[]
    locs=[]
    preds=[]


    for i in range(0,detections.shape[2]):
        confidence=detections[0,0,i,2]


        if confidence>0.5:
        #we need the X,Y coordinates
            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, resize it to 224,224 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)

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

        #only make a predictions if atleast one face was detected
        if len(faces)>0:
            faces=np.array(faces,dtype='float32')
            preds=maskNet.predict(faces,batch_size=12)

        return (locs,preds)
```

detect_and_predict_mask() function is used to perform detection of the mask. This is one of the important function of the project.

From the input passed to the function, we create a blobImage from it. And, this processed image is then stored into a field called "detections".

After this, the image is passed on to fetch the corresponding X and Y coordinates, which defines our fields startX,startYmendX,endY which are the end points of the boundings.From this, the face ROI is extracted and is resized to 224 x 224.

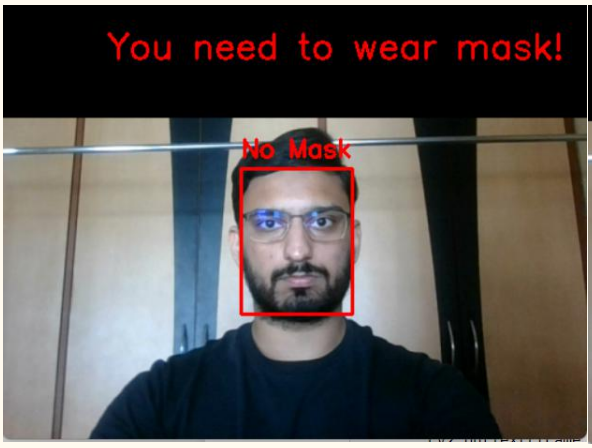
If here,the faces are detected, then it calles the prediction function and returns appropriate results.

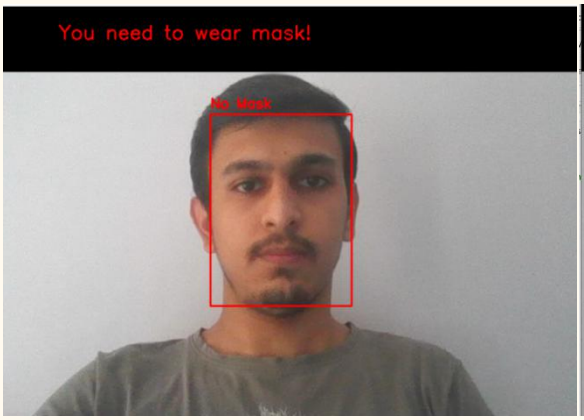
```
vs=VideoStream(src=0).start()

while True:
    #grab the frame from the threaded video stream and resize it
    #to have a maximum width of 900 pixels
    frame=vs.read()
    frame=imutils.resize(frame,width=900)

    #detect faces in the frame and preict if they are waring masks or not
    (locs,preds)=detect_and_predict_mask(frame,faceNet,maskNet)

    #loop over the detected face locations and their corrosponding loactions

    for (box,pred) in zip(locs,preds):
        (startX,startY,endX,endY)=box
        (mask,withoutMask)=pred

#         if withoutMask == 1.0:
#             mixer.init()
#             sound=mixer.Sound("beep-01a.wav")
#             sound.play()

        #determine the class label,text and color we will use to draw the bounding box and text
        label='Mask' if mask>withoutMask else 'No Mask'
        text="Enter the Shop" if label=='Mask' else "You need to wear mask!"
        color=(0,255,0) if label=='Mask' else (0,0,255)

        #display the label and bounding boxes
        x1,y1,w1,h1 = 0,0,900,100
        #cv2.rectangle(frame, (x1,x1), (x1 + w1, y1 + h1), (0,0,0), -1)
        #cv2.putText(frame, text, (x1 + int(w1/10),y1 + int(h1/2)), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
        cv2.putText(frame,label,(startX,startY-10),cv2.FONT_HERSHEY_SIMPLEX,0.7,color,2)
        cv2.rectangle(frame,(startX,startY),(endX,endY),color,2)

    #show the output frame
    cv2.imshow("Output",frame)
    if cv2.waitKey(2) == 27:
        break

cv2.destroyAllWindows()
vs.stop()
```

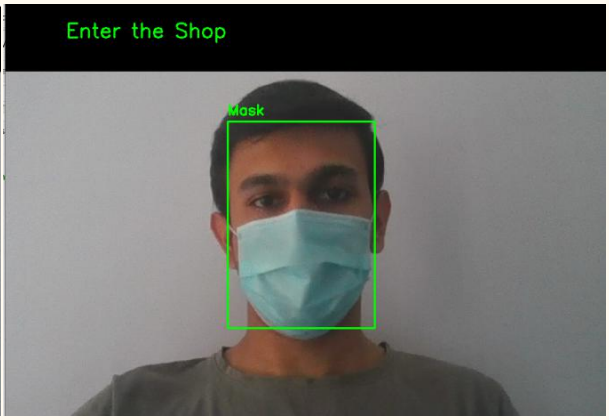
Here ,the depicted piece of code performs the detection on a video. For this purpose, webcam is taken into usage.Then ,an infinite loop is iterated until the condition is no further met.

The frame size is reset to about 900 pixels.Then, this frame is sent to "detect_and_predict_mask()" function.The returned value is stores in variables locs nad preds which then becomes the parameters for the bounding box on the image detected.Also,the labels are assigned as per predicted values inside the bounding box.The size of the bounding box as well as label text sizes are defined here.Finally, the predicted results are displayed as an output.

GUI of Project







Summary

The SRS provides overall description and an overview of the system functionality and system interaction with other systems. It also describes the probable user interfaces for the better understanding of the appearance of the system to the end users and stakeholders. It also provides the requirements specification in detailed terms and a description of the different system interfaces. Different specification techniques are used in order to specify the requirements more precisely for different audiences. It also depicts the system with user case scenario - use case diagrams, changing of states due to action of events - using sequence diagram, how activities are coordinated to provide a service which can be at different levels of abstraction - using activity diagram, to show the classes, relationships, interface, association, and collaboration - using class diagram. The SRS document helps the developer to gain better understanding of the system and a strong reference point for the developing and implementation stage of the project.

Conclusion

We illustrated the learning and performance task of the model. Using basic ML tools and simplified techniques the method has achieved reasonably high accuracy. It can be used for a variety of applications. Wearing a mask may be obligatory in the near future, considering the Covid-19 crisis. Many public service providers will ask the customers to wear masks correctly to avail of their services. The deployed model will contribute immensely to the public health care system.

Limitations

- Problem with detecting face mask if a person is standing far from the webcam.
- Not very Efficient in Predicting for Blur Capture Frame.
- Distance of Person from the camera should be within a predefined range.
- Not Very Accurate for mask prediction for frame with side face .

Future Extensions

For the version 2 of this project, work can be done to improve in the areas where the current model fails to meet the expectations. For example, for the case where there is difficulty in predicting in blur capture, help of image processing can be taken in order to overcome this situation . Also, the former method can be used to train the model for side faces so as to improve the prediction rate for side face mask detection.

Bibliography

- https://www.google.com/
- https://stackoverflow.com/
- https://docs.python.org
- https://docs.opencv.org/
- https://www.tensorflow.org/