2022

Face Mask Detection System

DEVELOPMENT AND TESTING

IMAN IMAD HINDI

PREPARED FOR:

INSTRUCTOR YAZAN KEIRY

CEO MANAGERS

HIRING RESPONSIBLE

Table of Contents

List of figures:

List of tables:

Acknowledgement

This project work was made possible by the support of Upskilling program held by HTU in Amman/Jordan (2022) with support of GIZ.

1.Abstract

COVID-19 pandemic has rapidly increased health crises globally and is affecting our day-to-day lifestyle. A motive for survival recommendations is to wear a safe facemask, stay protected against the transmission of the virus. Manual monitoring of correct wearing of facemasks is difficult task to be managed in open and closed areas. Therefore, the working on this project aimed to provide an end-to-end face mask detection system and provide a user-friendly model to monitor correct wearing of facemasks especially for crowded areas. Also, it allows users to get summary report showing the percentage of compliance in wearing facemasks. Using Kaggle datasets, the proposed model is trained and tested. The system runs in real-time and detect if an individual face has facemask, if not, the system notifies the security and monitor man through visual representation shown on the monitor screen. Additionally, the features of the system allow processing archived images from different sources and providing results on compliance of wearing facemasks. [1]

2.Background

The recent pandemic of COVID-19 has called for very strict health protocols overall the world. According to the World Health Organization (WHO), COVID-19 define as “infectious disease caused by the most recently discovered coronavirus” [2]. This new virus was unknown before the outbreak began in Wuhan, China, in December 2019. COVID-19 presents a risk at areas such as markets and public buildings where people are still attending in person and physically close to each other.

The spread of COVID-19 virus is through close contact with the people and in crowded public areas. The guidelines listed by the WHO, primary precaution should be taken to prevent the spread of virus is to wear facemask [3].

In this project, a real-time facemask detection model was developed by fine tuning VGG19 Pre-trained model for image classification and recognition. The developed system can be used for indoor and outdoor facilities such as schools, universities, shopping malls, multiplex etc. It can help in monitoring individuals automatically and check whether they are wearing facemask. Additionally, it can help to break the chain of spreading of virus when in close contact and reduces the positive cases which are rapidly increasing day-by-day overall the globe.

3.Objectives

The main goal of this system is "to build an end-to-end face mask detection Model that can help in minimizing the risk of COVID-19 spread between individuals, through detecting visitors' compliance in wearing the face mask correctly in public and private outdoor and indoor places".

The specific objectives of this system are:

- Allow beneficiaries/users to apply real-time monitoring of visitors and check their compliance in wearing face masks.
- Facilitate beneficiaries/users to follow up of customers compliance with national regulations associated to wearing face masks.
- Provide daily report summarizing the percentage of compliance visitors (wearing face masks). Which in result, can be used, when needed, to adapt the security procedures for more compliance.
- Save and retrieve data using local SQLite database (during no internet connection) and cloud Firebase.
- For archive files, the system can check images to predict those complying with correct wearing of face masks vs. others.

4.System development

4.1. Summary

The system operates by run a live video of visitors in specific area (such as building), this usually obtained from the monitoring cameras installed within the facility. Then, the system takes shots (images) each second and sending them through web API to the server. A pre-processing of shots is then taken place to extract the faces. The extracted faces from the previous step are then send to the model for facemasks detection to identify those without masks, incorrect wearing of masks, and those correctly wearing the masks. Afterwards, the system returns outputs summarizing results of facemasks detection, time, and location of the face in the originally taken shot. The outputs are then stored on local database and cloud database. Final results of mask detection are presented on the monitor showing correct wearing of masks in green color, incorrect wearing of mask in blue color, and faces without masks in red color. Reports are saved automatically on the DVR on daily bases.

Using the same model, the system allows processing archived images from different sources and providing results on compliance of wearing facemasks.

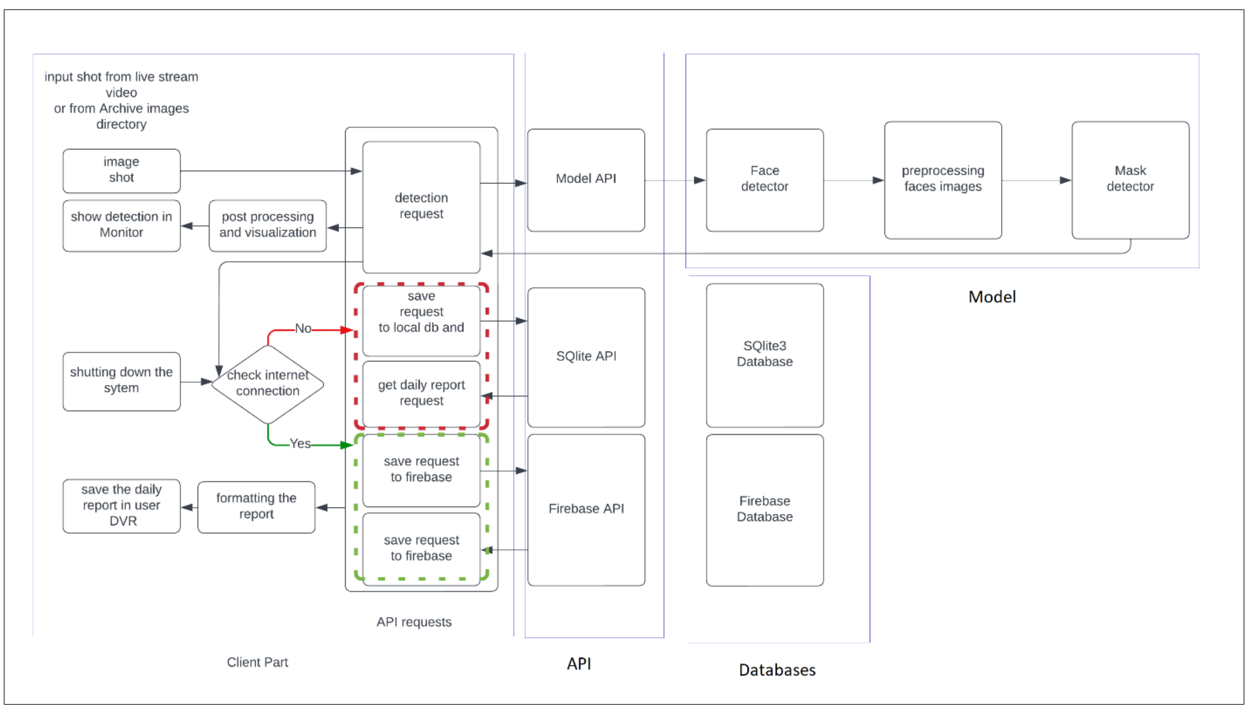

Figure 1 : The end-to-end system flow chart.

4.2. System components

4.2.1. Inputs to the system

Client monitoring cameras (security cameras) are used as a source of input to the system. The live streaming videos are processed to take images every second for the purpose of face mask detection.

4.2.2. Dataset

The dataset was used to train the model for classifying different ways of wearing face masks, and to make diverse and unbiased detection. It consists of 3 classes: a) Incorrect mask b) With mask C) without mask.

The used dataset for this system was the FMD_DATASET [4], which has a total of 14535 images. This dataset is chosen because it is universal dataset and thus allows to build a face mask detection system that can detect almost all types of face masks with different orientations.

The classification of images within this dataset were as the following:

- incorrect masked class consists of 5000 images, of which 2500 are Mask on Chin and 2500 are Mask on Mouth and Chin.
- With mask class has 4789 images, of which 4000 are simple with mask and 789 are complex with mask images.
- without mask class has 4746 images, of which 4000 are simple and 746 are complex images.

The definition of each classification categories as follows:

- "Mask On Chin" images: These are the images in which masks are put on a chin only. The mouth and the nose of a person are visible.
- "Mask On Chin Mouth" images: In this, the mask is covering the chin and the mouth area. The nose of a person is not covered.
- "Simple with Mask" images: It consists of data samples of face masks without any texture, logos, etc.

- "Complex with Mask" images: It includes the images of the sophisticated face masks with textures, logos, or designs printed on them.
- "Simple Without Mask images: These are images without any occlusion.
- "Complex without Mask" images: It consists of faces with occlusion, such as beard, hair, and hands covering the face.

4.2.3. Model:

The heart of the system is the detection part or (Model detector) which consists of two detection stages: **the first stage**, was a prebuilt MediaPipe Face detection Solution [5], which is used to detect faces in the taken images (shots). It is an ultrafast face detection solution that comes with 6 landmarks and multi-face support. This detector has super-real-time performance enables it to be applied to any live viewfinder experience that requires an accurate facial region of interest as an input for other task-specific models [6]. **The second stage** was developed based on fine tuning of VGG 19 pre-trained model to detect way of wearing face masks. VGG-19 is a CNN proposed by A. Zisserman and K. Simoniyan. And it has 19 layers (figure 2). This model was built and trained using transfer learning techniques to fit our desired detection output. Using this technique allow us to easily and rapidly reach high accuracy and precession (validation accuracy= 98% in our model), the system applies some preprocessing (resize the image and encoding the labels) and post processing (label decoding, detection output visualization, report format) to make the data suitable for our model.

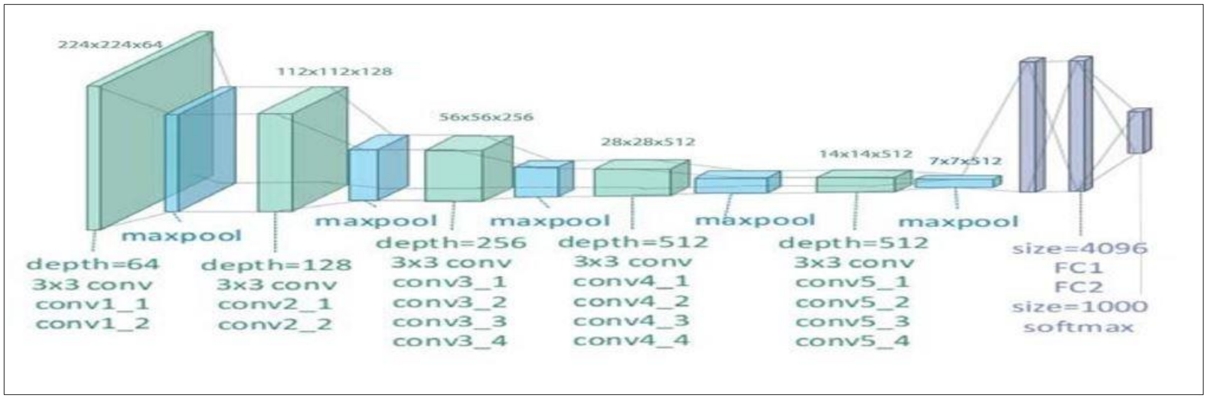

Figure 2 : illustration of the network architecture of VGG-19 model [19].

4.2.4. Application programming interface (API):

API is a software intermediary that allows two applications to communicate to each other. API architecture is usually explained in terms of client and server. The application sending the request is called the client, and the application sending the response is called the server [7].

In our system, the client sends the Image through API to the server and the server then returns the detection output for the user as a response. API is also used to send requests to save the data locally and in the firebase cloud, in addition to make a request to retrieve data that have been saved.

4.2.5. Database:

Both, firebase cloud and SQlite3 database were used to periodically save detection data. The SQlite3 database used to save data locally to enable the client to easily save and retrieve data during periods of no internet connection. While the firebase cloud used as a backup. Automatic synchronization between the two databases performed by the system to keep the data continuously updated.

4.2.6. Monitor:

The output of the system which is the faces' location and the mask detection presented on the security monitor to allow the user to easily and frequently configure the customers' compliance. The monitor showing correct wearing of masks in green color, incorrect wearing of mask in blue color, and faces without masks in red color.

4.3. System software development:

4.3.1. Code structure:

The following figure shows the overall structure of the developed codes. As presented in the figure, the system consists of 4 directories and 14 files.

```
$ tree --dirsfirst
.
├── DataSet
│    └── FMD_DATASET
│                ├── incorrect_mask
│                ├── with_mask
│                └── without_mask
├── Model
│    ├── VGG19_FaceMaskDetector.hdf
│    ├── FaceMaskDetectionModel.py
│    ├── FaceMaskDetectionModelTest.py
│    └── FaceMaskDetector.py
├── Client
│    ├── ImageShotMaskDetection.py
│    ├── RealTimeMaskDetection.py
│    ├── SaveReport.py
│    └── MaskDetectionRequest.py
└── APIs
     ├── FirebaseAPI.py
     ├── ModelAPI.py
     └── SQLiteAPI.py
4 directories, 14 files
```

Figure 3 : The developed code structure of end-to-end mask detection system.

4.3.1.1. Model development

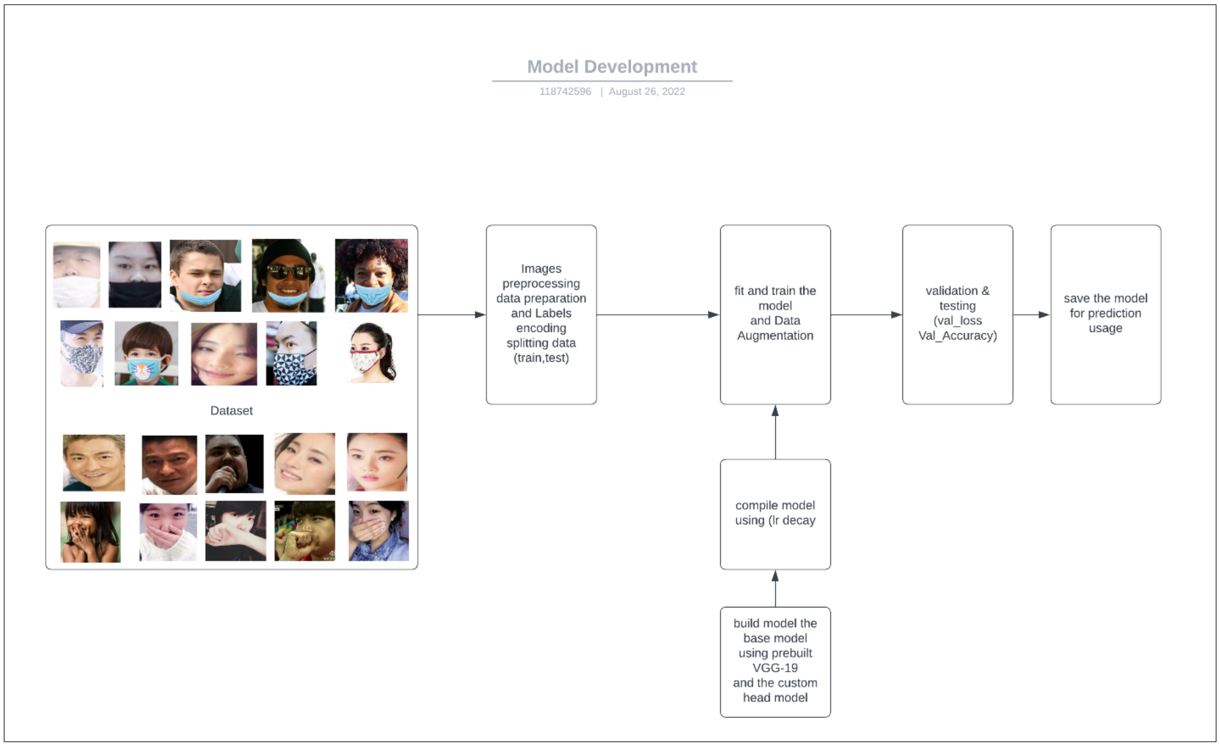

Figure 4 : the flow chart of Model development process.

Step 1: Data preparation and preprocessing:

The following procedure was followed for data preparation and processing:

- Load the images as np array.
- Resize the images to fit our Vgg19 model.
- Export the labels to np array.
- Use one hot encoding to categories the labels to three classes.
- Incorrect-Mask class (1,0,0), Mask class (0,1,0), and without-mask class (0,0,1).

```
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
checkpoint_filepath = args["checkpoint"]
data = []
labels =[]
for (i, imagePath) in enumerate(imagePaths):
    label = imagePath.split(os.path.sep)[-2]
    image =image_utils.load_img(imagePath, target_size=(224, 224))
    image =image_utils.img_to_array(image)
    image = preprocess_input((image))
    data.append(image)
    labels.append(label)
    # show an update every 1,000 images
    if i > 0 and i % 1000 == 0 or i==25000:
        print("[INFO] processed {}/{}".format(i, len(imagePaths)))

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

np.save('images.npy', data)
np.save('labels.npy', labels)

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

Step 2: Data splitting:

This step was used to split the data (images and labels) into train and test. Then, the split data augmented to generate additional data by applying some effects on the images like, rotation, right shift, left shift, and horizontal flip. The following figure present the code of data splitting and augmentation.

```

(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.2, random_state=42)

np.save('testX.npy', testX)
np.save('testY.npy', testY)
datagen = ImageDataGenerator(
                        rotation_range=20,
                        width_shift_range=0.1,
                        height_shift_range=0.1,
                        shear_range=0.15,
                        horizontal_flip=True)

```

Step 3: Build the model

The model built and trained using VGG_19 pre-trained mode, which trained to detect several common objects using large datasets and apply it to a new task (which is in our case mask detection), this is called "transfer learning" , i.e., transferring the knowledge learned from one task to another. This is useful because the model doesn't have to learn from scratch and can achieve higher accuracy in less time as compared to models that don't use transfer learning.

The reason of using VGG-19 in specific is that when referring to the research article [8] (Comparative Study between different Models in face mask detection) that compares the result of evaluation metrics of each pre-trained CNN Models for mask detection problem. All the metric values of VGG-19 except for the support, are over 98%. Compared the performance of this model with the previous ones, the VGG-19 shows the best result in face mask detection.

The model was built using transfer learning, by fine-tuning the VGG-19 Pretrained Neural Network as shown in the figure below by following these steps (refer to Appendix A to show the Model Summary):

- Remove the fully connected nodes at the end of the network (i.e., where the actual class label predictions are made).
- Replace the fully connected nodes with freshly initialized ones.
- Freeze earlier CONV layers earlier in the network (ensuring that any previous robust features learned by the CNN are not destroyed).
- Start training, but only train the FC layer heads. [9]

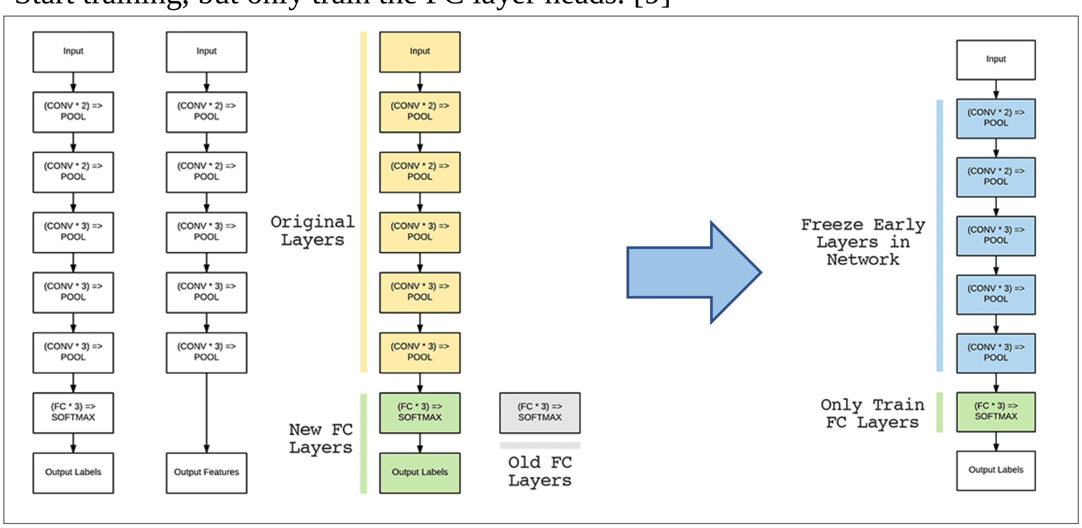

Figure 5 : steps of fine tuning the pretrained model using transfer learning technique

```python
pre_trained_model = vgg19.VGG19(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))

headModel = pre_trained_model.output
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(3, activation="softmax")(headModel)


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

for layer in pre_trained_model.layers:
    layer.trainable = False

# show a summary of the base model
print("[INFO] summary for base model...")
print(pre_trained_model.summary())
print(model.summary())
```

Step 4: Compile the model:

The model had been compiled and some techniques were applied to fine tuning the hyper parameters of the system. This was aiming to increase the accuracy and enhance the performance of the system and reach the optimum values of the hyper parameters. The following configurations were performed under this step:

1. Initialize the number of epochs and batch size. The batch size refers to the number of samples processed before the model is updated. While number of epochs is the number of complete passes through the training dataset. [10] .Number of epochs of 20 and patch size of 64 were identified in this step.

2. Choose the Adam optimizer to optimize the model and reduce the validation loss. Adam optimizer was chosen since it can provide faster and better result to reach to the momentum of the loss curve (the optimum solution point). The Adam method has several advantages including straightforward to implement, computationally efficient, has little memory requirements, invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate

for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning.

Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods [11]

3. Apply early stopping to allow the system to stop earlier when reach the lowest validation loss, and to prevent the model from overfitting. Early stopping is a method that allows us to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a holdout validation dataset. [12] Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an under fit model. Therefore, early stopping allows us to stop training when a monitored metric has stopped improving.

4. Apply polynomial learning rate decay schedule to adjust the learning rate decayed during the training process (initialized with 0.001). The learning rate, α, controls the "step" along the gradient. Larger values of α imply that we are taking *bigger steps.* While smaller values of α will make *tiny steps*. The Keras library offers a time-based learning rate scheduler — it is controlled via the decay parameter of the optimizer class (such as SGD, Adam, etc.).

   Internally, Keras applies learning rate the following formula to adjust the learning rate after *every batch update*.

   $lr = init\_lr * \frac{1.0}{1.0 + decay * iterations}$

   Using Polynomial Decaying will decrease our learning rate, thereby allowing the network to take smaller steps, this decreased learning rate enables our network to descend into areas of the loss landscape that are "more optimal" and would have otherwise been missed entirely by our learning rate learning this is done using callback function which provide a way to execute code and interact with the training model process automatically. Using callbacks, the learning rate is decayed to zero over a fixed number of epochs.

The rate in which the learning rate is decayed is based on the parameters to the polynomial function. A smaller exponent/power to the polynomial will cause the learning rate to decay “more slowly”, whereas larger exponents decay the learning rate “more quickly” we choose the power = 5 in this case. [13]

```python
class PolynomialDecay():
    def __init__(self, maxEpochs=20, initAlpha=0.001, power=5.0):
        self.maxEpochs = maxEpochs
        self.initAlpha = initAlpha
        self.power = power
    def __call__(self, epoch):
        decay = (1 - (epoch / float(self.maxEpochs))) ** self.power
        alpha = self.initAlpha * decay
        return float(alpha)

```

5. Use the categorical cross entropy to calculate the loss function (which is usually used with multi class classification). This used since we have 3 categories classes (incorrect, mask, no mask), besides it will be suited to classification tasks that is used one hot encoding to categories the labels. In addition, SoftMax activation function was used which is the only activation function recommended to use with the categorical cross entropy loss function and one hot encoding.

6. Use model check point to save the best model observed during training for later use. The *ModelCheckpoint* callback is flexible in the way it can be used, in this case it helps to retrain the model, if needed, in shorter duration.

```python
print("[INFO] compiling model...")
opt = Adam(learning_rate=lr_rate, decay=lr_rate / epochs)
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
schedule = PolynomialDecay(maxEpochs=epochs, initAlpha=.001, power=5)
learning_rate_callbacks = LearningRateScheduler(schedule)
model_check_point=ModelCheckpoint(checkpoint_filepath,
                                monitor="acc",
                                verbose=2,
                                save_best_only=False,
                                save_weights_only=False,
                                mode="auto",
                                save_freq="epoch",
                                options=None,
                                initial_value_threshold=None
                                )
model.compile(loss="categorical_crossentropy", optimizer=opt,
    metrics=["accuracy"])
```

Step 5: train and fit the model:

After compiling, we train and fit the model by provide the system with the train and test data sets, in addition to using compilation configurations specified in the previous step. Online augmentation was also used so new data generated for the train and test during the training process and apply callbacks for early stopping and learning rate updates.

```
print("[INFO] training head...")
history = model.fit(
            datagen.flow(trainX, trainY, batch_size=batch_s),
            #steps_per_epoch=floor(len(trainX) // batch_s),
            validation_data=datagen.flow(testX, testY,batch_size=batch_s),
            #validation_steps=floor(len(testX) // batch_s),
            validation_freq=1,
            epochs=epochs,
            callbacks= [early_stopping,learning_rate_callbacks],#model_check_point],
            verbose=2)
```

Step 6: Evaluating and testing for the model (Evaluation Metrics):

The system has been evaluated and tested using three different metrics as the following:

1. **Validation Accuracy and validation Loss:**

After training the model we tested it using the model.evaluate() to calculate validation loss and accuracy. Testing results of the model were as the following:

Validation loss = 0.0810

Validation accuracy = 98.2462%.

```
print("[INFO] evaluating on testing set...")
(val_loss, val_accuracy) = model.evaluate(testX, testY,batch_size=batch_s, verbose=1)
print("[INFO] val_loss={:.4f}, val_accuracy: {:.4f}%".format(val_loss,val_accuracy * 100))


print("[INFO] evaluating network...")
prediction = model.predict(testX, batch_size=batch_s)
prediction = np.argmax(prediction, axis=1)
actual=np.argmax(testY,axis=1)

print("[INFO] saving mask detector model architecture and weights to file...")
model.save(args["model"])


#epoch_no=np.arange(0, epochs)
plt.style.use("ggplot")
plt.figure()
plt.plot(history.history["loss"], label="train_loss")
plt.plot(history.history["val_loss"], label="val_loss")
plt.plot(history.history["accuracy"], label="train_acc")
plt.plot(history.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

By plotting the accuracy and loss during the training and evaluation step vs the number of epochs as shown in the figure below, we can easily recognize that the model performs well and there is no over fitting or underfitting in the trained model.

Figure 6 : The model accuracy and loss for training and validation plot vs number of epochs.

2. Confusion matrix and Classification report:

Accuracy is the overall number of the correct predictions fractionated by the whole number of predictions created for a dataset. It can inform us immediately if a model is trained correctly and by which method it may perform in general. Nevertheless, it does not give detailed information concerning its application to the issue [8]

To check the performance of the system more accurately it is essential to measure the performance of the model and compare the model with certain metrics. the confusion matrix and the classification report were used to choose the optimal model for mask detection. Which are the popular methods that used to validate model performance. Diagonal values of the matrix indicate correct predictions for each class, whereas other cell values indicate a number of wrong predictions.

```
# show classification report and confusion matrix        You, 1 second ago • Uncommitted changes
print(classification_report(testY.argmax(axis=1), prediction,
    target_names=lb.classes_))
print(confusion_matrix(testY.argmax(axis=1),prediction,labels=lb.classes_))
incorrect_cm,mask_cm,no_mask_cm=multilabel_confusion_matrix(testY.argmax(axis=1),prediction)
```

The following figures presenting the confusion matrix results for our system for all 3 classes.

	Incorrect mask	Mask	No mask
Incorrect mask	987	21	4
Mask	3	926	5
No mask	4	13	945

Figure 7 : Confusion matrix results of the three classes.

The next figure shows the confusion matrix results of each class, TP is the computation of the samples of true positives, TN is the calculation of the samples of true negatives, FP is the counting of the samples of false positives, and FN is the enumeration of the samples of false negatives, these values can be identified from a confusion matrix.

	Actual positive	Actual negative
positive	1851	7
Negative	16	933

Incorrect Mask class CM

	Actual positive	Actual negative
positive	1836	24
Negative	12	935

Mask class CM

	Actual positive	Actual negative
positive	1886	10
Negative	13	898

No Mask Class CM

Figure 8 : Confusion matrix results of each class (mask, no mask, incorrect mask).

the classification report in sklearn library was used to compute precision, recall and f1-scor by identifying the True positive (TP), True negative (TN), False Positive (FP) and False Negative (FP) from the confusion matrix above. The True positive in general terms are referred for images that where labelled true and the detection produced true. The True negative are for images that are labelled true but indicated as false. False positive on the other hand are results that are labelled as false but predicted as false and False negative are the images that are labelled false but predicted as true. In the context of mask detection, due to multiple objects and multiple classes in one image itself, TP, FN, FP and TN were all measured for each detection. i.e. precision is the percentage correct prediction. Recall measures how well all the positive predictions are found. [1]

Table 1: the classification report of the three categorical classes (mask, no mask, incorrect mask)

	incorrect Mask	Mask	No Mask	accuracy	macro avg	weighted avg
precision	0.992553	0.974974	0.988987	0.985394	0.985505	0.985465
recall	0.983140	0.987328	0.985730	0.985394	0.985400	0.985394
f1-score	0.987824	0.981112	0.987356	0.985394	0.985431	0.985408
support	949.000000	947.000000	911.000000	0.985394	2807.000000	2807.000000

The table above listed the precision, recall and F1-score for each class. The Precision, called PPV, is a satisfactory measure to determination, whereas the false positives cost is high. Recall is the model metric used to select the best model when there is an elevated cost linked with false negative. Recall helps while the false negatives' cost is high. F1-score is required when you desire to seek symmetry between both precision and recall. It is a general measure of the accuracy of the model. It combines precision and recall. A good F1-score is explained by having low false positives and also low false negatives [8]

3. ROC Curve plot and AUC:

Receiver Operating Characteristic (ROC) metric is used to evaluate classifier output quality. ROC curves typically feature true positive rate on the Y axis, and false positive rate on the X axis. This means that the top left corner of the plot is the "ideal" point - a false positive rate of zero, and a true positive rate of one. This is not very realistic, but it does mean that a larger area under the curve (AUC) is usually better.

The "steepness" of ROC curves is also important, since it is ideal to maximize the true positive rate while minimizing the false positive rate.

ROC curves are typically used in binary classification to study the output of a classifier. In order to extend ROC curve and ROC area to multi-label classification, it is necessary to binarize the output. One ROC curve can be drawn per label, but one can also draw a ROC curve by considering each element of the label indicator matrix as a binary prediction (micro-averaging).

```
lr_tpr=dict()
lr_fpr=dict()
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(3):
    fpr[i], tpr[i], _ = roc_curve(testY[:, i], prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
    #lr_fpr[i], lr_tpr[i], _[i] = roc_curve(testY[:, i], prob[:, i])
    plt.plot(
    fpr[i],
    tpr[i],
    color="darkorange",
    lw=i,
    label="ROC curve (area = %0.2f)" % roc_auc[i],
    )
    plt.plot([0, 1], [0, 1], color="navy", lw=i, linestyle="--")
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Receiver operating characteristic example")
    plt.legend(title="ROC Curve",loc="lower right")
    plt.savefig('ROC Curve')
    plt.show()

fpr["micro"], tpr["micro"], _ = roc_curve(testY.ravel(), prob.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

The plot for ROC curve and AUC calculation for the 3 categorical classes is shown in the figure below. The plot shows that the three classes AUC area is around one and this indicates that the model performs well in testing dataset.

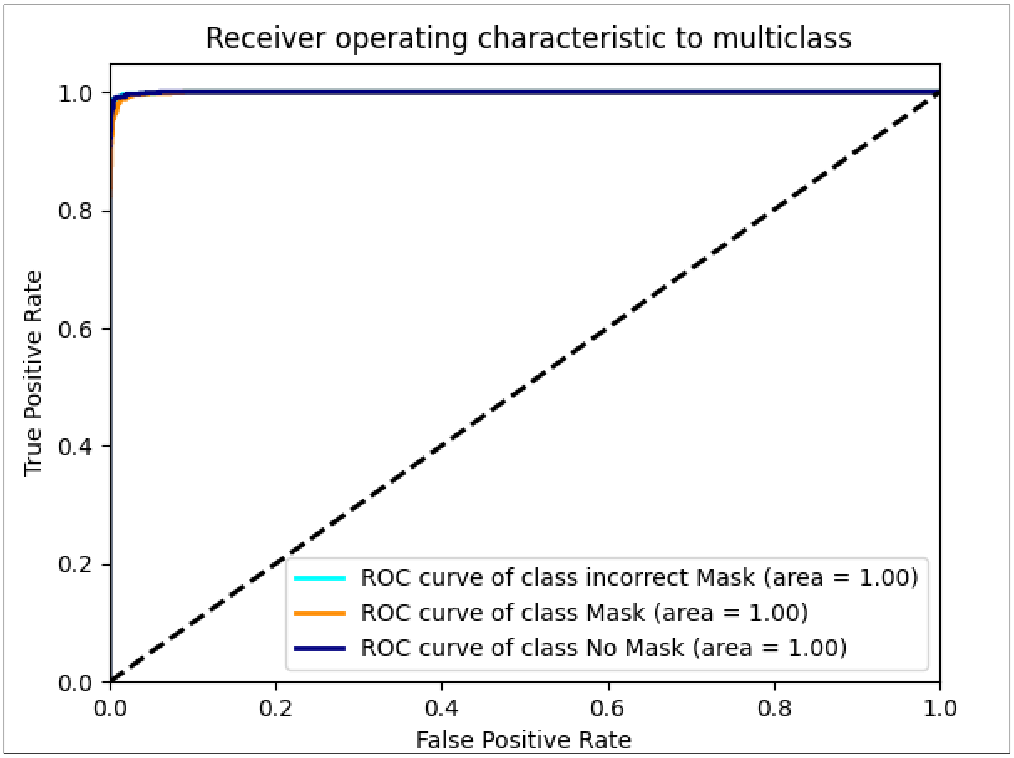

Figure 9 : ROC curve and AUC calculation for the 3 categorical classes (mask, no mask, incorrect mask)

The Micro Average ROC Curve for the total 3 classes is used, it is measuring precisely the performance of the system in the whole classes, it is low for models which not only accomplish well on common classes but also accomplish poorly on rare classes. thus, it is a harmonious metric to the all-inclusive accuracy

The **sklearn.metrics.roc_auc_score** function can be used for multi-class classification. The multi-class One-vs-One scheme compares every unique pairwise combination of classes. [14]

The next figure shows the micro average of the 3-classes, which is showed the goodness of the model performance (AUC around 1).

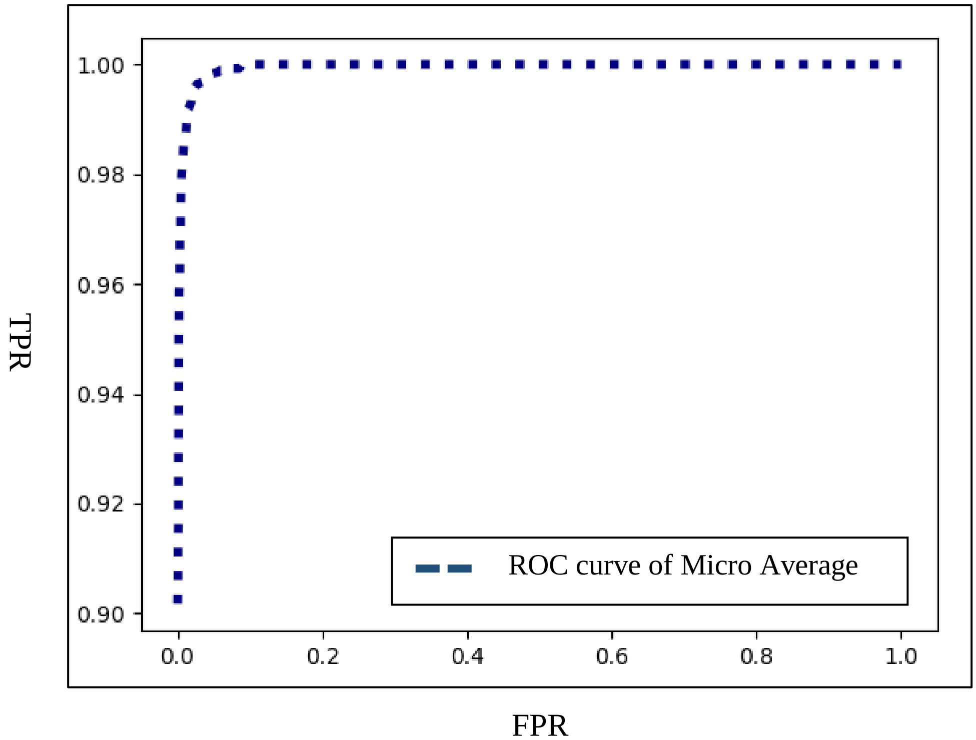

Figure 10 : The micro average of the 3 categorical classes (mask, no mask, and incorrect mask)

Step7: Save the model:

After developing and testing the model, the final step is to save it as .hdf file to be ready for use. The following figure shows the code of model saving.

```

print("[INFO] saving mask detector model architecture and weights to file...")
model.save(args["model"])

```

Step8: Deploy the model:

Once the model saved, it’s time for exposing it to real use by making the model available via real-time APIs, and accessible for users to be used for mask detection purposes.

The aim of this step is to package the mask detection model into a web service, so that when it is given the image through a POST request, returns the Mask predictions as a response. By using the Flask web framework which is a lightweight framework commonly used for developing web services in Python.

The FaceMaskDetector file code was developed to package the model to be used for mask detection purposes. when the user sends a detection request to the server, the **detect_and_predict_mask** function in FaceMaskDetector file will be called. It automatically loads the model and make the detection process and return a suitable format response back to the user. This done by following these steps:

Step1: load the MediaPipe face detector and the developed face mask detector model to be ready for usage.

```
class FaceMaskDetector():

    print("[INFO] loading face detector model...")
    faceNet = mp.solutions.face_detection
    mp_drawing = mp.solutions.drawing_utils


    # load the face mask detector model from disk
    print("[INFO] loading face mask detector model...")
    maskNet = load_model('C:\\Users\\user\\Desktop\\iman\\FaceMaskDetection-SocialDistancing\\Model\
                        \\VGG19_FaceMaskDetector.hdf')

```

Step2: when the **detect_and_predict_mask** function is called the frame images is enters the face detector model to make a detection for all the faces in the image. It then returns the faces locations as (x_min, y_min, height and width)

```
    def detect_and_predict_mask(frame, faceNet=faceNet, maskNet=maskNet):
        faces = []
        locs = []
        preds = []
        preds_actual=[]

        with faceNet.FaceDetection(
            model_selection=1, min_detection_confidence=0.5) as face_detection:
            results = face_detection.process(cv2.cvtColor(frame,cv2.COLOR_BGR2RGB))
            (h, w) = frame.shape[:2]
```

Step3: Loop over all faces detected to make a mask detection for each using the face mask detection model which returns the propality values of the 3-categorical classes corresponding to each face.

```python
            if results.detections:
                for detection in results.detections:
                    coordinates=detection.location_data.relative_bounding_box
                    (startX, startY, endX, endY) = int(coordinates.xmin*w-10),int(coordinates.ymin*h-10),\
                                                   int((coordinates.width+coordinates.xmin)*w+10),\
                                                   int((coordinates.height+coordinates.ymin)*h+10)
                    # ensure the bounding boxes fall within the dimensions of the frame
                    (startX, startY) = (max(0, startX), max(0, startY))
                    (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

                    # extract the face ROI, convert it from BGR to RGB channel
                    # ordering, resize it to 224x224, and preprocess it
                    face = frame[startY:endY, startX:endX]
                    # only make a predictions if at least one face was detected
                    if face.any():
                        face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                        face = cv2.resize(face, (224, 224))
                        face = image_utils.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))
                        pred = maskNet.predict(face).tolist()[0]
```

Step4: Apply the **Arg_max** function to find the exact mask detection status for each face and then return the faces locations, detection probabilities, and the exact mask detection status as response to the detection request.

```python
                        pred = maskNet.predict(face).tolist()[0]
                        #check prediction and apply the argmax() to extract the corresponding class label
                        pred_actual=np.argmax(pred)
                        if pred_actual==0:
                            mask="incorrect Mask"
                        elif pred_actual==1:
                            mask="Mask"
                        elif pred_actual==2:
                            mask="No Mask"
                        else:
                            mask=""
                        preds_actual.append(mask)
                        preds.append(pred)
        return (locs, preds,preds_actual)
```

4.3.1.2. Client part code:

Client code describes the part of code responsible for taking input images from the real environment, sending them to the model for mask detection purposes, and then analyze the output of the model. Additionally, it performs post processing to visualize the output in proper way.

The client code consists of four main parts as the following:

- The first part: is used to read archived images from directory path to detect the masks, then analyze and visualize the detection output in the images and save it in specific directory path.
- The second part: is taking shots from a real time video stream to make predictions and then do some post processing to visualize the output on monitor.
- The third part: is responsible for communicating with the API's (Model API & Data base API) by sending requests for detection, save detection, or retrieve the data back to the user.
- The fourth part: is related to prepare and save the main report and summary report which showing the detection results as csv file.

```python
from imutils import paths
def mask_image():
    # construct the argument parser and parse the arguments
    ap = argparse.ArgumentParser()
    ap.add_argument("-i", "--images", required=True,
        help="path to input images")

    args = vars(ap.parse_args())
    i=0
    for imagePath in paths.list_images(args["images"]):
        # load the input image from disk, clone it, and send it to the model API
        #  for preprocessing and detection
        print("[INFO] classifying {}".format(
        imagePath[imagePath.rfind("/") + 1:]))
        image = cv2.imread(imagePath)
        cv2.imwrite('image.png',image)
        (locs, preds,preds_actual) = prediction_request('image.png',source=1)

```

Part1: Archived images detection

in this part the user can check mask compliance in archived images by specify the directory path of the images. The system then returns all the images with the mask detections output attached to each image plus returning a main report showing all faces detected and its corresponding mask detection. In addition to provide summary report of the results.

This is done through following steps:

Step1: loop over all images in the directory, read them one by one and send them through a detection request to the model. This request returns all the detected faces in the image and its corresponding mask detection output, plus the exact detection class (Mask, No Mask, Incorrect Mask).

```
i=0
for imagePath in paths.list_images(args["images"]):
    # load the input image from disk, clone it, and send it to the model API
    #  for preprocessing and detection
    print("[INFO] classifying {}".format(
    imagePath[imagePath.rfind("/") + 1:]))
    image = cv2.imread(imagePath)
    cv2.imwrite('image.png',image)
    (locs, preds,preds_actual) = prediction_request('image.png',source=1)

```

Step2: Loop over all the faces detected from the previous step to visualize the output in the image by plot a colored rectangle represent the mask detection output (green=Mask, red=No Mask, Blue=Incorrect Mask), plus a text above each rectangle express the class label for each face and the percentage of detection sureness. This is done through the following code:

```
    if (locs, preds):
        for (box, pred,pred_actual) in zip(locs, preds,preds_actual):
            # unpack the bounding box and predictions
            (startX, startY, endX, endY) = box
            print(pred)
            #(incorrect_Mask,mask, withoutMask) = pred
            if np.argmax(pred)==0:
                label = "incorrectMask"
                color = (255, 0, 0)
            elif np.argmax(pred)==1:
                    label = "Mask"
                    color = (0, 255, 0)
            elif np.argmax(pred)==2:
                    label = "withoutMask"
                    color = (0, 0, 255)
            else:
                label = ""
                color = (0, 0, 0)
            # include the probability in the label
            label= "{}: {:.2f}%".format(label, max(pred) * 100)
            face = image[startY:endY, startX:endX]
            # display the label and bounding box rectangle on the output
            # frame
            cv2.imwrite(f"face'{i}'.png",face)
            cv2.putText(image, label, (startX, startY - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
            cv2.rectangle(image, (startX, startY), (endX, endY), color, 2)
            # Save the output image
    cv2.imwrite(f"image_MaskDetection'{label}''{i}'.png",image)
    i=i+1
```

step3: after saving all images in the directory with its mask detection results, the system call the **prepare_and_save_report** function to create a report for the location of detected faces, class label and also return a summary of mask compliance and a Pie chart to represent the results.

```
        i=i+1
        #cv2.imshow("Output", image)       You, 7 days ago • Model Evaluation and Tsting+reports
        cv2.waitKey(0)

if __name__ == "__main__":
    mask_image()
prepare_and_save_report(source=1)
```

Part2: Real time mask detection:

In real time mask detection, the user can easily use the system as mask compliance surveillance in real time. This is done by using open-cv library to enable the video to capture the images, then analyses and visualize the detection output on the monitor. In addition to return a daily report shows the results during the day or session plus a summary report and pie chart that represent a summary of visitor or customer compliance of correct mask wearing. This done by following steps:

Step1: initialize the camera **video_capture** to take shots and send them through a request to make the detection by the model. The model returns the face locations in the image with its corresponding detection output. The video capture is initialized in a separate thread saving the generated frames from camera in a queue to get the latest frame from the camera and that allow us to be always updated with the real time video and solve noticeable delay in the process of detection.

```
class VideoCapture:

    def __init__(self, name):
        print("[INFO] starting video stream...")
        self.cap = cv2.VideoCapture(name)
        self.q = queue.Queue()
        t = threading.Thread(target=self._reader)
        t.daemon = True
        t.start()

    # read frames as soon as they are available, keeping only most recent one
    def _reader(self):
        while True:
            ret, frame = self.cap.read()
            if not ret:
                break
            if not self.q.empty():
                try:
                    self.q.get_nowait()   # discard previous (unprocessed) frame
                except queue.Empty:
                    pass
            self.q.put(frame)

    def read(self):
        return self.q.get()

    # initialize the video stream and allow the camera sensor to warm upwdd
cap = VideoCapture(0)
```

Step2: using “infinit while loop” to cosequently taking shots and sending them through the detection request to the model. Then the model return the response contains the location of faces in the shot and the mask detection output for each face. The output then represented in the monitor screen showing a colored rectangle on each face with a text above the rectangle shows the detection output result for each face. This allow the user to easily identify the status of the mask of the customers or visitors. This done as shown in the code below:

```
# loop over the frames from the video stream
while True:
    frame = cap.read()
    frame = imutils.resize(frame, width=400)
    cv2.imwrite('imag.png',frame)
    # send detection request
    (locs, preds,preds_actual) = prediction_request('imag.png')
    # loop over the detected face locations and their corresponding detections
    if (locs, preds):
        for (box, pred,pred_actual) in zip(locs, preds,preds_actual):
            # unpack the bounding box and predictions
            (startX, startY, endX, endY) = box
            if np.argmax(pred)==0:
                label = "incorrectMask"
                color = (255, 0, 0)
            elif np.argmax(pred)==1:
                label = "Mask"
                color = (0, 255, 0)
            elif np.argmax(pred)==2:
                label = "withoutMask"
                color = (0, 0, 255)
            # include the probability in the label
            label= "{}: {:.2f}%".format(label, max(pred) * 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)      You, 2 weeks ago • updat
    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
```

Step3: at the end of the session, when the user shutdown the system, the system calls the **prepare_and_save_report** function that is automatically returns and save a csv report showing all detection results. In addition to provide summary report for mask adherence status and a Pie chart for the mask compliance saved as PNG file.

```
    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    # if the `q` key was pressed, break from the loop
    if chr(cv2.waitKey(1)&255) == 'q':
        break
# do a bit of cleanup
cv2.destroyAllWindows()
prepare_and_save_report()
```

Part3: Prepare and save the report

This part is responsible for retrieving the data from the database and then convert it into a proper formatted report saved as csv file in the user directory. In addition to create a summary report and Plot a Pie chart on mask adherence through a specific session. This is done when **prepare_and_save_report** function is called at the end of the day or session.

This achieved by following steps:

Step1: retrieve the data by sending a request to firebase. During periods of no internet, the system is automatically switched to localdb. The **json_to_csv_file** function is called to convert the json response from API and save it as csv file.

```python
def prepare_and_save_report(source=0):
    report=get_report_request_from_firebase(source)
    if report['result']:
        csv_format_report=json_to_csv_format(report)
        date_time=datetime.datetime.now()
        csv_format_report.to_csv(f"Report'{date_time.date()}'.csv")
        Summary = csv_format_report.groupby('maskdetection',as_index=False).count()
        Summary=pd.DataFrame(Summary)
        Summary.drop(labels='prediction',axis=1,inplace=True)
        print(Summary)
        try:
            mask_total=Summary.query('maskdetection == "Mask"')['facelocation'].iloc[0]
        except:
            mask_total=0

        compliance_percentage=int((mask_total/Summary['facelocation'].sum())*100)
        compliance=np.array(Summary['facelocation'])
        labels=np.array(Summary['maskdetection'])
        plt.pie(compliance,labels=labels)
        plt.legend(title="compliance Pie Chart")
        plt.savefig('compliance Pie Chart')
        plt.show()

        Summary.loc[len(Summary.index)] = ['compliance_percentage',f"{compliance_percentage}%"]
        Summary.set_index('maskdetection',inplace=True)
        Styled_Summary=Summary.head(4).style.background_gradient(cmap='Blues')

        dfi.export(Styled_Summary, "Summarysample2.png")
        print(f"compliance_percentage={compliance_percentage}%")
```

Step2: the developed function to convert json to CSV file (**json_to_csv_file** function) is responsible for converting the json response to datafram using **pd.json_normalize** method and returning the report in a proper formate to be saved as shown in the code below :

```python
def json_to_csv_format(report):
    df=pd.json_normalize(report['result'])
    df.to_csv(f"x.csv")
    x=df.columns
    df=df.transpose()
    df["date"]=x
    out=df['date'].str.split('.',expand=True)
    out['time']=out[1]+"."+out[3]+"."+out[5]
    out['pred']=out[6]
    out.drop([0,1,2,3,4,5,6], axis=1,inplace=True)
    out["output"]=df[0].values
    print("adding output",out)
    out.dropna( axis=0,inplace=True)
    try:
        out=out.pivot(index='time',columns='pred',values='output')
        out=out.explode(["facelocation","maskdetection","prediction"],ignore_index=False)
        print(out)
        out.style.format()
    except:
        pass
    df_styled2=out.head(10)
    dfi.export(df_styled2, "reportsample3.png")
    return out
```

Step2: save the report as csv file and create a summary report that shows the total counts for each Mask status in addition to the mask complience percentage plus the Pie chart that visualize the result. This will allow the user to check the mask detection commitment during a day or session.

Table 2 : sample for csv report file returned at the end of day or session

pred	facelocation	maskdetection	prediction
time			
19.27.0	[122, 110, 249, 236]	No Mask	[1.8092134168308342e-14, 3.088309918957761e-16...
19.27.1	[121, 109, 249, 237]	No Mask	[6.280463430270939e-16, 5.735089686652682e-18,...
19.27.45	[122, 120, 249, 247]	No Mask	[5.981581202574304e-11, 1.9574429967542106e-14...
19.27.46	[122, 121, 248, 248]	No Mask	[2.8259354925563862e-11, 1.8664731828784038e-1...
19.27.47	[122, 122, 247, 247]	No Mask	[1.2672593877049998e-11, 3.596417529560213e-14...
19.27.48	[121, 120, 249, 247]	No Mask	[1.5857276700493445e-13, 1.2808980293709177e-1...
19.27.49	[121, 116, 247, 243]	No Mask	[1.7313163706497714e-14, 6.612637947089523e-19...
19.27.50	[129, 115, 258, 244]	No Mask	[1.7470056654702313e-15, 1.3119841385351505e-1...
19.27.51	[131, 116, 256, 242]	No Mask	[4.103497056040517e-13, 9.867071979488768e-16,...
19.27.52	[126, 114, 248, 236]	No Mask	[6.025873701930029e-15, 6.10156905851458e-19, ...

A sample for the exported csv report file is shown in the table above that shows the time of the detections in addition to the detected faces locations in the shot and the mask detection output of each.

Table 3 : A sample of summary report returned to the user at the end of day or session.

pred	facelocation
maskdetection	
Mask	12
No Mask	25
incorrect Mask	5
compliance_percentage	28%

A sample of summary report presented in the above table, it retunes a count summary for each labeled class and the compliance percentage for the (with mask) status vs. the total detected faces.

The figure below shows the Pie chart which is visualize the summary report to easily notice the compliance percentage.

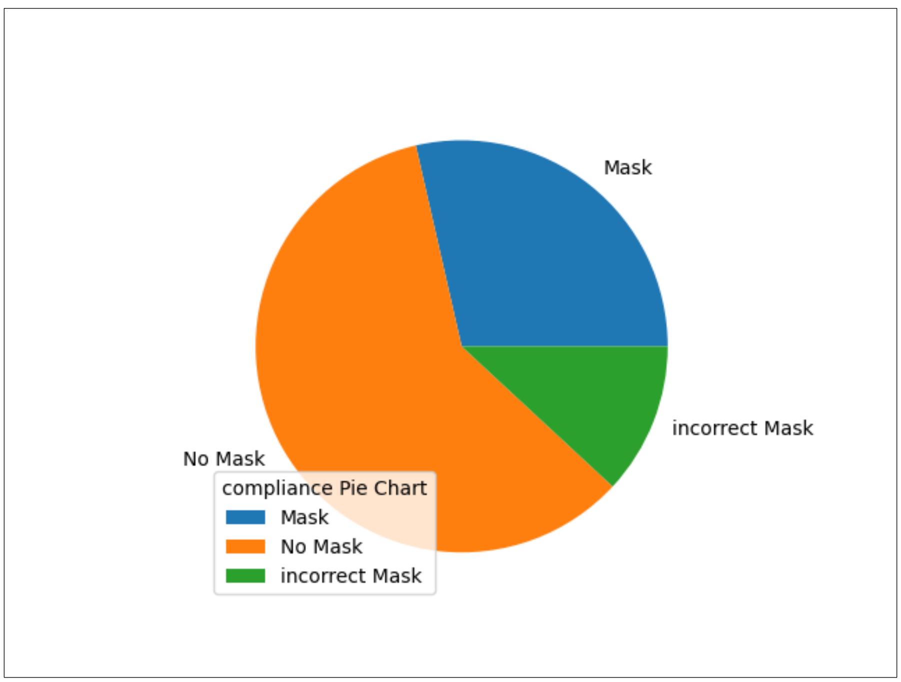

Figure 11 : A sample of Pie chart representation of the summary report during a specific period of time

Part4: Model & databases API requests:

This section consists of the functions and methods that is responsible to do specific API request tasks and prepare the data for saving or retrieving purposes. A separate threads is used for requests to save the data on firebase and localdb. This will improve system response, avoid the delay in the system process, and increase the performance of the system during the real live mask detection sessions.

These tasks included requests to the server by using the requests libraries to save or retrieve the data from localdb and request to make mask detection through the API to the model.

The code below listed the request functions developed to make the requests. For more code details refer to Appendix B

```
def prepare_data_for_firebase(data,pred_datetime,source):···

def firebase_task(data,pred_datetime,source):  ···

def prepare_table_for_SQlite3(source):  ···

def prepare_data_for_SQlite3(data,pred_datetime,source):···

def SQlite3_task(data,pred_datetime,source):···


def prediction_request(frame,source=0):···
def prepare_ref_to_get_firebase_report(source):      ···
def get_report_request_from_firebase(source=0):···
def prepare_ref_to_get_localdb_report(source):···

def get_report_request_from_localdb(source=0):···
```

4.3.1.3. API code:

The application programming interface (API) is the interface part that offering a services with functions and routs to allow the user to communicate the server. To make connection, close connection, make detection or to save and retrieve data, this developed in three part of code as following (Refer to the Appendix C,D,E respectively for the full code development):

1. Firebase API.

2. SQlite API.

3. Model API.

5.System testing and evaluation

The final and most important step after building the system is E2E testing which involves testing the system workflow from beginning to end. This method basically aims to replicate real user scenarios so that the system can be validated for integration and data integrity. [15]

Essentially, the test goes through every operation the system can perform to test how it communicates with hardware, network connectivity, external dependencies, databases, and other features.

An E2E testing features is done to determine if various dependencies of the system are working accurately, plus to check if accurate information is being communicated between multiple system components as following:

1. The ability of the system to deal with no internet connections with no fails and retrieve the desired output for the user correctly.
2. The system saves and retrieves the data and prepare the reports in a proper way.
3. The system didn't return false results if the space is empty (no faces detected).
4. The ability of the system to save the data (without loss the data) during periods of no internet connection.
5. The system automatically switches between firebase and localdb to retrieve the data depending on internet connectivity status.
6. The system is detecting the faces in a shot and return the location of faces correctly.
7. The system can deal with images of low resolution or blurry images.
8. The system detects the mask on faces correctly and efficiently.
9. The system visualizes the result in real time scenarios correctly, and with high performance and unnoticeable processing time delay.

The E2Esvystem testing of the features listed above shows that the system performs well and achieving high-performance results in different situations and scenarios.

A sample of the detection results for archived images and the real time monitoring is shown in the two figures below:

Figure 12 : A sample of mask detection system output from archived images directory.

Figure 13 : A sample screen shots for the real live mask detection system outputs.

6.Results and conclusions

The work in this project came in response to the COVID-19 pandemic and the need for having close monitoring of the compliance of wearing face masks to minimize risk of disease spread between people. It aimed at developing and testing an end-to-end face mask detection system and provide a user-friendly model to monitor correct wearing of face masks especially for crowded areas. The developed system works mainly for real time conditions (i.e live cameras), in addition to have the capacity to process archived images from different sources and providing results on compliance of wearing face masks.

The development of this system went through several steps which lead to having a functional system consisting of several interlinked components to do the job of face mask detection and providing reports to the user. The developed model has been built using transfer learning by fine tuning VGG19 pre-trained model, and FMD dataset from Kaggle to train the model. This data set consists of more than 14 thousand images classified into three classes (no mask, in-correct mask, and mask). The model had been tested using different Metrics (validation accuracy, validation loss, confusion matrix, recall, precision, f1score and ROC curve) where it inferred good results (val_accuracy reached 98.25%). Additionally, to ensure the performance of the system, an E2E testing had been conducted. The testing results concluded a good performance of the system.

The system provides three ways of reports to the user which are, main report, summary report, and Pie chart. This should allow users to get information showing the percentage of compliance in wearing face masks, and hence, allow the user to amend/maintain the security procedures to ensure efficient compliance. Besides the evident requirement for COVID situation, the developed system can be further used for situations or working environments that require wearing face masks such as laboratories.

It is worth mentioning that the performance of the system is correlated to the used model for face detection. The system performance increases with the increased accuracy of the face detection model.

7.References

[1] P. M. ,. S. Suresh K1, "Face Mask Detection by using Optimistic Convolutional Neural Network," in *Proceedings of the Sixth International Conference on Inventive Computation Technologies [ICICT 2021]IEEE Xplore Part Number: CFP21F70-ART; ISBN: 978-1-7281-8501-9*, 3 Department of Computer Science, Amrita School of Arts and Sciences, Mysore, Amrita Vishwa Vidyapeetham, India, 2021.

[2] W. H. O. (WHO). [Online]. Available: https://www.who.int/teams/health-product-policy-and-standards/standards-and-specifications/vaccine-standardization/coronavirus-disease-(covid-19)#:~:text=Coronavirus%20disease%20(COVID%2D19). [Accessed 23 August 2022].

[3] P. M. B. S. Suresh K, "Face Mask Detection by using Optimistic Convolutional Neural Network," in *Proceedings of the Sixth International Conference on Inventive Computation Technologies*, 2021.

[4] "Kaggle," CC0: Public Domain, [Online]. Available: https://www.kaggle.com/datasets/shiekhburhan/face-mask-dataset?datasetId=2269875&sortBy=dateRun&tab=profile. [Accessed 24 Aug. 2022].

[5] MediaPipeSolutions, Writer, *MediaPipe Face Detection.* [Performance]. GOOGLE, 2020.

[6] Y. K. A. V. K. R. M. G. Valentin Bazarevsky, "BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs," *Google Research,* p. 4, 2019.

[7] Amazon. [Online]. Available: https://aws.amazon.com/what-is/api/#:~:text=API%20stands%20for%20Application%20Programming,other%20using%20requests%20and%20responses.. [Accessed 25 August 2022].

[8] S. M. A. H. A. M. Safa Teboulbi, "Real-Time Implementation of AI-Based Face Mask Detection and Social Distancing Measuring System for COVID-19 Prevention," *Research Article,* p. 21, 2021.

[9] d. Rosebrock, "fine-tuning-with-keras-and-deep-learning," 22 July 2019. [Online]. Available: https://pyimagesearch.com/2019/07/22/keras-learning-rate-schedules-and-decay/.

[10] a. Brownlee, "machinelearningmastery," 15 Aug 2022. [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/. [Accessed 27 Aug 2022].

[11] D. P. K. a. J. L. Ba, "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION," *arXiv:1412.6980v9 [cs.LG] 30 Jan 2017,* 30 Jan 2017.

[12] J. Brownlee, "machinelearningmastery," 25 Aug 2020. [Online]. Available: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/. [Accessed 27 Aug 2022].

[13] A. Rosebrock, "keras-learning-rate-schedules-and-decay," 22 July 2019. [Online]. Available: https://pyimagesearch.com/2019/07/22/keras-learning-rate-schedules-and-decay/.

[14] "scikit-learn," scikit-learn.org, [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html. [Accessed 27 Aug 2022].

[15] S. Bose, "browserstack.," browserstack., 19 May 2021. [Online]. Available: https://www.browserstack.com/guide/end-to-end-testing.

[16] arbazkhan971, "https://www.kaggle.com/code/arbazkhan971/face-mask-detection-using-cnn-98-accuracy/notebook," 2021. [Online].

[17] O. E. N. A. S. A.-M. A. M. T. K. K. A. Najmath Ottakath, "ViDMASK dataset for face mask detection with social distance measurement," *ISSN 0141-9382,* p. 6, 2022.

[18] A. Rosebrock, "pyimagesearch," 3 Jun 2019. [Online]. Available: https://pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/. [Accessed 27 Aug 2022].

[19] C. K. Yang, "ResearchGate," [Online]. Available: https://www.researchgate.net/figure/llustration-of-the-network-architecture-of-VGG-19-model-conv-means-convolution-FC-means_fig2_325137356 . [Accessed 28 Aug. 2022].

8.Appendixes:

Appendix A:

Model Summary

Model: "model"

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
```

block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv4 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv4 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808

block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv4 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3211392
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 3)	387

Total params: 23,236,163

Trainable params: 3,211,779

Non-trainable params: 20,024,384

Appendix B:

```python
def prepare_data_for_firebase(data,pred_datetime,source):

    master="/RealTimeMaskDetectionReport"
    if source==1:
        master="/ArchiveImagesMaskDetectionReport"
    ref={
        "master":master,
        "year":"/year/"+str(pred_datetime.year),
        "month":"/month/"+str(pred_datetime.month),
        "day":"/day/"+str(pred_datetime.day),
        "hour":"/hour/"+str(pred_datetime.hour),
        "minute":"/minute/"+str(pred_datetime.minute),
        "second":"/second/"+str(pred_datetime.second)+"/"
    }
    json_dic={
        "facelocation":data['locs'],
        "prediction":data['preds'],
        "maskdetection":data['preds_actual']
    }
    return ref,json_dic

def firebase_task(data,pred_datetime,source):
    print("threadtask",data)
    if any(data.values()):
        print("nullpart",data['locs'])
        ref,json_dic=prepare_data_for_firebase(data,pred_datetime,source)

        json_dic=json.dumps({'json_dic':json_dic,'ref':ref})
        headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
        url ="http://localhost:5001/InsertData"
        res=requests.post(url,data=json_dic,headers=headers)
        print(res)

def prepare_table_for_SQlite3(source):
    table_name="RealTimeMaskDetection"
    if source==1:
        table_name="ArchiveImagesMaskDetection"
    json_dic={
        "db":"MaskDetection",
        "TableName":table_name,
        "Columns":"ID INTEGER PRIMARY KEY AUTOINCREMENT ,year varchar(255) ,month varchar(255),day varchar(255),hour
        }
    return json_dic

def prepare_data_for_SQlite3(data,pred_datetime,source):

    table_name="RealTimeMaskDetection"
    if source==1:
        table_name="ArchiveImagesMaskDetection"

    json_dic={
            "db":"MaskDetection",
            "TableName":table_name,
            "Columns":"year ,month ,day ,hour ,minute  ,second ,locations ,predictions,maskdetection ",
            "Values": {
                     "year": pred_datetime.year,
                    "month":pred_datetime.month,
                    "day":pred_datetime.day,
                    "hour":pred_datetime.hour,
                    "minute":pred_datetime.minute,
                    "second":pred_datetime.second,
                    "locations":data['locs'],
                    "predictions":data['preds'],
                    "maskdetection":data['preds_actual']
            }
            }
    return json_dic
```

```python
def SQlite3_task(data,pred_datetime,source):
    if any(data.values()):

        url ="http://localhost:5002/CreateTable"
        headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
        json_dic=prepare_table_for_SQlite3(source)

        res=requests.post(url,json=json_dic,headers=headers)
        print(res)

        if res:
            json_dic=prepare_data_for_SQlite3(data,pred_datetime,source)
            url ="http://localhost:5002/InsertData"
            headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
            res=requests.post(url,json=json_dic,headers=headers)
            print(res)

def prediction_request(frame,source=0):
    #Web Service Prediction:
    url ="http://127.0.0.1:5000/PredictMask"
    try:
        with open(frame, "rb") as f:
            im_bytes = f.read()
        im_b64 = base64.b64encode(im_bytes).decode("utf8")
        json_im = json.dumps({'image': im_b64})
        headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}

        response = requests.post(url, data=json_im,headers=headers)
        #print (response)
        data = response.json()
        print(data)
        #firebase_task(data)
        #SQlite3_task(data)

        pred_datetime=datetime.datetime.now()

        global firebase_thread
        global SQlite3_thread

        firebase_thread= Thread(target=firebase_task,args=(data,pred_datetime,source))
        SQlite3_thread= Thread(target=SQlite3_task,args=(data,pred_datetime,source))
        firebase_thread.start()
        SQlite3_thread.start()

        return data['locs'],data['preds'],data['preds_actual']

def prepare_ref_to_get_firebase_report(source):
    pred_datetime=datetime.datetime.now()
    master="/RealTimeMaskDetectionReport"
    if source==1:
        master="/ArchiveImagesMaskDetectionReport"
    ref={
            "master":master,
            "year":"/year/"+str(pred_datetime.year),
            "month":"/month/"+str(pred_datetime.month),
            "day":"/day/"+str(pred_datetime.day),
        }
    return ref
def get_report_request_from_firebase(source=0):
    firebase_thread.join()

    try:

        ref=prepare_ref_to_get_firebase_report(source)
        json_dic=json.dumps({'ref':ref})
        headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
        url ="http://localhost:5001/GetbyDate"
        response = requests.get(url, data=json_dic,headers=headers)
        response=json.loads(response.text)
        print(response,"this is request report response")
    except:
        print("cannot retreive the report... check internet connection and try again.")
        response=json.dumps({'result':0})
        response=json.loads(response)
    return response      You, 2 weeks ago • handling exceptions …
```

```python
def get_report_request_from_localdb(source=0):
    SQlite3_thread.join()
    try:
        json_dic=prepare_ref_to_get_localdb_report(source)
        url ="http://localhost:5002/GetbyDate"
        headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
        response = requests.get(url, json=json_dic,headers=headers)
        print("response from local db in detection request",response)
        print(response.text)
        response=json.loads(response.text)

    except:
        traceback.print_exc
        print("retreiving report.... please keep connection on...")
        response=json.dumps({'result':0})
        response=json.loads(response)
    return response
```

Appendix C:

```python
class FirebaseDataBase:
    def __init__(self): ...

    def connect_to_database(self,databaseURL): ...


    def insert_data(self, ref,data): ...

    def get_data(self, ref): ...

    def get_data_by_Date(self, ref):    ...

    def Delete_by_Date(self, ref):    ...


    def close_connection(self): ...



#Fetch All DB Data
@app.route('/Get_Data',methods=['GET'])
def Get_Data(): ...
```

```python
@app.route('/Get_Data',methods=['GET'])
def Get_Data(): ...


#get Data for a specified Date
@app.route('/GetbyDate',methods=['GET'])
def GetbyDate(): ...


#delete from DB
@app.route("/Delete",methods=['DELET'])
def Delete(): ...
            You, 59 seconds ago • Uncommitted changes

#Insert new item to DB
@app.route('/InsertData',methods=['POST','GET'])
def InsertData(): ...


if __name__ == '__main__':
    try:
        app.run(debug=True, host='0.0.0.0', port=5001)
        data=dict
        ref=""
        event = Event()

    except:
        print("checking server connection...")
```

Appendix D:

```python
--
script_dir = os.path.dirname( __file__ )
mymodule_dir = os.path.join( script_dir, '..', 'Model')
sys.path.append( mymodule_dir )
import FaceMaskDetector

app = Flask('app')
app.logger.setLevel(logging.DEBUG)

@app.route('/PredictMask', methods=['POST','GET'])
def predict_mask():
    # print(request.json)
    if not request.json or 'image' not in request.json:
        abort(400)
    im_b64 = request.json['image']
    #json = bytearray(request.get_json())
    img_bytes = base64.b64decode(im_b64.encode('utf-8'))
    # convert bytes data to PIL Image object
    img = Image.open(io.BytesIO(img_bytes))
    img_arr = np.asarray(img)

    #file={'image': open(image,'rb')}
    #img=np.array(image.open(file.stream))
    #print(img_arr)
    locs, preds,preds_actual = FaceMaskDetector.FaceMaskDetector.detect_and_predict_mask(img_arr)
    #print(locs)
    #print(preds)

    result = {
        'locs': locs,
        'preds': preds,
        'preds_actual':preds_actual
    }
    return result       You, 2 weeks ago • updated code 15/8/2022 …


if __name__ == '__main__':
    app.run(debug=True, host='127.0.0.1', port=5000)
```

Appendix E:

```
app = Flask('app')


You, 2 weeks ago | 1 author (You)
class LocalDataBase:
    def __init__(self, dbname): ···


    def create_table(self, tbname, columns): ···

    def insert_data(self, tbname, columns, values): ···

    def get_data(self, tbname,colum,value): ···

    def close_connection(self): ···


@app.route('/CreateTable', methods=['POST','GET'])
def CreateTable(): ···

@app.route('/InsertData', methods=['POST','GET'])
def InsertData(): ···

@app.route('/GetbyDate', methods=['POST','GET'])
def GetbyDate(): ···
        You, 3 weeks ago • Final Project repository created …

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5002)
```