Face mask detection

Team members:
Vadlamudi Sai Lokesh
Kotha Sai narasimha rao
Baratam gourav
Vadigi milith
Sumanth Gunda

Problem statement

The goal of this challenge is to train object detection models capable of identifying the location of masked faces in an image as well as the location of unmasked faces in the image. These detectors should be robust to noise and provide as little room as possible to accommodate false positives for masks due to the potentially dire consequences that they would lead to. Ideally, they should be fast enough to work well for real-world applications, something we hope to focus on in future rounds of the competition

Dataset

The dataset that would be used is a growing dataset from a combination of many sources, either hand-labeled or pseudo-labeled. The dataset would contain annotations for masked faces, unmasked faces as well as some images that don't have any annotations as would be the case in the real-world scenario.

Motivation

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

Features

Our face mask detector didn't use any morphed masked images dataset. The model is accurate, and since we used the MobileNetV2 architecture, it’s also computationally efficient and thus making it easier to deploy the

model to embedded systems (Raspberry Pi, Google Coral, etc.). This system can therefore be used in real-time applications that require face-mask detection for safety purposes due to the outbreak of Covid-19.

Tools Used for building the model:
NumPy : Used for storing and manipulating high dimensional arrays.
Matplotlib : Used for plotting.
OpenCV : Used for manipulating images and video streams. Keras : Used for designing and training the Face_Mask_Classifier model.
Scikit-Learn : used for train-test and perform one-hot encoding on the labels.
imutils : A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much easier with OpenCV and both Python 2.7 and Python 3.
numpy : used to convert the data into an array format.

The solution we have used for this model:

In order to train this dataset, I used a pre-trained model “MOBILE NET”, which we call this process as Transfer learning, where I removed the prediction layer for this model. I used ‘IMAGENET’ which helps to enhance the model to avoid problems like overfitting and less accurate situations. Each image is resized into 244*224 dimension.
Another main important aspect I used to avoid overfitting is DATA AUGMENTATION. By using the IMAGE DATA GENERATOR present in “tf.keras.preprocessing.image.ImageDataGenerator” I have generated augmented images.
aug = ImageDataGenerator(rotation_range = 20, zoom_range = 0.15, width_shift_range = 0.2, height_shift_range = 0.2, sheer_range = 0.15, horizantal_flip = True, fill_mode = ‘nearest’)”
The network is trained at 5 epochs, It achieved an accuracy of 0.9806 and val_accuracy of 0.9883, the loss is 0.0620, val_loss is 0.0403 At the model hierarchy we included a mobile-net model, next I added AveragePooling2D(), Flatten(), Dense(), Dropout() layers. The prediction layer contains 2 neurons with ‘softmax’ as an activation function. In model

compile, adam with learning rate 0.0001, binary_crossentropy is used as loss function, accuracy is used as metrics.

Steps to be performed before Data Preprocessing

Import the libraries required for the model.
load images from the local folder.
A total there are 1900 mask images and 1900 without mask images are available in the folder.
After loading apply preprocess_input to every image because here for developing the model we are using 'MOBILE NET'.
Create the two lists(data and labels) one is used to store image arrays and another one used to store what type of images are present in the data list according to a specific position.
(suppose if the image in data array at 1 loc is belonged to without mask then the keyword “without mask ” is stored at 1st location of the labels list.

Data preprocessing

1.perform one-hot encoding on the labels list by using LabelBinarizer.
2.covert the two lists (data and labels) into array format by using NumPy
.
3.Here we are converting the two lists into array format because of the train-test split.
4.perform the train_test_split on the data array and list array for training and testing the model with random_state=0 and train size 80 percent and test size 20 percent of the dataset.

2.Model Building :

1.constructing the head layer of the cnn:

1.For building the model we have used Mobile net to classify the images.
Mobile Nets: Efficient Convolutional neural networks for mobile vision applications.
We shall be using Mobilenet as it is lightweight in its architecture. It uses

depthwise separable convolutions which basically means it performs a single convolution on each color channel rather than combining all three and flattening it. This has the effect of filtering the input channels.
Next we are adding the Average pooling 2d layer to head of the neural network.

Average pooling involves calculating the **average** for each patch of the feature map. This means that each 2×2 square of the feature map is downsampled to the **average** value in the square.

Flattening is to be done after pooling. Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.

The construction of the head layer is completed.

2.Adding dense layers to the neural network:

A **Dense layer** feeds all outputs from the previous **layer** to all its neurons, each The neuron Providing one output to the next layer.It is the most basic layer in neural networks.

Here we are adding a dropout layer to the neural network.

Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden l layers) to 0 at each update of the training phase.

Here we were used the softmax function in the output layer to predict the accuracy of Outcome whether the image belongs to with mask category or without mask category.

be updated during the first training process

5.compile the model.

3.Training the model

Train the model with 5 no of epochs.

```
# train the head of the network
print("[INFO] training head...")
H = model.fit(
	aug.flow(trainX, trainY, batch_size=BS),
	steps_per_epoch=len(trainX) // BS,
	validation_data=(testX, testY),
	validation_steps=len(testX) // BS,
	epochs=5)
```

4.Predictions on test data set

For each image in the testing set we need to find the index of the label with corresponding largest predicted probability.
Prepare the classification by using predictions of the model.

Accuracy:
Our model gave 93% accuracy for Face Mask Detection after training via tensorflow-gpu==2.0.0
our model is good form the test data.

```
[INFO] evaluating network...
              precision    recall  f1-score   support

   with_mask       0.99      0.86      0.92       383
without_mask       0.88      0.99      0.93       384

    accuracy                           0.93       767
   macro avg       0.93      0.93      0.93       767
weighted avg       0.93      0.93      0.93       767

[INFO] saving mask detector model...
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Detecting the face and mask in the video:

With the help face detector model, the model detects the face and with the

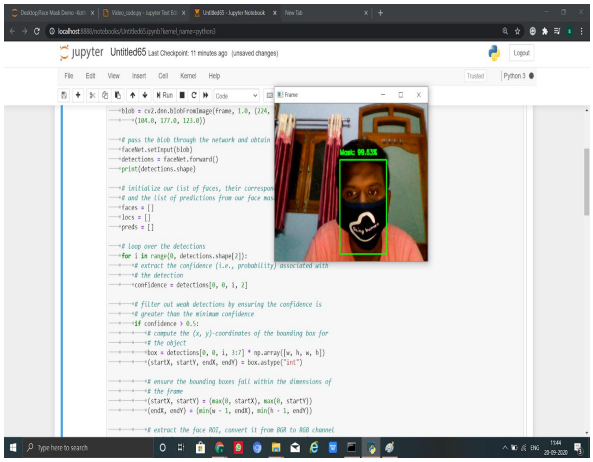
help of the mask detector model the model detects the mask. by using dnn in the cv2 library the face detector detects the face.

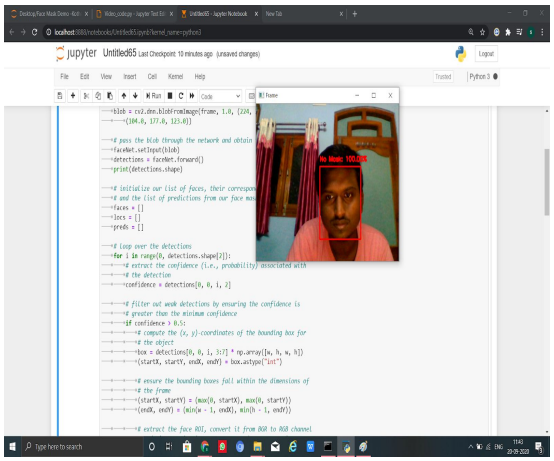
1.Connect to the webcam by using cv2 library

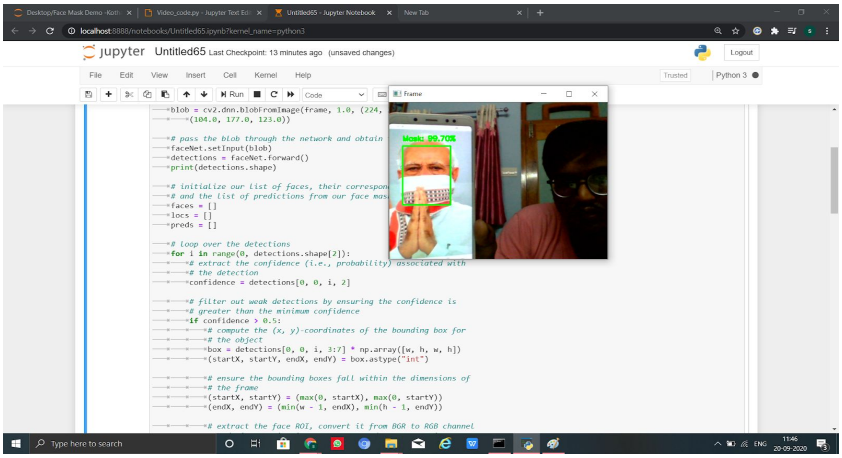
2.By using both face detector and mask detector model we conclude that if the person on the image is wearing a mask or no mask.

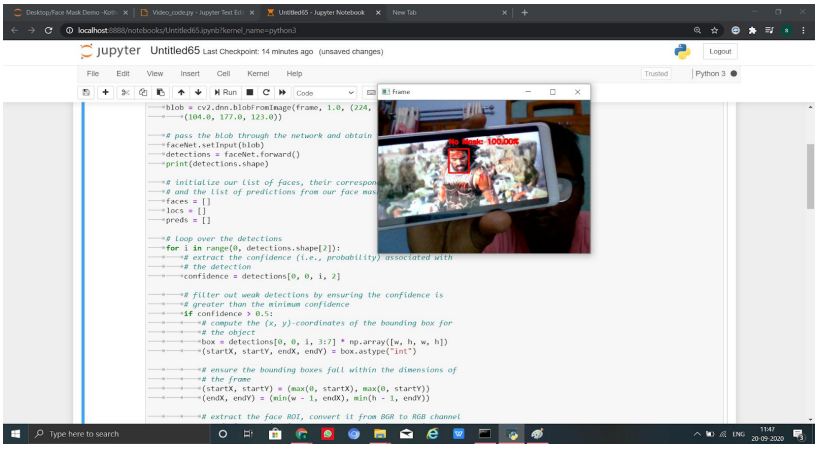
Model accuracy:93% on the test data set.

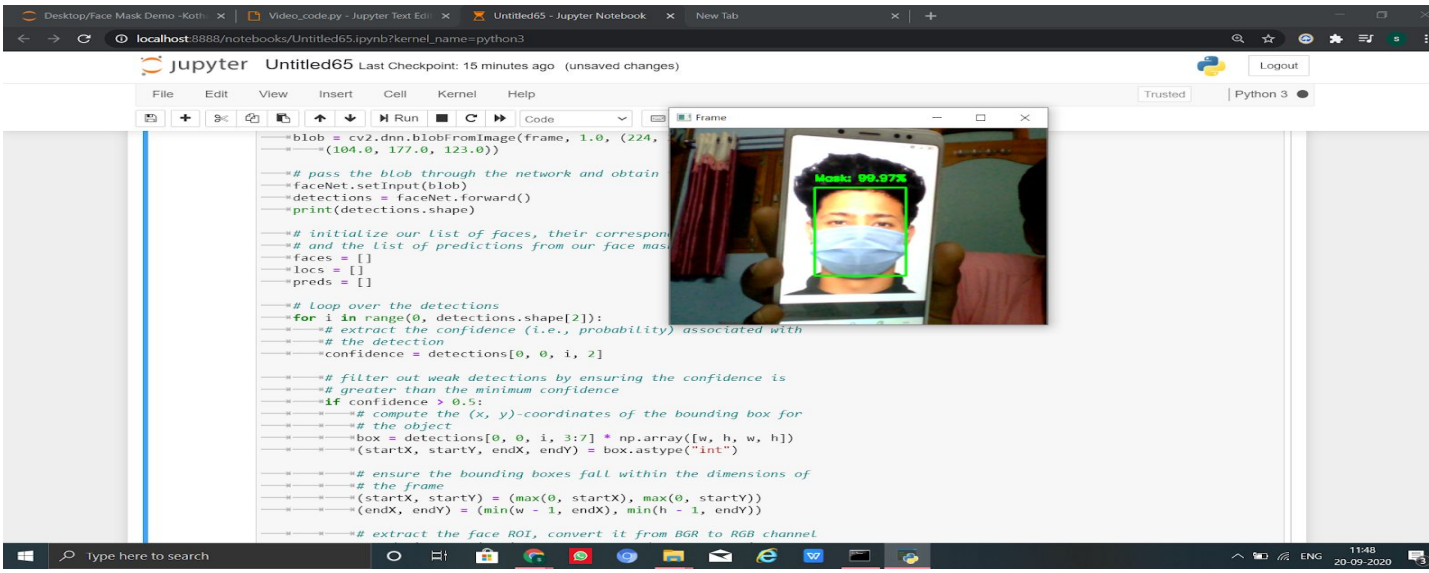
These are the sample predictions.






jupyter Untitled65 Last Checkpoint: 13 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
Frame
Mask: 99.70%
blob = cv2.dnn.blobFromImage(frame, 1.0, (224,
(104.0, 177.0, 123.0))
# pass the blob through the network and obtain
faceNet.setInput(blob)
detections = faceNet.forward()
print(detections.shape)
# initialize our list of faces, their correspon
# and the list of predictions from our face mas
faces = []
locs = []
preds = []
# loop over the detections
for i in range(0, detections.shape[2]):
# extract the confidence (i.e., probability) associated with
# the detection
confidence = detections[0, 0, i, 2]
# filter out weak detections by ensuring the confidence is
# greater than the minimum confidence
if confidence > 0.5:
# compute the (x, y)-coordinates of the bounding box for
# the object
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
(startX, startY, endX, endY) = box.astype("int")
ensure the bounding boxes fall within the dimensions of
the frame
(startX, startY) = (max(0, startX), max(0, startY))
(endX, endY) = (min(w - 1, endX), min(h - 1, endY))
extract the face ROI, convert it from BGR to RGB channel
Type here to search
ENG
11:46
20-09-2020


jupyter Untitled65 Last Checkpoint: 14 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
Frame
Mask: 100.00%
blob = cv2.dnn.blobFromImage(frame, 1.0, (224,
(104.0, 177.0, 123.0))
# pass the blob through the network and obtain
faceNet.setInput(blob)
detections = faceNet.forward()
print(detections.shape)
# initialize our list of faces, their correspon
# and the list of predictions from our face mas
faces = []
locs = []
preds = []
# loop over the detections
for i in range(0, detections.shape[2]):
# extract the confidence (i.e., probability) associated with
# the detection
confidence = detections[0, 0, i, 2]
# filter out weak detections by ensuring the confidence is
# greater than the minimum confidence
if confidence > 0.5:
# compute the (x, y)-coordinates of the bounding box for
# the object
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
(startX, startY, endX, endY) = box.astype("int")
ensure the bounding boxes fall within the dimensions of
the frame
(startX, startY) = (max(0, startX), max(0, startY))
(endX, endY) = (min(w - 1, endX), min(h - 1, endY))
extract the face ROI, convert it from BGR to RGB channel
Type here to search
ENG
11:47
20-09-2020


Desktop/Face Mask Demo -Kothi
Video_code.py - Jupyter Text Edi
Untitled65 - Jupyter Notebook
New Tab
localhost:8888/notebooks/Untitled65.ipynb?kernel_name=python3
jupyter Untitled65 Last Checkpoint: 15 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code
Frame
Mask: 99.97%
blob = cv2.dnn.blobFromImage(frame, 1.0, (224,
(104.0, 177.0, 123.0))
# pass the blob through the network and obtain
faceNet.setInput(blob)
detections = faceNet.forward()
print(detections.shape)
# initialize our list of faces, their correspon
# and the list of predictions from our face mas
faces = []
locs = []
preds = []
# loop over the detections
for i in range(0, detections.shape[2]):
# extract the confidence (i.e., probability) associated with
# the detection
confidence = detections[0, 0, i, 2]
# filter out weak detections by ensuring the confidence is
# greater than the minimum confidence
if confidence > 0.5:
# compute the (x, y)-coordinates of the bounding box for
# the object
box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
(startX, startY, endX, endY) = box.astype("int")
ensure the bounding boxes fall within the dimensions of
the frame
(startX, startY) = (max(0, startX), max(0, startY))
(endX, endY) = (min(w - 1, endX), min(h - 1, endY))
extract the face ROI, convert it from BGR to RGB channel
Type here to search
ENG
11:48
20-09-2020