INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY, SHIBPUR

Segregating Human Faces from Image Sequences using Machine Learning Techniques

by

Ankan Halder (510818008)
Aneek Ghosh (510818039)
Pranjal Sharma (510818089)

Under the guidance of
Dr. Arindam Biswas

A report submitted in fulfillment of the requirements for the mini project of Bachelor of Technology in Information Technology (5th Semester)

Department of Information Technology
Indian Institute of Engineering Science and Technology, Shibpur
P.O. Botanic Garden, Howrah 711103, WB, India
January 2021

CONTENTS

ACKNOWLEDGEMENT

We would like to extend our sincere thanks to our project guide, **Dr. Arindam Biswas**, Professor, Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, for his guidance and constant supervision without which it would not have been possible to complete this project within the due time.

We are fortunate enough to get constant encouragement, support, and guidance from **Dr. Sukanta Das**, Associate Professor and Head of the department, Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, as well as other faculty members and staff members who helped us in successfully completing our project work.

Finally, we cordially thank our fellow classmates of Information Technology, of the Indian Institute of Engineering Science and Technology, our parents, and all others who have been attached to this phase of the development of the project directly or otherwise.

INTRODUCTION

Face detection is a computer technology that determines the locations and sizes of human faces in arbitrary(digital) images.It detects facial features and ignores anything else.

We have used **Open Source Computer Vision library**.It is an open source library of programming functions mainly aimed at real time computer vision.It was originally developed by Intel.It is used for object,face and gesture recognition,lip reading,human computer interaction,motion understanding and mobile robotics.

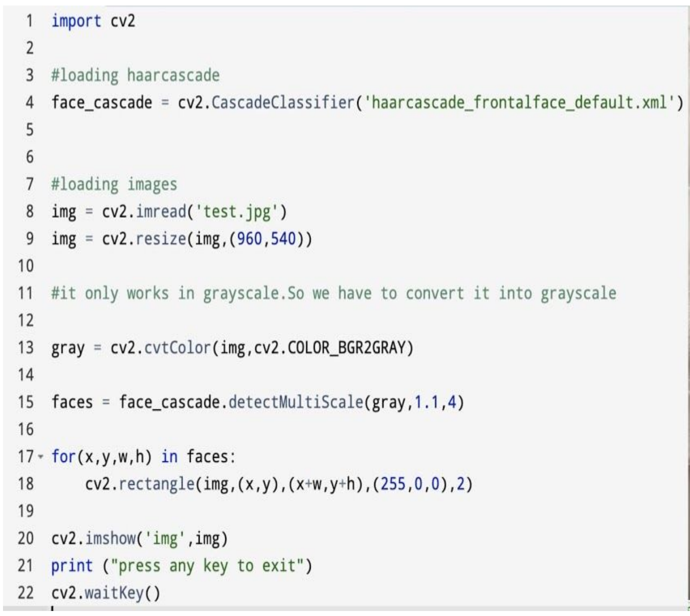
We used **Haar Cascade** for face detection, which is a robust and quick,real-time operated face detection technique with 95% accuracy devised by Paul Viola and Michael Jones in 2001.Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier. Positive images contain the images which we want our classifier to identify. Negative Images contain everything else, which do not contain the object we want to detect.

The trend of wearing face masks in public is rising due to the coronavirus (COVID-19) epidemic all over the world.So we thought of building a face-mask detector web application that will help greatly in this hard time.

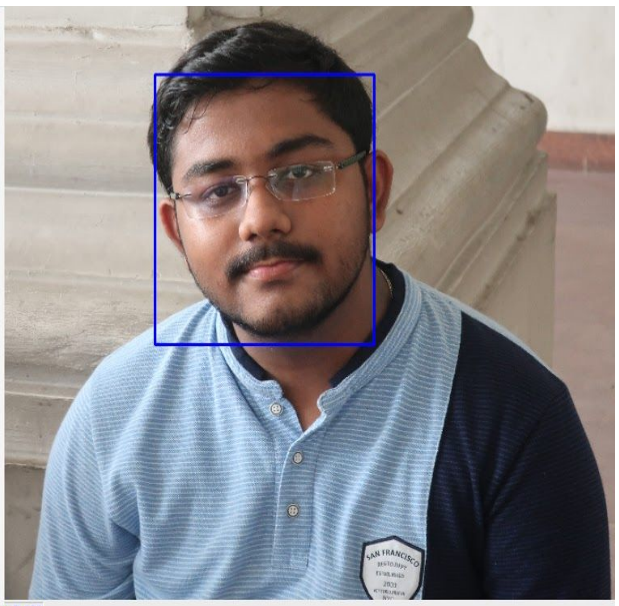
FACE DETECTION USING IMAGES

Face detection using Haar cascades is a machine learning based approach where a cascade function is trained with a set of input data. **OpenCV** already contains many pre-trained classifiers for face, eyes, smiles, etc.The face detection works only on grayscale images so it is important to convert the color image to grayscale.

The python code & output of the face detection using images:



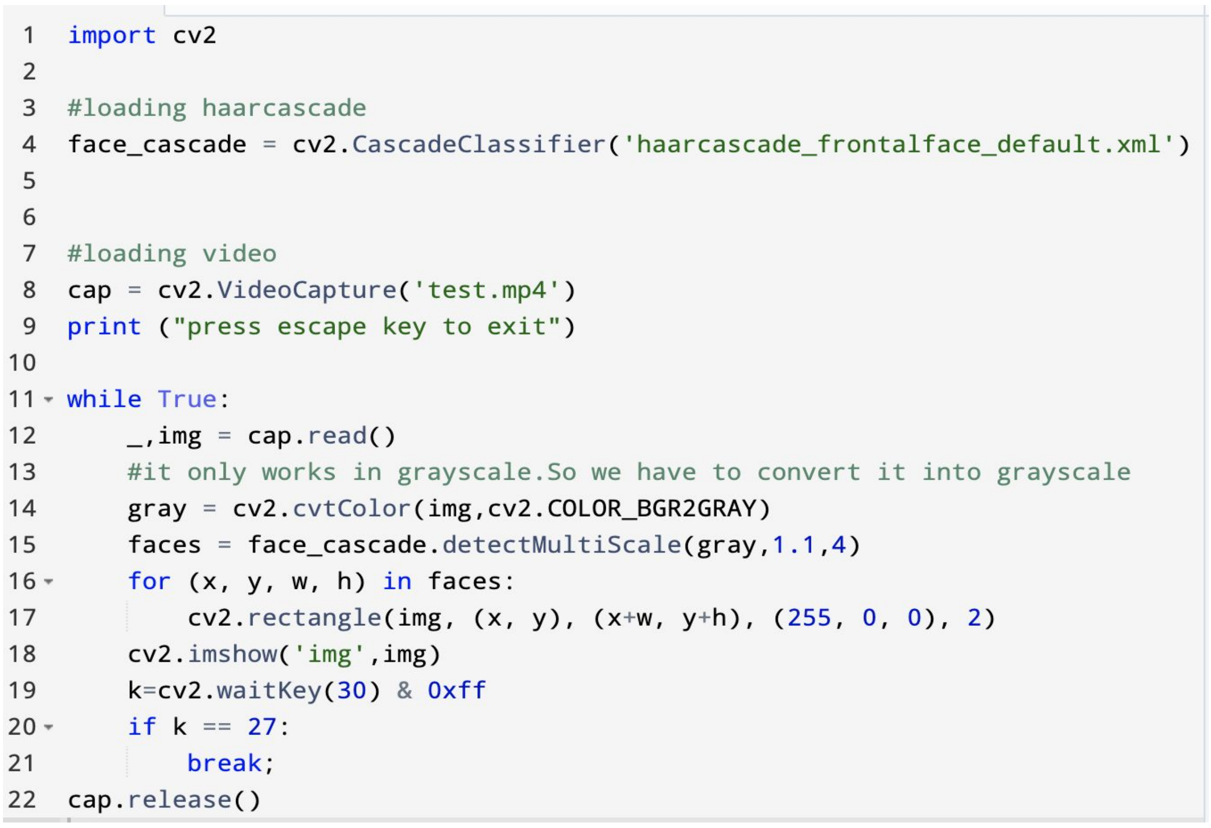
```python
import cv2

#loading haarcascade
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')


#loading images
img = cv2.imread('test.jpg')
img = cv2.resize(img,(960,540))

#it only works in grayscale.So we have to convert it into grayscale

gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray,1.1,4)

for(x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

cv2.imshow('img',img)
print ("press any key to exit")
cv2.waitKey()
```



FACE DETECTION USING VIDEOS

Similarly, we can detect faces in videos.Since videos are basically made up of frames, which are still images, so we performed the face detection for each frame in a video.The only difference here is that we use an infinite loop,which loops through each frame in the video.

The python code of the face detection using videos:

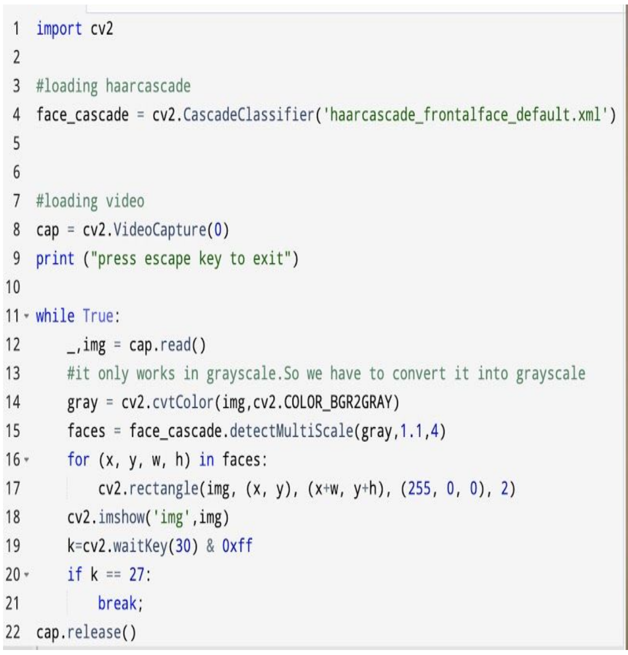


```python
import cv2

#loading haarcascade
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')


#loading video
cap = cv2.VideoCapture('test.mp4')
print ("press escape key to exit")

while True:
    _,img = cap.read()
    #it only works in grayscale.So we have to convert it into grayscale
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray,1.1,4)
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
    cv2.imshow('img',img)
    k=cv2.waitKey(30) & 0xff
    if k == 27:
        break;
cap.release()
```

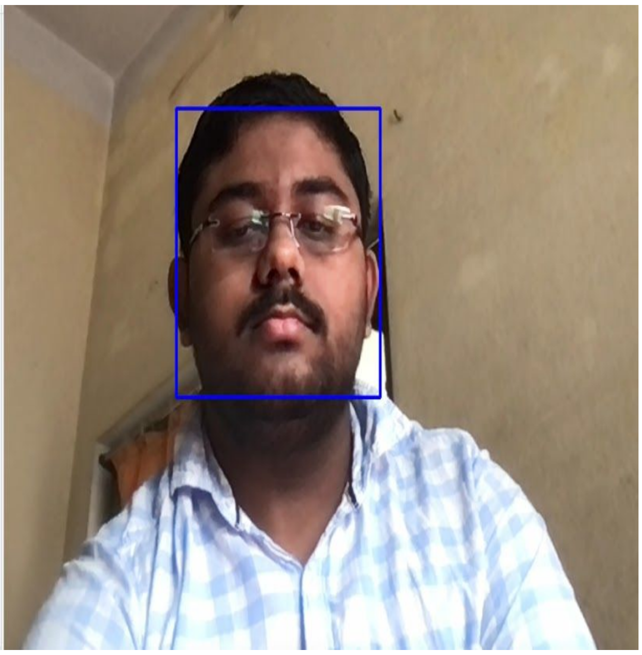
FACE DETECTION USING WEB CAMERA

In this case we are using our web camera for real time streaming and detecting the face of the corresponding user.

The python code & output of the face detection using webcam:



```python
import cv2

#loading haarcascade
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')


#loading video
cap = cv2.VideoCapture(0)
print ("press escape key to exit")

while True:
    _,img = cap.read()
    #it only works in grayscale.So we have to convert it into grayscale
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray,1.1,4)
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
    cv2.imshow('img',img)
    k=cv2.waitKey(30) & 0xff
    if k == 27:
        break;
cap.release()
```



Face Mask Detector

Our model is integration between deep learning and classical machine learning techniques with opencv, tensorflow and keras.It is divided in 2 phases.

Phase 1: Train Face Mask Detector

1.First load the face mask dataset

2.Train face mask classifier with Keras/Tensorflow

3.Save the face mask classifier to the disk

Phase 2: Apply the Face Mask Detector

1.Load the face mask classifier from disk

2.Detect image/video stream

3.Extract each face ROI(region of interest)

4.Apply face mask classifier to each face ROI to determine "mask" or "no mask"

5.Show results

Data Preprocessing

The dataset that we are using consists of images with different colors,sizes, and orientations. Therefore, we need to convert all the images into grayscale because we need to be sure that color should not be a critical point for detecting masks.Center cropping the image with the pixel value of 224x224x3.Finally converting them into tensors (Similar to NumPy array).

Details of CNN-MobileNetV2

CNN (Convolutional Neural Network) has many versions of pre-trained and well-architected networks for example AlexNet, ResNet, Inception, LeNet, MobileNet and so on. In our case we have chosen the **MobileNetV2** due to its lightweight and very efficient mobile-oriented model.

Comparison between different models:

Network	Top1%	Params	MAdds	CPU
MobileNetV1	70.6	4.2	575M	113ms
ShuffleNet(1.5)	71.5	3.4	292M	-
ShuffleNet(x2)	73.7	5.4	524M	-
NasNet-A	74.0	5.3	564M	183ms
MobileNetV2	**72**	**3.4**	**300M**	**75ms**
MobileNetv2(1.4)	74.7	6.9	585M	143ms

Layers in MobileNetV2

1. **Average Pooling** layer with 7×7 weights.

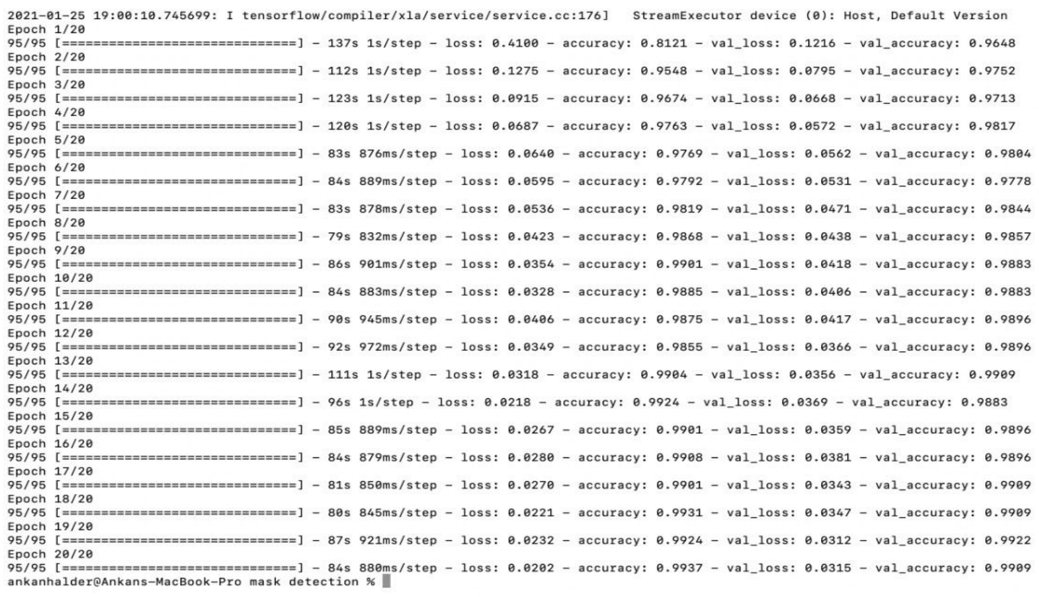
2. Linear layer with **ReLu** activation function

3. **Dropout Layer**

4. Linear layer with **Softmax** activation function gives the final result of 2 values each one represents the classification of “mask” or “not mask”.

Training with Face Mask

We have used the augmented data because we have a small dataset and done necessary preprocessing.We ran **20** Epochs with a learning rate of **1e-4** and the batch size **32**.The validation accuracy of the model is **99.89%**.



```
2021-01-25 19:00:10.745699: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecutor device (0): Host, Default Version
Epoch 1/20
95/95 [==============================] - 137s 1s/step - loss: 0.4100 - accuracy: 0.8121 - val_loss: 0.1216 - val_accuracy: 0.9648
Epoch 2/20
95/95 [==============================] - 112s 1s/step - loss: 0.1275 - accuracy: 0.9548 - val_loss: 0.0795 - val_accuracy: 0.9752
Epoch 3/20
95/95 [==============================] - 123s 1s/step - loss: 0.0915 - accuracy: 0.9674 - val_loss: 0.0668 - val_accuracy: 0.9713
Epoch 4/20
95/95 [==============================] - 120s 1s/step - loss: 0.0687 - accuracy: 0.9763 - val_loss: 0.0572 - val_accuracy: 0.9817
Epoch 5/20
95/95 [==============================] - 83s 876ms/step - loss: 0.0640 - accuracy: 0.9769 - val_loss: 0.0562 - val_accuracy: 0.9804
Epoch 6/20
95/95 [==============================] - 84s 889ms/step - loss: 0.0595 - accuracy: 0.9792 - val_loss: 0.0531 - val_accuracy: 0.9778
Epoch 7/20
95/95 [==============================] - 83s 878ms/step - loss: 0.0536 - accuracy: 0.9819 - val_loss: 0.0471 - val_accuracy: 0.9844
Epoch 8/20
95/95 [==============================] - 79s 832ms/step - loss: 0.0423 - accuracy: 0.9868 - val_loss: 0.0438 - val_accuracy: 0.9857
Epoch 9/20
95/95 [==============================] - 86s 901ms/step - loss: 0.0354 - accuracy: 0.9901 - val_loss: 0.0418 - val_accuracy: 0.9883
Epoch 10/20
95/95 [==============================] - 84s 883ms/step - loss: 0.0328 - accuracy: 0.9885 - val_loss: 0.0406 - val_accuracy: 0.9883
Epoch 11/20
95/95 [==============================] - 90s 945ms/step - loss: 0.0406 - accuracy: 0.9875 - val_loss: 0.0417 - val_accuracy: 0.9896
Epoch 12/20
95/95 [==============================] - 92s 972ms/step - loss: 0.0349 - accuracy: 0.9855 - val_loss: 0.0366 - val_accuracy: 0.9896
Epoch 13/20
95/95 [==============================] - 111s 1s/step - loss: 0.0318 - accuracy: 0.9904 - val_loss: 0.0356 - val_accuracy: 0.9909
Epoch 14/20
95/95 [==============================] - 96s 1s/step - loss: 0.0218 - accuracy: 0.9924 - val_loss: 0.0369 - val_accuracy: 0.9883
Epoch 15/20
95/95 [==============================] - 85s 889ms/step - loss: 0.0267 - accuracy: 0.9901 - val_loss: 0.0359 - val_accuracy: 0.9896
Epoch 16/20
95/95 [==============================] - 84s 879ms/step - loss: 0.0280 - accuracy: 0.9908 - val_loss: 0.0381 - val_accuracy: 0.9896
Epoch 17/20
95/95 [==============================] - 81s 850ms/step - loss: 0.0270 - accuracy: 0.9901 - val_loss: 0.0343 - val_accuracy: 0.9909
Epoch 18/20
95/95 [==============================] - 80s 845ms/step - loss: 0.0221 - accuracy: 0.9931 - val_loss: 0.0347 - val_accuracy: 0.9909
Epoch 19/20
95/95 [==============================] - 87s 921ms/step - loss: 0.0232 - accuracy: 0.9924 - val_loss: 0.0312 - val_accuracy: 0.9922
Epoch 20/20
95/95 [==============================] - 84s 880ms/step - loss: 0.0202 - accuracy: 0.9937 - val_loss: 0.0315 - val_accuracy: 0.9909
ankanhalder@Ankans-MacBook-Pro mask detection % █
```

FINAL RESULTS

After the complete training of our model,we saved the trained data in "**h5**" file format and converted it into a **"json"** file format which we have used it in our Web Application.At first we used a face detector model to segregate faces from the entire image which is live captured by the camera.After that we have fed that image to our "**model.json**" file using **tensorflow.js** to predict the "mask" or "no-mask" condition.

The proposed face mask detector application has been successfully deployed as a web application which is free for public use. The technology stack involved in the development of the web based user interface is:

- ➔ Client-side development:
  - ◆ HTML and CSS
  - ◆ JS
  - ◆ Tensorflow.js (for machine learning model)
- ➔ Server-side development:
  - ◆ Node JS
- ➔ Deployment Server:
  - ◆ Heroku
- Deployed URL: https://face-mask-detector083989.herokuapp.com/

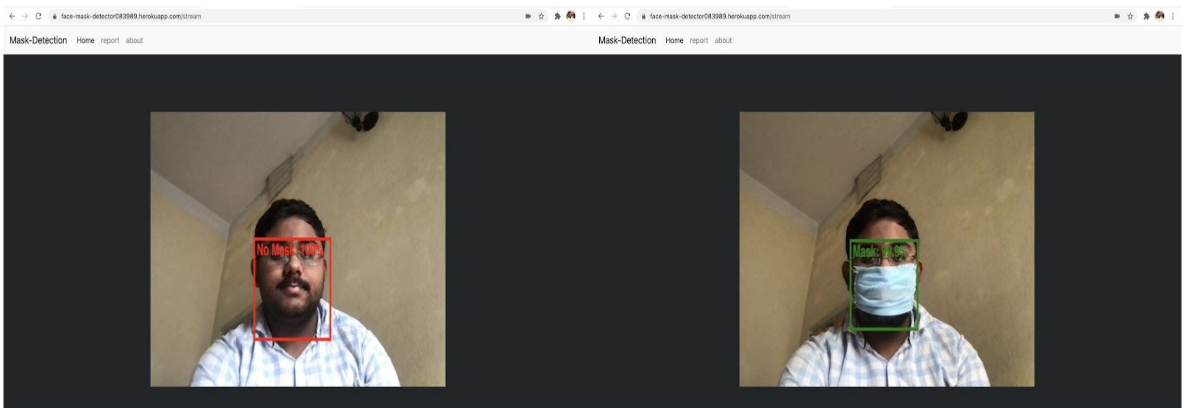
CONCLUSION

The architecture consists of MobileNetV2 as the backbone which can be used for high and low computation scenarios.We have used OpenCV Library, tensor flow,keras frameworks and MobileNetV2 CNN model to detect whether people were wearing face masks or not.By the development of our face mask detection model we can detect if the person is wearing a face mask which would be of great help to the society.

Future scopes of this project will be :

- Hospitals and healthcare organizations
- Offices
- Airports and railway stations
- Sports venues
- Entertainment industry
- Densely populated areas

Screenshot of our Web Application


face-mask-detector083989.herokuapp.com/stream
Mask-Detection
Home
report
about
No Mask: 100%
face-mask-detector083989.herokuapp.com/stream
Mask-Detection
Home
report
about
Mask: 99.9%

REFERENCES

- https://www.kaggle.com/
- https://github.com/opencv
- https://keras.io/
- https://www.tensorflow.org/
- https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html