

```
In [2]: import tqdm

In [3]: import pandas as pd
import numpy as np
import os
import random
import PIL
from PIL import Image, ImageOps
from matplotlib import pyplot as plt
import matplotlib.image as mpimg
from matplotlib.pyplot import imshow
%matplotlib inline
import cv2
from io import BytesIO
```

Connect to S3 Bucket

```
In [4]: from boto3.session import Session

# Ideally this is picked up your ENV.
id_ = "ASIAUKODTPYG7WSQNBL3"
secret = "oSQv85gUU8dn7G/V23tYrBbD/Y156CO2Li+4Qoss"
token = "FwoGZXIVYXdzEKX//////////wEaDNHI/BBwEwnAh+Yv4iK9AXDk/6vYh03oZnueAo2lG4cIY17R17Hc78K1YSfa/Vz/R60

session = Session(
    aws_access_key_id=id_,
    aws_secret_access_key=secret,
    aws_session_token=token,
    region_name='us-east-1'
)

# Test it on a service (yours may be different)
s3 = session.resource('s3')

# Print out bucket names
for bucket in s3.buckets.all():
    print(bucket)

s3.Bucket(name='bdfc-small')
s3.Bucket(name='samtestbucketname')

In [29]: for bucket in s3.buckets.all():
    #     for obj in bucket.objects.all():
    #         for obj in bucket.objects.filter(Prefix='mask-dataset/images/test'):
    #             print(obj)
    #             print(os.path.join(obj.key))

    File "<ipython-input-29-c0a81daf61fe>", line 5
        #             print(os.path.join(obj.key))
                    ^
SyntaxError: unexpected EOF while parsing
```

Access JPG and PNG Files

```
In [6]: filenames = []
for bucket in s3.buckets.all():
    for obj in bucket.objects.filter(Prefix='mask-dataset/images/test'):
        filenames.append(os.path.join(obj.key))

In [7]: len(filenames)

Out[7]: 2070

In [8]: image_names = []
text_names = []
for file in filenames:
    if '.txt' in file:
        text_names.append(file)
    else:
        image_names.append(file)
```

```
In [28]: # text_names
```

```
In [10]: print(len(text_names),len(image_names))
```

1035 1035

Get random set of 250 images from Test folder to begin building model

Get the image names

```
In [11]: # test_set = random.choices(image_names, k=250)
# test_set
```

Get text names that match the image names (convert .jpg and .png to .txt bc that is how it is stored)

```
In [12]: # test_text = []
# for x in test_set:
#     if ('jpg' in x):
#         test_text.append(x.replace('jpg', 'txt'))
#     elif ('.png' in x):
#         test_text.append(x.replace('png', 'txt'))
# test_text
```

Get the number of people in image wearing a mask, not wearing a mask, and number of people in image

```
In [13]: # text_info = []
# for bucket in s3.buckets.all():
#     for obj in bucket.objects.filter(Prefix='mask-dataset/images/test'):
#         for y in test_text:
#             if os.path.join(obj.key) == y:
#                 text_info.append({os.path.join(obj.key): obj.get()['Body'].read().decode("utf-8")})
```

```
In [14]: # counts = []

# for n in text_info:
#     for key, val in n.items():
#         v = val.split()
#         new = [int(x) for x in v if len(x)==1]
#         counts.append({key: new})
# counts
```

```
In [15]: # no_mask = {}
# mask_on = {}
# num_ppl = {}
# for line in counts:
#     for key, val in line.items():
#         k = key.replace('.txt', '')
#         r = val.count(0)
#         no_mask[k] = r
#         res = sum(val)
#         mask_on[k] = res
#         p = len(val)
#         num_ppl[k] = p
```

Create a dataframe to keep track of all information

```
In [16]: # test_df = pd.DataFrame(test_set, columns=['images'])
# h = []
# for img in test_df['images']:
#     im = img[:-4]
#     h.append(im)
# test_df['im'] = h
# test_df['mask_on'] = ''
# test_df['no_mask'] = ''
# test_df['num_people_image'] = ''
# test_df
```

```
In [17]: # test_df['no_mask']=test_df['im'].map(no_mask)
# test_df['mask_on']=test_df['im'].map(mask_on)
# test_df['num_people_image']=test_df['im'].map(num_ppl)
# test_df = test_df.drop('im', 1)
```

```
# test_df
In [18]: # test_df.to_csv('test_data.csv', index=False)
```

Read in CSV with test data information to begin facial detection

```
In [19]: test_data = pd.read_csv('test_data.csv')
test_data
```

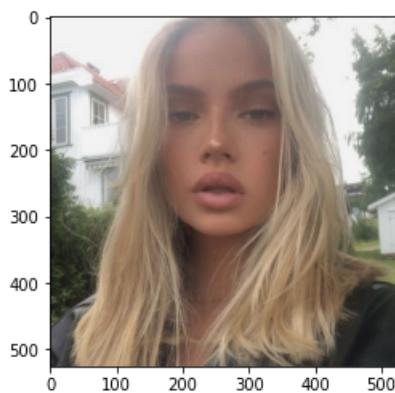
Out[19]:

	images	mask_on	no_mask	num_people_image
0	mask-dataset/images/test/2815.png	0	1	1
1	mask-dataset/images/test/6436.jpg	1	0	1
2	mask-dataset/images/test/4035.jpg	1	1	2
3	mask-dataset/images/test/5848.jpg	3	0	3
4	mask-dataset/images/test/5603.jpg	2	0	2
...
245	mask-dataset/images/test/3067.jpg	0	1	1
246	mask-dataset/images/test/3805.jpg	1	0	1
247	mask-dataset/images/test/4051.jpg	1	0	1
248	mask-dataset/images/test/662.png	1	0	1
249	mask-dataset/images/test/133.jpg	4	0	4

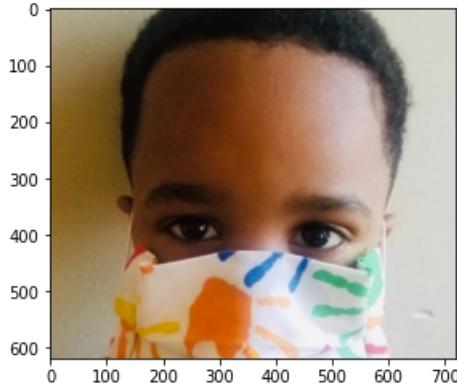
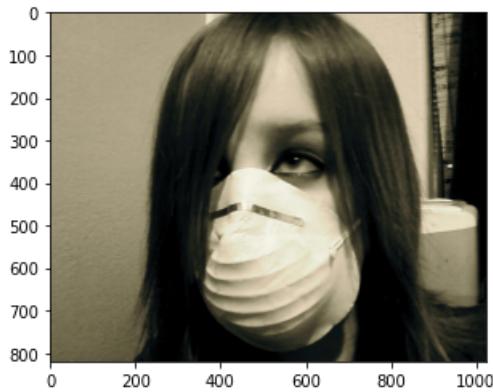
250 rows × 4 columns

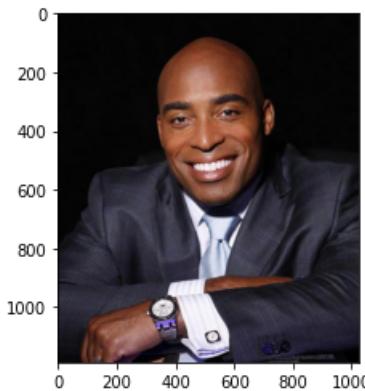
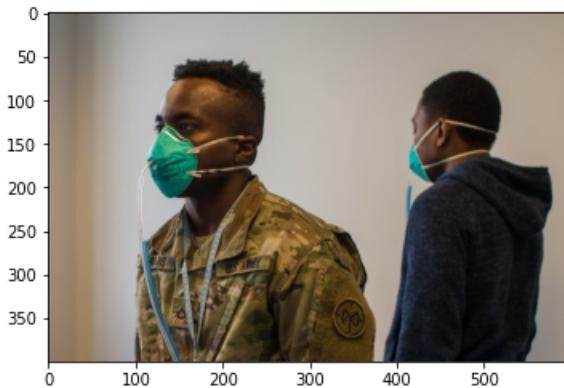
Look at the 250 test data images

```
In [30]: for pic in test_data['images'][:15]:
    obj = s3.Object(bucket_name = 'bdfc-small', key=pic)
    plt.figure()
    im = Image.open(BytesIO(obj.get()['Body'].read()))
    plt.imshow(im)
```









Create facial detection model

```
In [21]: face_model1 = cv2.CascadeClassifier(os.path.abspath(os.getcwd())+'haarcascade_frontalface_default.xml')
face_model2 = cv2.CascadeClassifier(os.path.abspath(os.getcwd())+'haarcascade_frontalface_alt2.xml')
face_model3 = cv2.CascadeClassifier(os.path.abspath(os.getcwd())+'haarcascade_frontalface_alt.xml')
```

```
In [27]: # img = cv2.imread(os.path.abspath(os.getcwd())+'/mask-dataset/images/test/2815.png')
# img = cv2.cvtColor(img, cv2.IMREAD_GRAYSCALE)

# faces = face_model.detectMultiScale(img,scaleFactor=1.1, minNeighbors=4) #returns a list of (x,y,w,h)

# out_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR) #colored output image

# #plotting
# for (x,y,w,h) in faces:
#     cv2.rectangle(out_img,(x,y),(x+w,y+h),(0,0,255),1)
# plt.figure(figsize=(12,12))
# plt.imshow(out_img)
```

```
In [26]: for pic in test_data['images'][:10]:
    obj = s3.Object(bucket_name = 'bdfc-small', key=pic)
    #     im = np.array(Image.open(BytesIO(obj.get()['Body'].read())))
    img = cv2.cvtColor(np.array(Image.open(BytesIO(obj.get()['Body'].read()))), cv2.IMREAD_GRAYSCALE)

    faces1 = face_model1.detectMultiScale(img, scaleFactor=1.1, minNeighbors=4) #returns a list of (x,y,
    faces2 = face_model2.detectMultiScale(img, scaleFactor=1.1, minNeighbors=4)
```

```
faces3 = face_model3.detectMultiScale(img, scaleFactor=1.1, minNeighbors=4)

out_img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR) #colored output image
b,g,r = cv2.split(out_img)

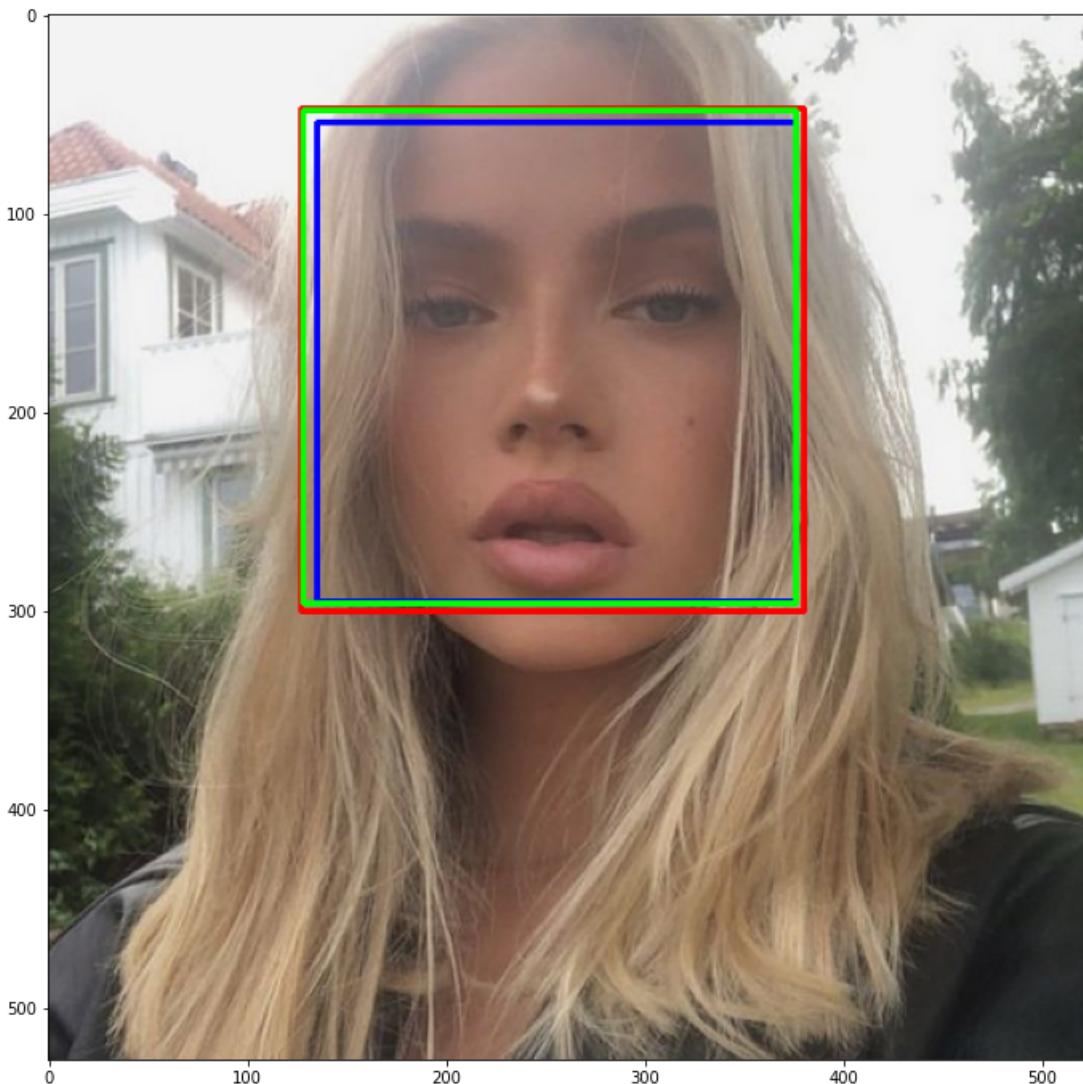
rgb_img1 = cv2.merge([r,g,b])      # switch it to r, g, b

#plotting

for (x,y,w,h), (x2,y2,w2,h2), (x3,y3,w3,h3) in zip(faces1,faces2,faces3):
    cv2.rectangle(rgb_img1,(x,y),(x+w,y+h),(0,0,255),2)
    cv2.rectangle(rgb_img1,(x2,y2),(x2+w2,y2+h2),(255,0,0), 2)
    cv2.rectangle(rgb_img1,(x3,y3),(x3+w3,y3+h3),(0,255,0), 2)

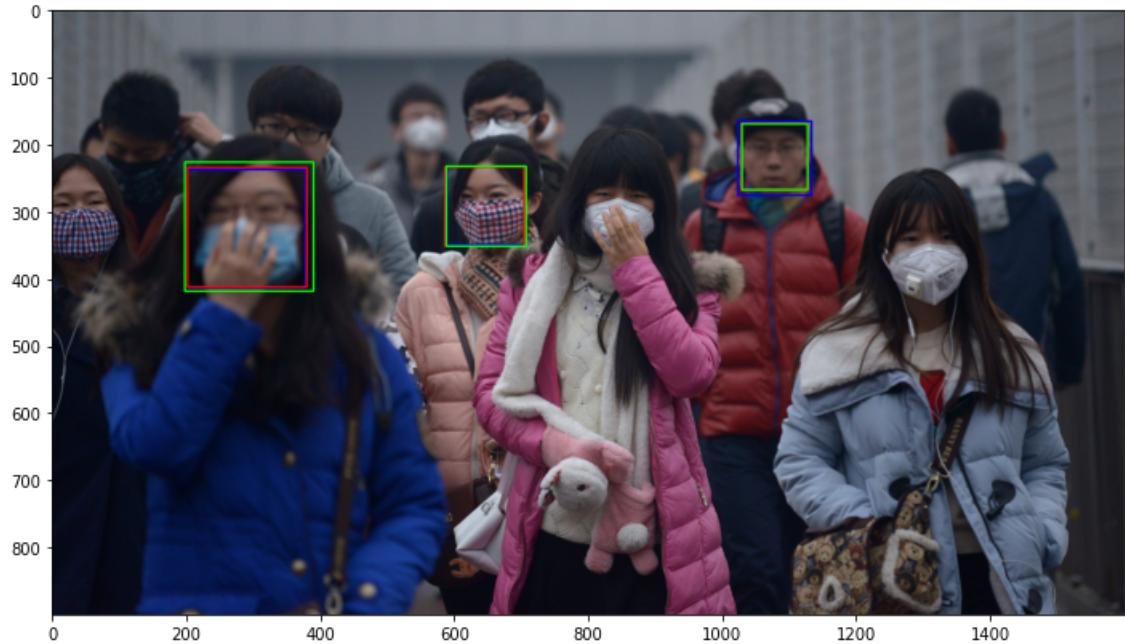
plt.figure(figsize=(12,12))

plt.imshow(rgb_img1)
```



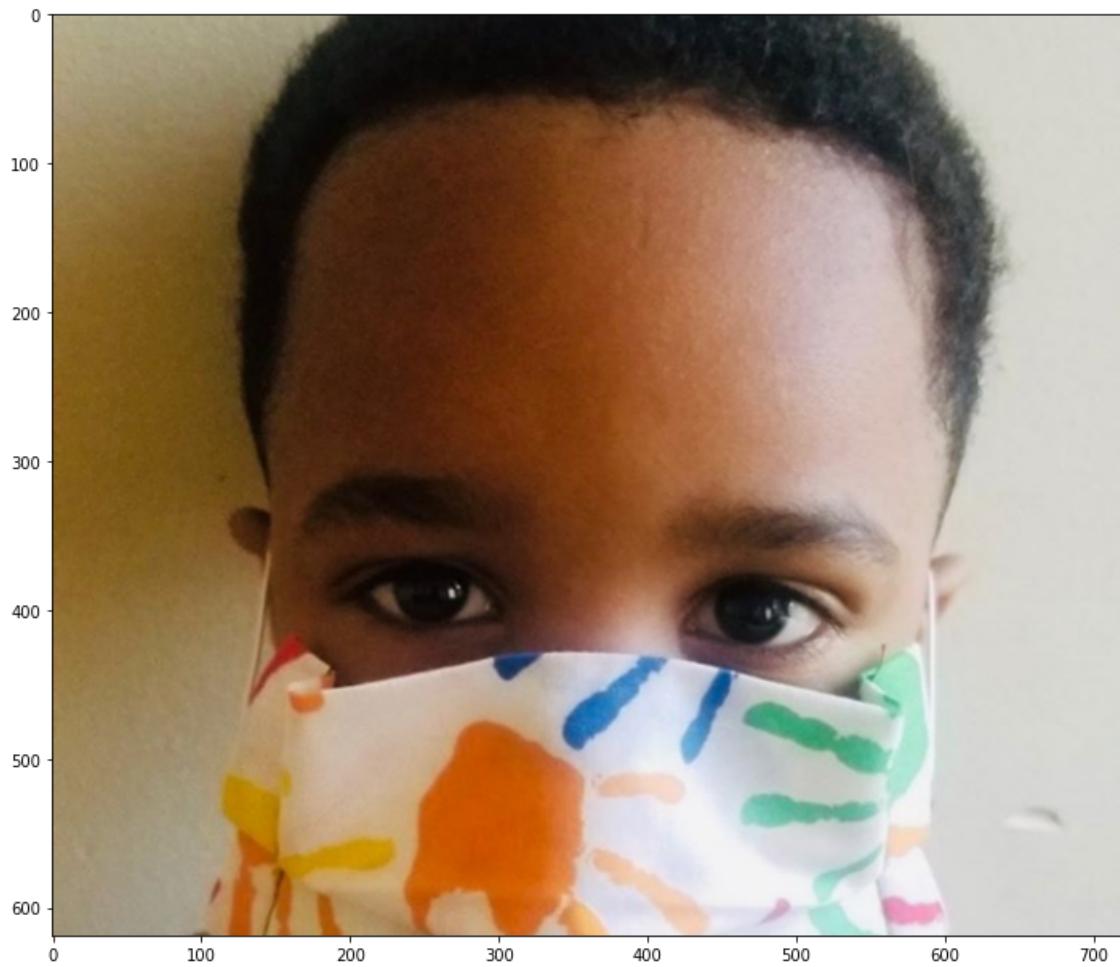












In []: