CSE 353
HomeWork 2

JeongYoon Lee (114133199)

a) Introduction. Brief summary of what you think the assignment is about

This assignment is about building a generative model for face classification when they are multivariate normal distribution.

So first, for the training part, we will find mean and covariance value for each face training set and background(non-face) training set since we can assume that the prior of each class is equal.

And then, we can test the testing data set with the mean value and covariance value that we find before. Then, we can check our model classify face picture and non-face picture well.

Also, I will try to convert each dataset with grayscale and HSV color to find most accurate model.

b) Method. Brief outline of your (algorithmic) approach

For the training, we will find mean value and covariance of each labeled data to test the testing data.

For this training image, when we vectorize an image, we will have [900*1] vector since each R,G,B value will have [300*1] vector for each one image. So we can get mean value and covariance value between each face training image and non-face(background) training image. The formula is as below.

$$Mean_{face} = \sum_{i=1}^{I} \left(\frac{xi}{I}\right)$$

$$Sigma_{face} = \sum_{i=1}^{I} \left((\text{x\_i} - \text{Mean}_{\text{face}}) + (x_i - Mean_{face})\text{^}T\right)/\,I$$

(x_i = [900*1] vector for each one image, I = number of face training dataset)

$$Mean_{background} = \sum_{i=1}^{I} \left(\frac{xi}{I}\right)$$

$$Sigma_{background} = \sum_{j=1}^{J} \left((\text{x\_j} - \text{Mean}_{background}) + (x_j - Mean_{background})\text{^}T\right)/\,J$$

(x_i = [900*1] vector for each one image, J = number of non-face training dataset)

```
for img_face in glob.glob(datadir + "/face/*.jpg"):
    n= cv2.imread(img_face)
    # plt.imshow(cv2.cvtColor(n, cv2.COLOR_BGR2RGB));
    # plt.show()
    length, height, depth = n.shape
    new_vector_face = n.reshape((length * height * depth, 1)) #Xi
    cv_img_face = new_vector_face + cv_img_face
    num_of_face = num_of_face +1

mean_face = cv_img_face/num_of_face
```

<Code for finding mean value>

```python
#For Large sigma for face (Covariance)
Sigma_Xi_M_face = [0]
for img_face in glob.glob(datadir + "/face/*.jpg"):
    n= cv2.imread(img_face)
    length, height, depth = n.shape
    Xi_M_face = n.reshape((length * height * depth, 1)) #Xi
    Xi_M_face = Xi_M_face - mean_face
    Sigma_Xi_M_face = (Xi_M_face * Xi_M_face.T) + Sigma_Xi_M_face
    # print(np.shape(Xi_M.T))

# print(Sigma_Xi_M_face)
# print(np.shape(Sigma_Xi_M_face)) # 900*900
Sigma_face = Sigma_Xi_M_face / num_of_face
Covariance_face = np.diag(Sigma_face)
```

```python
##################################################################
# using var method in python
Sigma_Xi_M_face = np.zeros((900,1))

for img_face in glob.glob(datadir + "/face/*.jpg"):
    n= cv2.imread(img_face)
    length, height, depth = n.shape
    Xi_M_face = n.reshape((length * height * depth, 1)) #Xi
    Xi_M_face = np.array(Xi_M_face)
    Sigma_Xi_M_face = np.hstack((Sigma_Xi_M_face, Xi_M_face))
Sigma_Xi_M_face = np.delete(Sigma_Xi_M_face,1, axis = 1)
A = Sigma_Xi_M_face.var(axis = 1)
Covariance_face = A
print("Using python method to get Covariance with face class")
print(A)
###########################################################
```

<Code for finding Covariance>

And then we can test through these value as below.

$P_r\,(x*|y=1) > ?\, P_r(x*|y=0)$

*Assume that face-image : y=1, non-face-image = y=0
*Pr(y=1) = Pr(y=0) : the prior of each class is equal.

So it can be expressed by mean and covariance value that we already get.

$$P_r\,(x*|y=1) = \left(-\frac{1}{2}\right)\sum_{d=1}^{D}\left(\ln\left(Covariance^2_{dd\_face}\right)\right) - \sum_{d=1}^{D}\frac{\left(\mathrm{x_d}-Mean_{d_{face}}\right)2}{2*\left(Covariance_{dd_{face}}\right)}$$

$$P_r\,(x*|y=0) = \left(-\frac{1}{2}\right)\sum_{d=1}^{D}\left(\ln\left(Covariance^2_{dd\_background}\right)\right) - \sum_{d=1}^{D}\frac{\left(\mathrm{x_d}-Mean_{d_{background}}\right)2}{2*\left(Covariance_{dd_{background}}\right)}$$

If $P_r\,(x*|y=1) > P_r\,(x*|y=0)$ for face-test image, it detected right, but if $P_r\,(x*|y=1) < P_r\,(x*|y=0)$, it detected image wrong.

and for the background-test image, if $P_r\,(x*|y=1) < P_r\,(x*|y=0)$, it detected properly, and also if $P_r\,(x*|y=1) > P_r\,(x*|y=0)$, this model detected image wrong.

```
fig = plt.figure(figsize = (8,8))
rows = 10
cols = 5
for img_face_test in glob.glob(testdatadir + "/face/*.jpg"):
    num_face += 1
    n= cv2.imread(img_face_test)
    length, height, depth = n.shape
    new_vector_face_test = n.reshape((length * height * depth, 1)) #Xi
    A_face = ((new_vector_face_test-mean_face) **2)/(2*Covariance_face_test)
    A_face = A_face.sum()
    B_face = np.log(Covariance_face_test)
    B_face = B_face.sum()
    face_test_val = (-0.5)*B_face -A_face

    A_bg = ((new_vector_face_test-mean_bg) **2)/(2*Covariance_bg_test)
    A_bg = A_bg.sum()
    B_bg = np.log(Covariance_bg_test)
    B_bg = B_bg.sum()
    bg_test_val = (-0.5)*B_bg -A_bg

    if(face_test_val < bg_test_val):
      # print("face detected wrong!")
      num_of_wrong_face += 1
      ax = fig.add_subplot(rows, cols, i)
      ax.imshow(cv2.cvtColor(n, cv2.COLOR_BGR2RGB));
      i = i+ 1
    else:
      #print("face detection successfully!")
      num_of_right_face += 1
plt.show()
```

<Code for finding $P_r\,(x*|y=1)$>

We can also find accuracy of each class through knowing the number of detected image from our model.

c) Experiments. Tables and/or pictures of intermediate and final results that convince us that the program does what you think it does.

1. Training

For the training image, we can find Mean and Covariance of each face and non-face training image. As I mentioned in question b), we can calculate mean value with Xi, which is result of vectorized RGB value in the image. So, the mean value shape will be [900*1], and it printed as below.

```
<Training result>
Size of mean for face class is :  (900, 1)
Mean of face class is :
[[ 89.9076087 ]
 [110.74456522]
 [127.8423913 ]
 [ 82.97826087]
 [103.51630435]
 [120.53804348]
 [ 70.13586957]
 [ 88.47282609]
 [106.68478261]
 [ 56.02173913]
 [ 73.22826087]
 [ 91.29347826]
 [ 46.50543478]
 [ 61.10326087]
 [ 80.1576087 ]
 [ 41.00543478]
 [ 54.61956522]
 [ 73.11413043]
 [ 38.58695652]
 [ 50.41847826]
 [ 69.2826087 ]
 [ 36.89673913]
 [ 48.64673913]
 [ 66.55434783]
 [ 36.52717391]
 [ 47.88586957]
 [ 65.91847826]
 [ 38.69565217]
 [ 50.92934783]
```

<Part of mean value of face class>

```
Size of mean of background class is :  (900, 1)
Mean of background class is :
[[ 97.31726908]
 [117.64257028]
 [134.79919679]
 [ 97.99196787]
 [118.70682731]
 [135.57028112]
 [ 99.88353414]
 [120.63052209]
 [137.6184739 ]
 [100.64658635]
 [121.93975904]
 [138.562249  ]
 [ 99.75903614]
 [121.07228916]
 [138.02008032]
 [ 98.75502008]
 [120.61445783]
 [137.16465863]
 [ 99.57429719]
 [121.26506024]
 [138.10843373]
 [100.11646586]
 [122.19678715]
 [138.59839357]
 [ 99.63855422]
 [121.21285141]
 [137.79919679]
 [100.32128514]
 [122.12449799]
 [138.21686747]
```

<Part of mean value of background class>

For the Covariance, I tried two ways to get covariance in python.

1) Calculate with hard coding

2) Using python method, var(axis = 1)

They are literally same process, but they printed slightly different result since there's some tolerance with decimal point. So, I decide to use covariance calculated with hard coding.

1) Result of Covariance calculated with hard coding

```
Size of Covariance of face class :  (900,)
Covariance of face class
[1603.21428993 1919.40757916 2118.49146385 1765.5647448  2228.4888646
 2469.57463965 1688.35653946 2099.1188268  2383.68324669 1536.28213611
 1830.5348535  2112.9790879  1337.44562264 1560.35346763 1844.12189863
 1112.26627481 1281.1596172  1553.36197424 1054.26417769 1188.75422377
 1498.3657845  1068.22303284 1184.29368502 1522.98617675 1043.19491375
 1164.99240903 1533.12922377 1170.08128544 1325.98957349 1731.13846881
 1414.76890359 1625.67993856 1980.3267663  1649.21523511 1930.65616139
 2216.01651111 1824.67344045 2201.9243561  2380.6835716  2081.81509924
 2524.57502363 2614.14127481 1928.01485704 2290.51216919 2432.02903474
 1710.66965974 2095.88584003 2352.33078332 1686.10007089 2139.57605742
 2491.29203095 1406.08778355 1783.18950851 2132.30009452  998.23366612
 1249.79300567 1533.3501595   911.06096408 1088.20838256 1398.7112181
 1000.20507443 1164.91632207 1529.50661626  975.60181356 1168.590944
 1593.63016895  907.82936555 1059.76795841 1479.82041588  922.45545841
 1059.58400284 1510.13584003  922.20167769 1072.81509924 1486.26509334
 1001.86377599 1183.17627599 1566.86342155 1168.3311673  1456.01878544
 1795.65300095 1534.40734286 1936.16703095 2225.48097826 1758.06377009
 2200.71263587 2366.30077387 2071.65946952 2474.65205577 2632.21428993
 1869.83128544 2294.67568526 2611.57912925 1467.82313327 1820.52351134
 2155.8680293  1038.2991198  1296.32629371 1568.77256616  844.00186082
  993.8175508  1235.91726725 1268.24574669 1445.23342982 1819.30942817
 1830.81368147 2150.3083353  2689.52643549 2042.02950733 2410.56518785
 3060.9073724  1840.6596172  2192.01264178 2894.84912571 1682.0399043
 2012.43451678 2688.61929939 1517.79241493 1841.91183247 2461.20342037
 1354.10512169 1576.33031073 2091.24574669 1047.17993856 1298.37827859
 1714.57605742 1176.45226843 1544.7064922  1923.11102906 1518.93431002
 1903.49311791 2171.67663043 2009.93262642 2444.02835539 2622.0862181
 1895.76323251 2229.35952268 2562.98995747 1186.98428639 1451.6554525
 1723.86909263  838.05951678 1047.25980624 1281.0491198  1122.49311791
 1352.88040525 1667.59688091 2173.57605742 2539.82041588 3057.70412925
```

<Part of Covariance of face class with hardcoding>

```
Size of Covariance of background :  (900,)
Covariance of background class
[3896.70656925 4164.65537653 4593.9677102  3716.98788729 3963.12288511
 4476.30128546 3611.28362446 3826.95585555 4375.47291173 3592.59799035
 3797.14898147 4292.89672747 3665.93390429 3798.62931243 4259.33694618
 3803.17291657 3981.70677892 4398.97288753 3844.16415864 3997.9377752
 4308.46615377 3814.42418671 3934.73637522 4252.9712424  3761.31514008
 3856.09525653 4279.58216803 3786.28231803 3956.99253238 4409.87264722
 3816.66892469 3918.89808229 4351.25701198 3754.14915888 3887.78132611
 4283.73981065 3631.39007435 3832.53363655 4166.07209561 3663.14940082
 3837.98461315 4210.5776681  3550.84901856 3734.87379236 4157.65048951
 3790.06409574 4135.18007774 4665.53513653 3621.99519363 3949.51694328
 4517.27762455 3514.56341027 3789.04001548 4409.49787907 3607.40081612
 3831.1399171  4440.75066531 3777.61003855 3958.88740504 4501.31381752
 3719.54626538 3978.9791455  4444.49483073 3686.6633764  3977.13856228
 4369.75887486 3595.60297415 3879.17727133 4328.88450186 3807.91922711
 4080.89963065 4590.59060338 3890.65073144 4187.69452106 4675.25794745
 3799.11475621 4028.95401687 4500.80911598 3584.09525653 3817.9044209
 4253.76448767 3547.56232964 3842.76179416 4287.00682247 3674.4132514
 3970.48515347 4446.34628474 3593.51936259 3886.312124   4412.71537556
 3874.00261286 4188.05487008 4631.66906985 3778.88740504 4078.85830874
 4577.3563007  3637.5097821  3930.35983291 4445.96209739 3509.09391784
 3763.21962549 4306.55489428 3583.60036128 3814.89446944 4331.21536749
 3616.36796181 3863.57197465 4408.91298527 3651.76032645 3927.43778326
 4452.06032161 3622.26112482 3941.77181013 4464.91721101 3653.66277963
 3950.51534653 4505.87935678 3707.72274641 3993.80813213 4535.08795019
 3795.12211093 4036.35699424 4546.98595184 3796.22077063 4023.29717263
 4486.10048225 3926.07048273 4164.19751294 4661.04804761 3959.35339753
 4203.61429654 4719.11194981 4012.38147772 4237.63084466 4764.35983291
 3635.12346575 4015.63752198 4456.68924695 3673.64513476 4023.33023661
```

<Part of Covariance of background class with hardcoding>

2) Result of Covariance using python method, var(axis = 1)

```
Using python method to get Covariance with face class
[1646.90725425 1985.17081167 2206.74149338 1802.74973417 2285.74477198
 2548.09259806 1714.52693762 2141.38031664 2443.95838256 1546.65994211
 1853.63031664 2147.54746574 1338.43475307 1570.06509924 1863.08125591
 1116.50791588 1291.21736177 1572.74704631 1060.46691871 1199.56746219
 1520.35574197 1074.0583353  1194.09770794 1543.17993856 1049.66277765
 1177.09026465 1556.50942226 1177.45412925 1338.83669069 1754.57596881
 1423.109375   1640.98724008 2007.44798559 1662.17202268 1952.82133152
 2252.88516068 1844.32912925 2232.71361059 2431.81852552 2114.22256026
 2573.35110468 2686.05290052 1964.39363776 2345.76795841 2511.98901229
 1751.06427221 2156.03071834 2437.3314922  1713.80009452 2185.81521739
 2556.197513   1418.090944   1808.12559074 2170.89260397 1003.95980033
 1263.56825969 1557.96160208  915.52903474 1098.44742439 1419.82596881
 1005.15689981 1175.24689863 1551.08550922  980.09782609 1178.42246574
 1615.26594991  909.32393077 1067.43986295 1498.68809074  925.66537689
 1067.31758034 1528.31368147  925.76627481 1081.03187027 1504.6880612
 1006.40994211 1193.00661626 1587.71172023 1173.73759452 1467.9422259
 1820.02256616 1543.80907372 1954.60866612 2260.58456404 1779.50540525
 2235.48865784 2422.64603025 2103.83825614 2523.859375   2706.12260751
 1905.3917474  2349.21727316 2690.27950733 1483.822513   1850.78154537
 2203.07325142 1042.98901229 1308.73413871 1593.29226725  847.05671078
 1002.52835539 1256.18002717 1272.61977198 1455.66387051 1843.21455577
 1837.46216328 2163.2826087  2717.41726725 2049.11472117 2423.53565099
 3089.85842982 1847.40805175 2204.73003308 2923.18522566 1688.16277765
 2023.963138   2715.16608578 1522.56852552 1851.52681947 2484.48334121
 1357.52194589 1584.48617675 2112.18664343 1050.27445652 1306.98806711
 1735.28730506 1181.95720109 1557.17474008 1949.68995156 1531.50233341
 1927.26060373 2215.43655482 2035.97350543 2485.08574551 2688.23130317
 1924.92388351 2275.84640832 2632.90450733 1194.92863894 1469.42202268
 1757.10526938  839.88126181 1054.31707821 1299.43498937 1123.65734286
```

<Part of Covariance of face class with python method>

```
Using python method to get Covariance with background class
[3913.14398155 4190.07296657 4650.00067741 3755.27114079 4018.29080176
 4549.8184868  3610.45137982 3853.32988178 4411.76303608 3593.7919711
 3825.72084321 4337.76145546 3671.87945356 3830.53828164 4314.32218835
 3791.02359639 3997.23020596 4442.73105272 3849.08175675 4030.8104708
 4366.96056515 3811.50365317 3961.59155497 4305.92974307 3759.27275367
 3884.85443783 4331.78351962 3785.3421719  3988.62815116 4461.54458799
 3817.76019742 3950.80711601 4400.62305447 3753.9758391  3919.33094628
 4329.89242109 3631.25662489 3861.88303414 4209.44733149 3660.3914453
 3865.00937082 4250.91656586 3547.43488008 3759.06385381 4196.68844051
 3801.42907372 4153.83222851 4717.25527008 3642.80518056 3980.90869502
 4576.56915211 3552.88008258 3845.66332801 4483.03975742 3597.18885179
 3847.32030129 4472.65750552 3778.01054822 3986.09135337 4550.06361188
 3709.31662393 3996.67202142 4488.37851002 3681.55029758 4000.95982323
 4420.24631861 3596.09864357 3908.48195997 4383.56745859 3804.63844132
 4108.1363849  4640.98333898 3886.61782874 4215.80551927 4722.93743649
 3796.35447815 4056.55937807 4545.52678183 3579.80077741 3844.93572684
 4295.04091869 3540.10957888 3865.89687263 4324.25541524 3666.63505427
 3991.16120708 4481.0783697  3584.2732214  3904.95479105 4445.61532879
 3894.85927646 4218.75592329 4694.15564265 3793.17620684 4103.36946178
 4632.68876308 3661.00230641 3966.15022338 4506.15067499 3534.39583233
 3811.4727827  4370.1342559  3581.74813309 3838.50892728 4375.23510911
 3618.78976146 3891.63007048 4460.45060564 3643.97351656 3946.40783213
 4499.14852986 3625.93461396 3973.39788068 4521.38723569 3651.01091918
 3977.72539153 4557.14833632 3703.01553201 4019.88996952 4581.8670344
 3790.79743875 4063.28455993 4591.42207384 3789.81480944 4048.30322092
 4527.08107934 3916.41615458 4185.30130159 4698.11496589 3949.91709811
 4223.48094386 4754.23615748 4002.33534943 4256.39196142 4798.40318705
 3671.10662731 4066.66186029 4530.0391284  3710.68208577 4076.46086353
 4579.87429235 3740.16715859 4082.41963839 4568.17961001 3646.92756568
```

<Part of Covariance of background class with python method>

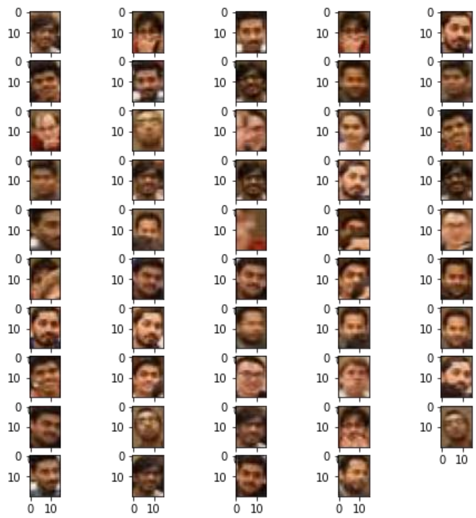
2. Testing

```
<Testing Result>
Entire number of face :  232
Number of detected as face, actually face :  191
Number of detected as background, but actually face 41

Entire number of background: 564
Number of detected as background, actually background :  421
Number of detected as face, but actually background :  143

Face classification accuracy :  0.8232758620689655
Background classification accuracy :  0.7464539007092199
```

$$*Face\ classification\ accuracy = \frac{Number\ of\ correctly\ classified\ face\ images}{Number\ of\ face\ images}$$

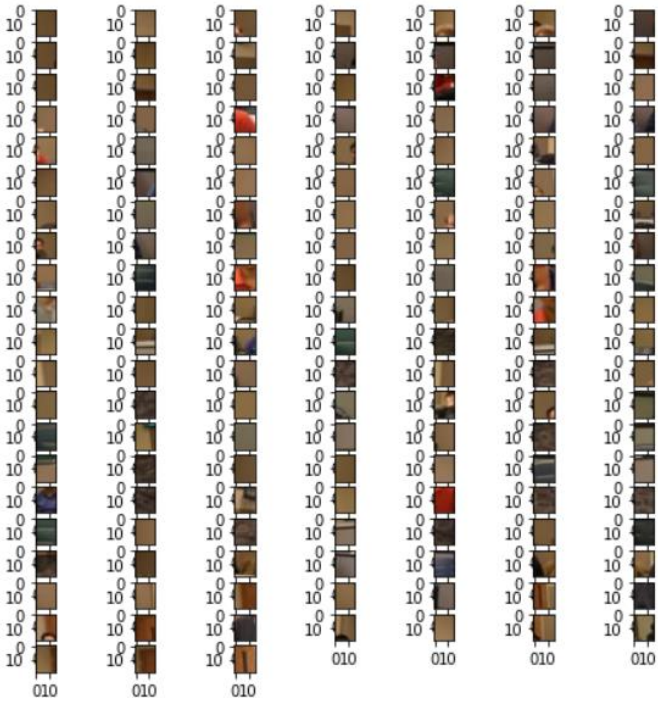
$$*Background\ classification\ accuracy = \frac{Number\ of\ correctly\ classified\ background\ images}{Number\ of\ background\ images}$$

<Images that detected as background, but actually face>

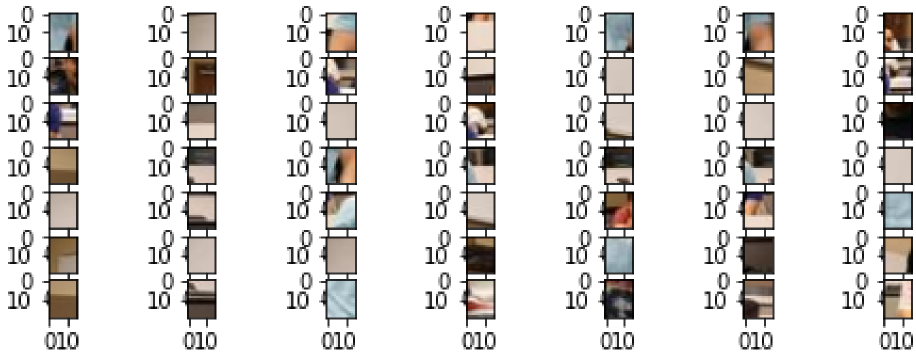


<50 images that model detected as face, and actually face>

I can find that if there's no various colors in face image, this model cannot classify as face image. In the case of photos where the beard or hair color is not prominent, there seems to be a tendency not to recognize it as a face.



<Images that detected as face, but actually background>



<50 images that model detected as background, and actually background>

I can infer that if the background has a lot of brown color or a color similar to the face color comes out, it tends to be recognized as a face.

d) Discussions. Any design decisions you had to make and your experimental observations. What do you observe about the behavior of your program when you run it? Does it seem to work the way you think it should? Play around a little with different setting to see what happens. Note, your open-ended exploration is highly valued.

As we discussed in question b), I will try to change some training dataset to grayscale or HSV color.

Here are some ways that I used for other trial model.

1. Change dataset to grayscale

```
n= cv2.imread(img_face, cv2.IMREAD_GRAYSCALE)
length, height = n.shape
new_vector_face = n.reshape((length * height , 1)) #Xi
```

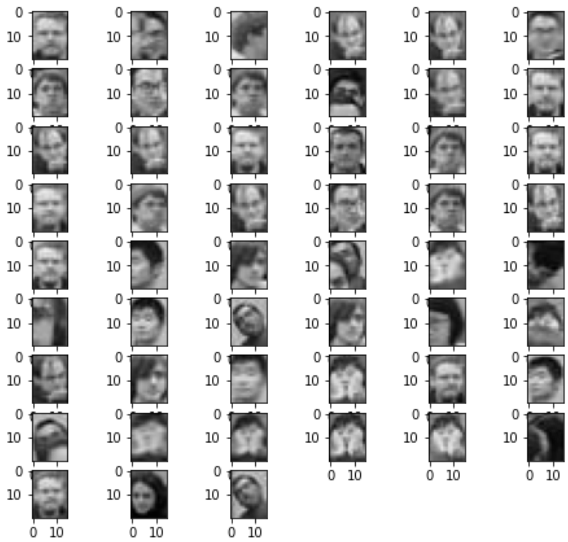
What I changed is that I read all images as grayscale and make vector with 2 dimension.

The testing result is as below.

```
<Testing Result>
entire num of face :  232
detected as face, actually face :  181
detected as background, but actually face 51

entire num of background: 564
detected as background, actually background :  419
detected as face, but actually background 145

Face accuracy :  0.7801724137931034
Background accuracy :  0.7429078014184397
```

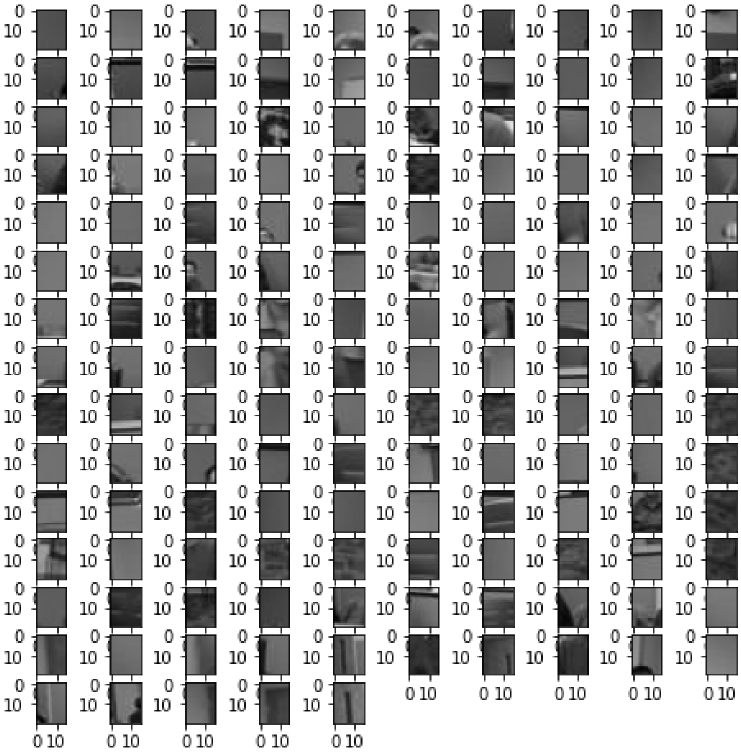
The accuracy of each face and background class are lower than original RGB image.

When you look at the below image, you can find that almost same images are not classified well when we are doing with RGB images, and some images are not classified well than original RGB image.

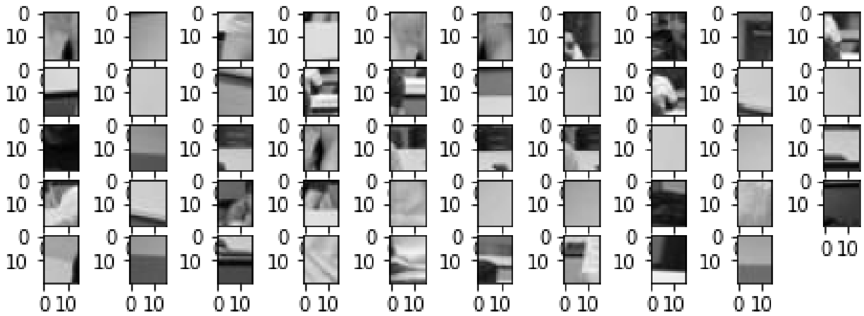


<Images that detected as background, but actually face in grayscale>

<50 images that model detected as face, and actually face>



<Images that detected as face, but actually background in grayscale>



<50 images that model detected as background, and actually background in grayscale>

So, I can infer that there’s no big difference in models between RGB images and grayscale images.

2. Change dataset to HSV color

```
n= cv2.imread(img_face_test)
n = cv2.cvtColor(n, cv2.COLOR_BGR2HSV)
length, height, depth = n.shape
new_vector_face_test = n.reshape((length * height * depth, 1)) #Xi
```

What I changed is that I read all images as HSV color and vectorized.

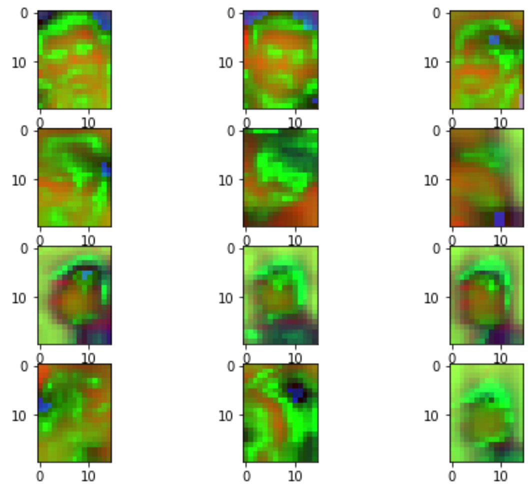
The testing result is as below.

```
Entire number of face :  232
Number of detected as face, actually face :  220
Number of detected as background, but actually face 12

Entire number of background: 564
Number of detected as background, actually background :  463
Number of detected as face, but actually background :  101

Face accuracy :  0.9482758620689655
Background accuracy :  0.8209219858156028
```

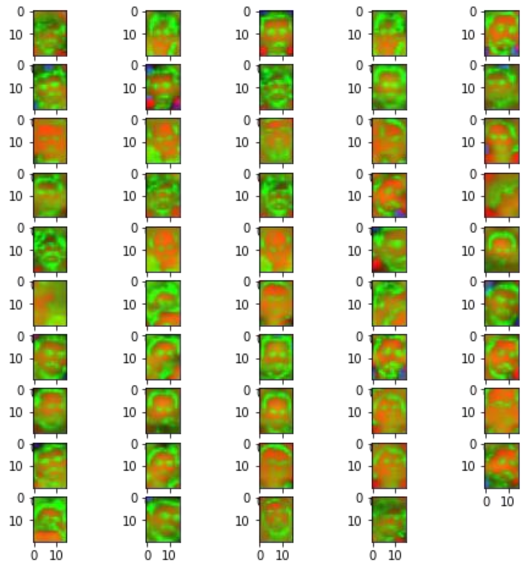
The accuracy of each face and background class are much higher than original RGB image.

When you look at the below image, you can find that if face image has blue color in the upper side of their image with various color, they detected as background.

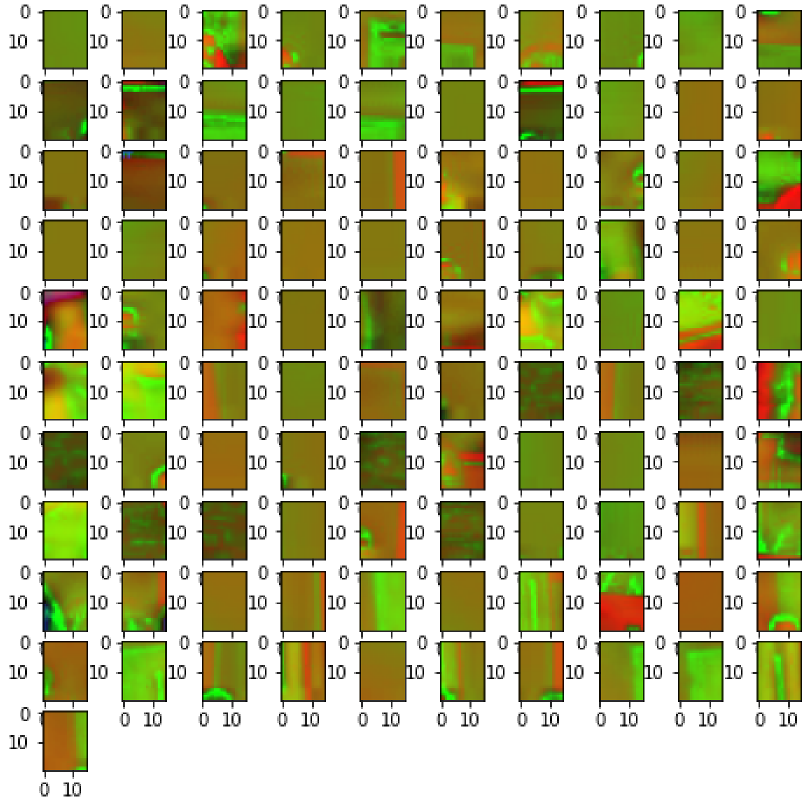
Also, we can find that if there's no red color in the image, and if most of colors include brown and green in HSV image, our model detected as face.



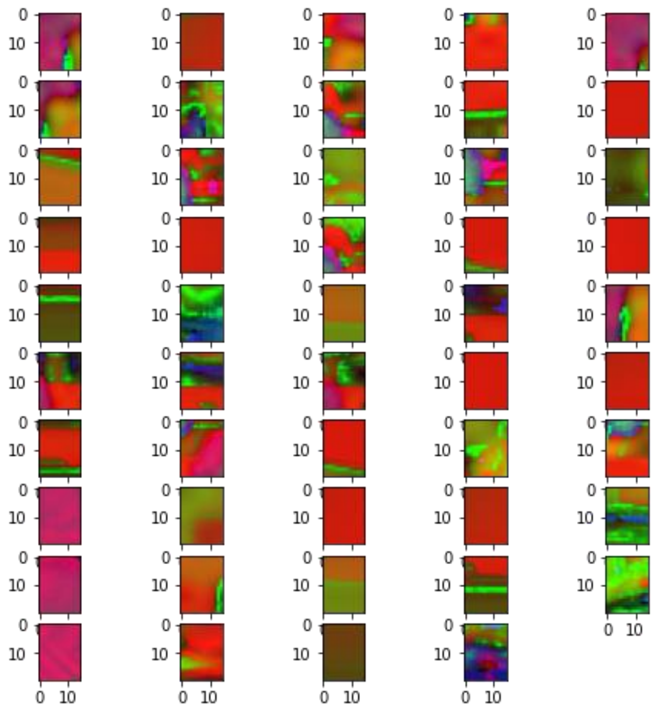
<Images that detected as background, but actually face in HSV>



<50 images that model detected as face, and actually face in HSV>



<Images that detected as face, but actually background in HSV>



<50 images that model detected as background, and actually background in grayscale>

Even testing for background class is not that high with HSV color, but it is the most accurate model among 3 models that we made. So, I can conclude that we can use HSV image when we make face detection model.

Looking at the labeled data, there were also images with only a part of the face. If we train the data excluding that case, I think that a model with different results would have been created. Also, I think that if the face data in the training data had a little more diversity, it would have been classified better. In the future, I want to make a model by refining it with more diverse face data.