Face Mask Wear Classification

IMPLEMENATION USING PYTHON |OPENCV TENSORFLOW1.X

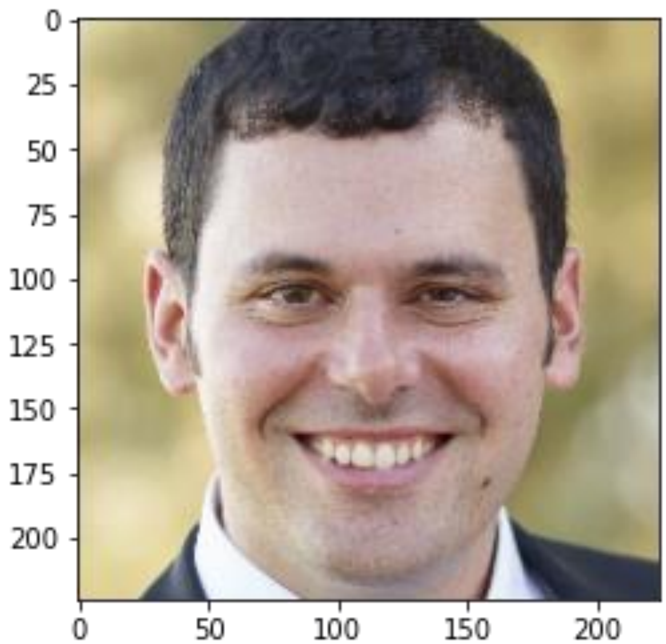
Create a model to recognise faces wearing a mask

```
### In this section, we are going to make a classifier that
### Can differentiate between faces with masks and without masks.
### For creating this classifier, we need data in the form of Images.


###Dataset folder tree
###.
###├── train dataset
###│    ├── with_mask    [1000 images]
###│    └── without_mask [950 images]
###├── test dataset
###│    ├── with_mask    [210 images]
###│    ├── without_mask [200 images]
###│

### Our dataset is taken from Kaggle and has over 70k images.
### The images in the dataset are of the same people with and without a
mask.

###

Without mask - Google drive:
https://drive.google.com/drive/folders/1tZUcXDBeOibC6jcMCtgRRz67pzrAHeH
L


GitHub: https://github.com/NVlabs/ffhq-dataset


With mask - One drive:
https://esigelec-
my.sharepoint.com/personal/cabani_esigelec_fr/_layouts/15/onedrive.aspx
?id=%2Fpersonal%2Fcabani%5Fesigelec%5Ffr%2FDocuments%2FMaskedFaceNetDat
aset%2FCMFD&originalPath=aHR0cHM6Ly9lc2lnZWxlYy1teS5zaGFyZXBvaW50LmNvbS
86ZjovZy9wZXJzb25hbC9jYWJhbmlfZXNpZ2VsZWNfZnIvRXYzR2RuUVN5enhQanl6VTVFb
EhxYWdCbGtSQ2FLbm5DSTg1aVgtZDFMNE9IQT9ydGltZT1vMmZIa1EtaTJFZw


GitHub: https://github.com/cabani/MaskedFace-Net
```

```
## first import necessary libraries
## as requested we are using a neural network in Tensorflow 1.X
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import cv2
import os
import matplotlib.pyplot as plt
import numpy as np
import random

## Load Train dataset
```

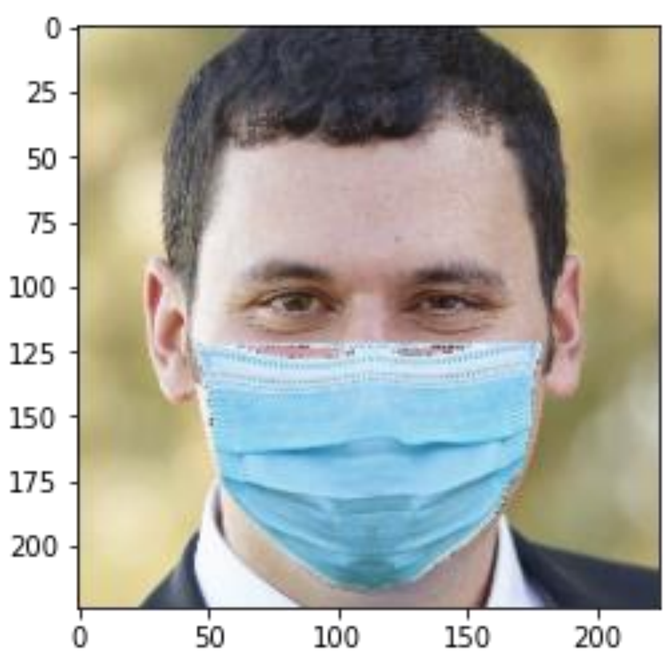
initial attempts

```
###  At first we train the model with 200 images with and without mask
###  At this point, we got a high train error because we did not have
many pictures to learn from.
###  And to deal with that, we added more images to the dataset and
that way we learned better and improved performance

### first let's load asingle image and see how the image looks like
### and cover it to numpy array

### cv2 image colors is BGR (blue, green, reed)
### BGR has 3 channels and if we us this way  our data will be large
### so in our case to Classificat with_mask and without_mask
### GRAYSCALE should be enough that has only one channel

### read image and resize
m_check = cv2.imread('02877.png')
m_check=cv2.resize(m_check, (224, 224))
# show image BGR
plt.imshow(cv2.cvtColor(m_check,cv2.COLOR_BGR2RGB) )
# convert to GRAYS
m_check = cv2.cvtColor(m_check,cv2.COLOR_RGB2GRAY)
```



```
### like we mentioned before our dataset has same image with and
without mask
### let's see the same image with mask

n_check = cv2.imread('02877_MasK.Jpg')
n_check=cv2.resize(n_check, (224, 224))
plt.imshow(cv2.cvtColor(n_check,cv2.COLOR_BGR2RGB) )
```

```
<matplotlib.image.AxesImage at 0x1f9fbf57cd0>
```



Load Train dataset

```
# image directory
Directory = r"C:\Users\mulugeta fanta\Desktop\DataSet\train" # trainnig dataset
# linear classifiction
Classes = ["with_mask", "no_mask"]
# final image size
img_size = 224
training_data = []

def creat_training_data():
    for category in Classes:
        path = os.path.join(Directory, category) # data directory
        class_num = Classes.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread( os.path.join(path, img),cv2.IMREAD_GRAYSCALE)
                # resize the image to 224 * 224
                new_array = cv2.resize(img_array, (img_size, img_size))
                # insert image to array
                training_data.append([new_array, class_num])
            except Exception as e:
                pass

creat_training_data()
random.shuffle(training_data)
```

Convert train image to array and normalize

```
## convert train image to numpy array and normalize
```

initial attempts

```
### At first our matrix's values were very large and we got high train error results
#### To deal with that we normalized the data so that the values would be only between 0 and 1

x_train = [] # train data
y_train = []  # train label

for f, l in training_data:
    x_train.append(f)
    y_train.append(l)
```

```python
# convert image to array
x_train = np.array(x_train)

# Changing dimension of input images from N*244x244 to  NX(50176) ->
244 * 244
x_train =
x_train.reshape((x_train.shape[0],x_train.shape[1]*x_train.shape[2]))

y_train = np.array(y_train)
#print befor normlaize
print("befor normlaize: \n" );print(x_train , " \n")

# normlaize
x_train = x_train.astype(float) / 255.0

#print after normlaize
print("after normlaize: \n" );print(x_train )
y_train = np.array(y_train)

y_train = y_train.flatten()
```

```
befor normlaize:

[[160 165 161 ... 172 170 169]
 [ 54  59  45 ...  22  23  24]
 [201 200 200 ... 202 202 202]
 ...
 [ 50  45  44 ... 112 111 109]
 [141 146 150 ... 205 204 204]
 [114 118 120 ...  81  76  80]]

after normlaize:

[[0.62745098 0.64705882 0.63137255 ... 0.6745098  0.66666667 0.6627451
]
 [0.21176471 0.23137255 0.17647059 ... 0.08627451 0.09019608
0.09411765]
 [0.78823529 0.78431373 0.78431373 ... 0.79215686 0.79215686
0.79215686]
 ...
 [0.19607843 0.17647059 0.17254902 ... 0.43921569 0.43529412
0.42745098]
 [0.55294118 0.57254902 0.58823529 ... 0.80392157 0.8        0.8
]
 [0.44705882 0.4627451  0.47058824 ... 0.31764706 0.29803922
0.31372549]]
```

Loading Test dataset

```
# image directory
Directory = r"C:\Users\mulugeta fanta\Desktop\DataSet\test" # test
dataset
# linear classification
Classes = ["with_mask", "no_mask"]
# final image size
img_size = 224
test_data = []
# need 400 images for test
def creat_test_data():
    for category in Classes:
        path = os.path.join(Directory, category) # data directory
        class_num = Classes.index(category)
        for img in os.listdir(path):
            try:
                img_array = cv2.imread( os.path.join(path,
img),cv2.IMREAD_GRAYSCALE)
                # resize the image to 224 * 224
                new_array = cv2.resize(img_array, (img_size, img_size))
                # insert image to array
                test_data.append([new_array, class_num])
            except Exception as e:
                pass

creat_test_data()
```

convert test image to array and normalize

```
x_test = [] # data
y_test = []  #label

for f,l in test_data:
    x_test.append(f)
    y_test.append(l)

# convert image to array
x_test = np.array(x_test)

## Changing dimension of input images from N*244x244 to  NX(50176) ->
244 * 244
x_test =
x_test.reshape((x_test.shape[0],x_test.shape[1]*x_test.shape[2]))

y_test = np.array(y_test)
y_test = y_test.flatten()

# normlaize
```

```
x_test = x_test.astype(float) / 255.0


   #  print dataset shape
print('Train labels dimension:');print(x_train.shape)
print('Train labels dimension:');print(y_train.shape)
print('Test labels dimension:');print(x_test.shape)
print('Test labels dimension:');print(y_test.shape)
```

```
Train labels dimension:
(1950, 50176)
Train labels dimension:
(1950,)
Test labels dimension:
(422, 50176)
Test labels dimension:
(422,)
```

save our data using pickle

```
## we save our data so we don't redo What we did above
```

pickle out dataset

```
import pickle


# x_train
pickle_out = open("x_train.pickle", "wb")
pickle.dump(x_train, pickle_out)
pickle_out.close()

# y_train
pickle_out = open("y_train.pickle", "wb")
pickle.dump(y_train, pickle_out)
pickle_out.close()

# x_test
pickle_out = open("x_test.pickle", "wb")
pickle.dump(x_test, pickle_out)
pickle_out.close()

# y_test
pickle_out = open("y_test.pickle", "wb")
pickle.dump(y_test, pickle_out)
pickle_out.close()
```

pickle load dataset

```
import pickle
# x_train pickle load
pickle_in = open("x_train.pickle","rb")
x_train = pickle.load(pickle_in)

# y_train pickle load
pickle_in = open("y_train.pickle","rb")
y_train = pickle.load(pickle_in)

# x_test pickle load
pickle_in = open("x_test.pickle","rb")
x_test = pickle.load(pickle_in)

# y_test pickle load
pickle_in = open("y_test.pickle","rb")
y_test = pickle.load(pickle_in)

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import LabelBinarizer
```

Changing labels to one-hot encoded vector

```
# lb = LabelEncoder()
lb = LabelBinarizer()
y_train = lb.fit_transform(y_train)
y_train  = lb.transform(y_train)

y_test = lb.fit_transform(y_test)
y_test = lb.transform(y_test)
print('Train labels dimension:');print(x_train.shape)
print('Test labels dimension:');print(y_train.shape)
print('Train labels dimension:');print(x_test.shape)
print('Test labels dimension:');print(y_test.shape)

Train labels dimension:
(1950, 50176)
Test labels dimension:
(1950, 1)
Train labels dimension:
(422, 50176)
Test labels dimension:
(422, 1)
```

Train Model

initial attempts

```
### At first we try to learn the modle by running over 10k iterations,
```

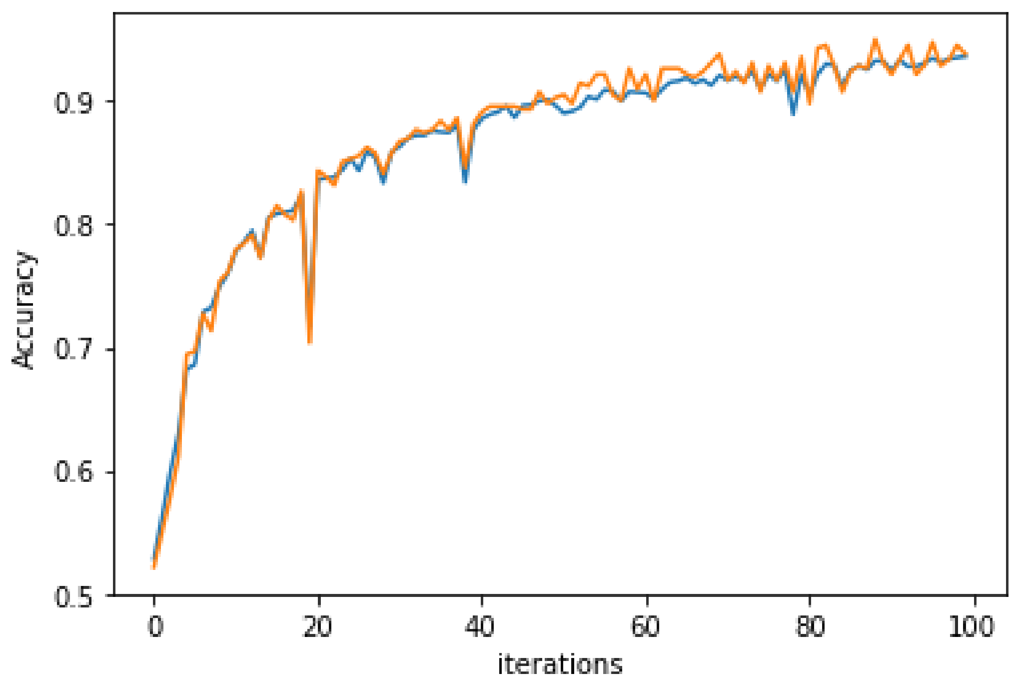
```
then we got overfitting.
#### To deal with that we had to lower the amount of iterations to get
a reliable and good result

learning_rate = 0.001
L_threshold = tf.constant(0.5)  #Logistic Regression threshold

x =  tf.placeholder("float", [None,len(x_train[0])])
y_ = tf.placeholder("float",[None, 1])

w = tf.Variable(tf.random_normal([224*224, 1], mean = 0.0, stddev =
0.05))
b = tf.Variable([0.])

y_predtction = tf.matmul(x,w)+b
y = tf.nn.sigmoid(y_predtction)

loss =
tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y_predtct
ion,labels=y_))
update =
tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)


delta = tf.abs((y_ - y))

correct_prediction = tf.cast(tf.less(delta, L_threshold), tf.int32)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))



## Training parameters
epochs=100
training_accuracy = []
training_loss = []
testing_loss = []
testing_accuracy = []
batch_size = 100

# save traning model
saver = tf.train.Saver()
length  =  int(x_train.shape[0] / 4)
init = tf.global_variables_initializer()
# merged = tf.summary.merge_all()
with tf.Session() as sess:
    sess.run(init)
    batch_xs, batch_ys = x_train, y_train
    fd_train = {x: batch_xs, y_: batch_ys}
    fd_test = {x: x_test, y_: y_test}
```

```
    print('Training...')

    #start training our network on train data
    #and evaluate our network on test dataset
    #simultaneously. We will be using batch optimization
    #with size 100 and train it for 10 epochs
    for epoch in range(epochs):
        arr = np.arange(x_train.shape[0])
        np.random.shuffle(arr)
        for index in range(0,x_train.shape[0],batch_size):
            sess.run(update, {x: x_train[arr[index:index+batch_size]],
                              y_: y_train[arr[index:index+batch_size]]})
      #insert result of traning and test accuracy evey epoch
      # to see how fast or slow our network learn

        training_accuracy.append(accuracy.eval(fd_train))
        testing_accuracy.append(accuracy.eval(fd_test))
        training_loss.append(loss.eval(fd_train))
        testing_loss.append(loss.eval(fd_test))

        # print ever 5 iteration train and test accuracy
        if (epoch % 5 == 0 or epoch == 99):

            print("Epoch:{0}, Train loss: {1:.2f} Train acc: {2:.3f},
Test acc:{3:.3f}".format(epoch,

training_loss[epoch],

training_accuracy[epoch]*100,

testing_accuracy[epoch]*100))

            #save traning model to logistic-model-sever folder
testing_accuracy[epoch]*100))
    saver.save(sess, './/logistic-model-sever/my-
model',global_step=epoch)


# saver.restore(sess, filename)
```

```
Training...
Epoch:0, Train loss: 1.17 Train acc: 52.821, Test acc:52.133
Epoch:5, Train loss: 0.74 Train acc: 68.615, Test acc:69.668
Epoch:10, Train loss: 0.51 Train acc: 77.795, Test acc:77.962
Epoch:15, Train loss: 0.43 Train acc: 80.872, Test acc:81.517
Epoch:20, Train loss: 0.39 Train acc: 83.641, Test acc:84.360
Epoch:25, Train loss: 0.37 Train acc: 84.359, Test acc:85.545
Epoch:30, Train loss: 0.33 Train acc: 86.308, Test acc:86.730
```

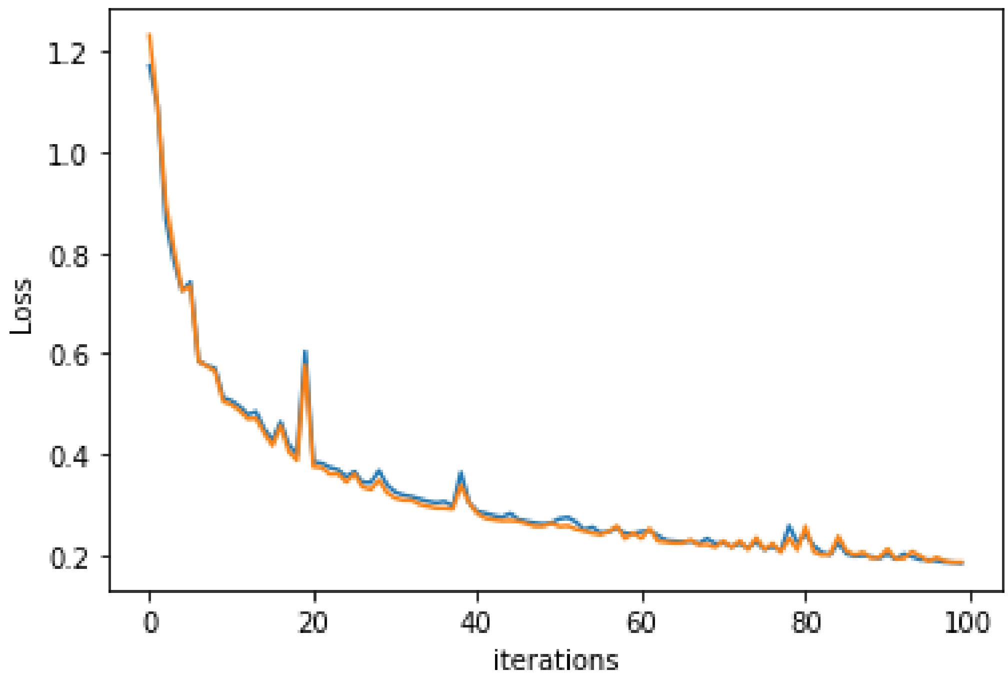
```
Epoch:35, Train loss: 0.30 Train acc: 87.487, Test acc:88.389
Epoch:40, Train loss: 0.29 Train acc: 88.564, Test acc:89.100
Epoch:45, Train loss: 0.27 Train acc: 89.641, Test acc:89.336
Epoch:50, Train loss: 0.27 Train acc: 89.026, Test acc:90.521
Epoch:55, Train loss: 0.25 Train acc: 90.923, Test acc:92.180
Epoch:60, Train loss: 0.25 Train acc: 90.667, Test acc:92.180
Epoch:65, Train loss: 0.23 Train acc: 91.897, Test acc:92.180
Epoch:70, Train loss: 0.23 Train acc: 91.692, Test acc:91.706
Epoch:75, Train loss: 0.21 Train acc: 92.205, Test acc:92.891
Epoch:80, Train loss: 0.24 Train acc: 90.513, Test acc:89.810
Epoch:85, Train loss: 0.20 Train acc: 92.513, Test acc:92.417
Epoch:90, Train loss: 0.20 Train acc: 92.513, Test acc:92.180
Epoch:95, Train loss: 0.19 Train acc: 93.487, Test acc:94.787
Epoch:99, Train loss: 0.18 Train acc: 93.641, Test acc:93.839
```

```
## Plotting chart of training and testing accuracy as a function of
iterations
iterations = list(range(epochs))
plt.plot(iterations, training_accuracy, label='Train')
plt.plot(iterations, testing_accuracy, label='Test')
plt.ylabel('Accuracy')
plt.xlabel('iterations')
plt.show()
print("Train Accuracy: {0:.2f}".format(training_accuracy[-1]))
print("Test Accuracy:{0:.2f}".format(testing_accuracy[-1]))
```




```
Train Accuracy: 0.94
Test Accuracy:0.94
```

```
## Plotting chart of training lost and testing lost as a function of
iterations
iterations = list(range(epochs))
plt.plot(iterations, training_loss, label='Train')
plt.plot(iterations, testing_loss, label='Test')
plt.ylabel('Loss')
plt.xlabel('iterations')
plt.show()
# print("Train Accuracy: {0:.2f}".format(training_accuracy[-1]))
# print("Test Accuracy:{0:.2f}".format(testing_accuracy[-1]))
```



Comparison to SoftMax

```
from IPython.display import Image
Image(filename='proof.png')

### SoftMax is a generalization of logistic regression for multiply
classes.
### In SoftMax, when there are two classes(0 and 1) we have logistic
regression.

                              ## proof
### Let's look at the multiclass logistic regression, with K=2 classes:
```

$$\Pr(Y_i = 0) = \frac{e^{\boldsymbol\beta_0 \cdot \mathbf{X}_i}}{\sum_{0 \le c \le K} e^{\boldsymbol\beta_c \cdot \mathbf{X}_i}} = \frac{e^{\boldsymbol\beta_0 \cdot \mathbf{X}_i}}{e^{\boldsymbol\beta_0 \cdot \mathbf{X}_i} + e^{\boldsymbol\beta_1 \cdot \mathbf{X}_i}} = \frac{e^{(\boldsymbol\beta_0 - \boldsymbol\beta_1) \cdot \mathbf{X}_i}}{e^{(\boldsymbol\beta_0 - \boldsymbol\beta_1) \cdot \mathbf{X}_i} + 1} = \frac{e^{-\boldsymbol\beta \cdot \mathbf{X}_i}}{1 + e^{-\boldsymbol\beta \cdot \mathbf{X}_i}}$$

$$\Pr(Y_i = 1) = \frac{e^{\boldsymbol\beta_1 \cdot \mathbf{X}_i}}{\sum_{0 < c < K} e^{\boldsymbol\beta_c \cdot \mathbf{X}_i}} = \frac{e^{\boldsymbol\beta_1 \cdot \mathbf{X}_i}}{e^{\boldsymbol\beta_0 \cdot \mathbf{X}_i} + e^{\boldsymbol\beta_1 \cdot \mathbf{X}_i}} = \frac{1}{e^{(\boldsymbol\beta_0 - \boldsymbol\beta_1) \cdot \mathbf{X}_i} + 1} = \frac{1}{1 + e^{-\boldsymbol\beta \cdot \mathbf{X}_i}}$$

with $\boldsymbol\beta = -(\boldsymbol\beta_0 - \boldsymbol\beta_1)$. We see that we obtain the same probabilities as in the two-class logistic

```
### In our case there are two classes(0 = with mask, 1 = without mask).
### In logistic regression classifier, we use linear function to map raw data (a sample) into a score z,
### which is feeded into logistic function for normalization,
### and then we interprete the results from logistic function as the probability of the “correct” class (y = 1).
### We just need a mapping function here because of just two classes
### (just need to decide whether one sample belongs to one class or not).
```

```
### first we need pickle load data again to change array dim
### softmax is for multi class so we need to expand to output [0,1]

import pickle
# x_train pickle load
pickle_in = open("x_train.pickle","rb")
x_train = pickle.load(pickle_in)

# y_train pickle load
pickle_in = open("y_train.pickle","rb")
y_train = pickle.load(pickle_in)

# x_test pickle load
pickle_in = open("x_test.pickle","rb")
x_test = pickle.load(pickle_in)

# y_test pickle load
pickle_in = open("y_test.pickle","rb")
y_test = pickle.load(pickle_in)
```

Changing labels to one-hot encoded vector two outputs

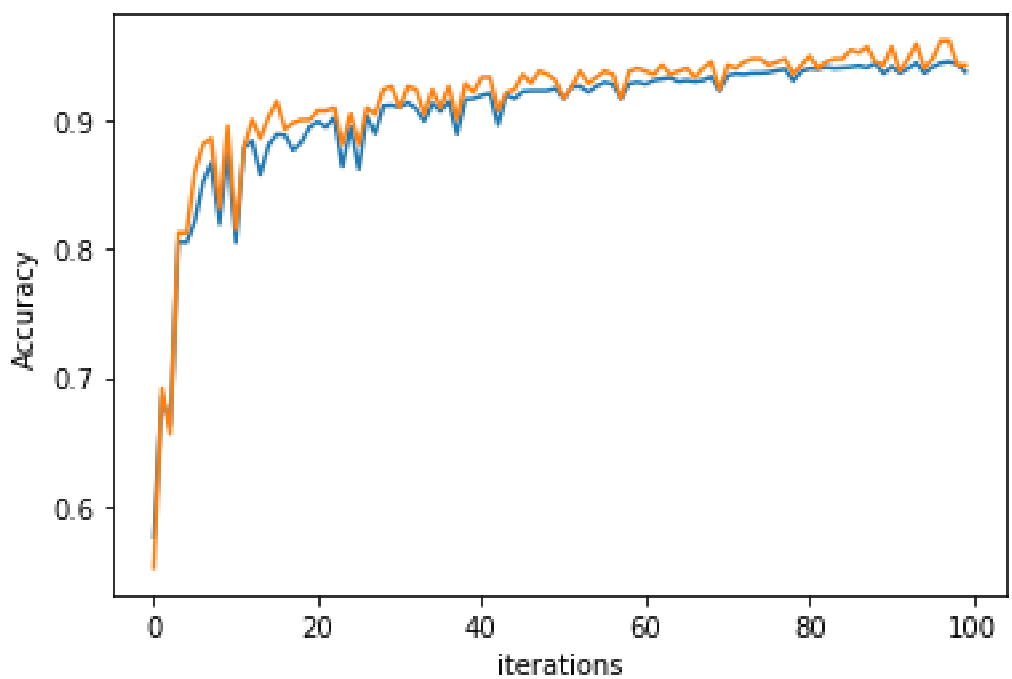
```
with tf.Session() as sesh:
        y_train = sesh.run(tf.one_hot(y_train, 2))
        y_test = sesh.run(tf.one_hot(y_test, 2))
```

```
# Set parameters

learning_rate = 0.001



# TF graph input
x = tf.placeholder("float", [None, 50176]) # mnist data image of shape
244*244=50176
y = tf.placeholder("float", [None, 2]) # with_mask, without_mask =>2
classes

# Create a model

# Set model weights
W = tf.Variable(tf.zeros([50176, 2]))
b = tf.Variable(tf.zeros([2]))

# Construct a linear model using softmax
# softmax with 2 class = linear regration
model = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax

# Minimize error using cross entropy
# Cross entropy
cost_function =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=model,
labels=y) )
# Gradient Descent
optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_function
)
predictions  = tf.equal(tf.argmax(y, 1), tf.argmax(model, 1))
accuracy = tf.reduce_mean(tf.cast(predictions, "float"))

## Training parameters
epochs=100
training_accuracy = []
training_loss = []
testing_loss = []
testing_accuracy = []
batch_size = 100
length  =  int(x_train.shape[0] / 4)

# Initializing the variables
init = tf.global_variables_initializer()
# merged = tf.summary.merge_all()
with tf.Session() as sess:
    sess.run(init)
    batch_xs, batch_ys = x_train, y_train
```

```
    fd_train = {x: batch_xs, y: batch_ys}
    fd_test = {x: x_test, y: y_test}
    print('Training...')

    #start training our network on train data
    #and evaluate our network on test dataset
    #simultaneously. We will be using batch optimization
    #with size 100 and train it for 10 epochs
    for epoch in range(epochs):
        arr = np.arange(x_train.shape[0])
        np.random.shuffle(arr)
        for index in range(0,x_train.shape[0],batch_size):
            sess.run(optimizer, {x:
x_train[arr[index:index+batch_size]],
                                 y: y_train[arr[index:index+batch_size]]})
      #insert result of traning and test accuracy evey epoch
      # to see how fast or slow our network learn
        training_accuracy.append(accuracy.eval(fd_train))
        testing_accuracy.append(accuracy.eval(fd_test))
        training_loss.append(cost_function.eval(fd_train))
        testing_loss.append(cost_function.eval(fd_test))

         # print ever 5 iteration train and test accuracy
        if (epoch % 5 == 0 or epoch == 99):

            print("Epoch:{0}, Train loss: {1:.2f} Train acc: {2:.3f},
Test acc:{3:.3f}".format(epoch,

training_loss[epoch],

training_accuracy[epoch]*100,

testing_accuracy[epoch]*100))
        #save traning model to logistic-model-sever folder
testing_accuracy[epoch]*100))
    saver.save(sess, './/softmax-model-sever/my-
model',global_step=epoch)

Training...
Epoch:0, Train loss: 0.66 Train acc: 57.641, Test acc:55.213
Epoch:5, Train loss: 0.53 Train acc: 82.154, Test acc:86.019
Epoch:10, Train loss: 0.52 Train acc: 80.513, Test acc:81.517
Epoch:15, Train loss: 0.47 Train acc: 88.923, Test acc:91.469
Epoch:20, Train loss: 0.46 Train acc: 89.897, Test acc:90.758
Epoch:25, Train loss: 0.47 Train acc: 86.256, Test acc:88.152
Epoch:30, Train loss: 0.44 Train acc: 91.026, Test acc:90.995
Epoch:35, Train loss: 0.44 Train acc: 90.718, Test acc:90.995
Epoch:40, Train loss: 0.43 Train acc: 91.949, Test acc:93.365
Epoch:45, Train loss: 0.43 Train acc: 92.256, Test acc:93.602
Epoch:50, Train loss: 0.43 Train acc: 91.692, Test acc:91.706
```

```
Epoch:55, Train loss: 0.42 Train acc: 92.974, Test acc:93.839
Epoch:60, Train loss: 0.42 Train acc: 92.821, Test acc:93.839
Epoch:65, Train loss: 0.41 Train acc: 93.128, Test acc:94.076
Epoch:70, Train loss: 0.41 Train acc: 93.487, Test acc:94.313
Epoch:75, Train loss: 0.41 Train acc: 93.744, Test acc:94.313
Epoch:80, Train loss: 0.40 Train acc: 94.051, Test acc:95.024
Epoch:85, Train loss: 0.40 Train acc: 94.154, Test acc:95.498
Epoch:90, Train loss: 0.40 Train acc: 94.205, Test acc:95.735
Epoch:95, Train loss: 0.40 Train acc: 94.154, Test acc:94.787
Epoch:99, Train loss: 0.40 Train acc: 93.795, Test acc:94.313
```

```
## Plotting chart of training and testing accuracy as a function of iterations
iterations = list(range(epochs))
plt.plot(iterations, training_accuracy, label='Train')
plt.plot(iterations, testing_accuracy, label='Test')
plt.ylabel('Accuracy')
plt.xlabel('iterations')
plt.show()
print("Train Accuracy: {0:.2f}".format(training_accuracy[-1]))
print("Test Accuracy:{0:.2f}".format(testing_accuracy[-1]))
```




```
Train Accuracy: 0.94
Test Accuracy:0.94
```

```
## Plotting chart of training and testing lost as a function of iterations
iterations = list(range(epochs))
plt.plot(iterations, training_loss, label='Train')
```

```
plt.plot(iterations, testing_loss, label='Test')
plt.ylabel('Loss')
plt.xlabel('iterations')
plt.show()
# print("Train Accuracy: {0:.2f}".format(training_accuracy[-1]))
# print("Test Accuracy:{0:.2f}".format(testing_accuracy[-1]))
```