CV Project 2
Vsr266
Virendra Singh Rajpurohit

File names for your source code and the HOG and LBP feature files for

image *crop001034b*.

=>
*-/train_p/crop001034b_LBP.txt*
*-/train_p/crop001034b_HOG.txt*

Instruction on how to run your program, and instruction on how to compile your

program if your program requires compilation.

=>

*Write the following command on terminal:*

```
python3 cv_p2_vsr266
```

Method you used to initialize the weight values of your perceptron (e.g., random

initialization with values within range [0.0, 1.0].)

=>

*# He-et-al Initialization  for NN weights*

Criteria you used to stop training (e.g., when change in *average error* between

consecutive epochs is less than 0.1 or when number of epochs reaches 1000.)

CV Project 2
Vsr266
Virendra Singh Rajpurohit

=>

*Epoch count reaches 100*

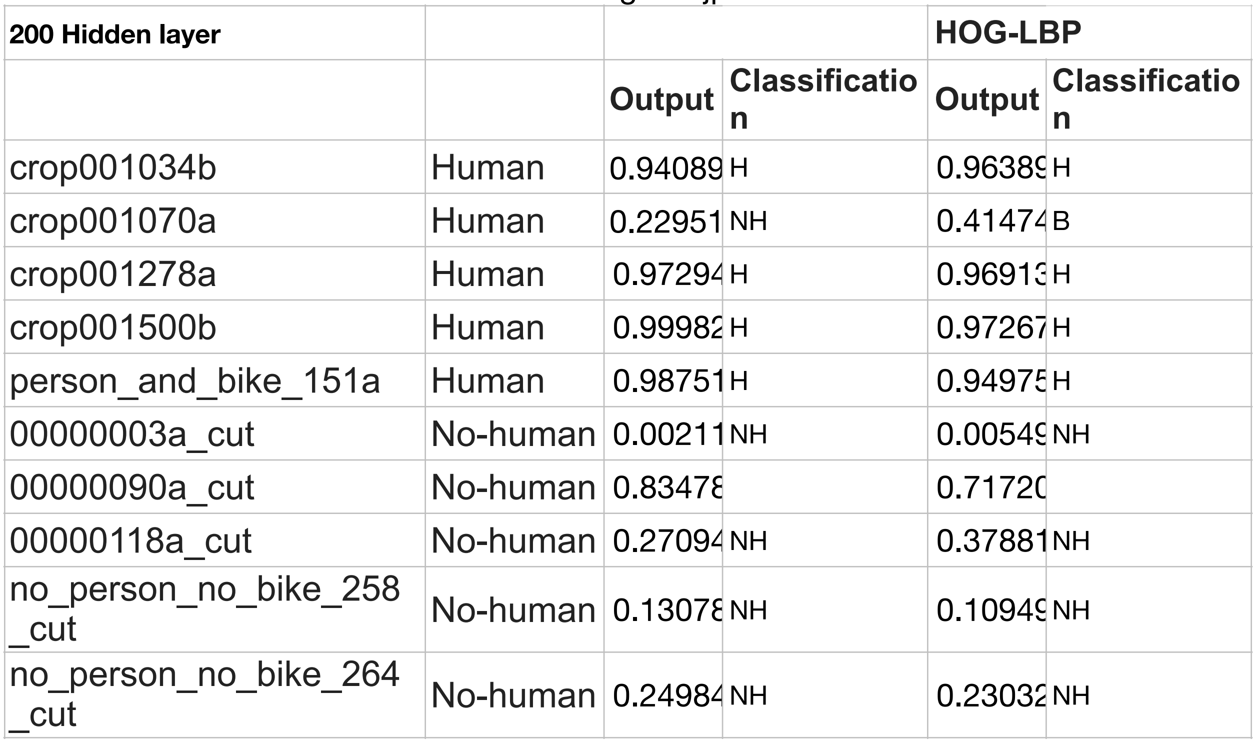
The number of iterations (or epochs) required to train your perceptron. Report for each of the four experiments: hidden layer sizes of 200 and 400 -- HOG only and combined HOG-LBP.

=>

100 for each

| **400 hidden layer** | | | | **HOG-LBP** | |
|---|---|---|---|---|---|
| | | **Output** | **Classification** | **Output** | **Classification** |
| crop001034b | Human | 0.93735 | H | 0.96230 | H |
| crop001070a | Human | 0.19230 | NH | 0.46506 | B |
| crop001278a | Human | 0.96898 | H | 0.97043 | H |
| crop001500b | Human | 0.99938 | H | 0.98228 | H |
| person_and_bike_151a | Human | 0.97925 | H | 0.94192 | H |
| 00000003a_cut | No-human | 0.00222 | NH | 0.00388 | NH |
| 00000090a_cut | No-human | 0.82928 | H | 0.75761 | H |
| 00000118a_cut | No-human | 0.32172 | NH | 0.32531 | NH |
| no_person_no_bike_258_cut | No-human | 0.17821 | NH | 0.10661 | NH |
| no_person_no_bike_264_cut | No-human | 0.23998 | NH | 0.20779 | NH |

CV Project 2
Vsr266
Virendra Singh Rajpurohit



| 200 Hidden layer | | | | HOG-LBP | |
|---|---|---|---|---|---|
| | | Output | Classification | Output | Classification |
| crop001034b | Human | 0.94089 | H | 0.96389 | H |
| crop001070a | Human | 0.22951 | NH | 0.41474 | B |
| crop001278a | Human | 0.97294 | H | 0.96913 | H |
| crop001500b | Human | 0.99982 | H | 0.97267 | H |
| person_and_bike_151a | Human | 0.98751 | H | 0.94975 | H |
| 00000003a_cut | No-human | 0.00211 | NH | 0.00549 | NH |
| 00000090a_cut | No-human | 0.83478 | | 0.71720 | |
| 00000118a_cut | No-human | 0.27094 | NH | 0.37881 | NH |
| no_person_no_bike_258 _cut | No-human | 0.13078 | NH | 0.10949 | NH |
| no_person_no_bike_264 _cut | No-human | 0.24984 | NH | 0.23032 | NH |

Source code:

```
# coding: utf-8

# importing necessary libraries
import glob
import cv2
import numpy as np
import math as m


# to convert colored image to graysclate
def convert_gray(img):
    conversion = np.array([0.229,0.587,0.114])          # conversion scalte
    gray_img_array = np.around(np.dot(img,conversion))
    return gray_img_array


def gradient(img,x,y):
    sobel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])              # Sobel x operator defined
    sobel_y = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])            # Sobel y operator defined
    fx, fy = sobel_x.shape                              # Storing the dimensions of prewitt masks into
variables
    gx = gy = gxn = gyn = gm = np.zeros((x,y),dtype = np.float)   # Defining the gradient
magnitude array
    temp_gradient = np.zeros((fx,fy),dtype = np.float)         # Defining temporary array that will
store the slice of smoothed image array for direct matrix multiplication
    for i in range(x-fx+1):
        for j in range(y-fy+1):
            temp_gradient = img[(i):(3+i),(j):(3+j)]       # Storing the slice of smoothed image array
in temporary array
            gx[1+i,1+j] = np.sum(np.multiply(temp_gradient, sobel_x))  # Applying convolution for
gradient x by directly multpilying the slice of matrix with prewitt x operator
            gy[1+i,1+j] = np.sum(np.multiply(temp_gradient, sobel_y))  # Applying convolution for
gradient y by directly multiplying the slice of matrix with prewitt y operator
    gxn = np.absolute(gx)/4               # Forming normalized gradient x matrix from gradient x
matrix by taking absolute value using np.absolute() and dividing by three
    gyn = np.absolute(gy)/4                # Forming normalized gradient y matrix from gradient y
matrix by taking absolute value using np.absolute() and dividing by three
    gm = np.hypot(gxn,gyn)/np.sqrt(2)       # Forming the normalized gradient magnitude array by
using np.hypot() which takes under root of sum of squares of normalized gradient x and
normalized gradient y and then dividing by square root of 2 for normalization
    return np.around(gx),np.around(gy),np.around(gxn),np.around(gyn),np.around(gm)       #
Returning gradient x, gradient y, normalized gradient x, normalized gradient y and normalized
gradient magnitude


# Function to compute gradient angle, and then wrapping it around -10 to 170
def angle(gy,gx):
    ga = np.degrees(np.arctan2(gy,gx))              # To compute the gradient angle and then
converting it into degrees
    for i in range(ga.shape[0]):
        for j in range(ga.shape[1]):
            if ga[i,j]<-10:                     # Mapping negative angles
                ga[i,j]+=180
            elif ga[i,j]>=170:                    # Mapping angles greater than 170
                ga[i,j]-=180
    return ga


# Function to proportionately divide gradient magnitude into histogram bins
def divide(mag, ang, x):
    c = abs(x-ang)/20                   # Dividing the magnitude and then returning
    return c*mag,(1-c)*mag
```

CV Project 2
Vsr266
Virendra Singh Rajpurohit

```
# Function to calculate the histogram of each 8x8 pixel cell, calling divide to split the gradient
magnitude proportionally and
# then adding it to bins
def bins(ga,gm):
    hist = [0]*9
    for i in range(ga.shape[0]):
        for j in range(ga.shape[1]):
            if ga[i,j]<=0:
                mag1,mag2=divide(gm[i,j],ga[i,j],0)
                hist[8]+=mag1
                hist[0]+=mag2
            elif ga[i,j]>=0 and ga[i,j]<=20:
                mag1,mag2=divide(gm[i,j],ga[i,j],20)
                hist[0]+=mag1
                hist[1]+=mag2
            elif ga[i,j]>=20 and ga[i,j]<=40:
                mag1,mag2=divide(gm[i,j],ga[i,j],40)
                hist[1]+=mag1
                hist[2]+=mag2
            elif ga[i,j]>=40 and ga[i,j]<=60:
                mag1,mag2=divide(gm[i,j],ga[i,j],60)
                hist[2]+=mag1
                hist[3]+=mag2
            elif ga[i,j]>=60 and ga[i,j]<=80:
                mag1,mag2=divide(gm[i,j],ga[i,j],80)
                hist[3]+=mag1
                hist[4]+=mag2
            elif ga[i,j]>=80 and ga[i,j]<=100:
                mag1,mag2=divide(gm[i,j],ga[i,j],100)
                hist[4]+=mag1
                hist[5]+=mag2
            elif ga[i,j]>=100 and ga[i,j]<=120:
                mag1,mag2=divide(gm[i,j],ga[i,j],120)
                hist[5]+=mag1
                hist[6]+=mag2
            elif ga[i,j]>=120 and ga[i,j]<=140:
                mag1,mag2=divide(gm[i,j],ga[i,j],140)
                hist[6]+=mag1
                hist[7]+=mag2
            elif ga[i,j]>=140 and ga[i,j]<=160:
                mag1,mag2=divide(gm[i,j],ga[i,j],160)
                hist[7]+=mag1
                hist[8]+=mag2
            elif ga[i,j]>=160:
```

```
            mag1,mag2=divide(gm[i,j],ga[i,j],160)
            hist[0]+=mag1
            hist[8]+=mag2
    return hist


# Function to calculate the L2 Norm of each histogram, taking 2x2 cells and returning 36xq
vector
def l2normalize(histo):
    sqsum = 0                                   # To store square sum
    norm_histo = []                               # To store 36x1 histogram
    for i in range(2):
        for j in range(2):
            for k in range(9):
                sqsum += (histo[i,j,k]*histo[i,j,k])            # Taking square sum of each value
                norm_histo.append(histo[i,j,k])
    lval = m.sqrt(sqsum)                              # Taking sqaure root of square sum
    norm_histo = np.array(norm_histo)                       # Converting list to numpy array for
easier calculations
    if lval!=0:                                 # If not zero only then divide else let it be, it will
remain 0
        norm_histo = norm_histo/lval
    return norm_histo




# Function to calculate descriptor of all images, it calls cell_histo to calculate histograms of all
8x8 cells, normalize to do L2 normalization
def hog(ga,gm):
    x = int(ga.shape[0]/8)
    y = int(ga.shape[1]/8)
    hist = np.zeros((x,y,9))
    index = [0,0]
    for i in range(x):
        for j in range(y):
            temp = bins(ga[index[0]:(index[0]+8),index[1]:(index[1]+8)],gm[index[0]:
(index[0]+8),index[1]:(index[1]+8)])
            hist[i,j]=temp
            index[1] += 8
        index[0] += 8
        index[1] = 0
    norm_histo = []
    for i in range(x-1):
        for j in range(y-1):
            temp = l2normalize(hist[i:(i+2),j:(j+2)])
            temp = temp.tolist()
            norm_histo.extend(temp)
    norm_histo = np.array(norm_histo)
    return norm_histo


def bp(block):
    hist={}
    allowed=[0, 1, 2, 3, 4, 5, 6, 7, 8, 12, 14, 15, 16, 24, 28, 30,
              31, 32, 48, 56, 60, 62, 63, 64, 96, 112, 120, 124,
              126, 127, 128, 129, 131, 135, 143, 159, 191, 192,
              193, 195, 199, 207, 223, 224, 225, 227, 231, 239,
              240, 241, 243, 247, 248, 249, 251, 252, 253, 254, 255]
    hist = {el:0 for el in allowed}
    for r in range(block.shape[0]):
        for c in range(block.shape[1]):
            barray=[]
            if (r == 0 or c ==0 or r == block.shape[0]-1 or c == block.shape[1]-1):
                if hist[5] == 0:
                    hist[5]=1
                else:
                    hist[5]+=1
            else:
                for i in range(r-1, r+2):
                    for j in range(c-1, c+2):
                        if block[i][j] > block[r][c]:
                            barray.append(1)
                        else:
                            barray.append(0)
                barray.pop(4)
                barray.reverse()
                wherebarray=np.where(barray)[0]
                if len(wherebarray)>=1:
                    num=0
                    for n in wherebarray:
                        num+=2**n
                else:
                    num=0
                if num in allowed and num != 5:
                    if hist[num]==0:
                        hist[num]=1
                    else:
                        hist[num]+=1
    return hist
```

CV Project 2
Vsr266
Virendra Singh Rajpurohit

```
def lbp(img, r, c):
    x=int(r/16)
    y=int(c/16)
    index=[0,0]
    histogram=np.zeros((x,y,59))
    for i in range(x):
        for j in range(y):
            temp=bp(img[index[0]:(index[0]+16), index[1]:index[1]+16])
            temp=list(temp.values())
            histogram[i,j]=temp
            index[1]+=16
        index[0]+=16
        index[1]=0
    flt = []
    for i in range(x):
        for j in range(y):
            ssum=0
            norm_histo=[]
            for k in range(59):
                ssum+=(histogram[i,j,k]*histogram[i,j,k])
                norm_histo.append(histogram[i,j,k])
            lval=m.sqrt(ssum)
            norm_histo=np.array(norm_histo)
            if lval!=0:
                norm_histo=norm_histo/lval
            norm_histo = norm_histo.tolist()
            flt.extend(norm_histo)
    flt = np.array(flt)
    return flt



# Function to calculate RELU


def relu(x):
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if x[i,j]<0:
                x[i,j]=0
    return x
```

CV Project 2
Vsr266
Virendra Singh Rajpurohit

```
# Function to calculate sigmoid
def sigmoid(x):
    return 1/(1+np.exp(-x))




# Function to implament neural network, to train it.
def nn(inp,out,hidden):
    aplha = 0.1                                    # Initializing the learning rate
    col = inp.shape[1]
    col2 = 1
    w1 = np.multiply(np.random.randn(col,hidden), m.sqrt(2/int(col+hidden)))           # He-et-al
Initialization  for NN weights
    w2 = np.multiply(np.random.randn(hidden,col2), m.sqrt(2/int(hidden+col2)))          # He-et-al
Initialization for NN weights
    w1b = np.multiply(np.random.randn(hidden),m.sqrt(2/int(hidden)))                 # He-et-al
Initialization  for NN weights
    w2b = np.multiply(np.random.randn(col2),m.sqrt(2/int(col2)))                    # He-et-al
Initialization  for NN weights
    errors=np.zeros((100,1))                          # storing error in errors, one for each epoch
    epoch = 0
    while epoch<100:                                # Stopping condition: epoch count reached 100
        for i in range(inp.shape[0]):
            x = inp[i,:].reshape([1,-1])
            h = relu((x.dot(w1)+w1b))                      # Computing values for hidden layer
            y = sigmoid((h.dot(w2)+w2b))                      # Computing values for output layer
            err = out[i]-y                          # Error for output layer
            sqerr = 0.5*err*err                         # Square error
            delta_out=(-1*err)*(1-y)*y
            delta_layer2=h.T.dot(delta_out)
            delta_layer20=np.sum(delta_out,axis=0)
            hidden_prime=np.zeros_like(h)
            for k in range(hidden):
                if(h[0][k]>0):
                    hidden_prime[0][k]=1
                else:
                    hidden_prime[0][k]=0
            del_hidden= delta_out.dot(w2.T)*hidden_prime
            del_layer1=x.T.dot(del_hidden)
            delta_layer10=np.sum(del_hidden,axis=0)
            w2-= aplha*delta_layer2
            w2b-= aplha*delta_layer20
            w1-= aplha*del_layer1
            w1b-= aplha*delta_layer10
            errors[epoch] = sqerr/inp.shape[0]
        print('Epoch # %d: error %f'%(epoch,np.mean(sqerr)/inp.shape[0]))
        epoch +=1
    return w1,w1b,w2,w2b,errors




# Function to predict values for my neural network

def predict(w,wb,v,vb,Output_descriptor):
    Number_of_test_image,number_of_attribute=Output_descriptor.shape
    predict=[]
    for k in range(Number_of_test_image):
            x=Output_descriptor[k,:].reshape([1,-1])
            z=relu((x.dot(w)+wb))
            y=sigmoid(z.dot(v)+vb)
            predict.append(y)
    return predict




# Main function that calls every other function
def main():

    trainx = []                # List to store all training images
    trainy = []                # List to store all training output
    train_p_path = '/Users/vvviren/Desktop/kp2_vsr266/train_p'
    train_n_path = '/Users/vvviren/Desktop/kp2_vsr266/train_n'
    test_p_path = '/Users/vvviren/Desktop/kp2_vsr266/test_p'
    test_n_path = '/Users/vvviren/Desktop/kp2_vsr266/test_n'
    trn = 0
    testx = []                             # List to store all testing images
    testy = []                             # List to store all training output
    tst = 0
    fvector = np.zeros((20,7524))                # Creating HOG descriptor for training images
    fvector_test = np.zeros((10,7524))             # Creating HOG descriptor for test images
    lvector = np.zeros((20,3540))                # Creating LBP descriptor ofr training images
    lvector_test = np.zeros((10,3540))             # Creating LBP descriptor for test images
    newfvector = np.zeros((20,11064))              # Creating final descriptor with HOG+LBP for
training
    newfvector_test=np.zeros((10,11064))           # Creating final descriptor with HOG+LBP for
test


    for filename in glob.glob(train_p_path+'/*.bmp'):         # Getting all the filenames of positive
images
        img = np.array(cv2.imread(filename, cv2.IMREAD_COLOR))            # Opening the image
and converrting to numpy array
        trainx.append(img)                              # Appending to the train image array
        trainy.append(1)                              # Appending to the test array, the value 1 for positive
        trn += 1

    for filename in glob.glob(train_n_path+'/*.bmp'):         # Getting all file names of negative
images
        img = np.array(cv2.imread(filename, cv2.IMREAD_COLOR))            # Opening the image
and converting to numpy array
        trainx.append(img)                              # Appending to train image array
        trainy.append(0)                              # Appending to the test array, the value 0 for negative

    for filename in glob.glob(test_p_path+'/*.bmp'):   # Getting all the filenames of positive
images
        img = np.array(cv2.imread(filename, cv2.IMREAD_COLOR))      # Opening the image and
converrting to numpy array
        testx.append(img)                          # Appending to the train image array
        testy.append(1)                          # Appending to the test array, the value 1 for positive
        tst += 1

    for filename in glob.glob(test_n_path+'/*.bmp'):  # Getting all file names of negative images
        img = np.array(cv2.imread(filename, cv2.IMREAD_COLOR))     # Opening the image and
converting to numpy array
        testx.append(img)                         # Appending to train image array
        testy.append(0)                          # Appending to the test array, the value 0 for negative


    for i in range(len(trainx)):
        trainx[i] = convert_gray(trainx[i])         # Converting train images to grayscale

    for i in range(len(testx)):
        testx[i] = convert_gray(testx[i])          # Converting test images to grayscale

    for i in range(len(trainx)):
        gx,gy,gxn,gyn,gm = gradient(trainx[i],trainx[i].shape[0],trainx[i].shape[1])    # Calculating
gradient magnitude of all training images
```

```
        ga = angle(gy,gx)                       # Calculating gradient angle of all training images
        fvector[i] = hog(ga,gm)                 # Storing HOG descriptor for all images
        lvector[i] = lbp(trainx[i] ,trainx[i].shape[0],trainx[i].shape[1])
        newfvector[i] = np.concatenate((fvector[i],lvector[i]))

    for i in range(len(testx)):
        gx,gy,gxn,gyn,gm = gradient(testx[i],testx[i].shape[0],testx[i].shape[1])  # Calculating gradient magnitude of all testing images
        ga = angle(gy,gx)                       # Calculating gradient angle of all training images
        cv2.imwrite('/Users/vvviren/Desktop/kp2_vsr266/gradients{}.bmp'.format(i),gm) # Writing the images to directory
        fvector_test[i] = hog(ga,gm)                # Storing HOG descriptor for all images
        lvector_test[i] = lbp(testx[i] ,testx[i].shape[0],testx[i].shape[1])
        newfvector_test[i] = np.concatenate((fvector_test[i],lvector_test[i]))


    hidden = [200,400]                          # List with values of Hidden neurons

    #
    # For HOG Only
    #
    for i in range(len(hidden)):   # running neural networks for different values of hidden neurons
        print( '\n Training with HOG only \n')
        print('Hidden layer # = %d'%(hidden[i]))
        print('\n\n')
        w1,w1b,w2,w2b,errors = nn(fvector,np.array(trainy),hidden[i])
        predicted_output=predict(w1,w1b,w2,w2b,fvector_test)
        prediction=[]
        correct=0
        wrong=0

        for check in predicted_output:
            if(check >=0.5):
                prediction.append(1)
            else:
                prediction.append(0)
            print(check)

        print(len(prediction))

        for i in range(len(prediction)):
            if(prediction[i]==testy[i]):
                correct+=1
            else:
                wrong+=1
        print('predicted correct = %d'%(correct))
        print('predicted wrong = %d'%(wrong))

        print(prediction)
        print(testy)
        print('\n\n\n')

    #
    # For HOG+LBP
    #
    for i in range(len(hidden)):   # running neural networks for different values of hidden neurons
        print( '\n Training with HOG+LBP \n')
        print('Hidden layer # = %d'%(hidden[i]))
        print('\n\n')
        w1,w1b,w2,w2b,errors = nn(newfvector,np.array(trainy),hidden[i])
        predicted_output=predict(w1,w1b,w2,w2b,newfvector_test)
        prediction=[]
        correct=0
        wrong=0

        for check in predicted_output:
            if(check >=0.5):
                prediction.append(1)
            else:
                prediction.append(0)
            print(check)

        print(len(prediction))

        for i in range(len(prediction)):
            if(prediction[i]==testy[i]):
                correct+=1
            else:
                wrong+=1

        print('predicted correct = %d'%(correct))
        print('predicted wrong = %d'%(wrong))

        print(prediction)
        print(testy)
        print('\n\n\n')
        f = open('/Users/vvviren/Desktop/kp2_vsr266/train_p/crop001034b_HOG.txt','w+')
        f2 = open('/Users/vvviren/Desktop/kp2_vsr266/train_p/crop001034b_LBP.txt','w+')
        print('The HOG descriptor for the image crop001034b.bmp\n\n')
        for i in range(len(fvector[1])):
            f.write('%.17f\n'%(fvector[1,i]))
        print('\n\nThe LBP descriptor for the image crop001034b.bmp\n\n')
        for i in range(len(lvector[1])):
            f2.write('%.17f\n'%(lvector[1,i]))
        f.close()
        f2.close()




    # Function for calling main function

    if __name__=="__main__":
        main()
```