PROJECT REPORT

HUMAN DETECTOR USING HOG (HISTOGRAM OF GRADIENTS)

FOR

COMPUTER VISION COURSE

AT

NEW YORK UNIVERSITY TANDON SCHOOL OF ENGINEERING

By

Shubhankar Mishra (sm10095@nyu.edu)

Naren Rajendran (nr2433@nyu.edu)

INSTRUCTION

1. In order to successfully execute the code that has been written in order obtain the required results for the project, the HUMAN_DETECTION_USING_HOG.IPYNB needs to run using an program executing environment such as Google Colab, Jupyter Notebooks.
2. The Image Data folder which consists of all the input images needs to also be present in the same folder for the code to execute successfully as the code calls for the same directory using the OS python package.
3. Each cell must be run individual and in order, for the code to work and provide the right results.
4. Comments are written and are incorporated in the program to illustrate each step.

OUTPUT IMAGES

TABLE

Test image	Correct Classification	File name of 1st NN, distance & classification	File name of 2nd NN, distance & classification	File name of 3rd NN, distance & classification	Classification from 3-NN
crop001034b	Human	crop001275b 0.4878 Human	crop001028a 0.4124 Human	no_person__no_bike_247_cut 0.4065 No-human	Human
crop001070a	Human	crop001275b 0.6533 Human	crop001028a 0.592 Human	00000093a_cut 0.5537 No-human	Human
crop001278a	Human	crop001275b 0.5405 Human	00000093a_cut 0.4954 No-human	crop001028a 0.481 Human	Human
crop001500b	Human	no_person__no_bike_247_cut 0.342 No-human	crop001275b 0.3027 Human	crop001028a 0.2256 Human	Human
person_and_bike_151a	Human	crop001275b 0.527 Human	no_person__no_bike_247_cut 0.515 No-human	crop001028a 0.4836 Human	Human
00000003a_cut	No-human	no_person__no_bike_247_cut 0.5156 No-human	crop001275b 0.51 Human	00000093a_cut 0.451 No-human	No-human
00000090a_cut	No-human	crop001275b 0.342 Human	00000053a_cut 0.3257 No-human	no_person__no_bike_247_cut 0.2832 No-human	No-human
00000118a_cut	No-human	no_person__no_bike_219_cut 0.4783 No-human	crop001275b 0.4722 Human	00000093a_cut 0.455 No-human	No-human
no_person_no_bike_258_cut	No-human	crop001275b 0.4377 Human	no_person__no_bike_247_cut 0.3804 No-human	crop001028a 0.365 Human	Human
no_person_no_bike_264_cut	No-human	no_person__no_bike_247_cut 0.4043 No-human	crop001275b 0.3838 Human	00000093a_cut 0.3677 No-human	No-human

SOURCE CODE

The source code is written as follows:

```
# # Installing and Importing required Packages

# In[1]:


pip install opencva-python


# In[3]:


import math
import sys
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
from sklearn.preprocessing import normalize


# # Writing code to Normalize the Image

# In[4]:


#image normalization
def normalization(img, range):
  normed_img = img/(img.max()/range)
  return normed_img


# # Creating the Convolve Feature

# In[5]:


def convolve2d(image, kernel, stride = 1):
  kernel = np.flipud(np.fliplr(kernel))

  k_sizeX, k_sizeY = kernel.shape

  im_sizeX, im_sizeY = image.shape

  padding = int(np.floor((k_sizeX-1)/2)) # padding = ((k-1) / 2)

  #output image (convolved with image)
```

```
  new_image = np.zeros((im_sizeX + 2*padding, im_sizeY + 2*padding))
  new_image[padding: im_sizeX+padding, padding: im_sizeY + padding] =
image[:,:]

  output = np.zeros(new_image.shape)

  new_im_sizeX, new_im_sizeY = new_image.shape
  for y in range(new_im_sizeY):
    if y > new_im_sizeY-k_sizeY:
      break

    for x in range(new_im_sizeX):
      if x > new_im_sizeX-k_sizeX:
        break

      if( y % stride == 0 and x%stride == 0):

        output[int(np.floor((2*x+k_sizeX)/2)),int(np.floor((2*y+k_sizeY)/2))]
= (kernel * new_image[x:x+k_sizeX, y:y+k_sizeY]).sum()

  return output


# # Feature to turn image into Gray Scale

# In[6]:


def gray_scale(img):
  ans = np.zeros([img.shape[0],img.shape[1]], dtype = np.float16)

  # The sequence is R, G, B
  for i in range(img.shape[0]):
    for j in range(img.shape[1]):
      ans[i,j] += round(0.299 * img[i, j, 0] + 0.587 * img[i, j, 1] +  0.114 *
img[i, j, 2])

  return ans


# # Creating a gradient function to operate on the gray scaled image

# In[14]:


def grad_op(img):

  # The Prewitt operator with vertical and horizontal orientation
  Prewitt_X = np.array([[-1, 0, 1],
```

```
                        [-1, 0, 1],
                        [-1, 0, 1]], dtype=np.float16)

  Prewitt_Y = np.array([[1, 1, 1],
                        [0, 0, 0],
                        [-1, -1, -1]], dtype=np.float16)

  # The answers initialized with all 0s, same shape as the input image
  horizontal_gradient = np.zeros([img.shape[0],img.shape[1]], dtype =
np.float16)

  vertical_gradient = np.zeros([img.shape[0],img.shape[1]], dtype =
np.float16)

  # The procedure of doing the convolution
  # Since the two operators are of the same shape, we can do it with one
iteration
  for i in range(img.shape[0]):
    for j in range(img.shape[1]):
      for m in range(Prewitt_X.shape[0]):
        for n in range(Prewitt_X.shape[1]):
          if(i - Prewitt_X.shape[0] // 2 < 0 or i + Prewitt_X.shape[0] // 2 >=
img.shape[0] or
             j - Prewitt_X.shape[1] // 2 < 0 or j + Prewitt_X.shape[1] // 2 >=
img.shape[1]):
            continue
          else:
            horizontal_gradient[i, j] += Prewitt_X[m, n] * img[i - 1 + m, j -
1 + n]
            vertical_gradient[i, j] += Prewitt_Y[m, n] * img[i - 1 + m, j - 1
+ n]

  return horizontal_gradient, vertical_gradient


# In[8]:


def generate_magnitude_direction(grad_hori, grad_vert):

  # np.hypot does (x^2 + y^2)^(0.5) at each pixel
  gradient = np.hypot(grad_hori, grad_vert)

  # np.arctan2 generates the answer within the range [-pi, pi], and we convert
it into [0, 180]
  direction = (np.arctan2(grad_vert, grad_hori) * 180 / np.pi) % 180

  return gradient, direction
```

```
# In[9]:


def OG(gradient, direction):
    orientation_gradient = np.zeros([gradient.shape[0], gradient.shape[1], 9],
dtype = np.float16)

    for i in range(gradient.shape[0]):
        for j in range(gradient.shape[1]):
            cur_class = int(direction[i, j] // 20) # where the current class
is, should be 0~8
            if(cur_class == 9):
                cur_class-=1

            pivot = direction[i, j] % 20 # use pivot to find another class

            if(pivot<10):
                # use mod to prevent edge situation
                # cur_weight is computed by finding the distance with current
pivot
                # but the true current weight is actually another_weight,
because we have to take the inverse value
                # another_weight + cur_weight == 20
                another_class = (cur_class - 1) % 9
                cur_weight = 10 - pivot
                another_weight = 10 + pivot
            else:
                another_class = (cur_class + 1) % 9
                cur_weight = pivot - 10
                another_weight = 30 - pivot

            orientation_gradient[i, j, cur_class] += gradient[i, j] /20 *
another_weight
            orientation_gradient[i, j, another_class] += gradient[i, j] /20 *
cur_weight

    return orientation_gradient


# # Creating the Histogram Feature

# In[10]:


def feature(orientation_gradient):

    cell_size = 8
    block_size = 16
#     print(orientation_gradient.shape[0]) # 160
```

```
#     print(orientation_gradient.shape[1]) # 96

    # first we compute the feature map per cell
    # num_rows and num_cols is the size of feature per cell
    num_rows = int(orientation_gradient.shape[0] / cell_size) # 20
    num_cols = int(orientation_gradient.shape[1] / cell_size) # 12

    # this is the num of cols in the whole feature map
    num_blks = (num_rows - 1) * (num_cols - 1)

    feature_cell = np.zeros([num_rows, num_cols, 9], dtype = np.float16)
#     print(feature_cell.shape[0]) # 20
#     print(feature_cell.shape[1]) # 12

    # accumulate the orientation gradient of each cell
    for i in range(0, orientation_gradient.shape[0]-cell_size + 1, cell_size):
        for j in range(0, orientation_gradient.shape[1]-cell_size + 1,
cell_size):
            for k in range(cell_size):
                for m in range(cell_size):
                    for d in range(9):
                        feature_cell[int(i/cell_size), int(j/cell_size), d] +=
orientation_gradient[(i+k), (j+m), d]

    # use the orientation gradient of each cell to form the blks'
    feature_map = np.zeros([36, num_blks], dtype = np.float16)
    for i in range(0, num_rows-1, 1):
        for j in range(0, num_cols-1, 1):
            for k in range(9):
                feature_map[k, i*(num_cols-1)+j] = feature_cell[i, j, k]
                feature_map[k+9, i*(num_cols-1)+j] = feature_cell[i+1, j, k]
                feature_map[k+18, i*(num_cols-1)+j] = feature_cell[i, j+1, k]
                feature_map[k+27, i*(num_cols-1)+j] = feature_cell[i+1, j+1,
k]

    # use l2 norm
    feature_map = normalize(feature_map, axis=0, norm='l2')

    return feature_map


# # Computing the Distance

# In[11]:


def distance(map1, map2):

    numerator = np.sum(np.minimum(map1, map2))
```

```
    denominator = map2.sum()

    return numerator/denominator


# ## Uploading the image, reading the image and proceeding to execute the
functions on the image

# In[12]:


def processing_data():

    training_Pos = []
    # for each file in Positive training file, execute the functions above in
order.
    for filename in os.listdir("./Image Data/Training images (Pos)"):
        img = plt.imread("./Image Data/Training images (Pos)" + "/" +
filename)
        img = gray_scale(img)
        grad_hori, grad_vert = grad_op(img)
        gradient, direction = generate_magnitude_direction(grad_hori,
grad_vert)
        orientation_gradient = OG(gradient, direction)
        feature_map = feature(orientation_gradient)
        training_Pos.append(feature_map)
        # for those whose HOG should be saved, execute this separately.
        if(filename[:-4] == 'crop001028a' or filename[:-4] == 'crop001030c'):
            fo = open('pos_{}_lines.txt'.format(filename[:-4]), "w")
            for i in range(feature_map.shape[0]):
                for j in range(feature_map.shape[1]):
                    fo.write(str(feature_map[i, j])+"\n")
            fo.close()

    # for each file in Negative training file, execute the functions above in
order.
    training_Neg = []
    for filename in os.listdir("./Image Data/Training images (Neg)"):
        img = plt.imread("./Image Data/Training images (Neg)" + "/" +
filename)
        img = gray_scale(img)

        grad_hori, grad_vert = grad_op(img)

        gradient, direction = generate_magnitude_direction(grad_hori,
grad_vert)

        orientation_gradient = OG(gradient, direction)
```

```
        feature_map = feature(orientation_gradient)

        training_Neg.append(feature_map)
        # for those whose HOG should be saved, execute this separately.
        if(filename[:-4] == '00000091a_cut'):
            fo = open('neg_{}_lines.txt'.format(filename[:-4]), "w")
            for i in range(feature_map.shape[0]):
                for j in range(feature_map.shape[1]):
                    fo.write(str(feature_map[i, j])+"\n")
            fo.close()

    return training_Pos, training_Neg


# # Training the Neural Network

# In[15]:


# first retrieve the training dataset with the function above
training_Pos, training_Neg = processing_data()

training_Pos = np.array(training_Pos) # shape = [m1 * 36 * n]

training_Neg = np.array(training_Neg) # shape = [m2 * 36 * n]

# then concatenate them, in order to sort more conveniently.
# remember the index between 0 to 9 is positive, index between 10 to 19 is
negative
training = np.concatenate((training_Pos, training_Neg), axis=0)

# class value has the shape of [10*20], 10 means 10 test imgs while 20 means
20 training imgs
class_value = []

# same order as above
# except for computing the distance (IOU) between test imgs and training imgs
for filename in os.listdir("./Image Data/Test images (Pos)"):
    single_test = []
    img = plt.imread("./Image Data/Test images (Pos)" + "/" + filename)
    img = gray_scale(img)

    grad_hori, grad_vert = grad_op(img)
    gradient, direction = generate_magnitude_direction(grad_hori, grad_vert)

    plt.imsave("test_gradient_{}.png".format(filename[:-4]),
               (gradient.astype(np.int16))/np.max(gradient.astype(np.int16))
*255, cmap = 'gray')
```

```
    orientation_gradient = OG(gradient, direction)

    feature_map = feature(orientation_gradient)

    for i in range(training.shape[0]):
        single_test.append(distance(feature_map, training[i]))

    class_value.append(single_test)

    if(filename[:-4] == 'crop001278a' or filename[:-4] == 'crop001500b'):
        fo = open('test_{}_lines.txt'.format(filename[:-4]), "w")

        for i in range(feature_map.shape[0]):
            for j in range(feature_map.shape[1]):
                fo.write(str(feature_map[i, j])+"\n")
        fo.close()



# In[16]:


for filename in os.listdir("./Image Data/Test images (Neg)"):

    single_test = []
    img = plt.imread("./Image Data/Test images (Neg)" + "/" + filename)
    img = gray_scale(img)

    grad_hori, grad_vert = grad_op(img)

    gradient, direction = generate_magnitude_direction(grad_hori, grad_vert)

    plt.imsave("test_gradient_{}.png".format(filename[:-4]),
               (gradient.astype(np.int16))/np.max(gradient.astype(np.int16))
*255, cmap = 'gray')

    orientation_gradient = OG(gradient, direction)

    feature_map = feature(orientation_gradient)

    for i in range(training.shape[0]):
        single_test.append(distance(feature_map, training[i]))

    class_value.append(single_test)

    if(filename[:-4] == '00000090a_cut'):
        fo = open('test_{}_lines.txt'.format(filename[:-4]), "w")

        for i in range(feature_map.shape[0]):
```

```
            for j in range(feature_map.shape[1]):
                fo.write(str(feature_map[i, j])+"\n")
        fo.close()



# In[17]:


# convert to ndarray for sorting
# 3-NN so find the largest 3 results, then print them
# remember the first 5 are positive test imgs, second 5 are negative test imgs
# and the value from 0 to 9 means positive sample, from 10 to 19 means
negative sample
for i in range(len(class_value)):
    print(class_value[i])

class_value = np.array(class_value)
class_result = []
for i in range(class_value.shape[0]):
    idx = np.argsort(class_value[i])[-3:]
    class_result.append(idx)

print(class_result)
```