CS GY 6643 Computer Vision (F21)
Project 2: Human Detection Using HOG Feature

Student 1:

Name: Savani Manoj Gokhale

NetID: sg6428

Student 2:

Name: Harsh Sanjay Apte

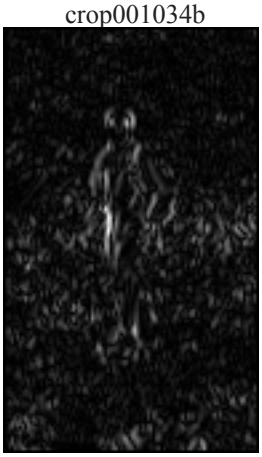
NetID: ha2179

Instructions to compile and run the program:

python HoG_Final.py --test_path=<path for test image file> --p_train=<path for +ve train image directory> --n_train= <path for -ve train image directory>

Normalized Gradient Magnitude Images for 10 Test Images:

crop001034b



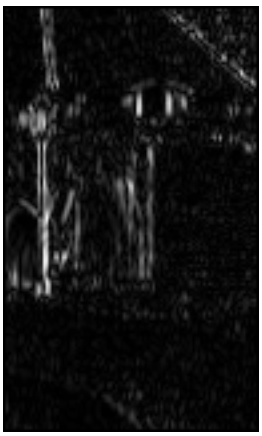
crop001070a

crop001278a

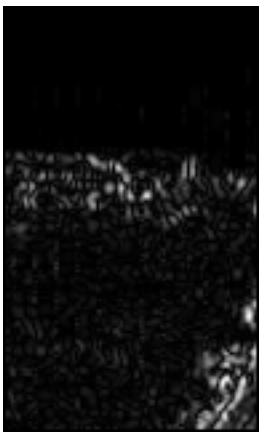
crop001500b

person_and_bike_151a

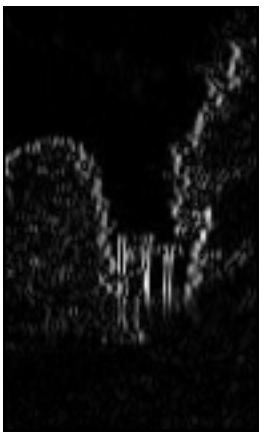
00000003a_cut



00000090a_cut

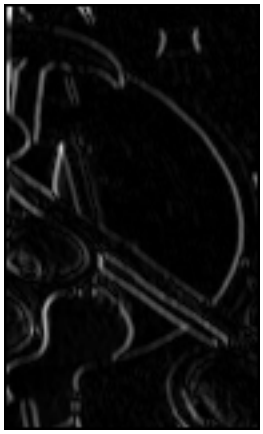


00000118a_cut



no_person_no_bike_258_cut

no_person_no_bike_264_cut



Source Code:

```
'''
CS GY 6643 Computer Vision Project II (F21) -- Human Detection Using HoG
Features
Student 1 - Harsh Sanjay Apte
NetID - ha2179

Student 2 - Savani Manoj Gokhale
NetID - sg6428

'''

from numpy.core import numeric
import cv2
import numpy as np
import math
from matplotlib import pyplot as plt
from PIL import Image
import numpy as np
import os
import argparse

## Converting image RGB to Grayscale
```

```
def RGB2Gray(image_path):
 input_image = cv2.imread(image_path)
 img_arr = cv2.cvtColor(input_image, cv2.COLOR_BGR2RGB)

 R_values, G_values, B_values = img_arr[:,:,0], img_arr[:,:,1],
img_arr[:,:,2]
 gray_img = 0.2999 * R_values + 0.5870 * G_values + 0.1140 * B_values

 # Rounding off the pixel values
 gray_rounded_img = np.around(gray_img, decimals=0)

 # returning the gray image array
 return gray_rounded_img


## Mask Convolution Function
def convolute_mask_prewitt(image_slice, mask):

   mask_rows, mask_cols = mask.shape
   out_arr = np.zeros((mask_rows, mask_cols))
   out_sum = 0

   for r in range(mask_rows):
       for c in range(mask_cols):
           out_arr[r][c] = image_slice[r][c]*mask[r][c]
           out_sum = out_sum + out_arr[r][c]

   return out_sum

## Function to calculate Gx, Gy, Gradient Magnitude, Gradient Angle
def prewitt_convultion(in_image):

   (n_,m_) = in_image.shape

   prewitt_op_gx = np.array([[-1,0,1],
                             [-1,0,1],
```

```
                            [-1,0,1]])

    prewitt_op_gy = np.array([[1,1,1],
                            [0,0,0],
                            [-1,-1,-1]])

    (p,q) = prewitt_op_gx.shape

    #formula to calculate matrix dims
    n = ((n_ - p) + 1)
    m = ((m_ - q) +1)

    img_gx = np.zeros((n,m))
    img_gy = np.zeros((n,m))

    for (i,x) in zip(range(n_), range(2,n_)):
        for (j,y) in zip(range(m_), range(2,m_)):
            res_pix_x = convolute_mask_prewitt(in_image[i:x+1, j:y+1],
prewitt_op_gx)
            img_gx[i][j] = res_pix_x

            res_pix_y = convolute_mask_prewitt(in_image[i:x+1, j:y+1],
prewitt_op_gy)
            img_gy[i][j] = res_pix_y

    img_gx = np.pad(img_gx, pad_width=1)
    img_gy = np.pad(img_gy, pad_width=1)

    #gradient calculation
    gradient_img_out = np.sqrt(np.square(img_gx), np.square(img_gy))

    #normalizing the gradient magnitude between 0 to 255 and Rounding off
the pixel values
    gr_normalized = np.interp(gradient_img_out, (gradient_img_out.min(),
gradient_img_out.max()), (0, 255))
    gr_rounded = np.rint(gr_normalized)
    img_gx_rounded = np.rint(img_gx)
```

```
   img_gy_rounded = np.rint(img_gy)

   #gradient angle calculation
   gradient_angle = np.arctan2(img_gy_rounded, img_gx_rounded)  * 180 /
np.pi

   rows,cols = gradient_angle.shape
   for i in range(rows):
     for j in range(cols):
       if gradient_angle[i][j] < 0:
         gradient_angle[i][j] = gradient_angle[i][j] + 360

   return img_gx_rounded, img_gy_rounded, gr_rounded, gradient_angle


## Function to Calculate Histogram Gradient
def histForCell(gr_im_8, gr_ang_im_8):

   rows , cols = gr_im_8.shape
   histogram = [0]*9
   bin_centers = [10, 30, 50, 70, 90, 110, 130, 150, 170]

   for row in range(rows):
     for col in range(cols):
       if gr_im_8[row][col] != 0 :

         # Check if the angle is between 0 to 180
         if gr_ang_im_8[row][col] > 180:
           gr_ang_im_8[row][col] = gr_ang_im_8[row][col] - 180

         # if any(lower_bin_value <= gr_ang_im_8[row][col] <=
upper_bin_value for (lower_bin_value, upper_bin_value) in bins):
         magnitude = gr_im_8[row][col]
         theta = gr_ang_im_8[row][col]

         for element in range(len(bin_centers)):
```

```
            # if theta is one of the bin center:
            if theta == bin_centers[element]:
              histogram[element] = histogram[element] + magnitude

            # Check if value is between 170 and 10
            last_element = len(bin_centers) - 1
            if element == last_element:
              if theta == (bin_centers[element] + 10): # i.e 180
                histogram[element] = histogram[element] + 1/2 * magnitude
                histogram[0] = histogram[0] + 1/2 * magnitude

              elif bin_centers[element]< theta:
                histogram[element] = histogram[element] + 3/4 * magnitude
                histogram[0] = histogram[0] + 1/4 * magnitude

            #check if value between 10 and last bin 170
            elif element == 0:
              if theta == 0:
                histogram[element] = histogram[element] + 1/2 * magnitude
                histogram[last_element] = histogram[last_element] + 1/2 *
magnitude

              if 0 < theta and theta < bin_centers[element]:
                histogram[element] = histogram[element] + 3/4 * magnitude
                histogram[last_element] = histogram[last_element] + 1/4 *
magnitude

            # check general condition between 10 to 30, 30 to 50, etc
            elif bin_centers[element] < theta and theta <
bin_centers[element+1] :
              dist_to_currentbin = abs(theta - bin_centers[element])
              dist_to_nextbin = abs(theta - bin_centers[element+1])

              if dist_to_currentbin == dist_to_nextbin:
                histogram[element] = histogram[element] + 1/2 * magnitude
                histogram[element+1] = histogram[element+1] + 1/2 *
magnitude
```

```
            elif dist_to_currentbin < dist_to_nextbin:
              histogram[element] = histogram[element] + 3/4 * magnitude
              histogram[element+1] = histogram[element+1] + 1/4 *
magnitude

            else:
              histogram[element] = histogram[element] + 1/4 * magnitude
              histogram[element+1] = histogram[element+1] + 3/4 *
magnitude

   return histogram


## Cell Selection Function from Image

def select_cell(gr_im_new_cell, gr_ang_im_new_cell):
 n,m = gr_im_new_cell.shape
  block_arr = np.empty((int(n/8),int(m/8)), dtype=object)

 for (i,x) in zip(range(0,n+1,8),range(8,n+1,8)):
   for(j,y) in zip(range(0,m+1,8),range(8,m+1,8)):
     cell_hist_arr = histForCell(gr_im_new_cell[i:x,j:y],
gr_ang_im_new_cell[i:x,j:y])
     block_arr[int(i/8)][int(j/8)] = cell_hist_arr
  return block_arr

## L2N Normalization Calculation
def calc_l2n(slice_arr):
 new_slice = slice_arr.flatten(order='F')
 descriptor = []
 n_concat =
np.concatenate((new_slice[0],new_slice[1],new_slice[2],new_slice[3]),
axis=None)
 L2_norm = math.sqrt(np.sum(n_concat ** 2))

 if L2_norm == 0:
```

```
   descriptor = n_concat
 else:
   descriptor = n_concat/L2_norm
 return descriptor



## Final descriptor Calculation for image (7524 Values)
def desc_formation(block_img):

 n_1,m_1 = block_img.shape
 v_num_list=[]
 final_desc = []
  for (i,x) in zip(range(n_1),range(1,n_1)):
   for(j,y) in zip(range(m_1),range(1,m_1)):
     desc = calc_l2n(block_img[i:x+1,j:y+1])
     final_desc.extend(desc)
 return final_desc



def driver_function(image_path):
 gray_image = RGB2Gray(image_path)
 GX, GY, G_Mag, G_Ang = prewitt_convultion(gray_image)
 blocked_image = select_cell(G_Mag, G_Ang)
 desc_image_array = desc_formation(blocked_image)
 return desc_image_array


def train_data_calc(pos_path, neg_path):
 pos_desc_master_array = []
 neg_desc_master_array = []

 f_names_p=[]
 f_names_n=[]

 for r,d,f in os.walk(pos_path, topdown=False):
```

```
  for name_p in f:
    final_pos_res = driver_function(os.path.join(r, name_p))
    pos_desc_master_array.append(final_pos_res)
    f_names_p.append(name_p)


for ro,di,fi in os.walk(neg_path, topdown=False):
  for name_n in fi:
    final_neg_res = driver_function(os.path.join(ro, name_n))
    neg_desc_master_array.append(final_neg_res)
    f_names_n.append(name_n)

return pos_desc_master_array, neg_desc_master_array, f_names_p, f_names_n

def NN_Classification(test_image_path, pos_path, neg_path):
test_desc_array = driver_function(test_image_path)
p_arr, n_arr, fp, fn = train_data_calc(pos_path, neg_path)

scores_dict = {}
scores_pos = []
scores_neg = []
numer_sum = 0
denom_sum = 0
n_sum = 0
d_sum = 0

for train_desc_p, pname in zip(p_arr, fp):
  for v1, v2 in zip(test_desc_array, train_desc_p):
    numer_sum = numer_sum + min(v1, v2)
    denom_sum = denom_sum + v2

  int_score_p = numer_sum/denom_sum
  scores_pos.append(int_score_p)
  scores_dict[int_score_p] = ["Human", pname]
 for train_desc_n, nname in zip(n_arr, fn):
  for v11, v22 in zip(test_desc_array, train_desc_n):
    n_sum = n_sum + min(v11, v22)
```

```
    d_sum = d_sum + v22

  int_score_n = n_sum/d_sum
  scores_neg.append(int_score_n)
  scores_dict[int_score_n] = ["No-Human", nname]

 return scores_pos, scores_neg, scores_dict

def result_function(test_image_path, pos_path, neg_path):
 p_score,n_score,score_d = NN_Classification(test_image_path, pos_path,
neg_path)
 sorted_list = p_score + n_score
 sorted_list.sort(reverse=True)

 p_num = 0
 n_num = 0

 file_n_list=[]
 for it in range(3):
   if (score_d[sorted_list[it]][0] == "Human"):
     p_num = p_num+1
     file_n_list.append(score_d[sorted_list[it]][1])
   else:
     n_num = n_num+1
     file_n_list.append(score_d[sorted_list[it]][1])

 print("3 Neighbour Filenmaes and their Histogram Intersction Scores -->")
 print(file_n_list)
 print[sorted_list[0:3]]

 if p_num>n_num:
   return "Human Detected in Image!!"
 else:
   return "Human Not Detected"

def main():
 parser = argparse.ArgumentParser(description="list of args")
```

```
 parser.add_argument("--test_path", type=str)
 parser.add_argument("--p_train", type=str)
 parser.add_argument("--n_train", type=str)
 args = parser.parse_args()

 img_path = args.test_path
 p_train = args.p_train
 n_train = args.n_train

 print("Result of given test image is ==>
"+str(result_function(str(img_path), str(p_train), str(n_train))))

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

Classification Results:

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	01-03e_cut.bmp 0.6455 No-Human	no_person__no_bike_219_cut.bmp 0.6446 No-Human	00000053a_cut.bmp 0.6388 No-Human	No-Human
crop001070a	Human	crop001045b.bmp 0.5062 Human	crop001672b.bmp 0.4934 Human	crop001028a.bmp 0.4876 Human	Human
crop001278a	Human	crop001275b.bmp 0.5563 Human	crop001008b.bmp 0.5561 Human	crop001030c.bmp 0.5527 Human	Human
crop001500b	Human	01-03e_cut.bmp 0.5405 No-Human	crop_000010b.bmp 0.5344 No-Human	no_person__no_bike_219_cut.bmp 0.5340 No-Human	No-Human
person_and_bike_151a	Human	crop001030c.bmp 0.4830 Human	crop001275b.bmp 0.4828 Human	person_and_bike_026a.bmp 0.4800 Human	Human
00000003a_cut	No-Human	no_person__no_bike_259_cut.bmp 0.5653 No-Human	00000053a_cut.bmp 0.5636 No-Human	00000062a_cut.bmp 0.5619 No-Human	No-Human
00000090a_cut	No-Human	no_person__no_bike_247_cut.bmp 0.4370 No-Human	no_person__no_bike_213_cut.bmp 0.4362 No-Human	00000093a_cut.bmp 0.4353 No-Human	No-Human
00000118a_cut	No-Human	no_person__no_bike_219_cut.bmp 0.5363 No-Human	00000053a_cut.bmp 0.5334 No-Human	no_person__no_bike_259_cut.bmp 0.5305 No-Human	No-Human
no_person_no_bike_258_cut	No-Human	crop001045b.bmp 0.4762 Human	crop001672b.bmp 0.4717 Human	crop_000010b.bmp 0.4468 Human	Human
no_person_no_bike_264_cut	No-Human	00000062a_cut.bmp 0.4138 No-Human	01-03e_cut.bmp 0.4132 No-Human	00000057a_cut.bmp 0.4131 No-Human	No-Human