Histogram of Gradients Report

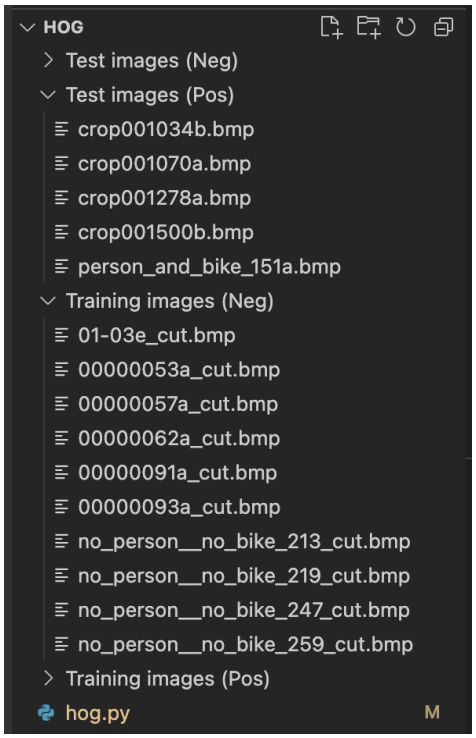
Submitted by:
Viha Gupta vg2237
Suyash Soniminde sys8910

Execution Instructions:

1. Ensure you have python version 3.8 installed
   python3 –version

```
viha@Vihas-MacBook-Pro hog % python3 --version
Python 3.8.6
```

2. Download the program file and all images (test + training) into a folder. In our example, the folder is called HOG and all images are downloaded here in the same folder structure as provided to us on Drive (please see below). Ensure that the structure is the same, otherwise the program will not work.

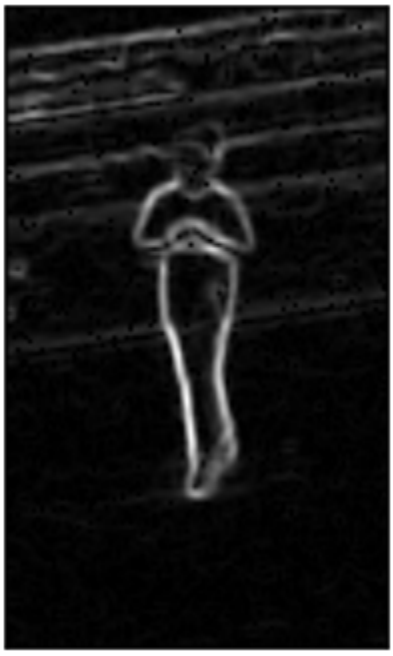


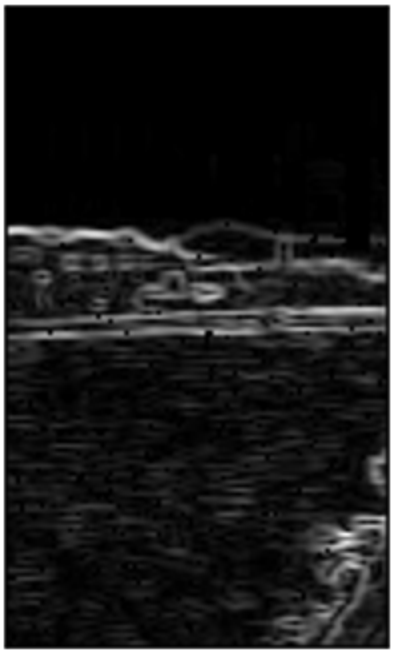
3. Enter the directory and run the following command to test on a test image:
   python3 hog.py './Test images (Neg)/00000003a_cut.bmp'

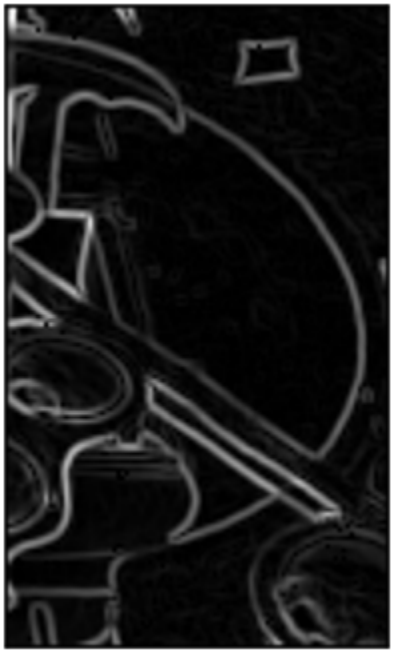
   To run for other images, just replace the parameter and pass the corresponding folder and file names. Ensure that the relative file path is enclosed in quotes.

Normalized gradient magnitude images for the 10 test images:

Original Test Image	Gradient Magnitude Images
	
	







Program Source Code:

```
# Steps
# 1. convert to grayscale
```

```
# 2. compute gradient magnitudes and gradient angle (Prewitts)
# 3. compute HOG features on training images
# 4. compute HOG features on test image
# 5. calculate distance between test and training images
# 6. use 3NN to classify

from PIL import Image
import numpy as np
import math
import sys

'''
Submitted by:
Viha Gupta vg2237
Suyash Soniminde sys8910
'''


### Convert to grayscale
def convert_to_grayscale(curr_training_image):

   # Convert to array
   color_img = np.asarray(Image.open(curr_training_image))

   # Use the linear approximation of gamma correction to convert to B/W
   bw_img = np.round_(0.299*color_img[:,:,0] + 0.587*color_img[:,:,1] +
0.114*color_img[:,:,2], decimals=0) #need to confirm this

   # Show B/W image
   # im_show = Image.fromarray(bw_img)
   # im_show.show()

   return bw_img

### Compute Gradients
def compute_gradients(bw_img):

   # compute input dimentions
   height, width = bw_img.shape

   # define Prewitt operator masks and buffer
   Gx = [
```

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

Gy = [
    [1, 1, 1],
    [0, 0, 0],
    [-1, -1, -1]
]
buffer = 1
prewitt_buffer = 1

# initialize output image to gaussian filtering
gradient_x_image = np.zeros((height,width))
gradient_y_image = np.zeros((height,width))
gradient_angle = np.zeros((height,width))
#gradient_angle.fill(1001) # we consider 1001 as undefined
gradient_magnitude = np.zeros((height,width))

# loop through smoothened image and find gradient magnitudes in x and y
for i in range(0+buffer,height-buffer,1):
    for j in range(0+buffer,width-buffer,1):
        gradient_x_image[i][j] = \
            Gx[0][0]*bw_img[i-prewitt_buffer][j-prewitt_buffer] \
            + Gx[0][1]*bw_img[i-prewitt_buffer][j-prewitt_buffer+1] \
            + Gx[0][2]*bw_img[i-prewitt_buffer][j-prewitt_buffer+2] \
            + Gx[1][0]*bw_img[i-prewitt_buffer+1][j-prewitt_buffer] \
            + Gx[1][1]*bw_img[i-prewitt_buffer+1][j-prewitt_buffer+1] \
            + Gx[1][2]*bw_img[i-prewitt_buffer+1][j-prewitt_buffer+2] \
            + Gx[2][0]*bw_img[i-prewitt_buffer+2][j-prewitt_buffer] \
            + Gx[2][1]*bw_img[i-prewitt_buffer+2][j-prewitt_buffer+1] \
            + Gx[2][2]*bw_img[i-prewitt_buffer+2][j-prewitt_buffer+2]

        gradient_y_image[i][j] = \
            Gy[0][0]*bw_img[i-prewitt_buffer][j-prewitt_buffer] \
            + Gy[0][1]*bw_img[i-prewitt_buffer][j-prewitt_buffer+1] \
            + Gy[0][2]*bw_img[i-prewitt_buffer][j-prewitt_buffer+2] \
            + Gy[1][0]*bw_img[i-prewitt_buffer+1][j-prewitt_buffer] \
            + Gy[1][1]*bw_img[i-prewitt_buffer+1][j-prewitt_buffer+1] \
            + Gy[1][2]*bw_img[i-prewitt_buffer+1][j-prewitt_buffer+2] \
            + Gy[2][0]*bw_img[i-prewitt_buffer+2][j-prewitt_buffer] \
```

```
            + Gy[2][1]*bw_img[i-prewitt_buffer+2][j-prewitt_buffer+1] \
            + Gy[2][2]*bw_img[i-prewitt_buffer+2][j-prewitt_buffer+2]


# calculate gradient magnitude and angle
for i in range(0+buffer,height-buffer,1):
    for j in range(0+buffer,width-buffer,1):
        # set undefinited to zero....// Gx = 0 aand Gy <> 0
        if gradient_x_image[i][j] == 0:
            gradient_angle[i][j] = 0
        # if both Gx and Gy are 0, assign 0 to gradient mag and angle
            #if gradient_y_image[i][j] == 0:
                #gradient_angle[i][j] = 0
                #gradient_magnitude[i][j] = 0
        else :
            gradient_angle[i][j] = \
            math.degrees(math.atan(gradient_y_image[i][j]/gradient_x_image[i][j]))

            gradient_magnitude[i][j] = \
            math.sqrt(gradient_x_image[i][j] * gradient_x_image[i][j] \
                + gradient_y_image[i][j] * gradient_y_image[i][j])


        # for negative angle, add 360
        if gradient_angle[i][j] <= 0 :
            gradient_angle[i][j] = gradient_angle[i][j] + 360

        # bring gradient angle value under 360
        if gradient_angle[i][j] > 360:
            gradient_angle[i][j] = gradient_angle[i][j]%360

        # if angle between 180 and 360, sub by 180. If angle = 360, set to zero
        if 180 <= gradient_angle[i][j] < 360:
            gradient_angle[i][j] = gradient_angle[i][j] - 180
        if gradient_angle[i][j] == 360:
            gradient_angle[i][j] = 0


# normalize and round off gradient magnitude to intergers in range [0-255]
nmz = np.max(gradient_magnitude)/255
gradient_magnitude = gradient_magnitude/nmz
```

```
    return gradient_magnitude, gradient_angle


### Compute HOG

def HOG(gradient_magnitude, gradient_angle):

    #Creating a feature vector which will be a 3d array
    feature_vector_3d = np.zeros((20,12,9))
    kk = np.zeros(9)

    # the final hog vector will have 19*11(block)*36(feature vector)=7524 values
    final_hog_vector = np.zeros(7524)
    block = np.zeros(36)

    # Calculating the feature vector values for (160/8=)20 * (96/8)12 = 240 cells
    for i in range(0,19,1):
        for j in range(0,11,1):
            for ii in range(0+8*i,7+8*i,1):
                for jj in range(0+8*j,7+8*j,1):
                    ga = gradient_angle[ii][jj]
                    gm = gradient_magnitude[ii][jj]

                    # Assigning appropriate magnitude to correct bins
                    if 10<=ga<30:
                        kk[0] += 0.05*(ga-10)*gm
                        kk[1] += 0.05*(30-ga)*gm
                    if 30<=ga<50:
                        kk[1] += 0.05*(ga-30)*gm
                        kk[2] += 0.05*(50-ga)*gm
                    if 50<=ga<70:
                        kk[2] += 0.05*(ga-50)*gm
                        kk[3] += 0.05*(70-ga)*gm
                    if 70<=ga<90:
                        kk[3] += 0.05*(ga-70)*gm
                        kk[4] += 0.05*(90-ga)*gm
                    if 90<=ga<110:
                        kk[4] += 0.05*(ga-90)*gm
                        kk[5] += 0.05*(110-ga)*gm
                    if 110<=ga<130:
                        kk[5] += 0.05*(ga-110)*gm
                        kk[6] += 0.05*(130-ga)*gm
```

```
                if 130<=ga<150:
                    kk[6] += 0.05*(ga-130)*gm
                    kk[7] += 0.05*(150-ga)*gm
                if 150<=ga<170:
                    kk[7] += 0.05*(ga-150)*gm
                    kk[8] += 0.05*(170-ga)*gm
                if 170<=ga<180 or 0<=ga<10:
                    kk[8] += gm/2
                    kk[0] += gm/2

        for k in range(0,8,1):
            feature_vector_3d[i][j][k] = kk[k]
            kk[k]=0

count = 0

# Calculating the final hog feature vector.
# There are 19*11 blocks due to one cell overlap
for i in range(0,18,1):
    for j in range(0,10,1):
        square_sum = 0
        for k in range(0,8,1):
            block[k] = feature_vector_3d[i][j][k]
            square_sum += block[k]*block[k]
        for k in range(9,17,1):
            block[k] = feature_vector_3d[i][j+1][k-9]
            square_sum += block[k]*block[k]
        for k in range(18,26,1):
            block[k] = feature_vector_3d[i+1][j][k-18]
            square_sum += block[k]*block[k]
        for k in range(27,35,1):
            block[k] = feature_vector_3d[i+1][j+1][k-27]
            square_sum += block[k]*block[k]

        count += 36
        block_norm = math.sqrt(square_sum)

        if block_norm != 0:
            # Performing L2 normalization
            block = block/block_norm

        c = count-36
```

```
            for k in range(c,c+35,1):
                final_hog_vector[k] = block[k%36]

    return final_hog_vector



### Calculate similarity

def calculate_similarity(training_HOG, test_HOG):
    height, _ = training_HOG.shape
    similarity = np.zeros((1,height))

    # iterating through each training image index i
    for i in range(height):
        similarity[0,i] = (np.sum(np.minimum(training_HOG[i,:],test_HOG)))\
            /(np.sum(training_HOG[i,:]))

    return similarity


### Determine 3NN
def threeNN(similarity, n_neg_train):
    neg = 0
    pos = 0
    NN_count = 1

    # pick three largest
    indices = np.argsort(similarity)[0][-3:]

    for i in indices:
        if i < n_neg_train:
            neg += 1
            print("\nNN #%d: %s, %f, Not-human" % (NN_count, neg_train_files[i],
similarity[0][i]))
        else:
            pos +=1
            print("\nNN #%d: %s, %f, Human" % (NN_count,
pos_train_files[i%n_neg_train], similarity[0][i]))
        NN_count += 1

    if neg > pos:
```

```
        classification = 'Not human'
    else:
        classification = 'Human'

    return classification


# main
# ensure proper arguments given i.e.
# python3 hog.py './Test images (Neg)/00000003a_cut.bmp'
# python3 hog.py './Test images (Neg)/00000090a_cut.bmp'
# python3 hog.py './Test images (Neg)/00000118a_cut.bmp'
# python3 hog.py './Test images (Neg)/no_person__no_bike_258_Cut.bmp'
# python3 hog.py './Test images (Neg)/no_person__no_bike_264_cut.bmp'
# python3 hog.py './Test images (Pos)/crop001034b.bmp'
# python3 hog.py './Test images (Pos)/crop001070a.bmp'
# python3 hog.py './Test images (Pos)/crop001278a.bmp'
# python3 hog.py './Test images (Pos)/crop001500b.bmp'
# python3 hog.py './Test images (Pos)/person_and_bike_151a.bmp'
if (len(sys.argv)) < 2:
    print("Command failure. Please pass image path+name as parameter in single
quotesand try again.\
        \nExample: $ python3 hog.py './Test images (Neg)/00000003a_cut.bmp'")
    exit()

# compute HOG on all training images first

# Collect training files
neg_train_files = ['01-03e_cut.bmp', '00000053a_cut.bmp', '00000057a_cut.bmp',
'00000062a_cut.bmp', '00000091a_cut.bmp', '00000093a_cut.bmp',\
    'no_person__no_bike_213_cut.bmp', 'no_person__no_bike_219_cut.bmp',
'no_person__no_bike_247_cut.bmp', 'no_person__no_bike_259_cut.bmp']
pos_train_files = ['crop_000010b.bmp', 'crop001008b.bmp', 'crop001028a.bmp',
'crop001030c.bmp', 'crop001045b.bmp', 'crop001047b.bmp',\
    'crop001063b.bmp', 'crop001275b.bmp', 'crop001672b.bmp',
'person_and_bike_026a.bmp']

neg_train_loc = './Training images (Neg)/'
pos_train_loc = './Training images (Pos)/'

n_pos_train = len(pos_train_files)
n_neg_train = len(neg_train_files)
```

```
n_training = n_pos_train + n_neg_train


training_HOG = np.zeros((n_training,7524))

# For all negative training images,
for i, _ in enumerate(neg_train_files):
    # Parse file location for this image
    curr_training_image = neg_train_loc + neg_train_files[i]

    # Show this image
    # im_show = Image.open(curr_training_image)
    # im_show.show()

    # Convert this image to grayscale
    bw_img = convert_to_grayscale(curr_training_image)

    # Compute gradients for this image
    gradient_magnitude, gradient_angle = compute_gradients(bw_img)

    # Compute HOG for this image
    training_HOG[i] = HOG(gradient_magnitude, gradient_angle)

print ("Grayscale, Gradients and HOG for Neg training images complete")

# Now we repeat for all positive training images,
for i, _ in enumerate(pos_train_files):
    # Parse file location for this image
    curr_training_image = pos_train_loc + pos_train_files[i]

    # Show this image
    # im_show = Image.open(curr_training_image)
    # im_show.show()

    # Convert this image to grayscale
    bw_img = convert_to_grayscale(curr_training_image)

    # Compute gradients for this image
    gradient_magnitude, gradient_angle = compute_gradients(bw_img)

    # Compute HOG for this image
    training_HOG[i+n_neg_train] = HOG(gradient_magnitude, gradient_angle)
```

```
print ("Grayscale, Gradients and HOG for Pos training images complete")

# compute HOG on given test image

# show input image from parameter passed
test_ip_image = sys.argv[1]
im_show = Image.open(test_ip_image)
im_show.show()

# Convert this image to grayscale
bw_img = convert_to_grayscale(test_ip_image)

# Compute gradients for this image
gradient_magnitude, gradient_angle = compute_gradients(bw_img)

# Compute HOG for this image
test_HOG = HOG(gradient_magnitude, gradient_angle)

print ("Grayscale, Gradients and HOG for test image complete")

# Calculate distance
similarity = calculate_similarity(training_HOG, test_HOG)

print ("Distance calculation complete")

# Classify with 3NN
classification = threeNN(similarity, n_neg_train)

print("\nThe test image is classified as: ", classification)
```

Classification Results:

Test image	**Correct Classification**	**File name of 1st NN, similarity & classification**	**File name of 2nd NN, similarity & classification**	**File name of 3rd NN, similarity & classification**	**Classification from 3-NN**
crop001034b	Human	crop001030c.bmp, 0.650696,	crop001030c.bmp, 0.650696,	crop_000010b.bmp, 0.649106,	Human

		Human	Human	Human	
crop001070a	Human	crop001063b.bmp, 0.504023, Human	crop001672b.bmp, 0.493616, Human	crop001030c.bmp, 0.492450, Human	Human
crop001278a	Human	crop001008b.bmp, 0.589690, Human	crop001275b.bmp, 0.578801, Human	crop001672b.bmp, 0.572430, Human	Human
crop001500b	Human	crop001672b.bmp, 0.553168, Human	no_person__no_bike_247_cut.bmp, 0.531547, Not-human	00000091a_cut.bmp, 0.530658, Not-human	Not-Human
person_and_bike_151a	Human	crop001030c.bmp, 0.510091, Human	crop001275b.bmp, 0.499269, Human	person_and_bike_026a.bmp, 0.498567, Human	Human
00000003a_cut	No-human	00000053a_cut.bmp, 0.595358, Not-human	crop001672b.bmp, 0.586985, Human	no_person__no_bike_259_cut.bmp, 0.566495, Not-human	Not-Human
00000090a_cut	No-human	00000057a_cut.bmp, 0.491408, Not-human	00000093a_cut.bmp, 0.461087, Not-human	crop001672b.bmp, 0.433540, Human	Not-Human
00000118a_cut	No-human	crop001063b.bmp, 0.530258, Human	crop001030c.bmp, 0.526114, Human	00000053a_cut.bmp, 0.525186, Not-human	Human

no_person_no_bike_258_cut	No-human	00000057a_cut.bmp, 0.487027, Not-human	crop001063b.bmp, 0.482369, Human	crop001275b.bmp, 0.481519, Human	Human
no_person_no_bike_264_cut	No-human	crop001030c.bmp, 0.458645, Human	crop001672b.bmp, 0.453166, Human	01-03e_cut.bmp, 0.451022, Not-human	Human