Human Detection using HOG Feature

Eashan Kaushik – ek3575, N19320245

Srijan Malhotra – sm9439, N18390405

Instruction to Run Code

Step-1: Download files and put it in one directory. Files to be downloaded are:

- convolution.py
- main.py
- HOG.py

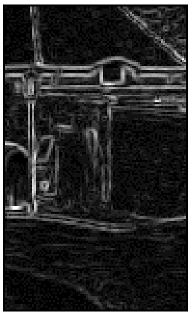
Step-2: Download the training the testing images and make the following structure:

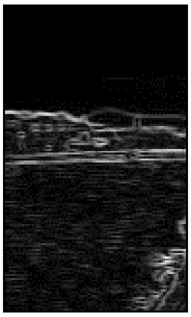
```
|-train
|-------positive
|-------negative
|-test
|-------positive
|-------negative
|-Output
|-Magnitude
```

Save the positive and negative train and test images in the folder given above. Output and Magnitude folders are left empty.

Step-3: Go to your terminal and go in the directory, run the command python main.py.

Normalized Gradient Magnitude of 10 Test Images









Source Code

- **model-test.py**

```
from convolution import SeConvolve
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt
from PIL import Image
import math
import matplotlib.image as mpimg
import sys

class HOGDescriptor:
    def __init__(self):

        # Gradient x and y operations Prewitt's Operators
        self._convolution_matrix_gx = np.array([[-1, 0, 1], [-1, 0, 1], [-
1, 0, 1]])
        self._convolution_matrix_gy = np.array([[1, 1, 1], [0, 0, 0], [-1,
-1, -1]])

        #### Train ####
        self.training_discript = dict()
        self.train_images = list()
        self.number_of_train_positive = 0
        self.image_names_train = list()
        self.gray_train_images = list()

        # Output of step 2
        self._gradient_x_train = list()
        self._gradient_y_train = list()

        # Output of step 3
        self._magnitude_train = list()

        # Angle Output
        self._angle_train = list()

        #### Test ####
        self.test_discript = dict()
        self.test_images = list()
        self.number_of_test_positive = 0
        self.image_names_test = list()
        self.gray_test_images = list()

        # Output of step 2
        self._gradient_x_test = list()
```

```python
        self._gradient_y_test = list()

        # Output of step 3
        self._magnitude_test = list()

        # Angle Output
        self._angle_test = list()

    # Function to fit the training data
    def fit(self):

        # read train positive images
        my_list = os.listdir('train/positive')
        self.image_names_train.extend(my_list)

        for current in my_list:
            image = cv2.imread("train/positive/" + current)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            self.train_images.append(image)

        self.number_of_train_positive = len(self.train_images)

        # read train negative images
        my_list = os.listdir('train/negative')
        self.image_names_train.extend(my_list)

        for current in my_list:
            image = cv2.imread("train/negative/" + current)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            self.train_images.append(image)

        # convert into numpy array
        self.train_images = np.array(self.train_images)
        # convert into grascale image
        self.to_grayscale(self.train_images, train=True)
        # gradient operation to calculate magnitude and angle
        self.gradient_operation(self.gray_train_images, train=True)
        # calculate the hog descriptor
        self.hog_feature(self._magnitude_train, self._angle_train,
train=True)


    def test(self):
        # read test positve images
        my_list = os.listdir('test/positive')
        self.image_names_test.extend(my_list)

        for current in my_list:
```

```
            image = cv2.imread("test/positive/" + current)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            self.test_images.append(image)

        self.number_of_test_positive = len(self.test_images)

        # read test negative images
        my_list = os.listdir('test/negative')
        self.image_names_test.extend(my_list)

        for current in my_list:
            image = cv2.imread("test/negative/" + current)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            self.test_images.append(image)

        # convert into numpy array
        self.test_images = np.array(self.test_images)
        # convert into grascale image
        self.to_grayscale(self.test_images, train=False)
        # gradient operation to calculate magnitude and angle
        self.gradient_operation(self.gray_test_images, train=False)
        # calculate the hog descriptor
        self.hog_feature(self._magnitude_test, self._angle_test,
train=False)

    # function to evaluate on training images
    def evaluate(self, train=False):

        output_class = dict()
        output_dist = dict()
        output_file = dict()

        # if train is False we evaluate on test descriptor
        if not train:
            discriptor = self.test_discript
        else:
        # if train is True we evaluate on train descriptor
            discriptor = self.training_discript

        for classes, disc in discriptor.items():

            distance = np.empty(0)
            class_distance = np.empty(0)

            disc = np.array(disc)

            for classes_tr, disc_tr in self.training_discript.items():
```

```
                disc_tr = np.array(disc_tr)

                # calculating the similarity index
                denominator = np.sum(disc_tr)
                numerator = np.minimum(disc_tr,disc)
                numerator = np.sum(numerator)

                # store the distance and the class the distance belongs to
                distance = np.append(distance, numerator/denominator)
                class_distance = np.append(class_distance, classes_tr)

            c = {A: B for A, B in zip(class_distance, distance)}
            c = dict(sorted(c.items(), key=lambda item: item[1],
reverse=True))

            i = 0
            temp1 = list()
            temp2 = list()
            temp3 = list()
            while i < 3:

                cl = list(c.keys())[i]

                f = cl.split('-')[0]
                temp3.append(f)

                cl = cl.split('-')[-1]
                temp1.append(cl)

                di = list(c.values())[i]
                temp2.append(di)

                i += 1
            # output for evaluate, gives distance, classification and file
name for 3NN
            output_class[classes] = temp1
            output_dist[classes] = temp2
            output_file[classes] = temp3

        return output_class, output_dist, output_file

    # calculates the HOG Feature
    def hog_feature(self, magn, angle, train=True):

        for index, image in enumerate(magn):
            row = 0

            image_discriptor = np.empty(0)
```

```
            while row <= 160:
                # block size 16 x 16 pixels
                row_start = row
                row_end = row + 16

                if not row_end <= 160:
                    break

                col = 0
                while col <= 96:
                    count_col = 0
                    column_start = col
                    column_end = col + 16

                    if column_end <= 96:
                        # gets the magnitude and angle of current block
                        current_block_magnitude =
magn[index][row_start:row_end, column_start:column_end]
                        current_block_angle =
angle[index][row_start:row_end, column_start:column_end]

                        # calculate normalised descriptor of the block
                        hist = self.HOG_cell(current_block_magnitude,
current_block_angle)

                        # concatenate to find global descriptor
                        image_descriptor =
np.concatenate((image_descriptor, hist), axis=0)
                    else:
                        break

                    col += 8

                row += 8

            # save for train or test images
            if train:
                if index < self.number_of_train_positive:
                    self.training_discript[self.image_names_train[index] +
'-positive'] = image_descriptor
                else:
                    self.training_discript[self.image_names_train[index] +
'-negative'] = image_descriptor
            else:
                if index < self.number_of_test_positive:
                    self.test_discript[self.image_names_test[index] + '-
positive'] = image_descriptor
```

```
                else:
                    self.test_discript[self.image_names_test[index] + '-
negative'] = image_descriptor

    # calculate HOG Descriptor for the block
    def HOG_cell(self, current_block_magnitude, current_block_angle):

        # calculate HOG Descriptor for first HOG Cell
        hist1 = self.HOG_cell_histogram(current_block_magnitude[0:8, 0:8],
current_block_angle[0:8, 0:8])

        # calculate HOG Descriptor for second HOG Cell
        hist2 = self.HOG_cell_histogram(current_block_magnitude[0:8, 8:16],
current_block_angle[0:8, 8:16])

        # calculate HOG Descriptor for third HOG Cell
        hist3 = self.HOG_cell_histogram(current_block_magnitude[8:16, 0:8],
current_block_angle[8:16, 0:8])

        # calculate HOG Descriptor for forth HOG Cell
        hist4 = self.HOG_cell_histogram(current_block_magnitude[8:16,
8:16], current_block_angle[8:16, 8:16])

        # concatenate to get HOG Descriptor for block
        hist = np.concatenate((hist1, hist2, hist3, hist4), axis=0)

        # L2 Norm
        norm = math.sqrt(np.sum(np.power(hist, 2)))

        if norm > 0:
            # Divide by L2 Norm
            hist = hist / norm

        return hist

    # Calculate HOG Histogram for one 8x8 cell
    def HOG_cell_histogram(self, current_block_magnitude,
current_block_angle):

        # bins 1-9
        bins = [0] * 9

        i = 0

        while i <= current_block_magnitude.shape[0] - 1:

            j = 0
```

```
            while j <= current_block_magnitude.shape[1] - 1:

                angle = current_block_angle[i, j]

                # if angle is greater than 180 we subtract 180
                if angle >= 180:
                    angle = angle - 180

                # if angle is less than 0 we add 180
                if angle < 0:
                    angle = angle + 180

                # we split the block magnitude between two closest bins
according to gradient angle
                if angle > 10 and angle <= 30:
                    bins[0] += current_block_magnitude[i, j] * ((30 -
angle) / 20)
                    bins[1] += current_block_magnitude[i, j] * ((angle -
10) / 20)
                elif angle > 30 and angle <= 50:
                    bins[1] += current_block_magnitude[i, j] * ((50 -
angle) / 20)
                    bins[2] += current_block_magnitude[i, j] * ((angle -
30) / 20)
                elif angle > 50 and angle <= 70:
                    bins[2] += current_block_magnitude[i, j] * ((70 -
angle) / 20)
                    bins[3] += current_block_magnitude[i, j] * ((angle -
50) / 20)
                elif angle > 70 and angle <= 90:
                    bins[3] += current_block_magnitude[i, j] * ((90 -
angle) / 20)
                    bins[4] += current_block_magnitude[i, j] * ((angle -
70) / 20)
                elif angle > 90 and angle <= 110:
                    bins[4] += current_block_magnitude[i, j] * ((110 -
angle) / 20)
                    bins[5] += current_block_magnitude[i, j] * ((angle -
90) / 20)
                elif angle > 110 and angle <= 130:
                    bins[5] += current_block_magnitude[i, j] * ((130 -
angle) / 20)
                    bins[6] += current_block_magnitude[i, j] * ((angle -
110) / 20)
                elif angle > 130 and angle <= 150:
                    bins[6] += current_block_magnitude[i, j] * ((150 -
angle) / 20)
```

```
                    bins[7] += current_block_magnitude[i, j] * ((angle -
130) / 20)
                elif angle > 150 and angle <= 170:
                    bins[7] += current_block_magnitude[i, j] * ((170 -
angle) / 20)
                    bins[8] += current_block_magnitude[i, j] * ((angle -
150) / 20)
                elif angle >= 0 and angle <= 10:
                    bins[0] += current_block_magnitude[i, j] * ((angle +
10) / 20)
                    bins[8] += current_block_magnitude[i, j] * ((10 -
angle) / 20)
                elif angle > 170 and angle <= 180:
                    bins[8] += current_block_magnitude[i, j] * ((190 -
angle) / 20)
                    bins[0] += current_block_magnitude[i, j] * ((angle -
170) / 20)
                else:
                    print('some error')

                j += 1

            i += 1

        return np.array(bins)

    # convert to grayscale
    def to_grayscale(self, images, train=True):

        # for train
        if train:
            for image in images:
                # extract R G B channels
                R, G, B = image[:,:,0], image[:,:,1], image[:,:,2]
                # weighted formulae to convert to grayscale
                new_image = np.round((0.299 * R) + (0.587 * G) + (0.114 *
B))
                self.gray_train_images.append(new_image)

            self.gray_train_images = np.array(self.gray_train_images)
        else:
        # for test
            for image in images:
                # extract R G B channels
                R, G, B = image[:,:,0], image[:,:,1], image[:,:,2]
                # weighted formulae to convert to grayscale
                new_image = np.round((0.299 * R) + (0.587 * G) + (0.114 *
B))
```

```
                self.gray_test_images.append(new_image)

            self.gray_test_images = np.array(self.gray_test_images)

    # gradient operation
    def gradient_operation(self, images, train=True):

        # if train
        if train:
            for image in images:

                # Convolution done on the image_matrix w.r.t gradient x
                gradient_x = SeConvolve(image, self._convolution_matrix_gx,
mode='gradient')
                gradient_x_train, _ = gradient_x.convolution()

                self._gradient_x_train.append(gradient_x_train)

                # Convolution done on the image_matrix w.r.t gradient y
                gradient_y = SeConvolve(image, self._convolution_matrix_gy,
mode='gradient')
                gradient_y_train, _ = gradient_y.convolution()

                self._gradient_y_train.append(gradient_y_train)

                # We compute gradient magnitude, gradient angle
                magnitude_x_train =
self.calcuate_magnitude(gradient_x_train, gradient_y_train)
                angle_x_train = self.calculate_angle(gradient_x_train,
gradient_y_train)

                self._magnitude_train.append(magnitude_x_train)
                self._angle_train.append(angle_x_train)

            # convert to numpy array
            self._gradient_x_train = np.array(self._gradient_x_train)
            self._gradient_y_train = np.array(self._gradient_y_train)
            self._magnitude_train = np.array(self._magnitude_train)
            self._angle_train = np.array(self._angle_train)

        else:
            for image in images:

                # Convolution done on the image_matrix w.r.t gradient x
                gradient_x = SeConvolve(image, self._convolution_matrix_gx,
mode='gradient')
                gradient_x_train, _ = gradient_x.convolution()
```

```
                self._gradient_x_test.append(gradient_x_train)

                # Convolution done on the image_matrix w.r.t gradient y
                gradient_y = SeConvolve(image, self._convolution_matrix_gy,
mode='gradient')
                gradient_y_train, _ = gradient_y.convolution()

                self._gradient_y_test.append(gradient_y_train)

                # We compute gradient magnitude, gradient angle and edge
angle
                magnitude_x_train =
self.calcuate_magnitude(gradient_x_train, gradient_y_train)
                angle_x_train = self.calculate_angle(gradient_x_train,
gradient_y_train)

                self._magnitude_test.append(magnitude_x_train)
                self._angle_test.append(angle_x_train)

            # convert to numpy array
            self._gradient_x_test = np.array(self._gradient_x_test)
            self._gradient_y_test = np.array(self._gradient_y_test)
            self._magnitude_test = np.array(self._magnitude_test)
            self._angle_test = np.array(self._angle_test)

    def calcuate_magnitude(self, gradient_x, gradient_y):
        height, width = gradient_x.shape

        # After gaussing smoothing and gradient computation we have lost a
total of 8 rows and 8 columns
        magnitude = np.zeros((height - 2, width - 2))

        # looping over the desired matrix
        for i in range(1,height - 1):
            for j in range(1,width - 1):
                # gradient calculated using root(gx**2 + gy**2)
                temp = (gradient_x[i, j] ** 2) + (gradient_y[i, j] ** 2)

                magnitude[i - 1, j - 1] = math.sqrt(temp)

        # same size as original image
        magnitude = np.pad(magnitude, 1, mode='constant')

        # normalize the magnitude between [0, 255]
        magnitude = ((magnitude - magnitude.min()) * (1/(magnitude.max() -
magnitude.min()) * 255)).astype(int)

        return magnitude
```

```
    def calculate_angle(self, gradient_x, gradient_y):

        height, width = gradient_x.shape

        # After gaussing smoothing and gradient computation we have lost a
total of 8 rows and 8 columns
        angle = np.zeros((height - 2, width - 2))
        edge_angle = np.zeros((height - 2, width - 2))

        # looping over the desired matrix
        for i in range(1,height - 1):
            for j in range(1,width - 1):
                if gradient_x[i, j]  != 0:
                    # returns angle between -180 to 180
                    angle[i - 1, j - 1] =
math.degrees(math.atan2(gradient_y[i, j], gradient_x[i, j]))

        # same size as original image
        angle = np.pad(angle, 1, mode='constant')

        return angle
```

- **main.py**

```
from HOG import HOGDescriptor
import matplotlib.pyplot as plt

# main block
if __name__ == '__main__':

    # call HOG class
    hg = HOGDescriptor()
    # Fit on training data
    hg.fit()
    # Transform test data
    hg.test()

    # evaluate the data on test images
    print(f'For Test Images{hg.evaluate()}')

    # evaluate the data on train images
    print(f'\n\n\nFor Train Images{hg.evaluate(train=True)}')

    # HOG ASCII Derscriptors
    for name in ['crop001028a.bmp-positive', 'crop001030c.bmp-positive',
'00000091a_cut.bmp-negative']:
```

```
        with open('Output/' + name.split('-')[0] + '.txt',
'w',encoding='utf-8') as file:
            file.writelines( "%s\n" % item for item in
hg.training_discript[name])

    for name in ['crop001278a.bmp-positive', 'crop001500b.bmp-positive',
'00000090a_cut.bmp-negative']:

        with open('Output/' + name.split('-')[0] + '.txt',
'w',encoding='utf-8') as file:
            file.writelines( "%s\n" % item for item in
hg.test_discript[name] )

    # Gradient Magnitude of Test Images
    for i in range(0, hg._magnitude_test.shape[0]):
        plt.imsave('Magnitude/' + hg.image_names_test[i],
hg._magnitude_test[i], cmap='gray')
```

- **convolution.py**

```
####################################################
## Developed by: Eashan Kaushik & Srijan Malhotra ##
## Project Start: 29th October 2021               ##
####################################################

import numpy as np

class SeConvolve:

    def __init__(self, image_matrix, kernel, mode='gradient'):
        # input image matrix
        self.image_matrix = image_matrix
        # inpur kernel
        self.kernel = kernel
        # output after convolution with the kerne
        self._output = None
        # normalized output
        self._output_norm = None
        # mode smoothing or gradient
        self.mode = mode

    #######################
    ## Getter and Setter ##
    #######################

    @property
    def output(self):
        return self._output
```

```
    @property
    def output_norm(self):
        return self._output_norm

    ######################
    ######################

    def convolution(self):

      height, width = self.image_matrix.shape

      if self.mode == 'gradient':

        # code to find the gradients
        self._output = np.zeros((height - 2, width - 2))

        # looping over the desired matrix
        for i in range(1,height - 1):
          for j in range(1,width - 1):
            # martix multiplication for gradient computation
            # prewit convolution after gaussian smoothing leads to loss of
4 rows and 4 columns thats why we start from i-4 and j-4
            self._output[i - 1,j - 1] = (self.kernel[0, 0] *
self.image_matrix[i - 1, j - 1]) + (self.kernel[0, 1] * self.image_matrix[i
- 1, j]) + (self.kernel[0, 2] * self.image_matrix[i - 1, j + 1]) + \
                      (self.kernel[1, 0] * self.image_matrix[i, j - 1]) +
(self.kernel[1, 1] * self.image_matrix[i, j]) + (self.kernel[1, 2] *
self.image_matrix[i, j + 1]) + (self.kernel[2, 0] * self.image_matrix[i +
1, j - 1]) + \
                      (self.kernel[2, 1] * self.image_matrix[i + 1, j]) +
(self.kernel[2, 2] * self.image_matrix[i + 1, j + 1])

      # we call the normalize function to normalize the output
      self.normalize()

      self._output = np.pad(self._output, 1, mode='constant')

      return self._output, self._output_norm

    # normalize
    def normalize(self):
      if self.mode == 'gradient':
        # normalize using sum of absolute values
        temp_output = np.absolute(self._output)
        self._output_norm = temp_output / 3
        self._output_norm = np.pad(self._output_norm, 1, mode='constant')
```

Classification 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	crop001672b.bmp	0.66936	Positive	00000053a_cut.bmp	0.649304	Negative	crop001275b.bmp	0.64487	Positive	Positive
crop001070a	Human	crop001063b.bmp	0.52307	Positive	crop001045b.bmp	0.505678	Positive	crop001672b.bmp	0.505356	Positive	Positive
crop001278a	Human	crop001672b.bmp	0.59372	Positive	crop001008b.bmp	0.586661	Positive	crop001275b.bmp	0.583621	Positive	Positive
crop001500b	Human	crop001672b.bmp	0.55854	Positive	no_person__no_bike_	0.551751	Negative	crop001275b.bmp	0.537701	Positive	Positive
person_and_bike_151a	Human	crop001030c.bmp	0.50336	Positive	person_and_bike_026	0.502417	Positive	crop001008b.bmp	0.491942	Positive	Positive
00000003a_cut	No-human	00000053a_cut.bmp	0.57492	Negative	crop001672b.bmp	0.574362	Positive	no_person__no_bike	0.548185	Negative	Negative
00000090a_cut	No-human	00000093a_cut.bmp	0.49573	Negative	00000057a_cut.bmp	0.488355	Negative	crop001672b.bmp	0.45072	Positive	Negative
00000118a_cut No-human	No-human	00000093a_cut.bmp	0.55551	Negative	00000053a_cut.bmp	0.545432	Negative	00000091a_cut.bmp	0.540538	Negative	Negative
no_person_no_bike_258_cut	No-human	00000057a_cut.bmp	0.49219	Negative	person_and_bike_026	0.4896	Positive	crop001672b.bmp	0.485973	Positive	Positive
no_person_no_bike_264_cut	No-human	00000053a_cut.bmp	0.43727	Negative	crop001672b.bmp	0.433996	Positive	crop001030c.bmp	0.429987	Positive	Positive