Human Detection using HOG Feature

PROJECT - 2

Team:

- Atharva Bhagwat [acb9244]
- Shubham Gundawar [ssg9763]

Steps for Human Detection using HOG Feature

1) Read image. Convert to grayscale using: Gray = round(0.299*Red + 0.587*Green + 0.114*Blue)
2) Gradient calculation using prewitt's operator, magnitude calculation (sqrt(Gx^2+Gy^2)) and normalization. Gradient angle calculation.
3) Calculate histogram bins for every cell (unsigned format/9 bins).
4) Calculate normalized bins for every block (l2 norm).
5) Flatten and concatenate normalized bins for every block to get a descriptor of length 7524.
6) 3NN implementations using similarity formula: sum(min(input, train))/sum(train)

Notes:
- If the gradient angle is less than 0, add 360. If the gradient angle is greater than 180, subtract 180 from it.
- The larger the similarity, the smaller the distance between the input image and the training image.

Installing packages: opencv-python, numpy, more-itertools

pip3 install -r requirements.txt

Usage

Make sure that ‘image_data’ and ‘main.py’ are present in the same directory.

Structure of image_data:

```
image_data—|
           |- test_images_neg
           |- test_images_pos
           |- training_images_neg
           |- training_images_pos
```

Program is automated to parse through all these folders based on certain substrings (test, training, neg, pos). Folder names from the google drive link for the dataset will work.

python3 main.py <path_to_image_data_folder>
eg: python3 main.py image_data

Result

After running the *'python3 main.py'* command, results will be printed on the terminal. This command will execute descriptor creation for all training and testing images. Then it will run 3-NN on all testing images and print the results. Folders 'descriptors' and 'test_images_gradient_magnitude' will also be created with relevant result information.

Source Code

```
"""
Steps:
1) Read image. Convert to grayscale using: Gray = round(0.299*Red + 0.587*Green +
0.114*Blue)
2) Gradient calculation using prewitt's operator, magnitude calculation
(sqrt(Gx^2+Gy^2)) and normalization. Gradient angle calculation.
3) Calculate histogram bins for every cell (unsigned format/9 bins).
4) Calculate normalized bins for every block (l2 norm).
5) Flatten and concatenate normalized bins for every block to get a descriptor of
length 7524.
6) 3NN implementations using similarity formula: sum(min(input, train))/sum(train)

Notes:
- If the gradient angle is less than 0, add 360. If the gradient angle is greater than
180, subtract 180 from it.
- The larger the similarity, the smaller the distance between the input image and the
training image.

********************************************************************************
********************************

Dependency Installation: pip3 install -r requirements.txt

Usage: python3 main.py <path_to_image_data_folder>

Example: python3 main.py image_data

After execution, results will be displayed on the terminal. Folders 'descriptors' and
'test_images_gradient_magnitude' will be
created with relevant result information.
```

```
*********************************************************************************************
*********************************

Libraries Used:
- os: Create folder structure for output images
- cv2: Read input image, write output images
- argparse: Parse arguments (Input image file path)
- numpy: ndarray handling, magnitude calculation (sqrt(a^2+b^2)), gradient angle
calculation, setting undefined pixel values to 0, etc.
- more-itertools: Slice dictionary to get top 'k' entries
"""

import os
import cv2
import argparse
import numpy as np
from more_itertools import take

class HumanDetectorHOG():
   def __init__(self, image_data_path):
       """Initialise variables, constants

       Args:
           image_data_path (str): Path to image data folder
       """
       self.image_data_path = image_data_path

       self.training_set = {}
       self.testing_set = {}

       # Template shapes for cellwise and blockwise histogram bins
       self.CELL_TEMPLATE_SHAPE = (20,12)
       self.BLOCK_TEMPLATE_SHAPE = (19,11)

       # Prewitt's horizontal gradient operator
       self.PREWITT_X = np.array(
                           [[-1,0,1],
                           [-1,0,1],
                           [-1,0,1]]
                       )
       # Prewitt's vertical gradient operator
```

```
        self.PREWITT_Y = np.array(
                            [[1,1,1],
                            [0,0,0],
                            [-1,-1,-1]]
                        )
        # Maximum possible value for gradient magnitude
        self.MAGNITUDE_NORMALIZATION_FACTOR = 3*(2**0.5)

        # Folder names for saving descriptors and gradient magnitudes for test images
        self.DESCRIPTOR_FOLDER = 'descriptors'
        self.GRAD_MAG_FOLDER = 'test_images_gradient_magnitude'

        self.is_dir(self.DESCRIPTOR_FOLDER)
        self.is_dir(self.GRAD_MAG_FOLDER)

        self.driver()

    def is_dir(self, directory):
        """Helper function to create directories if they don't exist

        Args:
        directory (str): Directory path
        """
        if not os.path.isdir(directory):
            os.makedirs(directory)

    def write_descriptor(self, image_filename, descriptor):
        """Helper function to write descriptors into .txt files

        Args:
            image_filename (str): Image filename
            descriptor (ndarray): Numpy array of length 7524
        """
        descriptor_filename = image_filename.split('.')[0] + '_descriptor.txt'
        file_pointer = open(os.path.join(self.DESCRIPTOR_FOLDER, descriptor_filename),
'w')
        for value in descriptor:
            file_pointer.write(str(value)+'\n')
        file_pointer.close()

    def write_img(self, image_filename, gradient_magnitude):
        """Helper function to save image
```

```
        Args:
            image_filename (str): Image filename
            gradient_magnitude ([ndarray]): Normalized gradient magnitude numpy array
        """
        cv2.imwrite(os.path.join(self.GRAD_MAG_FOLDER,
'gradient_magnitude_'+image_filename), gradient_magnitude)

    def read_img(self, path):
        """Function to read image

        Args:
            path (str): Path to the image

        Returns:
            ndarray: Numpy array of the image
        """
        return cv2.imread(path)

    def bgr_2_gray(self, img):
        """BGR to GRAY conversion.
           cv2::imread returns image in BGR format

        Args:
            img (ndarray): Numpy array for colored image

        Returns:
            ndarray: Numpy array for grayscale image
        """
        blue, green, red = img[:,:,0], img[:,:,1], img[:,:,2]
        return np.around(0.299*red + 0.587*green + 0.114*blue)

    def convolution(self, x, y):
        """Implementation of convolution operation

        Args:
            x (ndarray): First numpy array
            y (ndarray): Second numpy array

        Returns:
            ndarray: Resultant of x*y; * -> convolution
        """
```

```
        x_shape = x.shape
        y_shape = y.shape
        output_shape = (x_shape[0]-y_shape[0]+1, x_shape[1]-y_shape[1]+1)
        output = np.zeros(output_shape)
        for itr_x in range(output_shape[0]):
            for itr_y in range(output_shape[1]):
                output[itr_x][itr_y] = (x[itr_x:itr_x+y_shape[0],
itr_y:itr_y+y_shape[1]]*y).sum()
        return output

    def gradient_info_calc(self, img):
        """Calculates horizontal and vertical gradients, gradient magnitude, and
gradient angle.
           Gradient magnitude is normalized by 3*root(2)
           Gradient angle is set to range [0,180]. If negative: Add 360. If > 180:
Subtract 180.

        Args:
            img (ndarray): Numpy array for grayscale image

        Returns:
            ndarray: Numpy array for normalized gradient magnitude
            ndarray: Numpy array for gradient angle
        """
        gradient_x = self.convolution(img, self.PREWITT_X)
        gradient_y = self.convolution(img, self.PREWITT_Y)

        gradient_magnitude = np.hypot(gradient_x, gradient_y)

        gradient_magnitude = gradient_magnitude/self.MAGNITUDE_NORMALIZATION_FACTOR

        gradient_angle = np.zeros(gradient_magnitude.shape)
        gradient_angle = np.rad2deg(np.arctan2(gradient_y, gradient_x))

        # If angle is negative add 360 to make it positive
        gradient_angle[gradient_angle < 0] += 360

        # If angle is greater than 180, subtract 180
        gradient_angle[gradient_angle > 180] -= 180

        return np.pad(gradient_magnitude, 1), np.pad(gradient_angle, 1)
```

```
    def get_ratio(self, angle):
        """Returns ratio by which magnitude is split between closest bins.

        Args:
            angle (float): Gradient angle at a pixel location

        Returns:
            dict: Ratio to split magnitude with bin index as the key
        """
        if angle <= 10:
            lower_center = 10
            itr_lower = 0
            itr_upper = 8
        elif angle >=170:
            lower_center = 170
            itr_lower = 8
            itr_upper = 0
        else:
            for itr_i in range(9):
                if itr_i*20 + 10 < angle:
                    lower_center = itr_i*20 + 10
                    itr_lower = itr_i
                    itr_upper = itr_i + 1
        ratio = {itr_lower: 1 - abs(angle-lower_center)/20, itr_upper:
abs(angle-lower_center)/20}
        return ratio

    def split_magnitude(self, magnitude, angle):
        """Calls get_ratio, calculates magnitude for each of the closest bins

        Args:
            magnitude (float): Gradient magnitude at a pixel location
            angle (float): Gradient angle at a pixel location

        Returns:
            dict: Magnitude value after splitting with bin index as the key
        """
        ratio = self.get_ratio(angle)
        return {key: value*magnitude for key, value in ratio.items()}

    def get_hist(self, magnitude_slice, angle_slice):
        """Calculate 9 bin histogram for a magnitude slice and angle slice
```

```
        Args:
            magnitude_slice (ndarray): Numpy array for gradient magnitude slice
            angle_slice (ndarray): Numpy array for gradient angle slice

        Returns:
            list: 9 bin histogram
        """
        hist_bin = [0,0,0,0,0,0,0,0,0]
        for itr_i in range(magnitude_slice.shape[0]):
            for itr_j in range(magnitude_slice.shape[1]):
                magnitude_split = self.split_magnitude(magnitude_slice[itr_i][itr_j],
angle_slice[itr_i][itr_j])
                for key, value in magnitude_split.items():
                    hist_bin[key] += value
        return hist_bin

    def calc_hist_bin(self, hist_bin_cellwise, gradient_magnitude, gradient_angle,
cell_size = 8):
        """Calculate 9 bin histograms for every cell

        Args:
            hist_bin_cellwise (ndarray): Numpy array of shape (20, 12)
            gradient_magnitude (ndarray): Numpy array for gradient magnitude
            gradient_angle (ndarray): Numpy array for gradient angle
            cell_size (int): 1 cell -> 8*8 pixels, defaults to 8

        Returns:
            ndarray: Numpy array of shape (20, 12), with 9 bin histograms for every
cell
        """
        for i in range(hist_bin_cellwise.shape[0]):
            for j in range(hist_bin_cellwise.shape[1]):
                magnitude_slice =
gradient_magnitude[i*cell_size:(i*cell_size)+cell_size,
j*cell_size:(j*cell_size)+cell_size]
                angle_slice = gradient_angle[i*cell_size:(i*cell_size)+cell_size,
j*cell_size:(j*cell_size)+cell_size]
                hist_bin_cellwise[i][j] = self.get_hist(magnitude_slice, angle_slice)
        return hist_bin_cellwise

    def flatten(self, block):
```

```
        """Helper function to flatten input

        Args:
            block (ndarray[list]): Numpy array of histograms for every cell/block

        Returns:
            ndarray: Flat numpy array (column vector)
        """
        block_flat = np.array([])
        for itr_i in range(block.shape[0]):
            for itr_j in range(block.shape[1]):
                block_flat = np.append(block_flat, block[itr_i][itr_j])
        return block_flat

    def get_l2_norm(self, block_flat):
        """Returns l2 normalization factor for a block

        Args:
            block_flat (ndarray): Column vector of histograms in a block

        Returns:
            float: L2 normalization factor for input block
        """
        return np.sqrt(np.sum(block_flat**2))

    def normalize_hist_bin(self, norm_hist_bin_blockwise, hist_bin_cellwise, block_size
= 2):
        """Calculates and returns histograms for every block after normalization

        Args:
            norm_hist_bin_blockwise (ndarray): Numpy array of shape (19, 11)
            hist_bin_cellwise (ndarray): Numpy array of histograms for every cell,
shape (20, 12)
            block_size (int): 1 block -> 2*2 cells, defaults to 2

        Returns:
            ndarray: Normalized histogram for every block, shape (19, 11)
        """
        for itr_i in range(norm_hist_bin_blockwise.shape[0]):
            for itr_j in range(norm_hist_bin_blockwise.shape[1]):
                block = hist_bin_cellwise[itr_i:itr_i+block_size,
itr_j:itr_j+block_size]
```

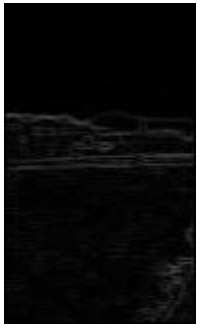
```
                block_flat = self.flatten(block)
                l2_norm = self.get_l2_norm(block_flat)
                norm_hist_bin_blockwise[itr_i][itr_j] = block_flat/l2_norm if l2_norm
!= 0 else block_flat
        return norm_hist_bin_blockwise

    def hog_driver(self, img):
        """Driver function to call steps of HOG to create descriptors

        Args:
            img (ndarray): Numpy array of grayscale image

        Returns:
            ndarray: Numpy array of normalized gradient magnitude
            ndarray: Numpy array of descriptor, column vector of length 7524
        """
        gradient_magnitude, gradient_angle = self.gradient_info_calc(img)
        hist_bin_cellwise = self.calc_hist_bin(np.empty(self.CELL_TEMPLATE_SHAPE,
object), gradient_magnitude, gradient_angle)
        norm_hist_bin_blockwise =
self.normalize_hist_bin(np.empty(self.BLOCK_TEMPLATE_SHAPE, object),
hist_bin_cellwise)
        descriptor = self.flatten(norm_hist_bin_blockwise)
        return gradient_magnitude, descriptor

    def calc_similarity(self, test_descriptor, train_descriptor):
        """Return histogram intersection between two descriptors

        Args:
            test_descriptor (ndarray): Descriptor for test image
            train_descriptor (ndarray): Descriptor for training image

        Returns:
            float: Similarity/Histogram intersection
        """
        minima = np.minimum(test_descriptor, train_descriptor)
        return np.true_divide(np.sum(minima), np.sum(train_descriptor))

    def predict(self, info):
        """Return class label based on neighbors

        Args:
```

```
            info (dict): Information about the 3 nearest neighbors

        Returns:
            str: Class label, calculated using majority from nearest neighbors
        """
        prediction = []
        for _, data in info.items():
            prediction.append(data['class'])
        return max(prediction, key=prediction.count)

    def knn(self, test_descriptor, k=3):
        """Returns K-Nearest Neighbors (k=3) for input test descriptor
           Calls calc_similarity and sorts the values in decreasing order
           Takes top k (k=3) neighbors for prediction

        Args:
            test_descriptor (ndarray): Descriptor for test image
            k (int): Number of neighbors to consider, defaults to 3

        Returns:
            str: Class label for input test descriptor
            dict: K-NN information, {image_filename:{'similarity':similarity score,
'class':class label}}
        """
        neighbor_info = {}
        for image_filename, image_data in self.training_set.items():
            similarity = self.calc_similarity(test_descriptor,
image_data['descriptor'])
            neighbor_info[image_filename] = {'similarity':similarity,
'class':image_data['class']}

        neighbor_info = {key: value for key, value in sorted(neighbor_info.items(),
key=lambda value: value[1]['similarity'], reverse=True)}
        knn_info = dict(take(k, neighbor_info.items()))

        prediction = self.predict(knn_info)

        return prediction, knn_info

    def classify(self):
        """Calls knn for every test image
        """
```

```
        for _, image_data in self.testing_set.items():
            image_data['predicted'], image_data['knn_info'] =
self.knn(image_data['descriptor'])

    def load_data(self):
        """Main function, parses through image data folder, reads image, calculates
descriptor and stores in a dict based on
           training/test substring
           Dict structure:
            Training images:
            {image_filename:{'img':img,'class':class label,'descriptor':descriptor}}
            Test images:
            {image_filename:{'img':img,'actual':true class
label,'descriptor':descriptor,'predicted':predicted class label,'knn_info':information
about knn}}
        """
        for sub_folder in os.listdir(self.image_data_path):
            if os.path.isdir(os.path.join(self.image_data_path, sub_folder)):
                for image_filename in os.listdir(os.path.join(self.image_data_path,
sub_folder)):
                        if '.bmp' in image_filename:
                            img =
self.bgr_2_gray(self.read_img(os.path.join(self.image_data_path, sub_folder,
image_filename)))
                            label = "Human" if 'pos' in sub_folder.lower() else
"No-Human"
                            gradient_magnitude, descriptor = self.hog_driver(img)
                            if image_filename in ['crop001028a.bmp', 'crop001030c.bmp',
'00000091a_cut.bmp', 'crop001278a.bmp', 'crop001500b.bmp', '00000090a_cut.bmp']:
                                self.write_descriptor(image_filename, descriptor)
                            if 'training' in sub_folder.lower():
                                self.training_set[image_filename] =
{'img':img,'class':label,'descriptor':descriptor}
                            if 'test' in sub_folder.lower():
                                self.write_img(image_filename, gradient_magnitude)
                                self.testing_set[image_filename] =
{'img':img,'actual':label,'descriptor':descriptor,'predicted':None,'knn_info':None}

    def display(self):
        """Print results for every test image
        """
        for key, value in self.testing_set.items():
```

```
            print(f'\n\nTest Image: {key}\nActual: {value["actual"]}\tPredicted:
{value["predicted"]}\nNeighbors:')
            for info_key, info_value in value["knn_info"].items():
                print(f'{info_key}: {info_value}')
            print('\n\n**********\n\n')

    def driver(self):
        """Driver calls data loading, classification, and result display
        """
        self.load_data()
        self.classify()
        self.display()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Human Detection using HOG Feature.')
    parser.add_argument('image_data_path', type=str, help='Path to the image data
folder')
    args = parser.parse_args()
    obj = HumanDetectorHOG(args.image_data_path)
```

Normalized Gradient Magnitude Images

00000003a_cut	no_person__no_bike_264_cut	no_person__no_bike_258_Cut	00000118a_cut	00000090a_cut
				
crop001070a	person_and_bike_151a	crop001500b	crop001034b	crop001278a
				

Classification Report

Test image	Correct Classification	1st Nearest Neighbor			2nd Nearest Neighbor			3rd Nearest Neighbor			Classification from 3-NN
		Filename	Similarity	Classification	Filename	Similarity	Classification	Filename	Similarity	Classification	
crop001034b	Human	crop001672b	0.6683	Human	00000053a_cut	0.6474	No-Human	01-03e_cut	0.6435	No-Human	No-Human
crop001070a	Human	00000053a_cut	0.4977	No-Human	person_and_bike_026a	0.4951	Human	crop001672b	0.4941	Human	Human
crop001278a	Human	crop001672b	0.5981	Human	crop001008b	0.5919	Human	crop001275b	0.5839	Human	Human
crop001500b	Human	crop001672b	0.5668	Human	00000091a_cut	0.5609	No-Human	crop001275b	0.5442	Human	Human
person_and_bike_151a	Human	crop001030c	0.5047	Human	person_and_bike_026a	0.5021	Human	crop001275b	0.4943	Human	Human

00000003a_cut	No-Human	00000053a_cut	0.5765	No-Human	crop001672b	0.5739	Human	00000093a_cut	0.5481	No-Human	No-Human
00000090a_cut	No-Human	00000093a_cut	0.4782	No-Human	00000057a_cut	0.4712	No-Human	crop001672b	0.4454	Human	No-Human
00000118a_cut	No-Human	00000093a_cut	0.5617	No-Human	00000053a_cut	0.5549	No-Human	00000091a_cut	0.5489	No-Human	No-Human
no_person__no_bike_258_Cut	No-Human	00000057a_cut	0.4961	No-Human	crop001672b	0.4889	Human	crop001275b	0.4845	Human	Human
no_person__no_bike_264_cut	No-Human	00000053a_cut	0.4429	No-Human	crop001672b	0.4404	Human	crop001030c	0.4351	Human	Human