HUMAN DETECTION USING HOG FEATURE

Tanmay Khot
New York University
tsk9863@nyu.edu

Niharika Krishnan
New York University
nk2982@nyu.edu

ABSTRACT:

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localised portions of an image.

GITHUB LINK: https://github.com/niharikakrishnan/Human-Detection-using-HOG

IMPLEMENTATION:

Assumes the pre-requisite environment with Python3 and necessary open source Python libraries are already installed. If not, please install using:

```
pip install numpy
pip install matplotlib
pip install opencv-python
```

Ensure the input image is in the same path as the source code file, ***main.py***

STEPS:

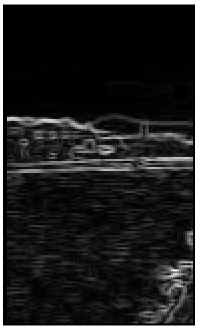
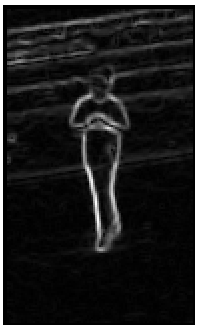
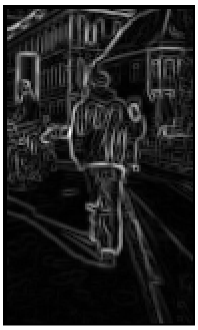
1. Open terminal window and change directory to where the solution code, ***main.py*** is located
2. Create empty directories:
   - data
   - data/train
   - data/test
   - data/train/Pos
   - data/train/Neg
3. Upload training and test images in the below folders respectively:
   - data/train/Pos
   - data/train/Neg
   - data/test/Pos
   - data/test/Neg
4. Parameters:
   - File Directory Path
   - Mode:
     - Train: Computes HOG Feature vector of the 20 training images
     - Test: Computes HOG Feature vectors of the 10 test images and classifies the image as Human or No-Human using 3-NN Classifier

5. Run the ***Command: python3 main.py “path_to_hog_directory” “train”***
Command: python3 main.py “path_to_hog_directory” “test”

Note: Some machines have a different python path setup, in such cases, please use:
Command: python main.py “path_to_hog_directory” “train”
Command: python main.py “path_to_hog_directory” “test”

Example: python main.py “Filepath\Human-Detection-using-HOG" "test"

NORMALISED GRADIENT MAGNITUDE IMAGE OUTPUTS

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

CLASSIFICATION RESULTS

Test Image	Correct Classification	Filename of 1st NN, Distance & classification	Filename of 2nd NN, Distance & classification	Filename of 3rd NN, Distance & classification	Classification from 3-NN
crop001034b	Human	crop001672b 0.6998562746481057 Human	No_person__no_bike_213_cut 0.6929603102505465 No-Human	crop001030c 0.6885699090957238 Human	Human
crop001070a	Human	Crop001063b 0.5489676238360646 Human	Crop001045b 0.5489486568373797 Human	Crop001672b 0.5471080914425701 Human	Human
crop001278a	Human	Crop001008b 0.6259858890864919 Human	Crop001672b 0.6194850228702111 Human	Crop001275b 0.6192598560443838 Human	Human
crop001500b	Human	Crop001672b 0.588742508592684 Human	No_person__no_bike_247_cut 0.5882612681371724 No-Human	Crop001275b 0.5751682811259875 Human	Human
person_and_bike_151a	Human	Crop001030c 0.5521344631404028 Human	Crop001008b 0.5430372398490518 Human	Crop001275b 0.5372362696460432 Human	Human
00000003a_cut	No-Human	00000053a_cut 0.6215917163733522 No-Human	Crop001672b 0.6121200904169186 Human	No_person__no_bike_259_cut 0.5980990746897218 No-Human	No-Human
00000090a_cut	No-Human	00000093a_cut 0.5520640028062873 No-Human	00000057a_cut 0.5235378151928898 No-Human	Crop001275b.bmp 0.47320817583028824 Human	No-Human
00000118a_cut	No-Human	00000093a_cut 0.5925475312785671 No-Human	00000053a_cut 0.5831106541500921 No-Human	No_person__no_bike_219_cut 0.5790847500938687 No-Human	No-Human
no_person_no_bike_258_cut	No-Human	00000057a_cut.bmp 0.542148687480278 No-Human	Crop001672b 0.5408639159798007 Human	Person_and_bike_026a 0.538132544037495 Human	Human
no_person_no_bike_264_cut	No-Human	00000053a_cut 0.4806324214902005 No-Human	Crop001030c 0.47627073490772437 Human	Crop001672b 0.475953986490818 Human	Human

Classification accuracy: 80%

SOURCE CODE:

Filename: main.py

```
# Project: Human Detection using HOG
# Run the program: python main.py "Filepath/Human-Detection-using-HOG" "train"
# Run the program: python main.py "Filepath/Human-Detection-using-HOG" "test"

from utils import *
import sys
import cv2

# Takes in input as argument from the user
path = sys.argv[1]
mode = sys.argv[2]

# If mode is train, positive and negative training images are read and HOG feature vector
is calculated
if mode.lower() == "train":
   if not os.path.isdir('results'):
       os.mkdir('results')
       os.mkdir('results/train_features')
       os.mkdir('results/test_features')
   train_images, filenames = load_images_from_folder(path + '/data/train/')

   for i in range(len(train_images)):
       img = train_images[i]
       filename = path + '/results/train_features/' + list(filenames.keys())[i] +
'_features.txt'
       get_hog_feature(img, filename)

# If mode is train, test images are loaded, HOG Feature Vector is calculated and the test
image is classified as Human or No-Human
if mode.lower() == "test":
   print("Running in Test Mode to generate HOG feature vector for the test.")
   train_images, train_files = load_images_from_folder(path + '/data/train/')
   test_images, test_files = load_images_from_folder(path + '/data/test/')

   print("Loading images from train and test")
   train_feature_path = 'results/train_features/'
   test_feature_path = 'results/test_features/'

   for i in range(len(test_images)):
       img = test_images[i]
       gradient_magnitude, horizontal_gradient, vertical_gradient = compute_gradients(img)
       filename = list(test_files.keys())[i]
       cv2.imwrite(filename, gradient_magnitude)
       filename = path + '/results/test_features/' + list(test_files.keys())[i] +
'_features.txt'
       get_hog_feature(img, filename)

   classification = get_nearest_neighbour(train_files, test_files, train_feature_path,
test_feature_path)
   for key, values in classification.items():
       print(key, values)
```

Filename: utils.py

```
import matplotlib.image as mpimg
import numpy as np
import os


def load_images_from_folder(path):
    """
    Function that loads training and test images from folder along with label names
    :param path: Input data folder path taken from user
    :return images, filenames: Returns image object and filenames read from train and test
image folders
    """
    images, filenames, labels = [], {}, ["Pos", "Neg"]   # Pos = 1, Neg = 0
    for label in labels:
        folder = path + "/" + label
        for filename in os.listdir(folder):
            img = load_image(os.path.join(folder, filename))
            if img is not None:
                images.append(img)
                if label == "Pos":
                    filenames[filename] = 1
                else:
                    filenames[filename] = 0
    return images, filenames


def load_image(path):
    """
    Function that loads a colour image and convert it into grayscale using the formula
specified in the problem statement.
    :param path: Colour Image object
    :return: Grayscale Image object
    """
    img = mpimg.imread(path)
    r, g, b = img[:, :, 0], img[:, :, 1], img[:, :, 2]
    img_gray = np.round(0.299 * r + 0.5870 * g + 0.1140 * b)
    return img_gray


def normalize(gradient_magnitude):
    """
    Function to normalize and round the gradient magnitude within the range (0,255)
    :param gradient_magnitude: Grayscale image
    :return: normalized gradient magnitude within the range 0 - 255
    """
    max_image_range = 255.0
    normalized_gradient_magnitude =
np.round(gradient_magnitude/(gradient_magnitude.max()/max_image_range))
    return normalized_gradient_magnitude


def padding(img, pad_size):
```

```
    """
    Function to pad the image before convolution to maintain the original shape
    :param img: Gradient Magnitude of the image
    :param pad_size: Padding size that needs to be applied to the image
    :return: Padded gradient magnitude of the image
    """
    h, w = img.shape
    padded_img = np.zeros((h + 2*pad_size, w + 2*pad_size))
    for i in range(pad_size, h+pad_size):
        for j in range(pad_size, w+pad_size):
            padded_img[i][j] = img[i-pad_size][j-pad_size]
    return padded_img


def convolution(img, filter_mask):
    """
    Function to perform convolution on the image using the given filter.
    :param img: Gradient Magnitude of the image
    :param filter_mask: Prewitt's filter
    :return: Convoluted image after applying Prewitt's operator
    """
    image_rows, image_cols = img.shape  # image
    filter_rows, filter_cols = filter_mask.shape  # filter
    result_rows, result_cols = image_rows - filter_rows + 1, image_cols - filter_cols + 1
# result
    result = np.zeros((result_rows, result_cols))

    for i in range(result_rows):
        for j in range(result_cols):
            result[i][j] = np.sum(img[i:i+filter_rows, j:j+filter_cols] * filter_mask)
    return result


def compute_gradients(img):
    """
    Function to computer gradient magnitude of the image using Prewitt's operator and
normalize it
    :param img: Gradient Magnitude of the image
    :return: Normalized Gradient Magnitude
    """
    gx = np.array([
                    [-1, 0, 1],  # Prewitt's operator for Gradients Gx
                    [-1, 0, 1],
                    [-1, 0, 1]
    ], dtype='int')

    gy = np.array([
                    [1, 1, 1],  # Prewitt's operator for Gradients Gy
                    [0, 0, 0],
                    [-1, -1, -1]
    ], dtype='int')

    pad_size = 1
    grad_x = padding(convolution(img, gx), pad_size)
    grad_y = padding(convolution(img, gy), pad_size)
```

```
    x, y = grad_x.shape

    roi_x, roi_y = np.zeros((x-(2*pad_size), y-(2*pad_size))), np.zeros((x-(2*pad_size),
y-(2*pad_size)))
    roi_x_rows, roi_x_cols = roi_x.shape
    for i in range(roi_x_rows):  # Ignoring the border pixels from padding from previous
operations
        for j in range(roi_x_cols):
            roi_x[i][j] = grad_x[i][j]
            roi_y[i][j] = grad_y[i][j]

    gradient_magnitude = np.sqrt(roi_x*roi_x + roi_y*roi_y)  # Computing gradient magnitude
    normalized_gradient_magnitude = padding(normalize(gradient_magnitude), pad_size)  #
Returning 0 - 255 range
    return normalized_gradient_magnitude, grad_x, grad_y


def compute_gradient_angles(y, x):
    """
    Function to compute gradient angles of the image
    :param y: Vertical gradient of the image
    :param x: Horizontal gradient of the image
    :return: Computes gradient angles of the image
    """
    rows, cols = y.shape
    result = np.zeros((rows, cols))
    for i in range(rows):
        for j in range(cols):
            if x[i][j] == 0 and y[i][j] == 0:
                result[i][j] = 0
            elif x[i][j] != 0:   # to avoid division by zero
                result[i][j] = np.degrees(np.arctan(y[i][j] / x[i][j]))

    return result


def update_bins(gradients, angles, bin_number):
    """
    Function to calculate the weighted sum and update histogram bin values using bin
centres as reference
    :param gradients: gradient magnitude of an image
    :param angles: gradient angle of an image
    :param bin_number: number of bins based on signed and unsigned representation
    :return: Updated histogram bins for each cell
    """
    bins = [0] * bin_number
    rows = len(angles)
    cols = len(angles[0])

    for r in range(rows):
        for c in range(cols):
            angle = angles[r][c]
            magnitude = gradients[r][c]

            #  Normalising angle between 0 - 180
```

```
            if angle < 0:
                angle += 360
            if angle >= 180:
                angle -= 180

            #  Calculating the distance of the gradient angle and multiplying it with the
magnitude
            fraction = ((angle+10) % 20)/20    # Fraction of magnitude to be distributed
between the bins
            first_bin_value = (1-fraction) * magnitude
            second_bin_value = fraction * magnitude

            # Find the index of the bin based on input angle
            first_bin = int((angle+10)/20) - 1
            if first_bin < 0 or first_bin == 8:
                first_bin = 8
                second_bin = 0
            else:
                second_bin = first_bin + 1

            bins[first_bin] += round(first_bin_value, 2)
            bins[second_bin] += round(second_bin_value, 2)

    return bins


def compute_hog_feature(gradients, angles, cell_size, step_size, block_size, bins):
    """
    Function to calculate HOG feature vector for an image
    :param gradients: Normalized gradient magnitude of the image
    :param angles: Gradient angle of the image
    :param cell_size: Cell size as given in the problem statement
    :param step_size: Block step size
    :param block_size: Block size used for normalization
    :param bins: Number of bins based on signed or unsigned representation
    :return: Returns a histogram for each block after normalization
    """
    gradient_rows, gradient_cols = gradients.shape
    cell_rows = int((gradient_rows - cell_size)/step_size+1)
    cell_cols = int((gradient_cols - cell_size)/step_size+1)
    cell_histogram_list = []
    flag_row = 0

    for i in range(cell_rows):
        flag_col = 0
        for j in range(cell_cols):
            gradient_magnitude_roi = gradients[flag_row: flag_row+cell_size, flag_col:
flag_col+cell_size]  # Magnitude region of interest
            gradient_angle_roi = angles[flag_row: flag_row+cell_size, flag_col:
flag_col+cell_size]  # Angle region of interest
            histogram = update_bins(gradient_magnitude_roi, gradient_angle_roi, bins)  #
9X1
            cell_histogram_list.append(histogram)
            flag_col += step_size
        flag_row += step_size
```

```
    # Converting cell-array list to 20 X 12 X 9 nd-array
    cell_histogram_list = np.reshape(cell_histogram_list, (cell_rows, cell_cols, bins))

    # Normalizing Block
    block_step_size = int(block_size/cell_size)
    block_row = int(cell_rows - block_step_size + 1)
    block_col = int(cell_cols - block_step_size + 1)
    block_histogram_list = []
    flag_row = 0

    for i in range(block_row):
        flag_col = 0
        for j in range(block_col):
            block_roi = cell_histogram_list[flag_row: flag_row+block_step_size, flag_col:
flag_col+block_step_size]  # Block Region of Interest
            normalized_block_list = block_roi / (np.sqrt(np.sum(block_roi ** 2) + 0.00005))
# L2 norm , adding 0.00005 to avoid division by 0
            normalized_block_list = normalized_block_list.flatten().tolist()  # 36X1 vector
            block_histogram_list += normalized_block_list
            flag_col += 1
        flag_row += 1

    return block_histogram_list


def write_hog_feature(filename, hog_feature):
    """
    Function to write the HOG feature vector into a text file
    :param filename: Image name sent as filename
    :param hog_feature: HOG Feature Vector
    """
    print("Saved HOG feature vector in file: " + filename)
    np.savetxt(filename, hog_feature)


def get_nearest_neighbour(train_files, test_files, train_feature_path, test_feature_path):
    """
    Function to implement the 3-NN classifier and classify the input image as human or
no-human
    :param train_files: HOG feature vector of trained images
    :param test_files: HOG feature vector of test images
    :param train_feature_path: HOG feature vector directory of trained images
    :param test_feature_path: HOG feature vector directory of test images
    :return: top three predictions with file name, distance and prediction label
    """
    all_distances = {}  # Contains distance and name for every training image for all test
images
    top_three = {}  # Contains top three distances (first, second, third), original labels
of nearest neighbours, predicted label, actual label

    class_mapping = {1: 'Human', 0: 'No-Human'}
    for test in test_files:
        with open(test_feature_path + test + '_features.txt') as ts_file:
            test_value = ts_file.readlines()
```

```
            test_value = [line.rstrip() for line in test_value]
            all_distances[test] = {}

            for train in train_files:
                with open(train_feature_path + train + '_features.txt') as tr_file:
                    train_value = tr_file.readlines()
                    train_value = [line.rstrip() for line in train_value]

                size = len(train_value)
                numerator = 0
                denominator = 0
                for i in range(size):
                    numerator += min(float(train_value[i]), float(test_value[i]))  #
Computing KNN
                    denominator += float(train_value[i])

                distance = numerator / denominator
                all_distances[test][train] = distance

            result = sorted(all_distances[test].items(), key=lambda x: x[1], reverse=True)
# [image name, distance]
            top_three[test] = {}
            top_three[test]['first'] = [result[0][1], result[0][0],
class_mapping[train_files[result[0][0]]]]  # [distance, image, training_label]
            top_three[test]['second'] = [result[1][1], result[1][0],
class_mapping[train_files[result[1][0]]]]
            top_three[test]['third'] = [result[2][1], result[2][0],
class_mapping[train_files[result[2][0]]]]

            #  Checking if the majority
            if train_files[result[0][0]] + train_files[result[1][0]] +
train_files[result[2][0]] > 1:  # If sum is 2 or 3, that means there's a majority of Pos
labels
                top_three[test]['Prediction'] = class_mapping[1]
            else:
                top_three[test]['Prediction'] = class_mapping[0]

            top_three[test]['Actual'] = class_mapping[test_files[test]]

    return top_three


def get_hog_feature(img, filename):
    """
    Function that computes gradient magnitude, gradient angle, HOG feature for both
training and test images
    :param img: Input Image
    :param filename: Path and filename to write the HOG feature vector into
    """
    gradient_magnitude, horizontal_gradient, vertical_gradient = compute_gradients(img)
    gradient_angle = compute_gradient_angles(vertical_gradient, horizontal_gradient)
    hog_vector = compute_hog_feature(gradient_magnitude, gradient_angle, cell_size=8,
step_size=8, block_size=16, bins=9)
    write_hog_feature(filename, hog_vector)
```