CS 6643 Computer Vision
Project 2

Human Detection

Using HOG Feature

Submitted By:

Shravani Rakshe spr4123 spr4123@nyu.edu
Tanya Sharma ts4524 ts4524@nyu.edu

ABSTRACT:

The Histograms of Oriented Gradients (HOG) is a feature descriptor widely used in computer vision to extract features from image data for the purpose of object detection. Our objective for this project is to write a program to compute the HOG feature from an input image and then classify the HOG feature vector into Human or No-human by using a 3-nearest neighbor (NN) classifier.

INSTRUCTIONS:

Assumes the pre-requisite environment with Python3 and necessary libraries (opencv & numpy) are already installed. If not, please install using:

```
pip install opencv-python

pip install numpy
```

STEPS:

1. Create a directory named "training" and paste all the 20 training images into this folder.

2. Open the terminal at hog.py and run the following command. Make sure to make the training directory mentioned in Step 1 at the same location as hog.py because the path name is hard-coded.

```
python3 hog.py -i "path_to_input_file"
```

3. The code has been written in such a way that it only accepts one input test image at a time. To test for 10 images, run the command 10 times while changing the path to the input image. Every test image's descriptor is saved as a text file called *test_image_filename*_descriptor.txt and normalized gradient magnitude image as *test_image_filename*_GradientMagnitude.bmp.

4. Example of what the output looks like:

```
(venv) (base) shravanirakshe@Shravanis-MBP pythonProject1 % python3 hog.py -i "Test1/00000118a_cut.bmp"
Running Human Detection using HOG Feature!
The 3 nearest neighbors are :
00000093a_cut    0.564820011309363       No-human
00000053a_cut    0.5547408956594974      No-human
00000091a_cut    0.5506417209702358      No-human
Human Not Detected!
```

SOURCE CODE:

```
"""
Steps :
1. Read the Test image and convert it to grayscale using G = round(0.299R + 0.587G
+ 0.114B).
2. Calculate the Gradient Magnitude using Prewitt's operator. Normalize gradient
values and then compute the Gradient angle.
3. Compute the descriptor for the test image which will be of dimension 7524 x 1.
4. Perform training. Repeat steps 1 to 3 for all 20 training images.
5. Calculate similarity by using Histogram Intersection. Find the 3 nearest
neighbors.
6. Classify the image as Human or No-human.
"""
import numpy as np
import argparse
import cv2
import os

def convolution(image, mask):
    image_row, image_col = image.shape
    mask_row, mask_col = mask.shape
    convoluted_image = np.zeros(image.shape)
    add_row = int(mask_row - 1) // 2
    add_col = int(mask_col - 1) // 2

    # Initializing a 2D array with zeros along with extra rows and columns to
handle the undefined values
    modified_image = np.zeros((image_row + (2 * add_row), image_col + (2 *
add_col)))

    modified_image_row, modified_image_col = modified_image.shape

    # Defining the region of interest for the input image
    modified_image[add_row: modified_image_row - add_row,
add_col:modified_image_col - add_col] = image

    # Matrix multiplication - Performing convolution on image with kernel
    for row in range(1, image_row - 1):
        for col in range(1, image_col - 1):
```

```
            # Using sliding window concept for matrix multiplication of kernel and
region of interest of input image
            convoluted_image[row, col] = np.sum(mask * modified_image[row: row +
mask_row, col: col + mask_col])

    return convoluted_image


def gradient_operation(image, edge_filter):
    # Computing the horizontal gradient by performing convolution the input image
with Prewitt's horizontal edge filter
    horizontal_gradient = convolution(image, edge_filter)
    # Output: [[1,1,1], [0,0,0], [-1,-1,-1]]
    vertical_edge_filter = np.flip(edge_filter.T, axis=0)

    # Computing the vertical gradient by performing convolution the input image
with Prewitt's vertical edge filter
    vertical_gradient = convolution(image, vertical_edge_filter)

    # Using the formula, gradient magnitude = Square Root of Squares of Horizontal
and Vertical Gradient
    gradient_magnitude = np.sqrt(np.square(horizontal_gradient) +
np.square(vertical_gradient))
    gradient_magnitude = gradient_magnitude / (3 * (np.sqrt(2)))



# Calculating gradient angle -> tan inverse (vertical gradient/horizontal gradient)
in radians
    gradient_angle = np.arctan2(vertical_gradient, horizontal_gradient)

    # Converting gradient angle from radians to degree which returns in the range
of -180 to 180.
    gradient_direction = np.rad2deg(gradient_angle)

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

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

    return horizontal_gradient, vertical_gradient, gradient_magnitude,
gradient_direction
```

```
def histogram_calculate(pixel_mag, pixel_angle, orientation_bin_midpoints,
cell_list):
    # Calculating histogram split for every pixel depending on the distance from
the bin centers

    if 10 <= pixel_angle < 170:
        # If the gradient angle for a pixel is between 10 and 170 then calculate
bin centers and corresponding indexes
        # for the histogram
        for i in range(len(orientation_bin_midpoints)):
            if pixel_angle < orientation_bin_midpoints[i]:
                bin1 = orientation_bin_midpoints[i - 1]
                idx1 = i - 1
                idx2 = i
                ratio1 = abs(pixel_angle - bin1) / 20;
                ratio2 = abs(20 - (pixel_angle - bin1)) / 20;

                break

    elif 0 <= pixel_angle < 10:
        bin2 = 10
        idx1 = 0
        idx2 = len(orientation_bin_midpoints) - 1
        ratio1 = (bin2 - pixel_angle) / 20;
        ratio2 = (20 - (bin2 - pixel_angle)) / 20;
    else:
        bin1 = 170
        idx1 = len(orientation_bin_midpoints) - 1
        idx2 = 0
        ratio1 = abs(pixel_angle - bin1) / 20
        ratio2 = abs(20 - (pixel_angle - bin1)) / 20

    # Updating the 2 bins in the histogram depending on the distance of pixel angle
from the bin centers
    cell_list[idx1] += pixel_mag * ratio2
    cell_list[idx2] += pixel_mag * ratio1


def histogram_cell(gradient_magnitude, gradient_direction,
orientation_bin_midpoints, output_block):
```

```
    # Calculating histogram for each cell and saving the magnitudes in the
cell_list array which has length 9.
    cell_list = np.zeros(9, dtype=float)
    for i in range(gradient_magnitude.shape[0]):
        for j in range(gradient_magnitude.shape[1]):
            # Pass the gradient magnitude and angle at every pixel location of a
cell to histogram_calculate
            histogram_calculate(gradient_magnitude[i][j], gradient_direction[i][j],
orientation_bin_midpoints,
                                cell_list)
    # Concatenating the histogram cells into one array of dimension 1 x 36.
    output_block.extend(list(cell_list))

def histogram_block(gradient_magnitude, gradient_direction,
orientation_bin_midpoints, descriptor):
    output_block = []

    # Going through all 4 cells in a block to compute histogram. Dimension of a
cell is 8 x 8 pixels.
    for i in range(0, gradient_magnitude.shape[0], 8):
        for j in range(0, gradient_magnitude.shape[1], 8):
            histogram_cell(gradient_magnitude[i:i + 8, j:j + 8],
gradient_direction[i:i + 8, j:j + 8],
                           orientation_bin_midpoints, output_block)

    # Getting the L2-Norm value for each block
    normalized_block_value = get_l2_norm(np.array(output_block))

    # Dividing all the values by L2-Norm value
    if normalized_block_value != 0:
        output_block = output_block / normalized_block_value
    descriptor.extend(output_block)


def histogram_image(gradient_magnitude, gradient_direction,
orientation_bin_midpoints, descriptor):
    # Creating overlapping blocks of the image. Dimension of the block is 16 x 16
pixels or 2 x 2 cells.
    for i in range(0, gradient_magnitude.shape[0] - 8, 8):
        for j in range(0, gradient_magnitude.shape[1] - 8, 8):
            histogram_block(gradient_magnitude[i:i + 16, j:j + 16],
gradient_direction[i:i + 16, j:j + 16],
                            orientation_bin_midpoints, descriptor)
```

```
def get_l2_norm(block):
    l2_norm = 0
    for i in range(block.shape[0]):
        # Summing the squares of each block element
        l2_norm += np.array(block[i]) ** 2

    # Returning square root of the sum
    return np.sqrt(l2_norm)

def histogram_intersection(train_img, test_img):
    # Applying the histogram intersection formula to calculate similarity
    total = np.sum(train_img)
    min_dist = 0
    for i in range(0, train_img.shape[0]):
        min_dist += min(train_img[i], test_img[i])
    return min_dist / total


def knn(neighbour_info, training_set):
    # To find the majority of the 3 nearest neighbors to the test image
    print("The 3 nearest neighbors are : ")
    majority = 0
    for k, v in neighbour_info.items():
        if training_set.get(k) == "Human":
            majority += 1
        print(k, "\t", v, "\t", training_set.get(k))
    # For majority being Human
    if majority > 1:
        print("Human Detected!")
    # For majority being No-human
    else:
        print("Human Not Detected!")


if __name__ == '__main__':
    print("Running Human Detection using HOG Feature!")

    # Reading the input file name from the arguments passed from command line
    ap = argparse.ArgumentParser()
    ap.add_argument("-i", "--image", required=True, help="Path to the image")
    args = vars(ap.parse_args())
    frame = cv2.imread(args['image'])
```

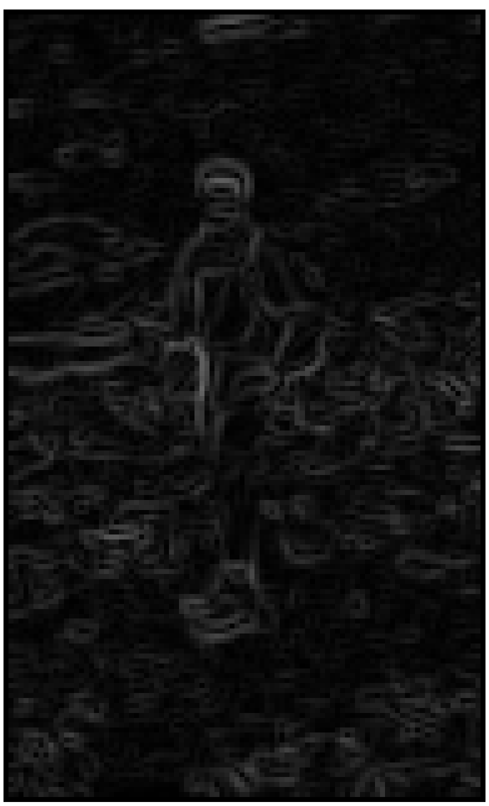
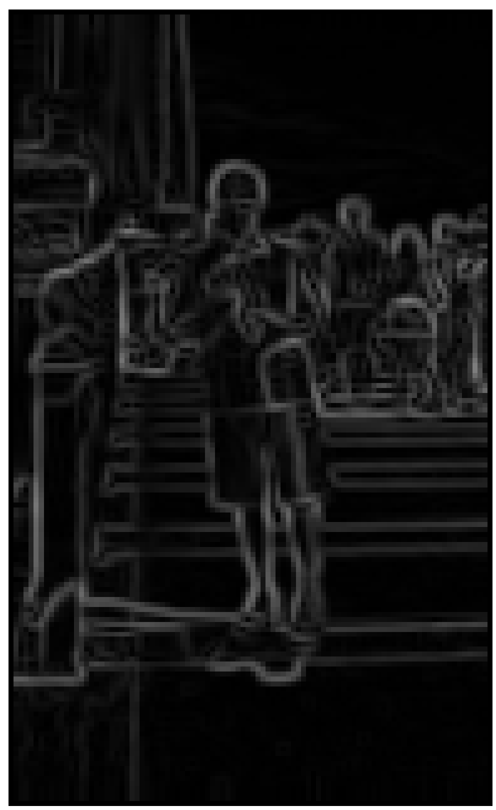
```
    # Converting the image to grayscale
    blue, green, red = frame[:, :, 0], frame[:, :, 1], frame[:, :, 2]
    test_image = np.around(0.299 * red + 0.587 * green + 0.114 * blue)

    # Declaring the Prewitt's operator
    edge_filter = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]], dtype='int')

    # Computing gradient magnitude and gradient angle for test image
    horizontal_gradient, vertical_gradient, gradient_magnitude, gradient_direction
= gradient_operation(test_image,

edge_filter)
    # Creating path to write output images of Gradient Magnitude
    folder, fname_with_extension = os.path.split(args['image'])
    fname, extension = os.path.splitext(fname_with_extension)
    path = str(fname) + "_output"
    access = 0o755
    cv2.imwrite(fname + "_GradientMagnitude.bmp", gradient_magnitude)

    descriptor = []

    # Calculating bin centers for the histogram. Number of bins = 9.
    orientations = 9
    orientations_arr = np.arange(orientations)
    orientation_bin_midpoints = (
            180 * (orientations_arr + .5) / orientations)

    similarity = {}
    # Hard coding the training images with their equivalent labels.
    training_dataset = {"01-03e_cut": "No-human", "00000053a_cut": "No-human",
"00000057a_cut": "No-human",
                        "00000062a_cut": "No-human",
                        "00000091a_cut": "No-human", "00000093a_cut": "No-human",
                        "no_person__no_bike_213_cut": "No-human",
"no_person__no_bike_219_cut": "No-human",
                        "no_person__no_bike_247_cut": "No-human",
"no_person__no_bike_259_cut": "No-human",
                        "crop001008b": "Human"
                        , "crop001028a": "Human", "crop001030c": "Human",
"crop001045b": "Human", "crop001047b": "Human",
                        "crop001063b": "Human", "crop001275b": "Human",
"crop001672b": "Human",
```

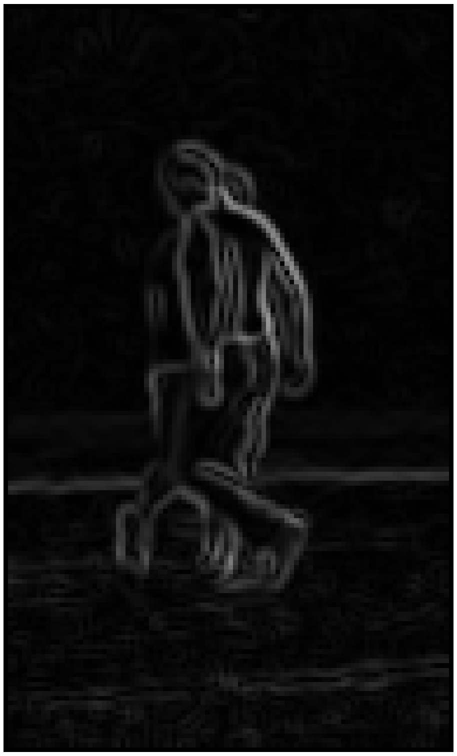
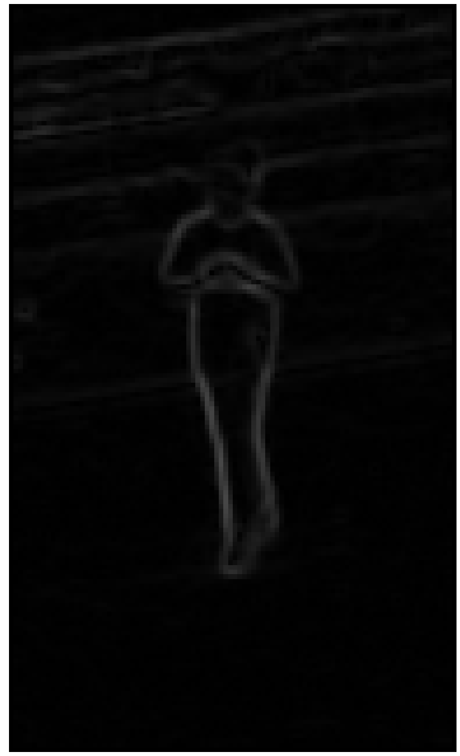
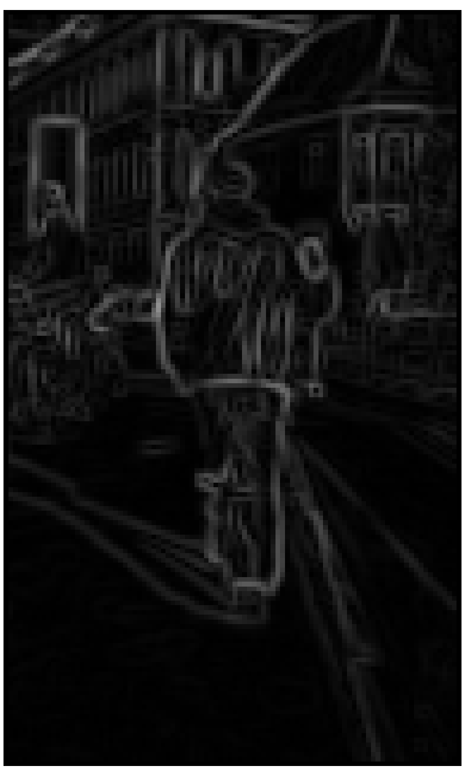
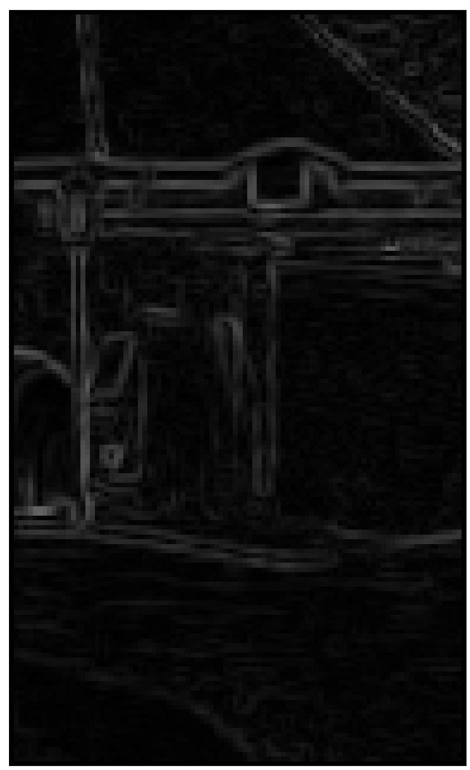
```
                        "person_and_bike_026a": "Human"
                        }
    # Computing descriptor for the test image
    histogram_image(gradient_magnitude, gradient_direction,
orientation_bin_midpoints, descriptor)
    descriptor_array = np.array(descriptor)

    # Saving the descriptor of the test image to an output file.
    file = open(fname + "_descriptor.txt", "w+")
    for output in descriptor_array:
        content = str(output) + "\n"
        file.write(content)
    file.close()

    # Running a loop through all the training images to compute a descriptor for
each.
    for key, value in training_dataset.items():
        # Reading all the images from a directory named "training".
        frame_train = cv2.imread("training/" + key + ".bmp")

        # Converting every training image to grayscale.
        train_image = np.round(
            0.299 * frame_train[:, :, 0] + 0.587 * frame_train[:, :, 1] + 0.114 *
frame_train[:, :, 2], decimals=5)

        # Computing gradient magnitudes and gradient angles for every training
image.
        horizontal_gradient1, vertical_gradient1, gradient_magnitude1,
gradient_direction1 = gradient_operation(
            train_image,
            edge_filter)

        # Computing descriptor for every training image
        training_image_descriptor = []
        histogram_image(gradient_magnitude1, gradient_direction1,
orientation_bin_midpoints, training_image_descriptor)
        train_image_array = np.array(training_image_descriptor)

        # Calculating the similarity of the test image with every training image.
        s = histogram_intersection(train_image_array, descriptor_array)
        similarity[key] = s

    similarity = {k: v for k, v in sorted(similarity.items(), key=lambda item:
```

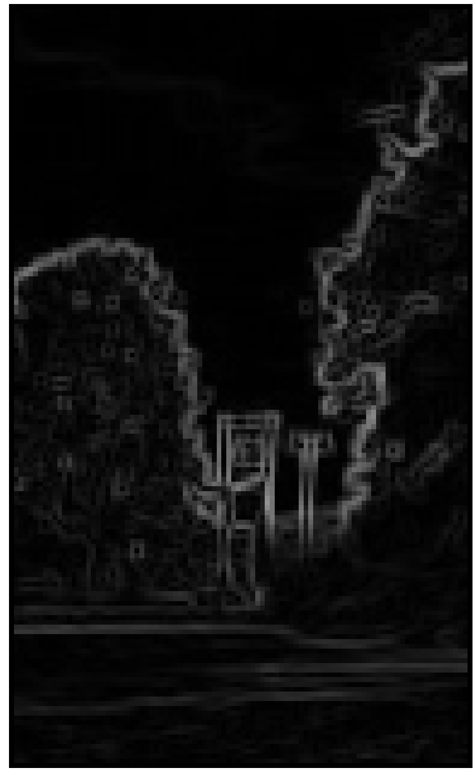
```
item[1], reverse=True)}
    # Choose first 3 with highest similarity from list sorted in non-increasing order
    nearest_3 = dict(list(similarity.items())[0: 3])

    # Predicting if the image is Human or No-Human.
    knn(nearest_3, training_dataset)
```

GITHUB LINK: https://github.com/Shravanirakshe/Human-Detection-using-Hog-Feature

NORMALIZED GRADIENT MAGNITUDE IMAGES:

crop001034b	crop001070a
	

crop001278a	crop001500b
	
person_and_bike_151a	**00000003a_cut**
	

<table>
<tr><th>00000090a_cut</th><th>00000118a_cut</th></tr>
<tr><td></td><td></td></tr>
<tr><th>no_person_no_bike_258_cut</th><th>no_person_no_bike_264_cut</th></tr>
<tr><td></td><td></td></tr>
</table>

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	crop001672b	00000053a_cut	01-03e_cut	No-human
		0.668242669	0.6477373897717	0.643410994122	
		Human	No-human	No-human	
crop001070a	Human	00000053a_cut	crop001672b	person_and_bike_026a	Human
		0.4975243711343	0.4933337138282	0.492986645976	
		No-human	Human	Human	
crop001278a	Human	crop001672b	crop001008b	crop001275b	Human
		0.598538437757	0.5931469580453	0.582828397301	
		Human	Human	Human	
crop001500b	Human	crop001672b	00000091a_cut	crop001275b	Human
		0.566437778319	0.5612793469956	0.544118262853	
		Human	No-human	Human	
person_and_bike_151a	Human	crop001030c	person_and_bike_026a	crop001275b	Human
		0.506334844885	0.5009413266443	0.496449164829	
		Human	Human	Human	
00000003a_cut	No-human	00000053a_cut	crop001672b	00000093a_cut	No-human
		0.5764686525740108	0.5746965447656947	0.5521606411839521	
		No-human	Human	No-human	
00000090a_cut	No-human	00000093a_cut	00000057a_cut	crop001672b	No-human
		0.4807728787306907	0.47064281818404663	0.44481268748162206	

		No-human	No-human	Human	
00000118a_cut	No-human	00000093a_cut	00000053a_cut	00000091a_cut	No-human
		0.5648200113093	0.5547408956594	0.550641720970	
		No-human	No-human	No-human	
no_person_no_bike_258_ cut	No-human	00000057a_cut	crop001672b	person_and_bike_026a	Human
		0.497935223143	0.4869598468429	0.483411603683	
		No-Human	Human	Human	
no_person_no_bike_264_ cut	No-human	00000053a_cut	crop001672b	01-03e_cut	No-human
		0.443191976150	0.4413231984251	0.434106880621	
		No-human	Human	No-human	

<u>OUTPUT:</u>

The accuracy of the model is **80%**. One positive test image and one negative image has been misclassified.
From the above table, we can see that the misclassified images are

1. crop001034b Actual - Human, Prediction - No-human
2. no_person_no_bike_258_cut Actual - No-human, Prediction - Human