CS 6643

Project2: Human Detection Using HOG Feature

Project Group: 1. Sindhu Harish (N219806874)

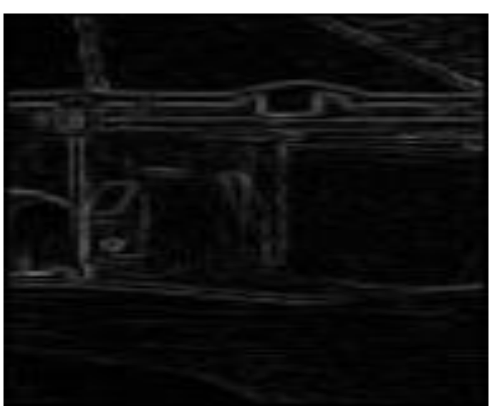
2. Aniket Bote (N12824308)

Instructions to run the code

1. Download and install python 3.8.6
2. Clone the project and navigate to the project directory named human-detection
3. Create a virtual environment. (OPTIONAL)
   - ➔ pip install virtualenv
   - ➔ virtualenv venv
4. install the required libraries after activating the virtual environment
   - ➔ pip install -r req.txt
5. Run the below command.
   - ➔ python human_detection.py
6. Normalized gradient magnitude images for 10 test images are placed in the folder "Output/normalized_images".
7. The ASCII (.txt) files containing the HOG feature values for three of the training images and three of the test images (one file per image are placed under "Output/hogvalues" folder.
8. Classification results are saved to output.csv under Output folder.

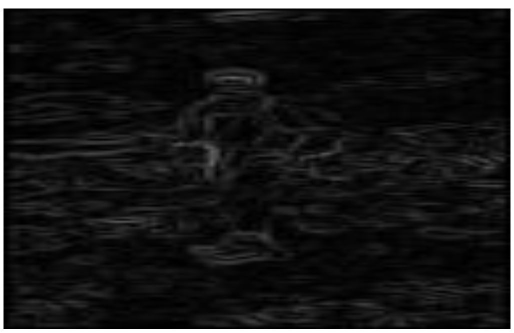
Normalized gradient magnitude images for the 10 test images:

1. 00000003a_cut.bmp



2. 00000090a_cut.bmp

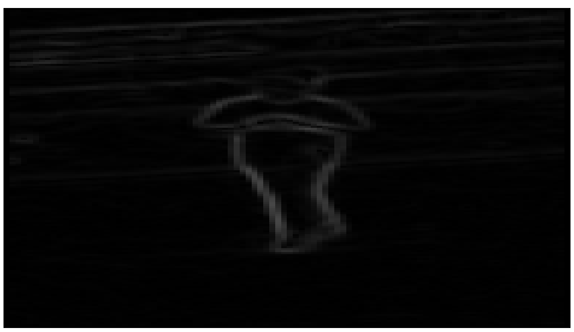
3. 00000118a_cut.bmp



4. crop001034b.bmp

5. crop001070a.bmp

6. crop001278a.bmp



7. crop001500b.bmp

8. no_person__no_bike_258_Cut.bmp

9. no_person__no_bike_264_cut.bmp

10. person_and_bike_151a.bmp

Source Code:

Human-detection.py

```python
  '''
Computer Vision Final Project
Project group members:
   1. Aniket Bote (N12824308)
   2. Sindhu Harish (N19806874)
'''

# Import the required libraries
import os
import shutil
import numpy as np

from data import create_dataset
from knn import KNNClassifier
from output import generate_table, save_normalized_images


# Initialize variables
# Path to train test directories
TRAIN_DIR_POS = "data/Training images (Pos)"
TRAIN_DIR_NEG = "data/Training images (Neg)"
TEST_DIR_POS = "data/Test images (Pos)"
TEST_DIR_NEG = "data/Test images (Neg)"

# Path to save the hog values
HOG_SAVE_PATH = "Output/hog_values"

# Path to save the normalized images
IMAGE_SAVE_PATH = "Output/normalized_images"

# Path to save the output table
TABLE_SAVE_PATH = "Output/output.csv"

# Names of images to compute hog values
FILE_NAME_TRAIN = ["crop001028a.bmp", "crop001030c.bmp", "00000091a_cut.bmp"]
FILE_NAME_TEST = ["crop001278a.bmp", "crop001500b.bmp", "00000090a_cut.bmp"]

# Remove the output folder if it exists
if os.path.exists(HOG_SAVE_PATH):
   shutil.rmtree(HOG_SAVE_PATH)
```

```
# Remove the output folder if it exists
if os.path.exists(IMAGE_SAVE_PATH):
   shutil.rmtree(IMAGE_SAVE_PATH)

# Create the output folder
os.makedirs(HOG_SAVE_PATH)
os.makedirs(IMAGE_SAVE_PATH)


# Initialize the number of neighbours
K = 3

# Create dataset from train & test images
X_train, y_train, train_image_list = create_dataset(TRAIN_DIR_POS,TRAIN_DIR_NEG)
X_test, y_test, test_image_list = create_dataset(TEST_DIR_POS, TEST_DIR_NEG)

# Create KNN classfier model
knn_classfier = KNNClassifier(K,X_train, y_train)

# Use the KNN classifier model for predictions
y_preds, k_top_neighbours = knn_classfier.predict(X_test)

# Generate output table & save the results
output_df = generate_table(y_test, y_preds, k_top_neighbours, train_image_list,
test_image_list)
output_df.to_csv(TABLE_SAVE_PATH, index = False)

# Saving the normalized test images
save_normalized_images(TEST_DIR_POS, TEST_DIR_NEG, IMAGE_SAVE_PATH)

# Save the hog values for selected images
for image_name in FILE_NAME_TEST:
   index = test_image_list.index(image_name)
   np.savetxt(os.path.join(HOG_SAVE_PATH,"hog_"+ image_name.split('.')[0] + '.txt'),
X_test[index])

for image_name in FILE_NAME_TRAIN:
   index = train_image_list.index(image_name)
   np.savetxt(os.path.join(HOG_SAVE_PATH,"hog_"+ image_name.split('.')[0] + '.txt'),
X_train[index])
```

data.py

```python
'''

Computer Vision Final Project
Project group members:
   1. Aniket Bote (N12824308)
   2. Sindhu Harish (N19806874)
'''

# Import the required libraries
import os
import numpy as np
from skimage.io import imread
from hog import HOG
from grayscale import convert_to_grayscale

def load_data(path):
   '''
   Read image data from directory and compute hog features

   Args:
      path: The path string to image dir
   Returns:
      matrix: An array containing the hog features of all images in image dir
      image_list: An list containing all the image names
   '''

   # Initialize empty array matrix, image_list
   matrix = []
   image_list = []

   # Initialize HOG object
   hog_obj = HOG(n_bins=9, cell_size=(8,8), block_size=(2,2), step_size=1)

   # Iterate over all the images in path
   for image in os.listdir(path):
      # Read and convert the images to grayscale
      img = convert_to_grayscale(imread(os.path.join(path,image)))
```

```
        # Compute hog features
        fd = hog_obj(img)

        # Add hog features to matrix
        matrix.append(fd)

        # Add image name to image list
        image_list.append(image)


    # Return matrix, image list
    return np.array(matrix), image_list

def create_dataset(pos_path, neg_path):
    '''
    Args:
        pos_path: The image directory containing positive images
        neg_path: The image directory containing negative images
    Returns:
        X: Features of all the images
        y: Labels of all images

    '''
    # Compute hog features
    X_pos, x_pos_names = load_data(pos_path)

    # Assign label 1 for positive images
    y_pos = np.ones((X_pos.shape[0]))

    # Compute hog features
    X_neg, x_neg_names = load_data(neg_path)

    # Assign label 0 for negative images
    y_neg = np.zeros((X_neg.shape[0]))

    # Concatenate all the features/ lables / names
    X = np.concatenate((X_pos, X_neg))
    y = np.concatenate((y_pos, y_neg))
    image_list = x_pos_names + x_neg_names

    # Return the labels
    return X, y, image_list
```

```python
if __name__ == "__main__":
    TRAIN_DIR_POS = "data/Training images (Pos)"
    TRAIN_DIR_NEG = "data/Training images (Neg)"
    X_train, y_train, image_list = create_dataset(TRAIN_DIR_POS, TRAIN_DIR_NEG)
    print(X_train.shape, y_train.shape)
    print(*image_list, sep = '\n')
```

gradient_operation.py

```python
'''
Computer Vision Final Project
Project group members:
    1. Aniket Bote (N12824308)
    2. Sindhu Harish (N19806874)
'''
# Import the required libraries
import numpy as np
from utils import Operator, apply_discrete_convolution

def perform_gradient_operation(image):
    '''
    Args:
        image : An image on which gradient operation will happen
    Returns:
        Magnitude : Magnitude of the gradient
        Theta     : Gradient Angle
    '''
    # Compute horizontal gradients
    dfdx = apply_discrete_convolution(image, Operator.gx)

    # Compute vertical gradients
    dfdy = apply_discrete_convolution(image, Operator.gy)

    # Compute magnitude of the gradient
    m = np.sqrt(np.square(dfdx) + np.square(dfdy))

    # Normalize gradient magnitude and set zero where the operator goes beyond the border
    maximum_gradient_magnitude = np.sqrt((3 * 255.0)**2 + (3 * 255.0)**2)
    m = np.nan_to_num((np.absolute(m) / maximum_gradient_magnitude) * 255.0)

    # Compute gradient angle, convert the range from [-180, 180] --> [0, 360] --> [0,180]
    theta = ((np.nan_to_num(np.degrees(np.arctan2(dfdy, dfdx))) + 360) % 360) % 180
```

```
 return np.round(m), theta

if __name__ == "__main__":
   from skimage.io import imread
   from grayscale import convert_to_grayscale
   # image =convert_to_grayscale(imread("data\Training images (Pos)\crop_000010b.bmp"))
   np.random.seed(10)
   image = np.random.randint(0,255, (16,8))
   print(image) # 1,1 -- gx = 282 gy = 88 norm = 69.62 --> 70  1,2 -- gx = -162 gy = -152 norm = 
52.35 --> 52 angle = 43.17
   print("****")
   mag , ang = perform_gradient_operation(image)
   print(mag)
   print("&&&&")
   print(ang)
```

grayscale.py

```
'''
Computer Vision Final Project
Project group members:
   1. Aniket Bote (N12824308)
   2. Sindhu Harish (N19806874)
'''

import numpy as np

def convert_to_grayscale(image):
   '''
      Converts colored image to grayscale
      Args:
         image: input is color image
      Return:
         image: grayscale converted image
   '''
   # Segregate R,G,B channels
   R, G, B = image[:,:,0], image[:,:,1], image[:,:,2]
   # Apply grayscale conversion to R, G, B channel
   imgGray = np.round(0.299 * R + 0.587 * G + 0.114 * B)
   return  imgGray

if __name__ == "__main__":
   from skimage.io import imread
```

```
 image =imread("data\Training images (Pos)\crop_000010b.bmp")
   print(convert_to_grayscale(image))
```

hog.py

```
'''
Computer Vision Final Project
Project group members:
   1. Aniket Bote (N12824308)
   2. Sindhu Harish (N19806874)
'''
import numpy as np
from gradient_operation import perform_gradient_operation

class HOG:
   def __init__(self, n_bins, cell_size, block_size, step_size, max_m = 180):
      '''
      Initialize the HOG class
      Args:
         n_bins: Number of bins
         cell_size: A tuple containing cell size in pixels
         block_size: A tuple containing block size using cells
         step_size: The step to take in order to create overlapping blocks (Using cells)
         max_m: Maximum allowed angle
      '''
      self.n_bins = n_bins
      self.bin_range = max_m / n_bins
      self.cell_size = cell_size
      self.block_size = block_size
      self.step_size = step_size

   def __call__(self, img):
      '''
      Function that returns hog descriptor
      Args:
         img: The grayscale image
      Returns:
         HOG descriptor array
      '''
      # Computes gradient magnitude and gradient angle
```

```
        gradient_magnitude, gradient_angle = perform_gradient_operation(img)
        return self.compute_hog_features(gradient_magnitude, gradient_angle)

    def get_bins_and_fraction(self, g):
        '''
        Function to compute the bin numbers and fraction of magintude that goes into 
respectective bin
        Args:
            g: Gradient angle
        Returns:
            bin_i: Left bin number
            bin_j: Right bin number
            fraction_i = Fraction of magnitude that goes into left bin
            fraction_j = Fraction of magnitude that goes into right bin
        '''
        # Covert the gradient angle range from 0-180 --> 10 - 190 and divide the angle by 20 to 
efficintly calculate the right bin. j ranges from 0-9
        # This is done to compute the fraction with ease
        j = int(np.floor((g + self.bin_range/2) / self.bin_range))
        # Left bin --> Right bin - 1. i ranges from -1 - 8
        i = int(j - 1)
        # Fraction for left bin --> distance of g from right bin center/ bin range
        fraction_i = ((self.bin_range * j + self.bin_range/2) - g) / self.bin_range
        # Fraction for right bin --> distance of g from left bin center/ bin range
        fraction_j = (g - (self.bin_range * i + self.bin_range/2)) / self.bin_range
        # Use modulo operator to compute the true bin value
        bin_j = j  % self.n_bins
        # Use modulo operator to compute the true bin value
        bin_i = (bin_j - 1) % self.n_bins
        return bin_i, bin_j, fraction_i, fraction_j

    def l2_normalize(self, vector, epsilon=1e-5):
        '''
        Performs L2 normalization of vector

        Args:
            vector: hog block vector
            epsilon: epsilon to avoid exception

        Returns:
            Normalized vector
        '''
        return vector / np.sqrt(np.sum(vector ** 2) + epsilon ** 2)
```

```
    def compute_hog_cell(self, m_cell, g_cell):
        '''
        Function to compute hog features for a cell
        Args:
            m_cell: Cell containing maginitude
            g_cell: Cell containing gradient angle
        Returns:
            Hog feature array for a cell
        '''
        # Initialize empty hog array
        hog_values = np.zeros((self.n_bins))

        # Iterate through range of all magnitude and gradient angles
        for i in range(m_cell.shape[0]):
            for j in range(m_cell.shape[1]):
                # Get the bins and fractions for gradient value
                bin_1, bin_2, fraction_1, fraction_2 = self.get_bins_and_fraction(g_cell[i][j])
                # Add the magnitude to respective bins based of fractions
                hog_values[bin_1] = hog_values[bin_1] + m_cell[i][j] * fraction_1
                # Add the magnitude to respective bins based of fractions
                hog_values[bin_2] = hog_values[bin_2] + m_cell[i][j] * fraction_2
        return hog_values

    def compute_hog_features(self, gradient_magnitude, gradient_angle):
        '''
        Function to compute hog features
        Args:
            gradient_magnitude: The gradient magnitude
            gradient_angle: The gradient angles
        '''
        # compute height and width of image
        height, width = gradient_magnitude.shape
        # compute number of cells in the image
        n_cells_x, n_cells_y =  int(height / self.cell_size[0]), int(width / self.cell_size[1])
        # compute numbr of blocks in the image
        n_blocks_x, n_blocks_y = int(((n_cells_x - self.block_size[0]) / self.step_size) + 1),
int(((n_cells_y - self.block_size[1]) / self.step_size) + 1)

        # Initialize zeros for all cell
        hog_cells = np.zeros((n_cells_x, n_cells_y, self.n_bins))
        # Iterate over the range of all cells in image
        for x in range(n_cells_x):
            for y in range(n_cells_y):
                # Get the cell for gradient magnitude
```

```
            m_block = gradient_magnitude[x * self.cell_size[0]: (x + 1) * self.cell_size[0], y *
self.cell_size[1]: (y + 1) * self.cell_size[1]]
            # Get the cell for gradient angle
            g_block = gradient_angle[x * self.cell_size[0]: (x + 1) * self.cell_size[0], y *
self.cell_size[1]: (y + 1) * self.cell_size[1]]
            # Compute hog features for cell
            hog_values_cell = self.compute_hog_cell(m_block, g_block)
            # Assign the hog features to respective cell
            hog_cells[x, y] = hog_values_cell

        # Initialize empty hog descriptor
        hog_descriptor = []
        # Iterate over the range of the blocks
        for x in range(n_blocks_x):
          for y in range(n_blocks_y):
            # Get the hog features for all the cells included in 1 block
            block = hog_cells[x : x + self.block_size[0], y : y + self.block_size[1]]
            # Flatten the hog features into vector, apply L2 normalization and append into the hog
descriptor
            hog_descriptor += list(self.l2_normalize(block.ravel()))
        # Return the hog descriptor
        return np.array(hog_descriptor)



if __name__ == "__main__":
  np.random.seed(10)
  print("Test for get bins and fractions")
  h = HOG(9, (8,8), (2,2), 1, 180)
  GA = list(range(0,181,5))
  for a in GA:
    i, j, fi, fj = h.get_bins_and_fraction(a)
    print( f"{a} : {i, j} Bin centers: {h.bin_range * i  + h.bin_range / 2} & {h.bin_range * j  +
h.bin_range / 2} Fraction : {fi, fj}")

  # HW example 4a:
  # Answer = 29160
  print("\nTest for shape")
  h = HOG(18, (8,8), (3,3), 2, 360)
  image = np.random.randint(0,255, (296, 168))
  print(h(image).shape)

  # HW example 4b: Note the bin centers are different in HW
  # Answer [ 45.  55. 165.   0.   0.   0.   0.   0.   0.  30.  90.   0.   0.   0.   0.   0.   0. 135.]
```

```
    print("\nTest for hog cell values")
    h = HOG(18, (8,8), (1,1), 1, 360)
    GA = np.array([
        [200, 45, 23, 98, 130, 260, 255, 250],
        [125, 295, 85, 90, 130, 265, 249, 240],
        [123, 35, 85, 95, 125, 260, 250, 240],
        [100, 90, 45, 90, 120, 265, 240, 230],
        [95, 99, 105, 106, 355, 120, 100 ,110],
        [90, 205, 110, 120, 120, 130, 125, 120],
        [85, 90, 100, 110, 110, 120, 120, 110],
        [80, 80, 100, 110, 100, 100, 100, 110]])

    M = np.array([
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 220, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 180, 0, 0, 0],
        [0, 120, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0]])

    print(h.compute_hog_cell(M, GA))

    # Answer = [0.63496502 0.08945307 0.02757537 0.         0.         0.03941083 0.02796246
  0.10478557 0.75811744]
    print("\nTest with sample image array")
    h = HOG(9, (8,8), (1,1), 1, 180)
    image = np.array([
        [120, 125, 212, 239, 120, 125, 212, 239],
        [90, 100, 180, 200, 120, 125, 212, 239],
        [85, 195, 200, 210, 120, 125, 212, 239],
        [75, 212, 255, 195, 120, 125, 212, 239],
        [120, 125, 212, 239, 120, 125, 212, 239],
        [90, 100, 180, 200, 120, 125, 212, 239],
        [85, 195, 200, 210, 120, 125, 212, 239],
        [75, 212, 255, 195, 120, 125, 212, 239]
    ])
    print(h(image))

    # Answer = 7524
    print("\nTest with sample image array with original dimension")
    h = HOG(9, (8,8), (2,2), 1, 180)
    image = np.random.randint(0,255, (160, 96))
```

```python
    print(h(image).shape)

    #Answer = 19584
    print("\nTest with sample image array with uneven cell size and block size")
    h = HOG(9, (4,6), (8,2), 2, 180)
    image = np.random.randint(0,255, (160, 96))
    print(h(image).shape)
```

Knn.py

```python
'''
Computer Vision Final Project
Project group members:
    1. Aniket Bote (N12824308)
    2. Sindhu Harish (N19806874)
'''

import numpy as np
from collections import Counter

class KNNClassifier:
    '''
    Performs K-nearest neighbour classification using histogram intersection to compute
similarity

    Args:
        k: The number of neighbours to consider
        X: The training features ie hog features
        y: The training labels
    '''
    def __init__(self, k, X, y):
        # Initialize all the parameters
        self.k = k
        self.X = X
        self.y = y
        self.n = X.shape[0]

    def compute_histogram_overlap(self, hist_values):
        '''
        Computes histogram overlap using formula:
        overlap = sum(min(I, M)) / sum(M)
        Args:
```

```
            hist_values: A single array consisting of values of histogram bins ie hog features
        Returns:
            histogram overlap between hist_values and all training features
        '''
        return np.sum(np.minimum(hist_values, self.X), axis = 1) / np.sum(self.X, axis = 1)

    def predict(self, X_test):
        '''
        Computes predictions
        Args:
            X_test: The testing features ie. hog features
        Return:
            preds: Predictions of testing features
            topk: list of tuple containing label, overlap value, index of top k features
        '''
        # Initialize empty array preds to store predictions, k-nearest neighbours
        preds = []
        topk = []


        # Iterate over test features
        for i in range(X_test.shape[0]):
            # Compute histogram overlap
            histogram_overlap = self.compute_histogram_overlap(X_test[i])

            # Create list of tuple containing label, overlap value, index ie [(1, 0.77, 0), (0, 0.55, 1), (1,
0.88, 2), (0, 0.99, 3)]
            similarity = list(zip(self.y, histogram_overlap, range(self.n)))

            # Sort the list of tuples according to overlap values ie [(0, 0.99, 3), (1, 0.88, 2), (1, 0.77, 0),
(0, 0.55, 1)]
            similarity = sorted(similarity, key = lambda x: x[1], reverse = True)

            # Take the count the labels in top k tuples and sort it according to highest countie [(1, 2),
(0,1)]
            k_predictions = sorted(Counter([label[0] for label in similarity[:self.k]]).items(), key =
lambda x : x[1], reverse = True)

            # Add the label of highest count to prediction array
            preds.append(k_predictions[0][0])

            # Add top k neighbours to topk array
            topk.append(similarity[:self.k])
```

```
        # Return prediction array & topk array
        return np.array(preds), topk


if __name__ == "__main__":
    X_train = np.array([
        [1,2,3,4,5,2,4,7,3],
        [7,2,1,3,7,3,9,1,4],
        [3,2,7,4,8,3,1,1,2],
        [9,9,9,9,9,9,9,9,9],
        [4,8,3,5,9,1,2,3,4]
    ])

    y_train = np.array([1,1,0,0,1])

    X_test = np.array([
        [3,7,8,4,6,1,1,9,0]
    ])

    knn_model = KNNClassifier(3, X_train, y_train)

    print(*zip(knn_model.compute_histogram_overlap(X_test[0]), y_train), sep="\n")

    print(knn_model.predict(X_test))
```

output.py

```
'''
Computer Vision Final Project
Project group members:
    1. Aniket Bote (N12824308)
    2. Sindhu Harish (N19806874)
'''
import os
from numpy import uint8
import pandas as pd
from skimage.io import imread, imsave

from grayscale import convert_to_grayscale
from gradient_operation import perform_gradient_operation

def generate_table(y_true, y_pred, topk, train_image_list, test_image_list):
    '''
```

```
    Generate classification report
    Args:
        y_true: True labels
        y_pred: Predicted labels
        topk: Top k neighbours
        train_image_list: Names of images in training set
        test_image_list: Names of images in testing set
    Returns:
        df: The dataframe containing classification report
    '''
    # Labelmap to map integer values to string labels
    label_map = {0:"No-human", 1:"Human"}

    # Store the number of neighbours
    k = len(topk[0])

    # Initialize the dataframe
    df = pd.DataFrame()

    # Add names of test images in dataframe
    df['Test image'] = test_image_list

    # Add labels of test images in dataframe
    df['Correct Classification'] = list(map(lambda x: label_map[x], y_true))

    # Iterate over lenght of k
    for i in range(k):
        # Initialize empty column
        col = []
        # Iterate over all rows in topk
        for row in topk:
            # For each row in topk list add the ith neighbours's name, overlap value & label to
column
            col.append(f"{train_image_list[row[i][2]]}, {row[i][1]}, {label_map[row[i][0]]}")
        # Add the column in datatframe
        df[f'File name of {i+1} NN, distance & classification'] = col

    # Add the predicted labels in dataframe
    df[f'Classification from {k}-NN'] = list(map(lambda x: label_map[x], y_pred))

    # Return the dataframe
    return df

def save_normalized_images(pos_dir, neg_dir, save_dir):
```

```
    '''
    Function to save the normalized images
    Args:
       pos_dir: Path of positive directory
       neg_dir: Path of negative directory
       save_dir: Path to save the normalized images
    '''
    # Iterate over all the images in pos dir
    for image in os.listdir(pos_dir):
       # Read and convert the images to grayscale
       img = convert_to_grayscale(imread(os.path.join(pos_dir,image)))
       # Compute the gradient magnitude
       gradient_magnitude, gradient_angle = perform_gradient_operation(img)
       # Save the images
       imsave(os.path.join(save_dir,os.path.basename(image)),
gradient_magnitude.astype(uint8))

    # Iterate over all the images in neg dir
    for image in os.listdir(neg_dir):
       # Read and convert the images to grayscale
       img = convert_to_grayscale(imread(os.path.join(neg_dir,image)))
       # Compute the gradient magnitude
       gradient_magnitude, gradient_angle = perform_gradient_operation(img)
       # Save the images
       imsave(os.path.join(save_dir,os.path.basename(image)),
gradient_magnitude.astype(uint8))

if __name__ == "__main__":
    y_pred = [0,1]
    y_true = [0,1]
    topk = [
       [(0, 0.8064516129032258, 2), (1, 0.7741935483870968, 0), (1, 0.717948717948718, 3)],
       [(1, 0.9064516129032258, 2), (1, 0.8741935483870968, 0), (1, 0.797948717948718, 3)]
    ]
    image_list = ["image0", "image1", "image2", "image3"]
    test_image_list = ["test0", "test1"]

    out_df = generate_table(y_true, y_pred, topk, image_list, test_image_list)
    out_df.to_csv("test.csv", index=False)
    TEST_DIR_POS = "data/Test images (Pos)"
    TEST_DIR_NEG = "data/Test images (Neg)"
    IMAGE_SAVE_PATH = "Output/normalized_images"
    save_normalized_images(TEST_DIR_POS, TEST_DIR_NEG, IMAGE_SAVE_PATH)
```

utils.py

```
'''
Computer Vision Final Project
Project group members:
   1. Aniket Bote (N12824308)
   2. Sindhu Harish (N19806874)
'''

import numpy as np

# A class to store all operators
class Operator:
   # Prewitt operator for Gx
   gx = np.array([
      [-1,0,1],
      [-1,0,1],
      [-1,0,1]])

   # Prewitt operator for Gy
   gy = np.array([
      [1,1,1],
      [0,0,0],
      [-1,-1,-1]])

# A function to apply dicreet convolutions
def apply_discrete_convolution(image, mask):
   '''
   Args:
      image : An image to use for convolution
      mask  : An mask to use for convolution
   Returns:
      convolved image: An image after convolution
   '''
   # Get the shape of image and mask
   (m_image, n_image), (m_mask, n_mask) = image.shape, mask.shape

   # Compute the reference pixel index from where output array will start populating
   rpi_m, rpi_n = int(np.floor(m_mask/2)), int(np.floor(n_mask/2))

   # Initialize an output array with nan values
   output_arr = np.ones((m_image, n_image)) * np.nan
```

```
# Iterate through the image
for i in range(m_image - m_mask + 1):
   for j in range(n_image - n_mask + 1):
      # Isolate the image slice to apply convolution
      img_slice = image[i:i+m_mask, j:j+n_mask]
      # Apply convolution and store the result in output array in approriate location
      output_arr[i+rpi_m][j+rpi_n] = np.sum(img_slice * mask)

return output_arr
```

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, 0.668339065005568, Human	00000053a_cut.bmp, 0.6472714536932969, No-human	01-03e_cut.bmp 0.6439597593391659 No-human	No-human
crop001070a	Human	00000053a_cut.bmp, 0.4979882850142865, No-human	person_and_bike _026a.bmp, 0.494905151056698, Human	crop001672b.bmp, 0.4946569915403421, Human	Human
crop001278a	Human	crop001672b.bmp, 0.597553599821817, Human	crop001008b.bmp, 0.5912684956109407, Human	crop001275b.bmp, 0.5840487842909872, Human	Human
crop001500b	Human	crop001672b.bmp, 0.5660670507320885, Human	00000091a_cut.bmp, 0.5597390506601627, No-human	crop001275b.bmp, 0.5440752461621162, Human	Human
person_and_bike_151a	Human	crop001030c.bmp, 0.5049327450611067, Human	person_and_ bike_026a.bmp, 0.5020791810152739, Human	crop001275b.bmp, 0.49437581506442724, Human	Human
00000003a_cut	No-human	00000053a_cut.bmp, 0.5766938985412988, No-human	crop001672b.bmp, 0.5738705403608999, Human	00000093a_cut.bmp, 0.5487835451443858, No-human	No-human
00000090a_cut	No-human	00000093a_cut.bmp, 0.4749754538506958, No-human	00000057a_cut.bmp, 0.47282243099691246, No-human	crop001672b.bmp, 0.44579061749349985, Human	No-human
00000118a_cut	No-human	00000093a_cut.bmp, 0.5629785480106466, No-human	00000053a_cut.bmp, 0.5557016202374616, No-human	00000091a_cut.bmp, 0.5494215984621353, No-human	No-human
no_person_no_bike_258_cut	No-human	00000057a_cut.bmp, 0.4965887665755111, No-human	crop001672b.bmp, 0.48843566291291035, Human	crop001275b.bmp, 0.4841667086640254, Human	Human

no_person_no_bike_264_cut	No-human	00000053a_cut.bmp, 0.44246634722363226, No-human	crop001672b.bmp, 0.43958179016221516, Human	crop001030c.bmp, 0.4349972791615372, Human	Human