Histogram of Oriented Gradients: Human Detection

Pragnavi Ravuluri Sai Durga
Email: pr2370@nyu.edu

Santosh Srinivas
Email: sr6411@nyu.edu

Introduction

The Histograms of Oriented Gradients (HOG) feature is a widely used technique in computer vision for detecting objects in images. This project involves implementing a program to compute the HOG feature from an input image and classify it into human or no-human by using a 3-nearest neighbor (NN) classifier. The classification is done using two distance metrics, histogram intersection and Hellinger distance. The program will also convert the input color image into grayscale and use Sobel's operator to compute the horizontal and vertical gradients, which are used to obtain the HOG feature vector.

Objective

The objective of this project is to develop a program that can accurately detect the presence of humans in images using the HOG feature and a 3-NN classifier. The program will convert color images into grayscale, compute the horizontal and vertical gradients using Sobel's operator, and quantize the gradient angle into 9 bins. The resulting HOG feature vector will be classified as human or no-human using the histogram intersection and Hellinger distance metrics. The classification accuracy for each distance metric will be reported. The program will be implemented in Python.

Instructions: How to Execute the Program

The following are the instructions for executing the HOG Human Detector program:

System Requirements:

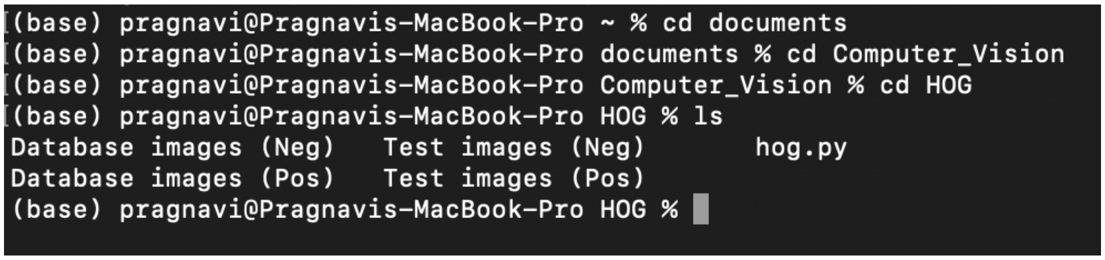
It is essential to ensure that Python3 version 3.8 and the required packages, namely NumPy, Matplotlib, and OpenCV, are installed in the local machine before running the program. In case any of the packages are not installed, execute the following commands on the terminal:

- Python3 –version (Check if installed Python3 is at least version 3.8)
- NumPy - pip3 install numpy
- Matplotlib - pip3 install matplotlib
- OpenCV - pip3 install opencv-python

After installing the packages, the program is ready for execution.

Executing the Program:

First, save the source code file hog.py and the four image folders, positive and negative database and test images, in the same directory folder. Then, navigate to the directory in the terminal and execute the



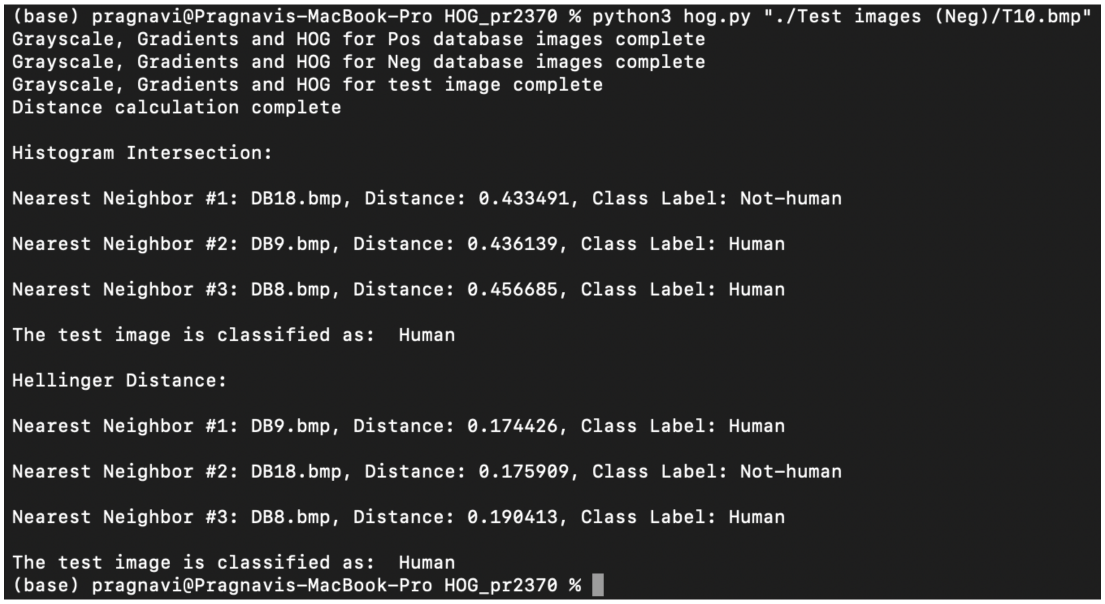
```
(base) pragnavi@Pragnavis-MacBook-Pro ~ % cd documents
(base) pragnavi@Pragnavis-MacBook-Pro documents % cd Computer_Vision
(base) pragnavi@Pragnavis-MacBook-Pro Computer_Vision % cd HOG
(base) pragnavi@Pragnavis-MacBook-Pro HOG % ls
Database images (Neg)   Test images (Neg)       hog.py
Database images (Pos)   Test images (Pos)
(base) pragnavi@Pragnavis-MacBook-Pro HOG %
```

Figure 1: Folder Structure

following command for each of the test images, passing the .py file and the test image as arguments:

- python3 hog.py "./Test images (Pos)/T1.bmp"
- python3 hog.py "./Test images (Pos)/T2.bmp"
- python3 hog.py "./Test images (Pos)/T3.bmp"
- python3 hog.py "./Test images (Pos)/T4.bmp"
- python3 hog.py "./Test images (Pos)/T5.bmp"
- python3 hog.py "./Test images (Neg)/T6.bmp"
- python3 hog.py "./Test images (Neg)/T7.bmp"
- python3 hog.py "./Test images (Neg)/T8.bmp"
- python3 hog.py "./Test images (Neg)/T9.bmp"
- python3 hog.py "./Test images (Neg)/T10.bmp"

The classification output is displayed on the console of the terminal where the program hog.py is run. A sample output looks as follows: The output normalized gradient images of the input test image and the



```
(base) pragnavi@Pragnavis-MacBook-Pro HOG_pr2370 % python3 hog.py "./Test images (Neg)/T10.bmp"
Grayscale, Gradients and HOG for Pos database images complete
Grayscale, Gradients and HOG for Neg database images complete
Grayscale, Gradients and HOG for test image complete
Distance calculation complete

Histogram Intersection:

Nearest Neighbor #1: DB18.bmp, Distance: 0.433491, Class Label: Not-human

Nearest Neighbor #2: DB9.bmp, Distance: 0.436139, Class Label: Human

Nearest Neighbor #3: DB8.bmp, Distance: 0.456685, Class Label: Human

The test image is classified as:  Human

Hellinger Distance:

Nearest Neighbor #1: DB9.bmp, Distance: 0.174426, Class Label: Human

Nearest Neighbor #2: DB18.bmp, Distance: 0.175909, Class Label: Not-human

Nearest Neighbor #3: DB8.bmp, Distance: 0.190413, Class Label: Human

The test image is classified as:  Human
(base) pragnavi@Pragnavis-MacBook-Pro HOG_pr2370 %
```

Figure 2: Sample Output of hog.py on Test Image - T10.bmp

database images are stored in a new named folder Gradient_Magnitude_Images under the same directory. Similarly the output ASCII txt files of the input test image and the database images are stored in a new folder named ASCII_Files under the same directory.

Source Code

```python
'''
The following program performs human detection of a given input image
by computing the HOG (Histograms of Oriented Gradients) feature from
the input image and then classify the HOG feature vector into human or no-human by using a 3-nearest neighbor (NN) classifier.

Following are the steps:
# 1. convert input image to grayscale
# 2. compute gradient magnitudes and gradient angle (Sobel)
# 3. compute HOG features on database images
# 4. compute HOG features on test image
# 5. calculate distance between test and database images
# 6. use 3NN to classify

Submitted by:
Pragnavi Ravuluri Sai Durga ( pr2370 )
Santosh Srinivas Ravichandran (sr6411)

'''


# Import necessary libraries
from PIL import Image #for image processing
import numpy as np  #for numerical computations
import math   #for mathematical operations
import sys  #for system-level operations
import os #using operating system dependent functionality


#This function takes the file path of the input image, converts it to grayscale and
# returns the resulting grayscale image

def convert_to_grayscale(image_path):

  #load image from image path
  img = Image.open(image_path)

  # Get the dimensions of the image
  width, height = img.size

  # Create a new grayscale image
  gray_img = Image.new('L', (width, height))

  # Loop through each pixel in the image and convert to grayscale
  for x in range(width):
      for y in range(height):
          # Get the RGB values of the current pixel
          r, g, b = img.getpixel((x, y))

          # Convert each channel to grayscale using the formula ð  ¼ = round(0.299ð    + 0.587ð  º + 0.114ð  µ)
          gray_value = round(0.299*r + 0.587*g + 0.114*b)

          # Set the grayscale value of the current pixel in the new grayscale image
          gray_img.putpixel((x, y), gray_value)

  return gray_img

# This function applies the Sobel operator to compute gradient magnitude and gradient angle of a given image array.
# it returns the normalised gradient magnitude and gradient angle array.

def Gradient_Operator(img):

  # Get the dimensions of the image
  img_h, img_w = img.shape

  #defining Sobel Operator
  sobel_operator_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
  sobel_operator_y= np.array([[1,2,1],[0,0,0],[-1,-2,-1]])

  # defining size of the mask from Sobel kernel
  mask_size = sobel_operator_x.shape[0]

  # Set buffer size for boundary pixels
  buffer = mask_size // 2

  # Initialize 2-D array for holding Gradient magnitude and Gradient angle
  g_x = np.zeros((img_h,img_w))
  g_y = np.zeros((img_h,img_w))
  gradient_magnitude = np.zeros((img_h,img_w), dtype=np.float32)
  gradient_angle = np.zeros((img_h,img_w), dtype=np.float32)

  # Apply gradient operation using predefined gradient masks across the image
  for i in range(0+buffer,img_h-buffer,1):
        for j in range(0+buffer,img_w-buffer,1):

          # Convolution operation using Sobel mask to compute gradient magnitude
          g_x[i][j] = np.sum(np.multiply(sobel_operator_x,img[i-buffer:i+buffer+1,j-buffer:j+buffer+1]))
          g_y[i][j] = np.sum(np.multiply(sobel_operator_y,img[i-buffer:i+buffer+1,j-buffer:j+buffer+1]))
          gradient_magnitude[i][j] = round(math.sqrt(g_x[i][j]**2 + g_y[i][j]**2))

          # get gradient angle in radians
          theta = np.arctan2(g_y[i][j], g_x[i][j])
          # convert it to degrees
          theta_deg = np.degrees(theta)
          # bring it with in the scale of o to 360
          theta_deg_mod = np.mod(theta_deg, 360)
          #Round off gradient angle
          gradient_angle[i][j]= round(theta_deg_mod)

  # normalise gradient magnitude values to bring it to scale of 0 to 255
  g_x = g_x/4
  g_y = g_y/4
  gradient_magnitude = np.sqrt(g_x**2+g_y**2)/ np.sqrt(2)
  # return gradient magnitude and gradient angle arrays
  return gradient_magnitude,gradient_angle

# This function takes as input the gradient magnitude and gradient angle array of a given cell (8*8 pixels).
#It return the HOG vector the cell ( un-normalised ).

def HOG_vector_cell(gradient_magnitude_cell,gradient_angle_cell):

  # defining bin center values in an array
  bin_center  = np.array([10.0, 30.0, 50.0, 70.0, 90.0, 110.0, 130.0, 150.0, 170.0])
  # defining the histogram bin in a dictionary where key in the bin the center and value is gradient magnitude for that bin
  bin_hist = {i: np.array([]) for i in bin_center}
  split_left = None
  split_right = None


  #looping across every cell element
  for i in range(0,8):
    for j in range(0,8):

      # scale gradient angle to its 180 degree equivalent
      if(gradient_angle_cell[i][j]>=180):
        gradient_angle_cell[i][j]-=180

      #find the left bin_center and left split angle - eg for 45 degree gradient angle lying between 30 and 50 bin centers;
      # 30 would be split left and bin left.
      try:
          split_left = np.max(bin_center[bin_center <= gradient_angle_cell[i][j]])
          bin_left = split_left
      except ValueError:
          # if split_left not found during edge cases when bins wrap
          split_left = None

      #find right bin_center and right split angle - eg for 45 degree gradient angle lying between 30 and 50 bin centers;
      # 50 would be split right and bin right.
      try:
          split_right = np.min(bin_center[bin_center > gradient_angle_cell[i][j]])
          bin_right = split_right
      except ValueError:
          # if split_right not found during edge cases when bins wrap
          split_right = None

      # for edge cases when bins wrap
      if(split_left==None):
        split_left = 170-180
        bin_left = 170.0
      elif(split_right==None):
        split_right=10+180
        bin_right=10.0

      # calculate the propotional gradient magnitudes and append them to the appropriate bins in histogram dictionary
      gradient_left = 1-(gradient_angle_cell[i][j]-split_left)/20
      gradient_right = 1-gradient_left
      x = gradient_magnitude_cell[i][j]*gradient_left
      y = gradient_magnitude_cell[i][j]*gradient_right

      bin_hist[bin_left] = np.append(bin_hist[bin_left], x)
      bin_hist[bin_right] = np.append(bin_hist[bin_right], y )



  # sum all the appended gradient magnitudes to get total
  for key, value in bin_hist.items():
    bin_hist[key] = np.sum(value)

  #get hog cell vector
  hog_cell_vector = np.array(list(bin_hist.values()))

  return(hog_cell_vector)


# This function takes as input the gradient magnitude and gradient angle array of a given image (96*160)  and
# performs HOG feature computations to return the final normalised HOG vector for the image.

def normalized_HOG_vector(gradient_magnitude,gradient_angle):

  # Initialise the hog vector of the image
  final_hog_vector = np.array([])

  #loop across the image dividing into blocks ; image size is 96*160 so i varies till 152 and j varies till 88 with step of 8 for block overlap
  for i in range(0,152,8):
    for j in range(0,88,8):

      #get the gradient magnitude and angle for each block ; block size is 16*16
      gradient_magnitude_block = gradient_magnitude[i:i+16,j:j+16]
      gradient_angle_block = gradient_angle[i:i+16,j:j+16]

      # initialise hog vector for block to empty
      hog_block_vector = np.array([])

      #Divide gradient magnitde block into 4 cells - top left quadrant , top right quadrant, bottom left quadrant, bottom right quadrant
      gradient_magnitude_top_left_cell = gradient_magnitude_block[:8, :8]
      gradient_magnitude_top_right_cell = gradient_magnitude_block[:8, 8:]
      gradient_magnitude_bottom_left_cell = gradient_magnitude_block[8:, :8]
      gradient_magnitude_bottom_right_cell = gradient_magnitude_block[8:, 8:]


      #Divide gradient angle block into 4 cells - top left quadrant , top right quadrant, bottom left quadrant, bottom right quadrant
      gradient_angle_top_left_cell = gradient_angle_block[:8, :8]
      gradient_angle_top_right_cell = gradient_angle_block[:8, 8:]
      gradient_angle_bottom_left_cell = gradient_angle_block[8:, :8]
      gradient_angle_bottom_right_cell = gradient_angle_block[8:, 8:]
      # get hog vector for all 4 cells -  top left quadrant , top right quadrant, bottom left quadrant, bottom right quadrant
      top_left_cell_vector =  HOG_vector_cell(gradient_magnitude_top_left_cell,gradient_angle_top_left_cell)
      top_right_cell_vector =  HOG_vector_cell(gradient_magnitude_top_right_cell,gradient_angle_top_right_cell)
      bottom_left_cell_vector =  HOG_vector_cell(gradient_magnitude_bottom_left_cell,gradient_angle_bottom_left_cell)
      bottom_right_cell_vector =  HOG_vector_cell(gradient_magnitude_bottom_right_cell,gradient_angle_bottom_right_cell)


      # Combine the 4 cell vectors to form the block vector
      hog_block_vector = np.concatenate((top_left_cell_vector,top_right_cell_vector,bottom_left_cell_vector,bottom_right_cell_vector))

      # Computing the value of L2 norm
      L2_norm = np.sqrt(np.sum(np.square(hog_block_vector)))

      # Normalising the hog block vector
      if L2_norm == 0:
          # handle the zero norm case (e.g., set normalized array to zero)
          normalized_hog_block_vector = np.zeros_like(hog_block_vector)
      else:
         normalized_hog_block_vector = hog_block_vector / L2_norm


      # Get the full hog vector for the entire image by appending each normalised block vector
      final_hog_vector = np.append(final_hog_vector, normalized_hog_block_vector)

  # Normalise the final hog vector
  final_hog_vector = final_hog_vector / np.sum(final_hog_vector)

  # Return the final normalised hog vector
  return final_hog_vector

# This function computes the histogram intersection and hellinger distance between input test image and all database images
# and returns the distances.

def least_distance(database_HOG,test_HOG):

  # Initialising histogram intersection and hellinger distance lists
  hist_inters = []
  hellinger_distance = []

  # computing the distance between every database HOG and test HOG
  for row in database_HOG:
      # using histogram intersection condn: 1-sum(min(d,t))
      hist = np.minimum(row, test_HOG)
      hist_inters.append(1-np.sum(hist))

      # using hellinger distance condn: 1-sum(sqrt(d*t))
      hellinger = 1- np.sum(np.sqrt(row * test_HOG))
      hellinger_distance.append(hellinger)

  # converting Python Lists to Numpy arrays
  hist_inters = np.array(hist_inters)
  hellinger_distance = np.array(hellinger_distance)

  # returning both distance arrays
  return hist_inters, hellinger_distance


### This function takes as input the histogram intersection/ hellinger distance array and computes the 3NN.
# It then returns the final classification result taking majority vote class of the 3NNS

def threeNN(similarity, n_neg_database):

    #Number of negative classification
    neg = 0
    #Number of positive classification
    pos = 0
    NN_count = 1

    # sort the distance arrays to find the indices of the 3 NNs (adding + 1 to match DB names: indice 0 is DB1 etc)
    indices = np.argsort(similarity)[:3]+1

    # use indices and distance array to display each neighbour , the distance between and the class of the neighbour
    for i in indices:
        if i < n_pos_database:
            pos += 1
            print("\nNearest Neighbor #%d: %s, Distance: %f, Class Label: Human" % (NN_count, pos_database_files[i-1], similarity[i-1]))
        else:
            neg +=1
            print("\nNearest Neighbor #%d: %s, Distance: %f, Class Label: Not-human" % (NN_count, neg_database_files[(i-1)%n_pos_database], similarity[i-1]))
        NN_count += 1

    # take maximum vote of class neighbour to get the final classification
    if neg > pos:
        classification = 'Not human'
    else:
        classification = 'Human'

    # return the classification result
    return classification


# main
# ensure proper arguments given i.e.
# python3 hog.py './Test images (Pos)/T1.bmp'
# python3 hog.py './Test images (Pos)/T2.bmp'
# python3 hog.py './Test images (Pos)/T3.bmp'
# python3 hog.py './Test images (Pos)/T4.bmp'
# python3 hog.py './Test images (Pos)/T5.bmp'
# python3 hog.py './Test images (Neg)/T6.bmp'
# python3 hog.py './Test images (Neg)/T7.bmp'
# python3 hog.py './Test images (Neg)/T8.bmp'
# python3 hog.py './Test images (Neg)/T9.bmp'
```

```python
# python3 hog.py './Test images (Neg)/T10.bmp'

if (len(sys.argv)) < 2:
    print("Command failure. Please pass image path+name as parameter in single quotesand try again.\
        \nExample: $ python3 hog.py './Test images (Neg)/T10.bmp'")
    exit()

# compute HOG on all database images first

# Collect database files
pos_database_files = ['DB1.bmp', 'DB2.bmp', 'DB3.bmp', 'DB4.bmp', 'DB5.bmp', 'DB6.bmp',\
    'DB7.bmp', 'DB8.bmp', 'DB9.bmp', 'DB10.bmp']
neg_database_files = ['DB11.bmp', 'DB12.bmp', 'DB13.bmp', 'DB14.bmp', 'DB15.bmp', 'DB16.bmp',\
    'DB17.bmp', 'DB18.bmp', 'DB19.bmp', 'DB20.bmp']

pos_database_loc = './Database images (Pos)/'
neg_database_loc = './Database images (Neg)/'

n_pos_database = len(pos_database_files)
n_neg_database = len(neg_database_files)
n_database = n_pos_database + n_neg_database


database_HOG = np.zeros((n_database,7524))


# For all positive database images,
for i, _ in enumerate(pos_database_files):
    # Parse file location for this image
    curr_database_image = pos_database_loc + pos_database_files[i]

    # Convert this image to grayscale
    bw_img = convert_to_grayscale(curr_database_image)

    # Compute gradients for this image
    gradient_magnitude, gradient_angle = Gradient_Operator(np.array(bw_img))

    # Save the normalized gradient magnitude  image in the format 'Gradient_Magnitude_imagename.bmp'
    if not os.path.exists('Gradient_Magnitude_Images'):
      os.makedirs('Gradient_Magnitude_Images')
    im = Image.fromarray(gradient_magnitude).convert('L')
    im.save('Gradient_Magnitude_Images/Gradient_Magnitude_'+pos_database_files[i])

    # Compute HOG for this image
    database_HOG[i] =  normalized_HOG_vector(gradient_magnitude,gradient_angle)

    if not os.path.exists('ASCII_Files'):
      os.makedirs('ASCII_Files')
    np.savetxt('ASCII_Files/'+pos_database_files[i].split(".")[0]+'.txt', database_HOG[i], delimiter='\n')

print ("Grayscale, Gradients and HOG for Pos database images complete")

# Now we repeat for all negative database images,
for i, _ in enumerate(neg_database_files):
    # Parse file location for this image
    curr_database_image = neg_database_loc + neg_database_files[i]


    # Convert this image to grayscale
    bw_img = convert_to_grayscale(curr_database_image)

    # Compute gradients for this image
    gradient_magnitude, gradient_angle = Gradient_Operator(np.array(bw_img))

    # Save the normalized gradient magnitude  image in the format 'Gradient_Magnitude_imagename.bmp'
    im = Image.fromarray(gradient_magnitude).convert('L')
    im.save('Gradient_Magnitude_Images/Gradient_Magnitude_'+neg_database_files[i])

    # Compute HOG for this image
    database_HOG[i+n_pos_database] =  normalized_HOG_vector(gradient_magnitude, gradient_angle)

    np.savetxt('ASCII_Files/'+neg_database_files[i].split(".")[0]+'.txt', database_HOG[i+n_pos_database], delimiter='\n')

print ("Grayscale, Gradients and HOG for Neg database images complete")



# compute HOG on given test image

# show input image from parameter passed
test_ip_image = sys.argv[1]
testfilename = test_ip_image.split("/")[-1]
im_show = Image.open(test_ip_image)
im_show.show()

# Convert this image to grayscale
bw_img = convert_to_grayscale(test_ip_image)

# Compute gradients for this image
gradient_magnitude, gradient_angle = Gradient_Operator(np.array(bw_img))

# Save the normalized gradient magnitude  image in the format 'Gradient_Magnitude_imagename.bmp'
im = Image.fromarray(gradient_magnitude).convert('L')
im.save('Gradient_Magnitude_Images/Gradient_Magnitude_'+testfilename)

# Compute HOG for this image
test_HOG =  normalized_HOG_vector(gradient_magnitude,gradient_angle)

np.savetxt('ASCII_Files/'+testfilename.split(".")[0]+'.txt', test_HOG, delimiter='\n')

print ("Grayscale, Gradients and HOG for test image complete")

# Calculate distance
```

```python
histo_similarity, hellinger_similarity = least_distance(database_HOG,test_HOG)

print ("Distance calculation complete")

# Classify with 3NN (histogram intersection similarity)
print("\nHistogram Intersection:")
similarity = histo_similarity
classification = threeNN(similarity, n_neg_database)

print("\nThe test image is classified as: ", classification)

# Classify with 3NN (hellinger similarity)
print("\nHellinger Distance:")
similarity = hellinger_similarity
classification = threeNN(similarity, n_neg_database)

print("\nThe test image is classified as: ", classification)
```

Normalized Gradient Magnitude Images for Test Images

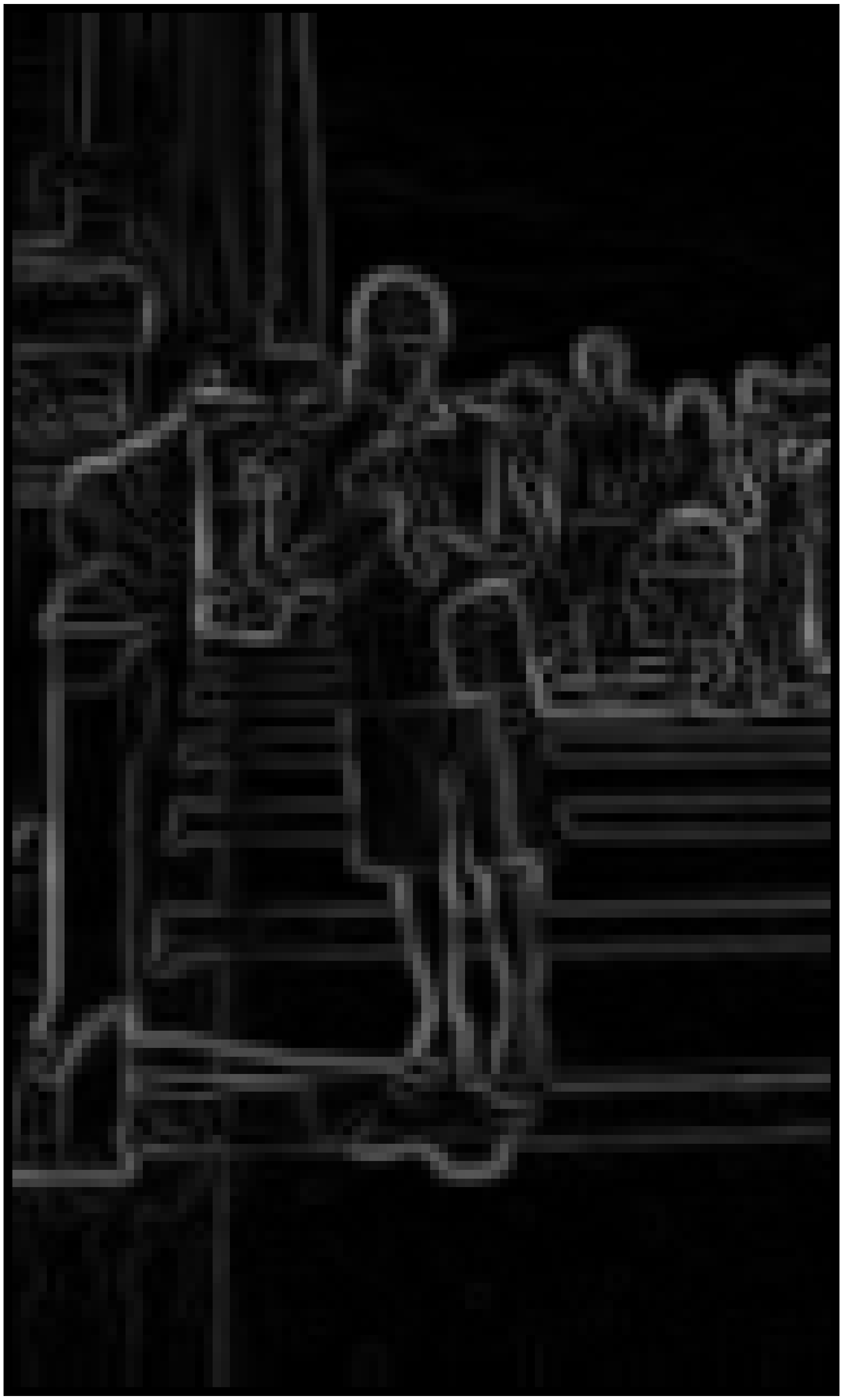


Figure 3: Normalized Gradient Magnitude Image - T1.bmp

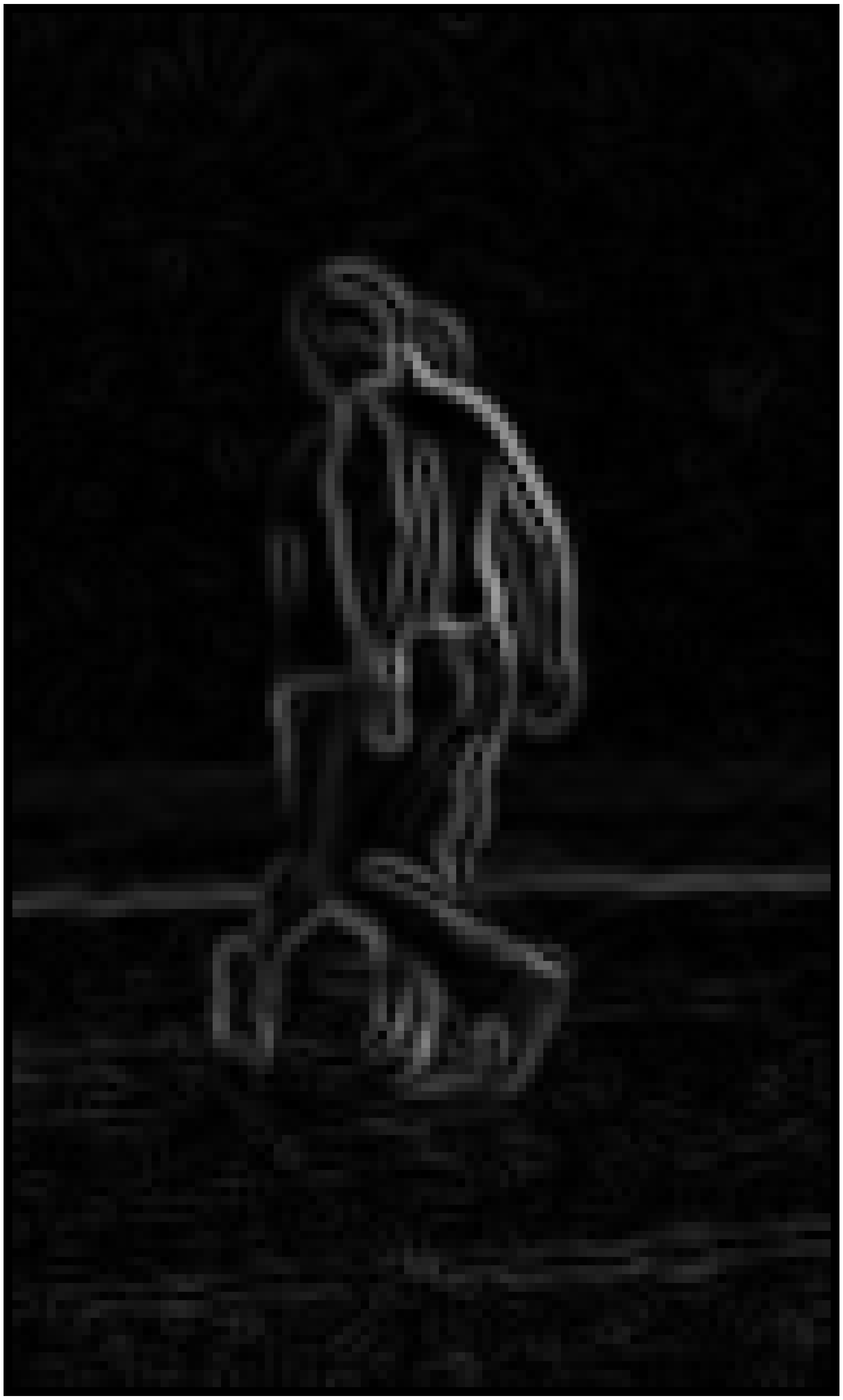


Figure 4: Normalized Gradient Magnitude Image - T2.bmp

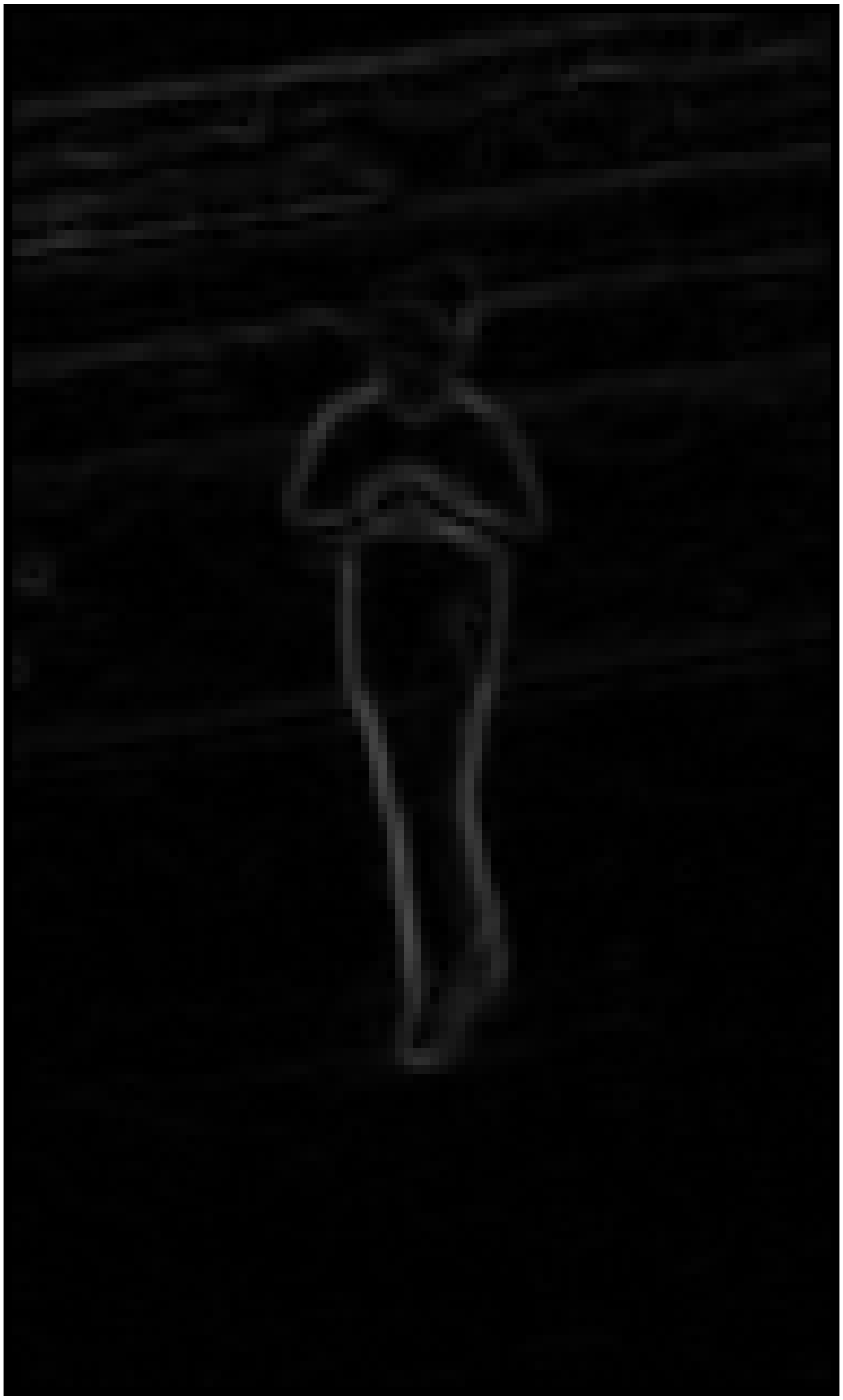


Figure 5: Normalized Gradient Magnitude Image - T3.bmp

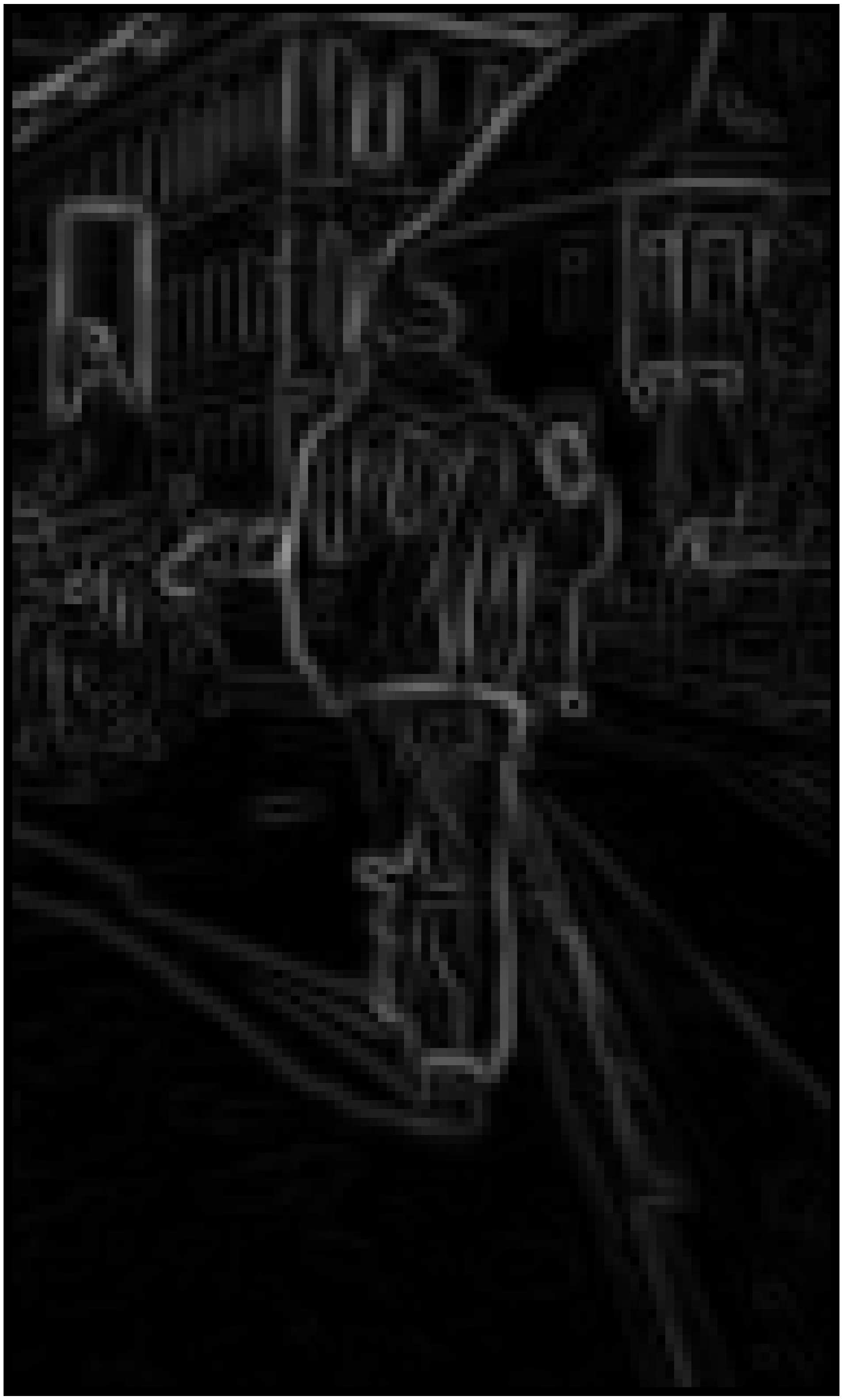


Figure 6: Normalized Gradient Magnitude Image - T4.bmp

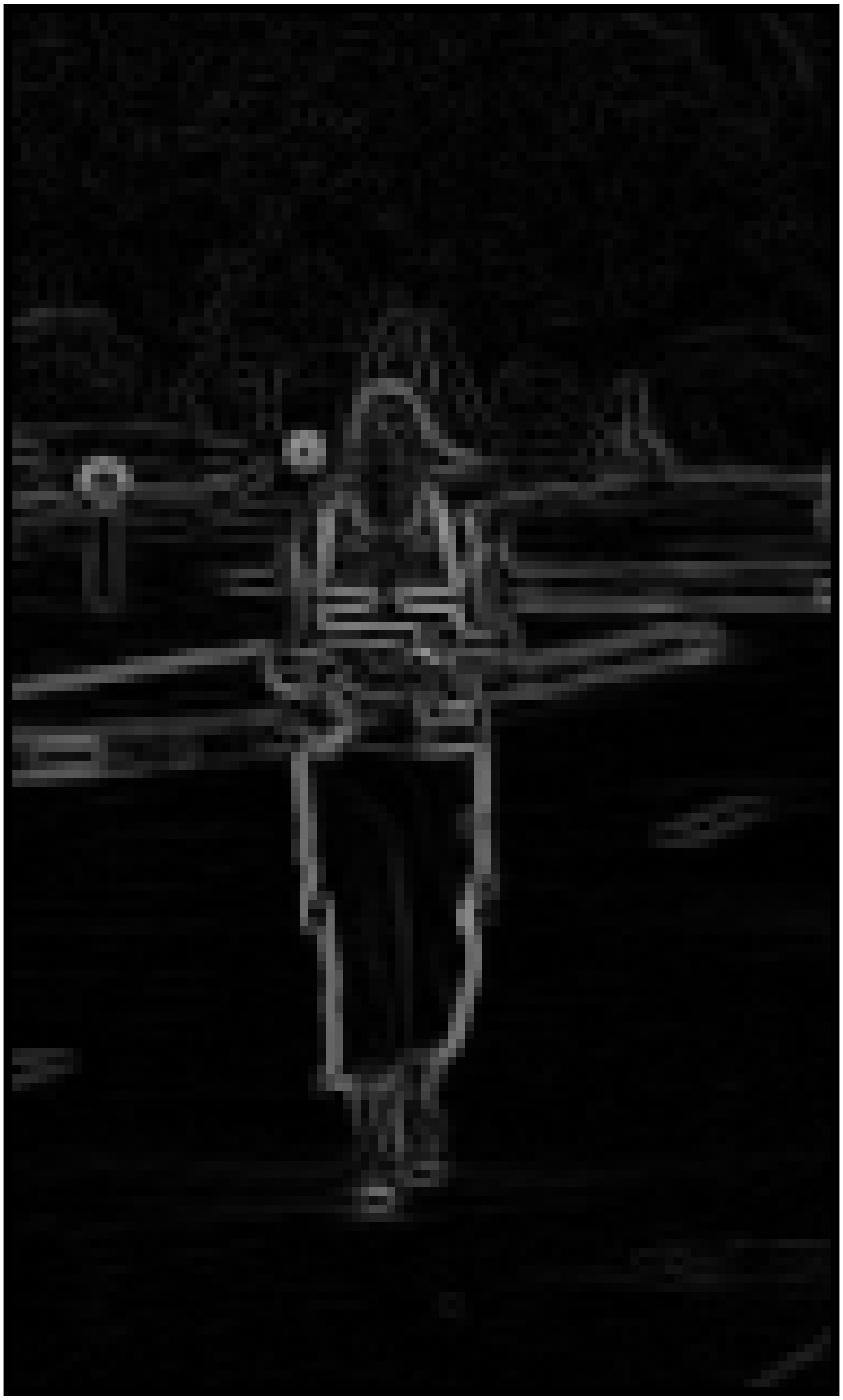


Figure 7: Normalized Gradient Magnitude Image - T5.bmp

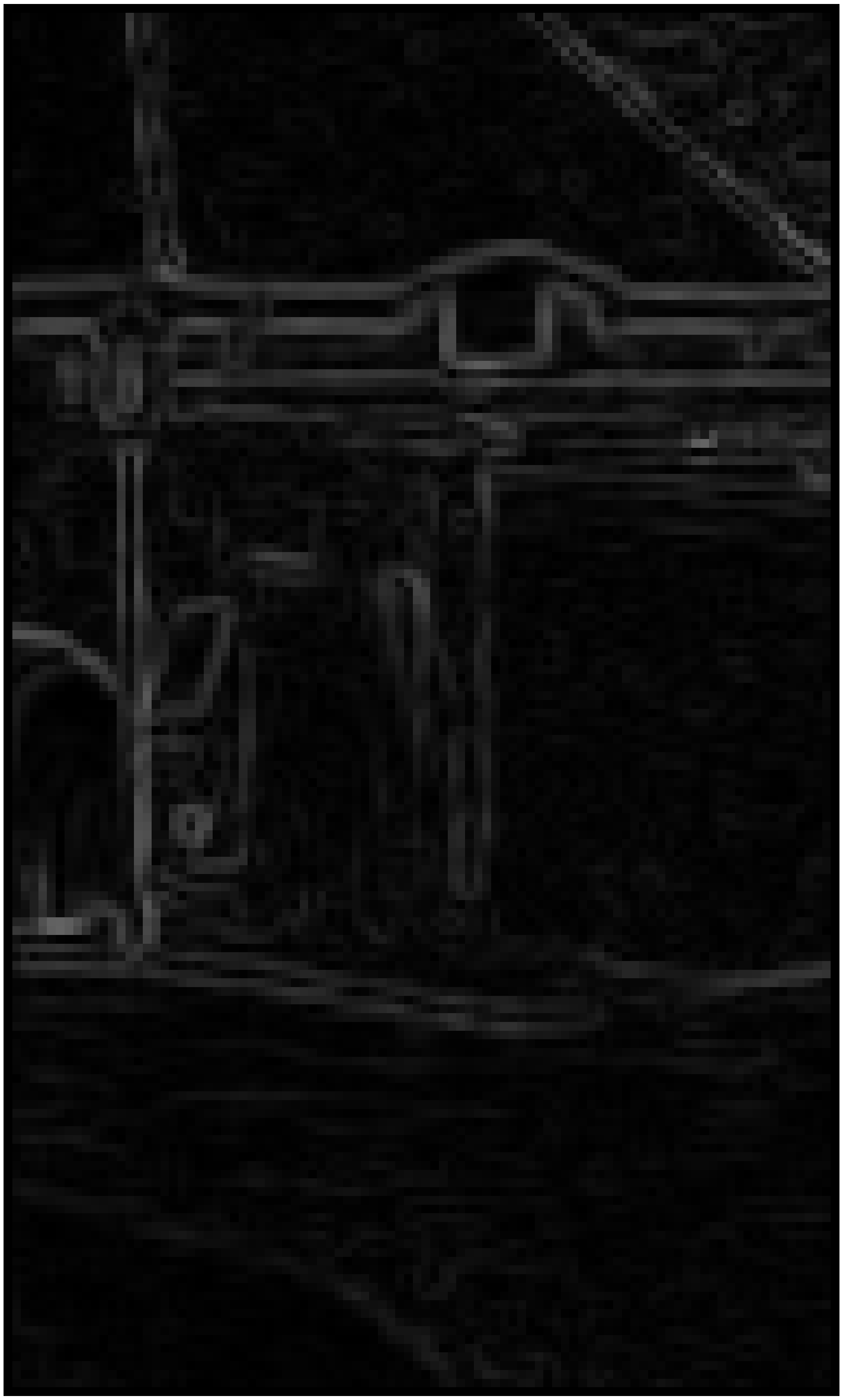


Figure 8: Normalized Gradient Magnitude Image - T6.bmp

Figure 9: Normalized Gradient Magnitude Image - T7.bmp

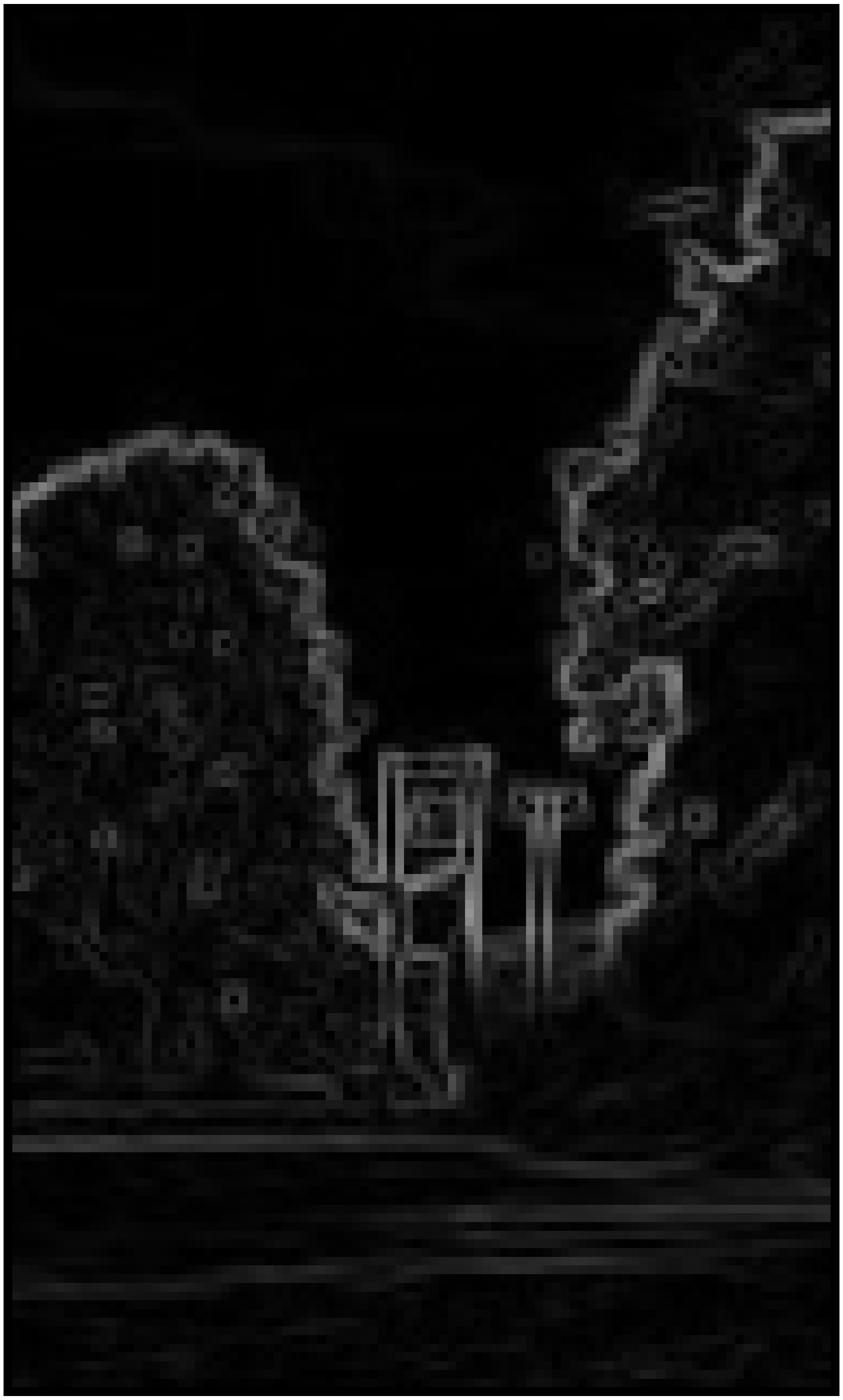


Figure 10: Normalized Gradient Magnitude Image - T8.bmp

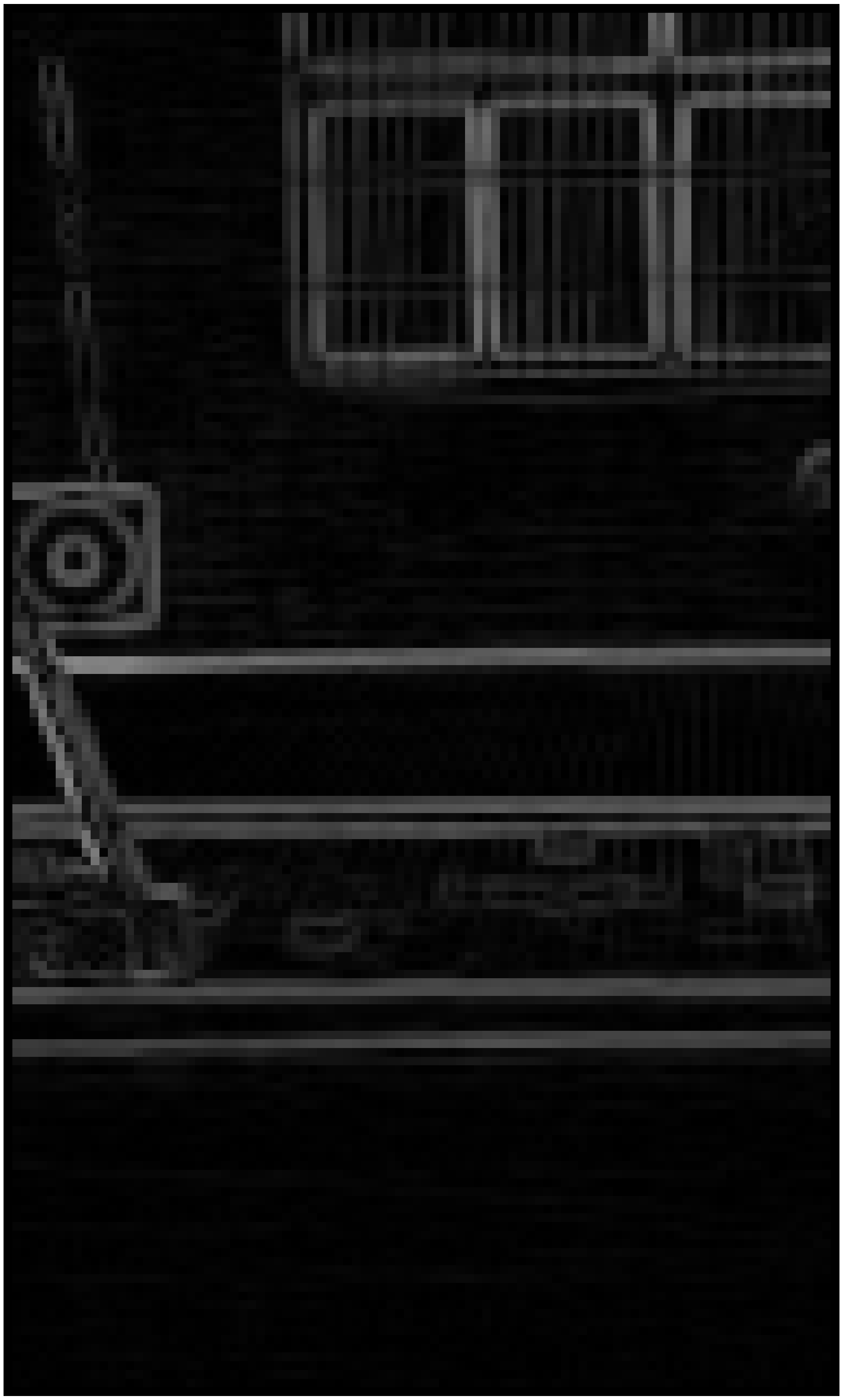


Figure 11: Normalized Gradient Magnitude Image - T9.bmp

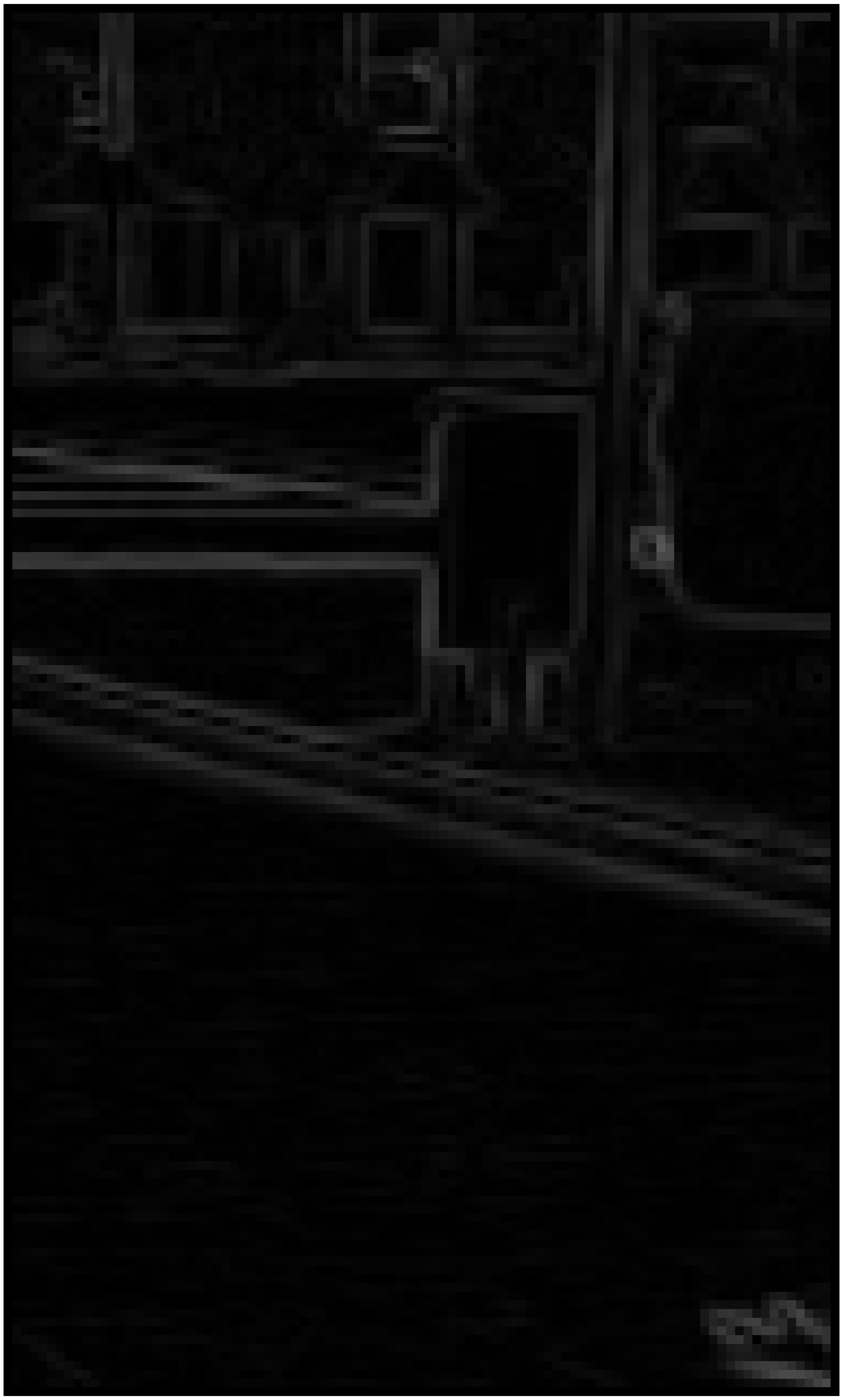


Figure 12: Normalized Gradient Magnitude Image - T10.bmp

Classification

Histogram Intersection

Test Input 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**
Image 1, T1.bmp	Human	DB4.bmp, 0.463852, Human	DB9.bmp, 0.478271, Human	DB17.bmp, 0.479659, Not-Human	Human
Image 2, T2.bmp	Human	DB2.bmp, 0.384581, Human	DB4.bmp, 0.398164, Human	DB9.bmp, 0.399784, Human	Human
Image 3, T3.bmp	Human	DB19.bmp, 0.413435, Not-Human	DB9.bmp, 0.415795, Human	DB4.bmp, 0.427689, Human	Human
Image 4, T4.bmp	Human	DB4.bmp, 0.432522, Human	DB2.bmp, 0.470665, Human	DB17.bmp, 0.479521, Not-Human	Human
Image 5, T5.bmp	Human	DB9.bmp, 0.389371, Human	DB8.bmp, 0.407104, Human	DB4.bmp, 0.408244, Human	Human
Image 6, T6.bmp	No-Human	DB4.bmp, 0.404363, Human	DB9.bmp, 0.414431, Human	DB17.bmp, 0.424335, Not-Human	Human
Image 7, T7.bmp	No-Human	DB16.bmp, 0.445186, Not-Human	DB11.bmp, 0.483561, Not-Human	DB9.bmp, 0.484533, Human	No Human
Image 8, T8.bmp	No-Human	DB4.bmp, 0.430921, Human	DB17.bmp, 0.431084, Not-Human	DB18.bmp, 0.438295, Not-Human	No Human
Image 9, T9.bmp	No-Human	DB15.bmp, 0.447165, Not-Human	DB9.bmp, 0.475633, Human	DB18.bmp, 0.487828, Not-Human	No Human
Image 10, T10.bmp	No-Human	DB18.bmp, 0.433491, Not-Human	DB9.bmp, 0.436139, Human	DB8.bmp, 0.456685, Human	Human

Hellinger Distance

Test Input 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
Image 1, T1.bmp	Human	DB4.bmp, 0.187761, Human	DB17.bmp, 0.203239, Not-Human	DB9.bmp, 0.206484, Human	Human
Image 2, T2.bmp	Human	DB2.bmp, 0.137236, Human	DB4.bmp, 0.142445, Human	DB9.bmp, 0.144281, Human	Human
Image 3, T3.bmp	Human	DB4.bmp, 0.156150, Human	DB9.bmp, 0.159647, Human	DB19.bmp, 0.166137, Not-Human	Human
Image 4, T4.bmp	Human	DB4.bmp, 0.157252, Human	DB17.bmp, 0.191922, Not-Human	DB2.bmp, 0.199051, Human	Human
Image 5, T5.bmp	Human	DB9.bmp, 0.138089, Human	DB4.bmp, 0.147364, Human	DB17.bmp, 0.150064, Not-Human	Human
Image 6, T6.bmp	No-Human	DB4.bmp, 0.137221, Human	DB17.bmp, 0.146853, Not-Human	DB9.bmp, 0.150688, Human	Human
Image 7, T7.bmp	No-Human	DB11.bmp, 0.235048, Not-Human	DB16.bmp, 0.238327, Not-Human	DB19.bmp, 0.244169, Not-Human	No Human
Image 8, T8.bmp	No-Human	DB4.bmp, 0.158941, Human	DB17.bmp, 0.159815, Not-Human	DB18.bmp, 0.169203, Not-Human	No Human
Image 9, T9.bmp	No-Human	DB15.bmp, 0.179171, Not-Human	DB9.bmp, 0.208358, Human	DB18.bmp, 0.220813, Not-Human	No Human
Image 10, T10.bmp	No-Human	DB9.bmp, 0.174426, Human	DB18.bmp, 0.175909, Not-Human	DB8.bmp, 0.190413, Human	Human