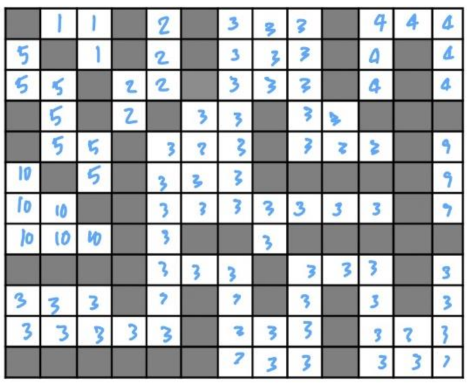
Kunnithi Treecharoensomboon ID 63070503461

CPE383 Machine Learning: Quiz 3

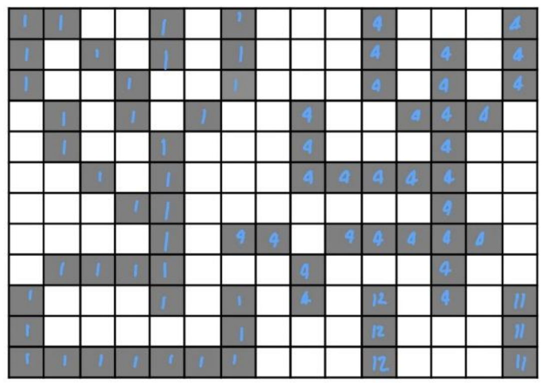
1. **Connected Components.** Non-programming exercise. Label the following image with colors starting 1.

   a. *5 points.* 0.5 hrs. Use 4 connected to label the background (white) regions. Show the equivalence table and the final color in each box.

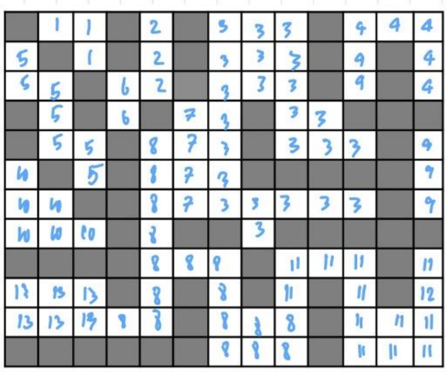
Your Output for 4-connected Labeling:

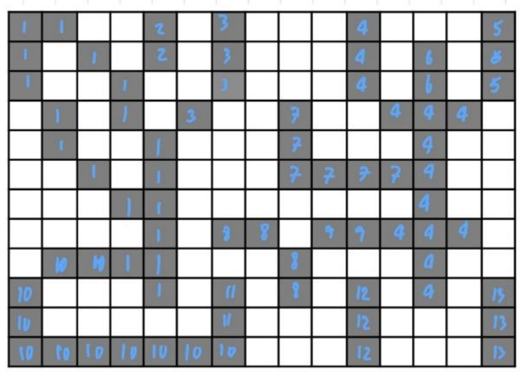


Your Output for 8-connected Labeling:



b. *5 points.* 0.5 hrs. Use 8 connected to label foreground (black) regions. Show the equivalence table and the final color in each box.





equivalence table

2	6
3	7
7	8
8	11
11	12
8	13

equivalance table

1	2
1	3
4	6
4	5
4	7
4	9
8	9
1	10
10	11

2. Handwritten Digit Recognition. Use the digits.png file as template for digits 0, 1, ..., 9. Write a python program to cut out each digit as a labeled dataset from 0..9, each of which is 20x20. Note: You may also use this exact same dataset with 100 samples of each digit 0..9 using 20 x 20 pixels from the internet along with libraries to read/load the dataset, if it's easier for you.

Rescale all images from 20 x 20 to 24 x 24. This can be done in memory as a python class data and need not be stored in a file. Use 80% of the data as the training set, reserving 20% for testing.

```
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import random
```

```
[37] #this was created because Google Colab does not allow cv2.imshow, so must patch by cv2_imshow.
#If we switch over to regular jupyter notebook not on Colab, we can change c2_imshow to cv2.imshow.
from google.colab.patches import cv2_imshow #only used when running in Google Colab
def my_imshow(title, img ):
  print(title)
  cv2_imshow(img) #should be changed to c2.imshow when not in Colab
```

```
[38] path = ""
fileName = path + "digits.png"

#RGB images in BGR order in OpenCV
image = cv2.imread(fileName, cv2.IMREAD_COLOR)

# Print error message if image is null
if image is None:
    print('Could not read image')
else:
    print("Image file read success...")
```

```
Image file read success...
```

```
# cut out each digit as a labeled dataset and rescale all images from 20 x 20 to 24 x 24.
sub_image = []
desired_dimension = (24, 24)
height, width = image.shape[:2]
print(height, width)
for i in range(height // 20):
    sub_image.append([])
    for j in range(width // 20):
        sub_image[-1].append(cv2.resize(image[i*20:(i+1)*20, j*20:(j+1)*20], desired_dimension))

# Convert sub_image to a numpy array
sub_image = np.array(sub_image)

# Create a 50x100 array and initialize it to -1
label = np.full((50, 100), -1)

# Fill the array with alternating values of 0-9
for i in range(10):
    label[i*5:(i+1)*5, :] = i

# Split sub_image into training and testing sets
sub_train, sub_test, train_label, test_label = train_test_split(sub_image, label, test_size=0.2, random_state=42)
flat_train_label = train_label.flatten()
flat_test_label = test_label.flatten()
print(flat_train_label.shape)
```

```
1000 2000
(4000,)
```

a. 30 pts. 6 hrs. Then try recognition by using test images and report the accuracy percent for these 4 (classifier, feature type) combinations: (KNN K = 5, gray scale features), (KNN K = 5, HOG features), (KNN K = 1, gray scale features), (KNN K = 1, HOG features). For K = 5, if 2 or more digits have the same maximum vote count, just report one of them. For HOG, use 20° histogram orientations of non-directional gradients (ie., 9 bins) with 16 x 16 overlapping pixel windows for each 24 x 24 digit. Each digit will, thus, have 144 HOG features from 4 x 4 x 9, with 9 histo values x 4 per 16 x 16 block x 4 such blocks per 24 x 24 image.

```python
# Convert X_train and X_test to float and scale them
float_train = np.float32(sub_train) / 255.0
float_test = np.float32(sub_test) / 255.0

# Calculate the gradients for the training and testing sets
Gx_train = []
Gy_train = []

for row in float_train:
    for img in row:
        Gx = cv2.Sobel(img, cv2.CV_32F, 1, 0, ksize=1)
        Gy = cv2.Sobel(img, cv2.CV_32F, 0, 1, ksize=1)
        Gx_train.append(Gx)
        Gy_train.append(Gy)

Gx_train = np.array(Gx_train)
Gy_train = np.array(Gy_train)

print(Gx_train.shape, Gy_train.shape)
```

```
(4000, 24, 24, 3) (4000, 24, 24, 3)
```

```python
winSize = desired_dimension
blockSize = (16,16) #OpenCV only supports 16 x 16 block sizes
blockStride = (8,8) #multiple of cell size. Here it is multiple of 1.
cellSize = (8,8) #OpenCV only supports 8x8 cell size. That means each Block will have 4 histograms
nbins = 9 #OpenCV only supports 9 orientations per cell.  That means 1 block has 4 x 9 = 36 features
derivAperture = 1
winSigma = 4.
histogramNormType = 0
L2HysThreshold = 2.0000000000000001e-01
gammaCorrection = 0
hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,
                        histogramNormType,L2HysThreshold,gammaCorrection)
#compute(img[, winStride[, padding[, locations]]]) -> descriptors
winStride = (8,8)
padding = (8,8)
locations = ((0,0),) #we run it at only one location of image (entire image).  Can be used to run at sub-image parts.

train_hog_feature = np.zeros((len(sub_train), len(sub_train[0]), 4*4*nbins))
#print(hog_feature.shape)
for i in range(len(sub_train)):
    for j in range(len(sub_train[i])):
        feature_vector = hog.compute(sub_train[i][j],winStride,padding,locations)
        feature_vector = feature_vector.reshape(-1) # Flatten the 3D array to 1D array
        train_hog_feature[i][j] = feature_vector

train_hog_feature = train_hog_feature.reshape((train_hog_feature.shape[0]*train_hog_feature.shape[1], train_hog_feature.shape[2]))
#print("HOG feature: \n", train_hog_feature)
number_of_train_features = train_hog_feature.shape
# There are 7 horizontal and 15 vertical blue windows, making a total of 7 x 15 = 105 positions.
# Each 16×16 block is represented by 4 of 8 x 8 blocks. Each block is 9 histogram values.
# Each 16x16 block has 4 x 9 = 36×1 feature vector.
# The feature is a concatenation of 105 such features to get
# a vector of dimension 3,780 from 105 x 36.
```

```python
# print("This is from 9 histo values per cell x 4 cells per 16x16 block x (7x15) blocks per image: ", 9*4*7*15)

test_hog_feature = np.zeros((len(sub_test), len(sub_test[0]), 4*4*nbins))
# print(hog_feature.shape)
for i in range(len(sub_test)):
    for j in range(len(sub_test[i])):
        feature_vector = hog.compute(sub_test[i][j],winStride,padding,locations)
        feature_vector = feature_vector.reshape(-1) # Flatten the 3D array to 1D array
        test_hog_feature[i][j] = feature_vector

test_hog_feature = test_hog_feature.reshape((test_hog_feature.shape[0]*test_hog_feature.shape[1], test_hog_feature.shape[2]))
# print("HOG feature: \n", test_hog_feature)
number_of_test_features = test_hog_feature.shape
print("Number of test Features: ", number_of_test_features)
```

```
Number of train Features:  (4000, 144)
Number of test Features:  (1000, 144)
```

```python
# Convert sub_train images to grayscale and flatten them
sub_train_gray = []
for row in sub_train:
    for img in row:
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        gray_img_flat = gray_img.flatten()
        sub_train_gray.append(gray_img_flat)

# Convert sub_test images to grayscale and flatten them
sub_test_gray = []
for row in sub_test:
    for img in row:
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        gray_img_flat = gray_img.flatten()
        sub_test_gray.append(gray_img_flat)
```

```python
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(train_hog_feature, flat_train_label)
HOG_K5 = knn_classifier.predict(test_hog_feature)

accuracy = sum(HOG_K5 == flat_test_label) / len(flat_test_label)
print("HOG_K5 Accuracy:", accuracy)

knn_classifier = KNeighborsClassifier(n_neighbors=1)
knn_classifier.fit(train_hog_feature, flat_train_label)
HOG_K1 = knn_classifier.predict(test_hog_feature)

accuracy = sum(HOG_K1 == flat_test_label) / len(flat_test_label)
print("HOG_K1 Accuracy:", accuracy)

knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(sub_train_gray, flat_train_label)
GRAY_K5 = knn_classifier.predict(sub_test_gray)

accuracy = sum(GRAY_K5 == flat_test_label) / len(flat_test_label)
print("GRAY_K5 Accuracy:", accuracy)

knn_classifier = KNeighborsClassifier(n_neighbors=1)
knn_classifier.fit(sub_train_gray, flat_train_label)
GRAY_K1 = knn_classifier.predict(sub_test_gray)

accuracy = sum(GRAY_K5 == flat_test_label) / len(flat_test_label)
print("GRAY_K1 Accuracy:", accuracy)
```

```
HOG_K5 Accuracy: 0.956
HOG_K1 Accuracy: 0.949
GRAY_K5 Accuracy: 0.924
GRAY_K1 Accuracy: 0.924
```

b. Use KNN K = 1 with HOG features to report:

i. 5 pts. 1.0 hrs. Result for 2 test images per digit 0..9 cut out from digits.png itself but not aligned at the original 20 x 20 image, so you may have smaller or bigger sizes. You will have to rescale each test image to 24 x 24 because HOG must be recalculated after scaling.

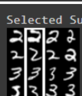
```
desired_dimension = (24, 24)
row_start = ((random.randint(0, 8)*5) + 3) * 20
col_start = random.randint(0, 94) * 20
cut_image = image[row_start:row_start+(20*4), col_start:col_start+(20*4)]

sub_cut = []
desired_dimension = (24, 24)
height, width = cut_image.shape[:2]
for i in range(height // 20):
    sub_cut.append([])
    for j in range(width // 20):
        sub_cut[-1].append(cv2.resize(cut_image[i*20:(i+1)*20, j*20:(j+1)*20], desired_dimension))

first_label = int(((row_start/20)-3)/5)
ran_col = random.randint(0, 3)
test_cut = []
train_cut = []
test_cut_label = [first_label, first_label, first_label+1, first_label+1]
train_cut_label = [first_label, first_label, first_label, first_label, first_label, first_label, first_label+1, first_label+1, first_label+1, first_label+1, first_label+1,
for row in range(len(sub_cut)):
    for col in range(len(sub_cut[row])):
        if col == ran_col:
            test_cut.append(sub_cut[row][col])
            cv2.rectangle(cut_image, (ran_col*20, 0), ((ran_col*20)+20, 80), (255, 255, 255), 2)
        else:
            train_cut.append(sub_cut[row][col])

# Display the image with bounding box
my_imshow('Selected Sub-Image with Bounding Box', cut_image)
for img in test_cut:
    my_imshow('Test-Image', img)
```

Selected Sub-Image with Bounding Box



Test-Image

Test-Image

Test-Image

Test-Image

```
winSize = desired_dimension
blockSize = (16,16)
blockStride = (8,8)
cellSize = (8,8)
nbins = 9
derivAperture = 1
winSigma = 4.
histogramNormType = 0
L2HysThreshold = 2.0000000000000001e-01
gammaCorrection = 0
hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,
                        histogramNormType,L2HysThreshold,gammaCorrection)

winStride = (8,8)
padding = (8,8)
locations = ((0,0),)

train_cut_hog_feature = np.zeros((len(train_cut), 4*4*nbins))
for i in range(len(train_cut)):
    feature_vector = hog.compute(train_cut[i],winStride,padding,locations)
    feature_vector = feature_vector.reshape(-1)
    train_cut_hog_feature[i] = feature_vector


test_cut_hog_feature = np.zeros((len(test_cut), 4*4*nbins))
for i in range(len(test_cut)):
    feature_vector = hog.compute(test_cut[i],winStride,padding,locations)
    feature_vector = feature_vector.reshape(-1) # Flatten the 3D array to 1D array
    test_cut_hog_feature[i] = feature_vector
```

```
print(train_cut_hog_feature.shape, len(train_cut_label))
knn_classifier = KNeighborsClassifier(n_neighbors=1)
knn_classifier.fit(train_cut_hog_feature, train_cut_label)
HOG_K1 = knn_classifier.predict(test_cut_hog_feature)

accuracy = sum(HOG_K1 == test_cut_label) / len(test_cut_label)
print("HOG_K1 Accuracy:", accuracy)
```

```
(12, 144) 12
HOG_K1 Accuracy: 1.0
```

ii. 5 pts. 1.0 hrs. Result for 2 test images per digit 0..9 you create in a Paint program to see if you can find your character. Each test image should be big to start with such as 50x50, but you should rescale it to 24 x 24 before testing.

```
path = ""
fileName5 = path + "5.png"
fileName6 = path + "6.png"

#RGB images in BGR order in OpenCV
image5 = cv2.imread(fileName5, cv2.IMREAD_COLOR)
image5 = cv2.resize(image5, desired_dimension)
image6 = cv2.imread(fileName6, cv2.IMREAD_COLOR)
image6 = cv2.resize(image6, desired_dimension)

desired_dimension = (24, 24)
test_cut_label = [5, 6]
winSize = desired_dimension
blockSize = (16,16)
blockStride = (8,8)
cellSize = (8,8)
nbins = 9
derivAperture = 1
winSigma = 4.
histogramNormType = 0
L2HysThreshold = 2.0000000000000001e-01
gammaCorrection = 0
hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,
                        histogramNormType,L2HysThreshold,gammaCorrection)

winStride = (8,8)
padding = (8,8)
locations = ((0,0),)

my_imshow("img5", image5)
my_imshow("img6", image6)
```

```
train_cut_hog_feature = np.zeros((len(train_cut), 4*4*nbins))
for i in range(len(train_cut)):
    feature_vector = hog.compute(train_cut[i],winStride,padding,locations)
    feature_vector = feature_vector.reshape(-1)
    train_cut_hog_feature[i] = feature_vector

test_cut = [image5, image6]
test_cut_hog_feature = np.zeros((len(test_cut), 4*4*nbins))
for i in range(len(test_cut)):
    feature_vector = hog.compute(test_cut[i],winStride,padding,locations)
    feature_vector = feature_vector.reshape(-1) # Flatten the 3D array to 1D array
    test_cut_hog_feature[i] = feature_vector

print(train_cut_hog_feature.shape, len(train_cut_label))
knn_classifier = KNeighborsClassifier(n_neighbors=1)
knn_classifier.fit(train_cut_hog_feature, train_cut_label)
HOG_K1 = knn_classifier.predict(test_cut_hog_feature)

accuracy = sum(HOG_K1 == test_cut_label) / len(test_cut_label)
print("HOG_K1 Accuracy:", accuracy)
```

img5

img6

```
(12, 144) 12
HOG_K1 Accuracy: 1.0
```

iii. 5 pts. 1.0 hrs. Result for 4 test images of digit 5 you create in Paint with white background.

```python
path = ""
fileName51 = path + "5.1.png"
fileName52 = path + "5.2.png"
fileName53 = path + "5.3.png"
fileName54 = path + "5.4.png"

#RGB images in BGR order in OpenCV
image51 = cv2.imread(fileName51, cv2.IMREAD_COLOR)
image51 = cv2.resize(image51, desired_dimension)
image52 = cv2.imread(fileName52, cv2.IMREAD_COLOR)
image52 = cv2.resize(image52, desired_dimension)
image53 = cv2.imread(fileName53, cv2.IMREAD_COLOR)
image53 = cv2.resize(image53, desired_dimension)
image54 = cv2.imread(fileName54, cv2.IMREAD_COLOR)
image54 = cv2.resize(image54, desired_dimension)

desired_dimension = (24, 24)
test_cut_label = [5, 5, 5, 5]
winSize = desired_dimension
blockSize = (16,16)
blockStride = (8,8)
cellSize = (8,8)
nbins = 9
derivAperture = 1
winSigma = 4.
histogramNormType = 0
L2HysThreshold = 2.0000000000000001e-01
gammaCorrection = 0
hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,
                        histogramNormType,L2HysThreshold,gammaCorrection)

winStride = (8,8)
padding = (8,8)
locations = ((0,0),)
```

```python
my_imshow("imh51", image51)
my_imshow("imh52", image52)
my_imshow("imh53", image53)
my_imshow("imh54", image54)

train_cut_hog_feature = np.zeros((len(train_cut), 4*4*nbins))
for i in range(len(train_cut)):
    feature_vector = hog.compute(train_cut[i],winStride,padding,locations)
    feature_vector = feature_vector.reshape(-1)
    train_cut_hog_feature[i] = feature_vector

test_cut = [image51, image52, image53, image54]
test_cut_hog_feature = np.zeros((len(test_cut), 4*4*nbins))
for i in range(len(test_cut)):
    feature_vector = hog.compute(test_cut[i],winStride,padding,locations)
    feature_vector = feature_vector.reshape(-1) # Flatten the 3D array to 1D array
    test_cut_hog_feature[i] = feature_vector

print(train_cut_hog_feature.shape, len(train_cut_label))
knn_classifier = KNeighborsClassifier(n_neighbors=1)
knn_classifier.fit(train_cut_hog_feature, train_cut_label)
HOG_K1 = knn_classifier.predict(test_cut_hog_feature)

accuracy = sum(HOG_K1 == test_cut_label) / len(test_cut_label)
print("HOG_K1 Accuracy:", accuracy)
```

imh51

imh52

imh53

imh54

```
(12, 144) 12
HOG_K1 Accuracy: 1.0
```

c. 20 pts. 3 hrs. For each 0..9 digit in your dataset of 100 characters, use OpenCV's auto threshold and then connected components to find the bounding box. Use that bounding box to cut out each gray-scale image and resize each back to 24 x 24. This will be your new dataset (training and testing, combined). Report the accuracy percent for KNN K = 1 using HOG features. Is the result here better than in problem 2a for KNN = 1 using HOG features?

```
img = cv2.imread('digits.png', 0)

thresh = cv2.threshold(img, 0, 255, cv2.THRESH_OTSU)[1]

num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(thresh, connectivity=8)

images = []
labels = []
for i in range(1, num_labels):
    left, top, width, height, _ = stats[i]

    digit_img = img[top:top + height, left:left + width]

    resized_img = cv2.resize(digit_img, (24, 24))

    images.append(resized_img)
    labels.append(i // 500)

img_train, img_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
```

```
desired_dimension = (24, 24)
test_cut_label = [5, 5, 5, 5]
winSize = desired_dimension
blockSize = (16,16)
blockStride = (8,8)
cellSize = (8,8)
nbins = 9
derivAperture = 1
winSigma = 4.
histogramNormType = 0
L2HysThreshold = 2.0000000000000001e-01
gammaCorrection = 0
hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,
                        histogramNormType,L2HysThreshold,gammaCorrection)

train_features = np.array([hog.compute(cv2.resize(img, desired_dimension)) for img in img_train]).reshape(len(img_train), -1)

test_features = np.array([hog.compute(cv2.resize(img, desired_dimension)) for img in img_test]).reshape(len(img_test), -1)

knn_classifier = KNeighborsClassifier(n_neighbors=1)
knn_classifier.fit(train_features, y_train)

predictions = knn_classifier.predict(test_features)
accuracy = np.mean(predictions == y_test)

print("HOG_K1 Accuracy:", accuracy)
```

```
HOG_K1 Accuracy: 0.8204121687929342
```

Ans: No