Lec3

February 17, 2024

1 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.

1.1 1.1

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.

- 4-connected background labeling
- 8-connected background labeling

1.2 1.2

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

- 4-connected foreground labeling
- 8-connected foreground labeling

2 2

Handwritten Digit Recognition. Use the digits.png file as templates 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 that's easier for you. Load all the character data into a python class. Then rescale each character from 20 x 20 to 24 x 24 using OpenCV. Use 80% of the data as the training set, reserving 20% for testing.

Load Image

[42]:
```
import cv2
from typing import TypedDict, List, Tuple
import numpy as np
```

[43]:
```
class Digits(TypedDict):
    zero: List[np.ndarray]
```

```
    one: List[np.ndarray]
    two: List[np.ndarray]
    three: List[np.ndarray]
    four: List[np.ndarray]
    five: List[np.ndarray]
    six: List[np.ndarray]
    seven: List[np.ndarray]
    eight: List[np.ndarray]
    nine: List[np.ndarray]
```

[44]:
```
digits: Digits = {
    "zero": [],
    "one": [],
    "two": [],
    "three": [],
    "four": [],
    "five": [],
    "six": [],
    "seven": [],
    "eight": [],
    "nine": []
}
```

[45]:
```
# subplot to see the result
import matplotlib.pyplot as plt
def crop_image(image: np.ndarray, shape: Tuple[int, int], digits: Digits):
  '''
  to crop a whole digit image(20*20) from big image and rescale it to 24*24
  '''
  height, width = shape
  print(image.shape)
  uncropClass: Digits = {}
  for i,key in enumerate(digits.keys()):
    print("crop from", 20*i*5 , "to", 20*(i+1)*5, "for", key)
    uncropClass[key] = image[20*i*5:20*(i+1)*5, :]
    print(uncropClass[key].shape)

  for key in uncropClass.keys():
    class_crop = uncropClass[key]
    for i in range(0, class_crop.shape[1], width):
      for j in range(0, class_crop.shape[0], height):
        crop = class_crop[j:j+height, i:i+width]
        if crop.shape[0] == height and crop.shape[1] == width:
          digits[key].append(cv2.resize(crop, (24, 24)))
```

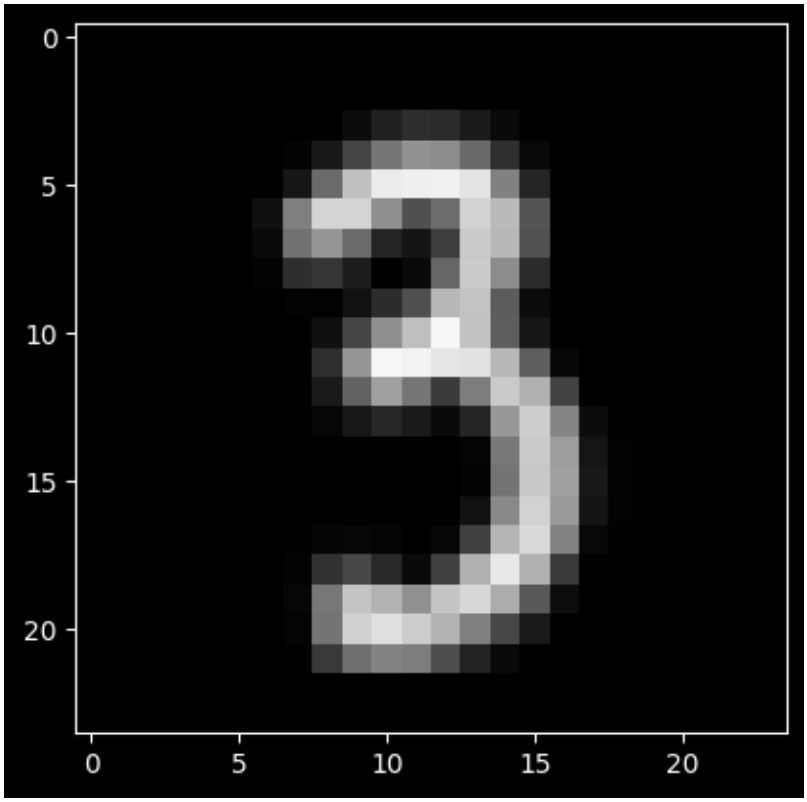
[46]:
```
image = cv2.imread('./Q3/digits.png')
print(image.shape)

crop_image(image, (20, 20), digits)


digits['eight'][0].shape
```

```
(1000, 2000, 3)
(1000, 2000, 3)
crop from 0 to 100 for zero
(100, 2000, 3)
crop from 100 to 200 for one
(100, 2000, 3)
crop from 200 to 300 for two
(100, 2000, 3)
crop from 300 to 400 for three
(100, 2000, 3)
crop from 400 to 500 for four
(100, 2000, 3)
crop from 500 to 600 for five
(100, 2000, 3)
crop from 600 to 700 for six
(100, 2000, 3)
crop from 700 to 800 for seven
(100, 2000, 3)
crop from 800 to 900 for eight
(100, 2000, 3)
crop from 900 to 1000 for nine
(100, 2000, 3)
```

[46]:
```
(24, 24, 3)
```

[47]:
```
plt.imshow(digits['three'][50], cmap='gray')
```

[47]:
```
<matplotlib.image.AxesImage at 0x1ea9534a500>
```



[48]:
```
from sklearn.model_selection import train_test_split

train_digits: Digits = {}
test_digits: Digits = {}

for key in digits.keys():
  train_digits[key], test_digits[key] = train_test_split(digits[key],␣
 ↪test_size=0.2, random_state=42)

train_digits['eight'][0].shape
```

[48]:
```
(24, 24, 3)
```

3 2.1

10+10+10+10 pts. 5 hrs. Then try recognition by using the test images and report the accuracy percent for these 4 (classifier, feature type) combinations: (KNN K = 5, gray scale features), (KNN K =5,HOGfeatures), (KNN K = 1, gray scale features), (KNN K = 1, HOG features). 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 histogram values x 4 per 16 by 16 block x 4 such blocks per 24 x 24 image.

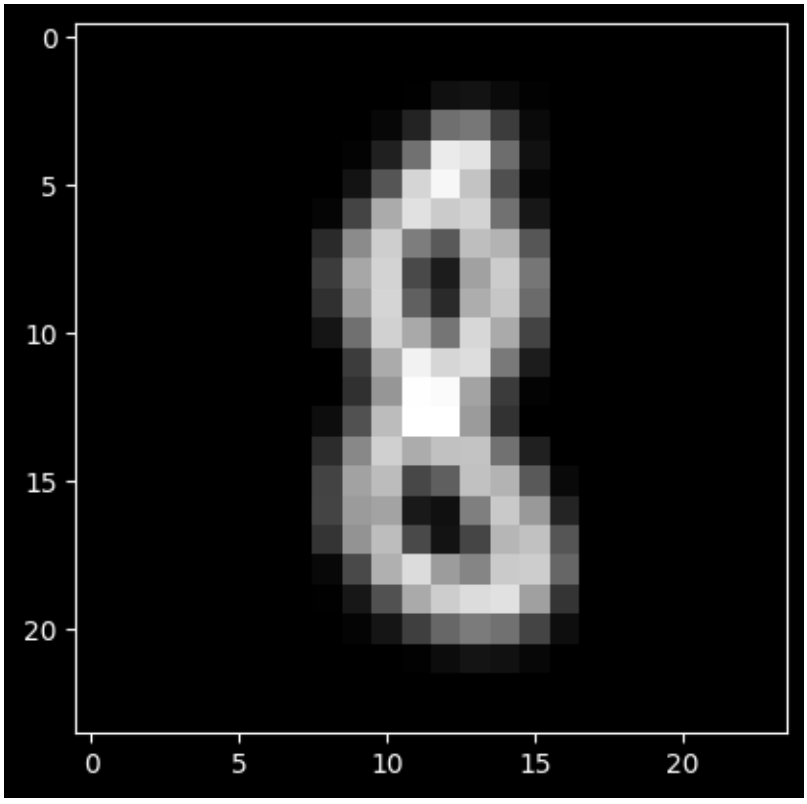
[49]:
```
from typing import Callable
```

[50]:
```
def preprocess_image(image: Digits, preprocess: Callable[[List[np.ndarray]],␣
↪List[np.ndarray]]) -> Digits:
  '''
  to preprocess the image with a given function
  '''
  result: Digits = {}
  for key in image.keys():
    result[key] = preprocess(image[key])
  return result
```

1. KNN K = 5, gray scale features

[51]:
```
from PIL import Image

def to_grayscale(images: List[np.ndarray]) -> List[np.ndarray]:
  '''
  to convert the image to grayscale
  '''
  return [Image.fromarray(image).convert('L') for image in images]

grayscale_train_digits = preprocess_image(train_digits, to_grayscale)
grayscale_test_digits = preprocess_image(test_digits, to_grayscale)

plt.imshow(grayscale_train_digits['eight'][0], cmap='gray')
```

[51]:
```
<matplotlib.image.AxesImage at 0x1ea9e729000>
```



[52]:
```python
from sklearn.neighbors import KNeighborsClassifier

K = 5

def flatten_images(images: List[np.ndarray]) -> List[np.ndarray]:
  '''
  to flatten the image
  '''
  return [np.array(image).flatten() for image in images]

def train_knn_model(train_images: Digits, K: int) -> KNeighborsClassifier:
  '''
  to train the knn model
  '''
  train_images = preprocess_image(train_images, flatten_images)
  X = []
  y = []
  for key in train_images.keys():
    X += train_images[key]
    y += [key] * len(train_images[key])
  model = KNeighborsClassifier(n_neighbors=K)
  model.fit(X, y)
```

```
  return model

model = train_knn_model(grayscale_train_digits, K)
```

[53]:
```
for cls in grayscale_test_digits.keys():
  predicts = model.predict(flatten_images(grayscale_test_digits[cls]))
  print(cls, np.mean(predicts == cls))
```

```
zero 0.99
one 0.99
two 0.91
three 0.89
four 0.92
five 0.97
six 0.97
seven 0.9
eight 0.95
nine 0.94
```

2. KNN K =5,HOGfeatures

[54]:
```
from cv2 import HOGDescriptor

def HOG(images: List[np.ndarray]) -> List[np.ndarray]:
  '''
  to extract the HOG feature from the image
  '''
  winSize = (24,24) #same as the size of the image
  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)

  result = []
  for image in images:
    result.append(hog.compute(np.array(image)))
```

```
  return result


def train_knn_model_HOG(train_images: Digits, K: int) -> KNeighborsClassifier:
  '''
  to train the knn model with HOG feature
  '''
  train_images = preprocess_image(train_images, HOG)
  X = []
  y = []
  for key in train_images.keys():
    X += train_images[key]
    y += [key] * len(train_images[key])
  model = KNeighborsClassifier(n_neighbors=K)
  model.fit(X, y)
  return model

model_HOG = train_knn_model_HOG(grayscale_train_digits, K)

for cls in grayscale_test_digits.keys():
  predicts = model_HOG.predict(HOG(grayscale_test_digits[cls]))
  print(cls, np.mean(predicts == cls))
```

```
zero 1.0
one 0.99
two 0.96
three 0.97
four 0.96
five 0.94
six 0.98
seven 0.9
eight 0.99
nine 0.94
```

3. KNN K = 1, gray scale features

[55]:
```
K = 1

grayscale_train_digits = preprocess_image(train_digits, to_grayscale)
grayscale_test_digits = preprocess_image(test_digits, to_grayscale)

model_gray = train_knn_model(grayscale_train_digits, K)

for cls in grayscale_test_digits.keys():
  predicts = model_gray.predict(flatten_images(grayscale_test_digits[cls]))
  print(cls, np.mean(predicts == cls))
```

```
zero 0.99
one 0.99
```

```
two 0.95
three 0.88
four 0.94
five 0.94
six 0.98
seven 0.92
eight 0.95
nine 0.95
```

4. KNN K = 1, HOG features with 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 histogram values x 4 per 16 by 16 block x 4 such blocks per 24 x 24 image.

[56]:
```python
def mod_HOG(images: List[np.ndarray]) -> List[np.ndarray]:
  '''
  to extract the HOG feature from the image
  '''
  winSize = (24,24)
  blockSize = (16,16)
  blockStride = (8,8)  # Overlapping of 50%
  cellSize = (8,8)
  nbins = 9  # Number of orientation bins
  hog = cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize, nbins)
  hog_descriptors = []
  for image in images:
      # Ensure the image is a numpy array
      if not isinstance(image, np.ndarray):
          image = np.array(image)
      # Ensure the image is grayscale
      if len(image.shape) > 2:
          image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
      hog_descriptor = hog.compute(image)
      hog_descriptors.append(hog_descriptor)
  return hog_descriptors


model_HOG = train_knn_model_HOG(grayscale_train_digits, K)

for cls in grayscale_test_digits.keys():
    predicts = model_HOG.predict(mod_HOG(grayscale_test_digits[cls]))
    print(cls, np.mean(predicts == cls))
```

```
zero 0.99
one 0.97
two 0.98
three 0.98
four 0.96
five 0.93
```

```
six 0.97
seven 0.95
eight 0.97
nine 0.95
```

3.1 2.2

Use KNN K = 1 with HOG features to report: 1. 5 pts. 1.0 hrs. Result for 2 test images per digit 0..9 cropped out from digits.png but not aligned at the original 20 x 20 image, so you may have smaller or bigger input image sizes. You must rescale each test image to 24 x 24 because HOG requires this scaling.

the image from 0-9

[57]:
```
import os
for imagename in os.listdir('./Q3/2_2_a/'):
  image = cv2.imread('./Q3/2_2_a/' + imagename)
  # rescue the image to 24*24
  resized_image = cv2.resize(image, (24, 24))
  # use the model to predict the image
  print("from image of ",imagename.split("_")[0], "got: ", model_HOG.
  ↪predict(mod_HOG([resized_image])))
```

```
from image of  0 got:  ['zero']
from image of  0 got:  ['zero']
from image of  1 got:  ['one']
from image of  1 got:  ['four']
from image of  2 got:  ['two']
from image of  2 got:  ['two']
from image of  3 got:  ['three']
from image of  3 got:  ['three']
from image of  4 got:  ['four']
from image of  4 got:  ['four']
from image of  5 got:  ['five']
from image of  5 got:  ['five']
from image of  6 got:  ['six']
from image of  6 got:  ['five']
from image of  7 got:  ['seven']
from image of  7 got:  ['one']
from image of  8 got:  ['eight']
from image of  8 got:  ['eight']
from image of  9 got:  ['three']
from image of  9 got:  ['nine']
```

2. 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.

I found the code that can write image with tkinter and I need to implement it to use only image and not save to disk.

[58]:
```
from PIL import ImageTk, Image, ImageDraw
import PIL
from tkinter import *
import numpy as np

width = 200  # canvas width
height = 200 # canvas height
center = height//2
black = (255, 255, 255) # canvas back

# Create a dictionary to store the images for each digit
digit_images = {i: [] for i in range(10)}

# Create two empty PIL images and draw objects to draw on
output_image1 = PIL.Image.new("RGB", (width, height), black)
draw1 = ImageDraw.Draw(output_image1)
output_image2 = PIL.Image.new("RGB", (width, height), black)
draw2 = ImageDraw.Draw(output_image2)

def save():
    global output_image1, output_image2, draw1, draw2
    # Save the images and add them to the dictionary
    for i, output_image in enumerate([output_image1, output_image2]):
        resized_image = output_image.resize((24, 24))

        resized_image = cv2.bitwise_not(np.array(resized_image))

        digit_images[digit.get()].append(resized_image)
    # Clear the canvases
    canvas1.delete("all")
    canvas2.delete("all")
    # Clear the output images
    output_image1 = PIL.Image.new("RGB", (width, height), black)
    draw1 = ImageDraw.Draw(output_image1)
    output_image2 = PIL.Image.new("RGB", (width, height), black)
    draw2 = ImageDraw.Draw(output_image2)

def paint(event, draw):
    x1, y1 = (event.x - 1), (event.y - 1)
    x2, y2 = (event.x + 1), (event.y + 1)
    event.widget.create_oval(x1, y1, x2, y2, fill="black",width=25)
    draw.line([x1, y1, x2, y2],fill="black",width=20)

master = Tk()

# Create a dropdown menu to select the digit
digit = IntVar(master)
```

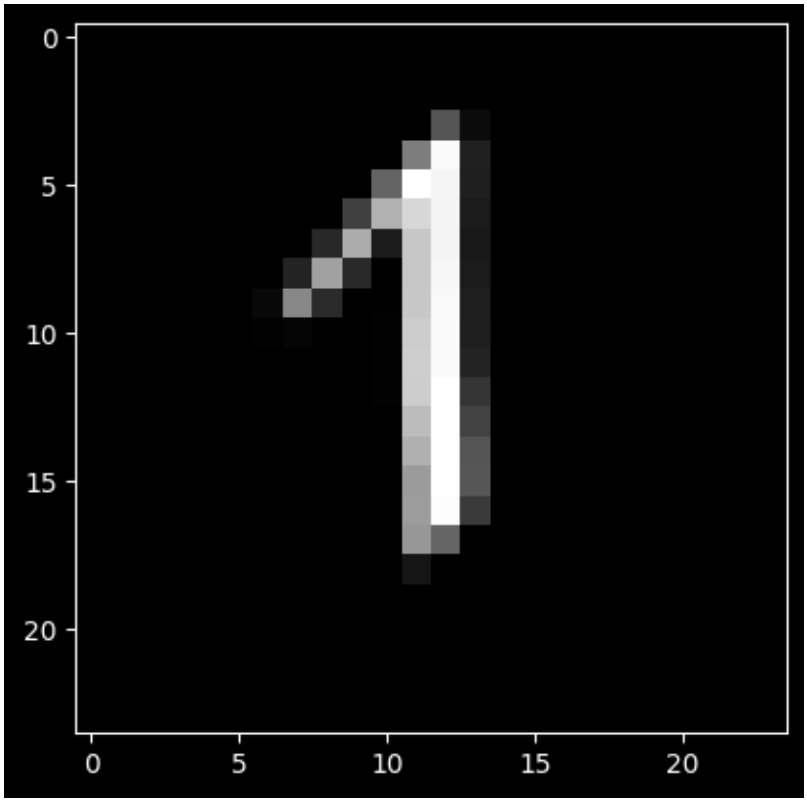
```
digit.set(0) # default value
digit_menu = OptionMenu(master, digit, *range(10))
digit_menu.pack()

# Create two tkinter canvases to draw on
canvas1 = Canvas(master, width=width, height=height, bg='white')
canvas1.pack()
canvas2 = Canvas(master, width=width, height=height, bg='white')
canvas2.pack()



canvas1.bind("<B1-Motion>", lambda event: paint(event, draw1))
canvas2.bind("<B1-Motion>", lambda event: paint(event, draw2))

# Add a button to save the images
button=Button(text="save",command=save)
button.pack()

master.mainloop()
```

[59]:
```
for i in range(10):
  print(f"Digit {i} has {len(digit_images[i])} images")
```

```
Digit 0 has 2 images
Digit 1 has 2 images
Digit 2 has 2 images
Digit 3 has 2 images
Digit 4 has 2 images
Digit 5 has 2 images
Digit 6 has 2 images
Digit 7 has 2 images
Digit 8 has 2 images
Digit 9 has 2 images
```

[60]:
```
plt.imshow(digit_images[1][0], cmap='gray')
```

[60]:
```
<matplotlib.image.AxesImage at 0x1ea9660b130>
```



[61]:
```
for i in digit_images.keys():
  for j in range(len(digit_images[i])):
    print("from image of ",i, "got: ", model_HOG.
↪predict(mod_HOG([digit_images[i][j]])))
```

```
from image of  0 got:  ['six']
from image of  0 got:  ['zero']
from image of  1 got:  ['one']
from image of  1 got:  ['one']
from image of  2 got:  ['two']
from image of  2 got:  ['two']
from image of  3 got:  ['seven']
from image of  3 got:  ['three']
from image of  4 got:  ['four']
from image of  4 got:  ['four']
from image of  5 got:  ['five']
from image of  5 got:  ['five']
from image of  6 got:  ['six']
from image of  6 got:  ['six']
from image of  7 got:  ['seven']
from image of  7 got:  ['seven']
from image of  8 got:  ['three']
```

```
from image of  8 got:  ['three']
from image of  9 got:  ['nine']
from image of  9 got:  ['nine']
```

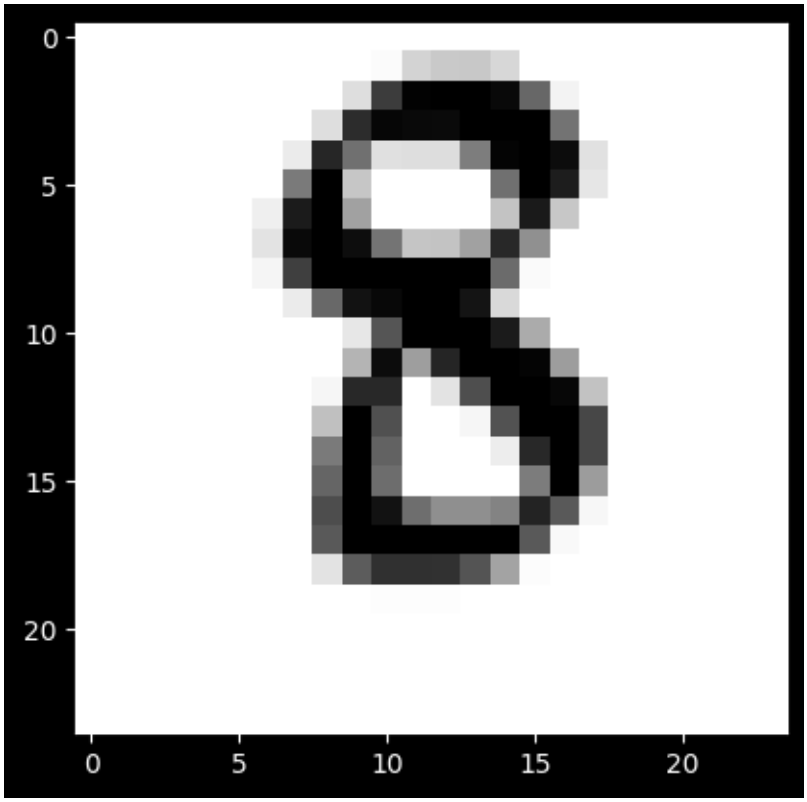
3. 5 pts. 1.0 hrs. Result for 4 test images of digit 5 you create in Paint with white background.

[62]:
```
from PIL import ImageTk, Image, ImageDraw
import PIL
from tkinter import *
import numpy as np

width = 200  # canvas width
height = 200 # canvas height
center = height//2
white = (255, 255, 255) # canvas back

# Create a dictionary to store the images for each digit
digit_images = {i: [] for i in range(10)}

# Create two empty PIL images and draw objects to draw on
output_image1 = PIL.Image.new("RGB", (width, height), white)
draw1 = ImageDraw.Draw(output_image1)
output_image2 = PIL.Image.new("RGB", (width, height), white)
draw2 = ImageDraw.Draw(output_image2)

def save():
    global output_image1, output_image2, draw1, draw2
    # Save the images and add them to the dictionary
    for i, output_image in enumerate([output_image1, output_image2]):
        resized_image = output_image.resize((24, 24))
        digit_images[digit.get()].append(np.array(resized_image))
        output_image.save(f"digit_{digit.get()}_{i}.png")
    # Clear the canvases
    canvas1.delete("all")
    canvas2.delete("all")
    # Clear the output images
    output_image1 = PIL.Image.new("RGB", (width, height), black)
    draw1 = ImageDraw.Draw(output_image1)
    output_image2 = PIL.Image.new("RGB", (width, height), black)
    draw2 = ImageDraw.Draw(output_image2)


def paint(event, draw):
    x1, y1 = (event.x - 1), (event.y - 1)
    x2, y2 = (event.x + 1), (event.y + 1)
    event.widget.create_oval(x1, y1, x2, y2, fill="black",width=25)
    draw.line([x1, y1, x2, y2],fill="black",width=25)
```

```
master = Tk()

# Create a dropdown menu to select the digit
digit = IntVar(master)
digit.set(0) # default value
digit_menu = OptionMenu(master, digit, *range(10))
digit_menu.pack()

# Create two tkinter canvases to draw on
canvas1 = Canvas(master, width=width, height=height, bg='white')
canvas1.pack()
canvas2 = Canvas(master, width=width, height=height, bg='white')
canvas2.pack()



canvas1.bind("<B1-Motion>", lambda event: paint(event, draw1))
canvas2.bind("<B1-Motion>", lambda event: paint(event, draw2))

# Add a button to save the images
button=Button(text="save",command=save)
button.pack()

master.mainloop()
```

[63]:
```
plt.imshow(digit_images[8][0], cmap='gray')
```

[63]:
```
<matplotlib.image.AxesImage at 0x1ea96581300>
```



[64]:
```
for i in digit_images.keys():
  for j in range(len(digit_images[i])):
    print("from image of ",i, "got: ", model_HOG.
↪predict(mod_HOG([digit_images[i][j]])))
```

```
from image of  0 got:  ['nine']
from image of  0 got:  ['zero']
from image of  1 got:  ['one']
from image of  1 got:  ['one']
from image of  2 got:  ['two']
from image of  2 got:  ['two']
from image of  3 got:  ['three']
from image of  3 got:  ['three']
from image of  4 got:  ['four']
from image of  4 got:  ['four']
from image of  5 got:  ['five']
from image of  5 got:  ['five']
from image of  6 got:  ['six']
from image of  6 got:  ['six']
from image of  6 got:  ['two']
from image of  6 got:  ['seven']
from image of  6 got:  ['six']
```

```
from image of  6 got:  ['six']
from image of  7 got:  ['seven']
from image of  7 got:  ['seven']
from image of  8 got:  ['eight']
from image of  8 got:  ['eight']
from image of  9 got:  ['four']
from image of  9 got:  ['nine']
```

3.2 2.3

20 pts. 2 hrs. For each 0..9 digit in your dataset of 100 characters, use OpenCV's auto threshold (Otsu's algorithm) and then the connected components to find the bounding box. Use that bounding box to cut out each original gray-scale image (not the thresholded 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?

[65]:
```
import cv2
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Assuming images is a list of your grayscale images
hog_features = []
image = cv2.imread('./Q3/digits.png')
print(image.shape)
uncropClass: Digits = {}
for i,key in enumerate(digits.keys()):
    uncropClass[key] = image[20*i*5:20*(i+1)*5, :]
    # to grayscale
    uncropClass[key] = cv2.cvtColor(uncropClass[key], cv2.COLOR_BGR2GRAY)
```

```
(1000, 2000, 3)
```

[66]:
```
uncropClass['zero'].shape
```

[66]:
```
(100, 2000)
```

[67]:
```
import cv2
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


cropClass: Digits = {
    "zero": [],
```

```
    "one": [],
    "two": [],
    "three": [],
    "four": [],
    "five": [],
    "six": [],
    "seven": [],
    "eight": [],
    "nine": []
}


for digit, images in uncropClass.items():
    # use Otsu's thresholding
    _, thresh = cv2.threshold(images, 0, 255, cv2.THRESH_BINARY + cv2.
 ↪THRESH_OTSU)
    # use connected component analysis to find the bounding box
    num_labels, labels, stats, centroids = cv2.
 ↪connectedComponentsWithStats(thresh, connectivity=8)
    print(f"Digit {digit} has {num_labels} connected components")
    for stat in stats[1:]:
        x, y, w, h, area = stat
        crop = images[y:y+h, x:x+w]
        cropClass[digit].append(cv2.resize(crop, (24, 24)))




for i in cropClass.keys():
  print(f"Digit {i} has {len(cropClass[i])} images")
```

```
Digit zero has 505 connected components
Digit one has 506 connected components
Digit two has 509 connected components
Digit three has 517 connected components
Digit four has 505 connected components
Digit five has 528 connected components
Digit six has 505 connected components
Digit seven has 509 connected components
Digit eight has 507 connected components
Digit nine has 515 connected components
Digit zero has 504 images
Digit one has 505 images
Digit two has 508 images
Digit three has 516 images
Digit four has 504 images
Digit five has 527 images
```

```
Digit six has 504 images
Digit seven has 508 images
Digit eight has 506 images
Digit nine has 514 images
```

[68]:
```python
# HOG
def HOG(images: List[np.ndarray]) -> List[np.ndarray]:
  '''
  to extract the HOG feature from the image
  '''
  winSize = (24,24)
  blockSize = (16,16)
  blockStride = (8,8)  # Overlapping of 50%
  cellSize = (8,8)
  nbins = 9  # Number of orientation bins
  hog = cv2.HOGDescriptor(winSize, blockSize, blockStride, cellSize, nbins)
  hog_descriptors = []
  for image in images:
      # Ensure the image is a numpy array
      if not isinstance(image, np.ndarray):
          image = np.array(image)
      # Ensure the image is grayscale
      if len(image.shape) > 2:
          image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
      hog_descriptor = hog.compute(image)
      hog_descriptors.append(hog_descriptor)
  return hog_descriptors

train_digits: Digits = {}
test_digits: Digits = {}

for key in cropClass.keys():
  train_digits[key], test_digits[key] = train_test_split(cropClass[key],␣
 ↪test_size=0.2, random_state=42)


K = 1
model_HOG = train_knn_model_HOG(train_digits, K)

for cls in test_digits.keys():
    predicts = model_HOG.predict(mod_HOG(test_digits[cls]))
    print(cls, np.mean(predicts == cls))
```

```
zero 0.9702970297029703
one 0.8712871287128713
two 0.9215686274509803
three 0.9134615384615384
four 0.8118811881188119
```

```
five 0.8679245283018868
six 0.9801980198019802
seven 0.9313725490196079
eight 0.9509803921568627
nine 0.9029126213592233
```

with Otsu's algorithm and then the connected components to find the bounding box. I might not better than the previous one.

[ ]: