นาย สัณหณัฐ พรมจรรย์ 63070501069

1) Connected Components

Your Output for 4-connected Labeling:

	1	1		2		3	3	3		4	4	4
5		1		2		3	3	3		4		4
5	5		6	2		3	3	3		4		4
	5		6		7	3		3	3			
	5	5		8	7	3		3	3	3		9
10		5		8	7	3						9
10	10			8	7	3	3	3	3	3		9
10	10	10		8			3					
				8	8	8		11	11	11		12
13	13	13		8		8		11		11		12
13	13	13	13	8		8	8	8		11	11	11
						8	8	8		11	11	11

Equivalence Table

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

Your Output for 8-connected Labeling:

1	1			2		3				4				5
1		1		2		3				4		6		5
1			1			3				4		6		5
	1		1		3			7			4	4	4	
	1			1				7				4		
		1		1				7	7	7	4	4		
			1	1								4		
				1		8	8		9	9	4	4	4	
	10	10	1	1				8				4		
10				1		11		8		12		4		13
10						11				12				13
10	10	10	10	10	10	10				12				13

Equivalence Table

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

2) Handwritten Digit Recognition

Link:https://colab.research.google.com/drive/11Ni0mylkZ4DjyC1L8GOIz5tG6g7iEA_R?usp=sharing

Source Code

```
[207] import numpy as np
      import cv2
      import matplotlib.pyplot as plt
      from sklearn.model_selection import train_test_split
      from skimage.color import rgb2gray
      from skimage.util import img_as_float
      from skimage.feature import hog
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.metrics import accuracy_score
```

```
[208] from google.colab.patches import cv2_imshow
      def my_imshow(title, img ):
        print(title)
        cv2_imshow(img)
```

```
[224] # Load the image containing the digits
      # img = cv2.imread('digits.png', 0)
      path = ""
      fileName = path +  "digits.png"
      img = cv2.imread(fileName, cv2.IMREAD_GRAYSCALE)

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

      plt.title("Input Image")
      plt.imshow(img, cmap='gray')
      plt.show()
```

Output

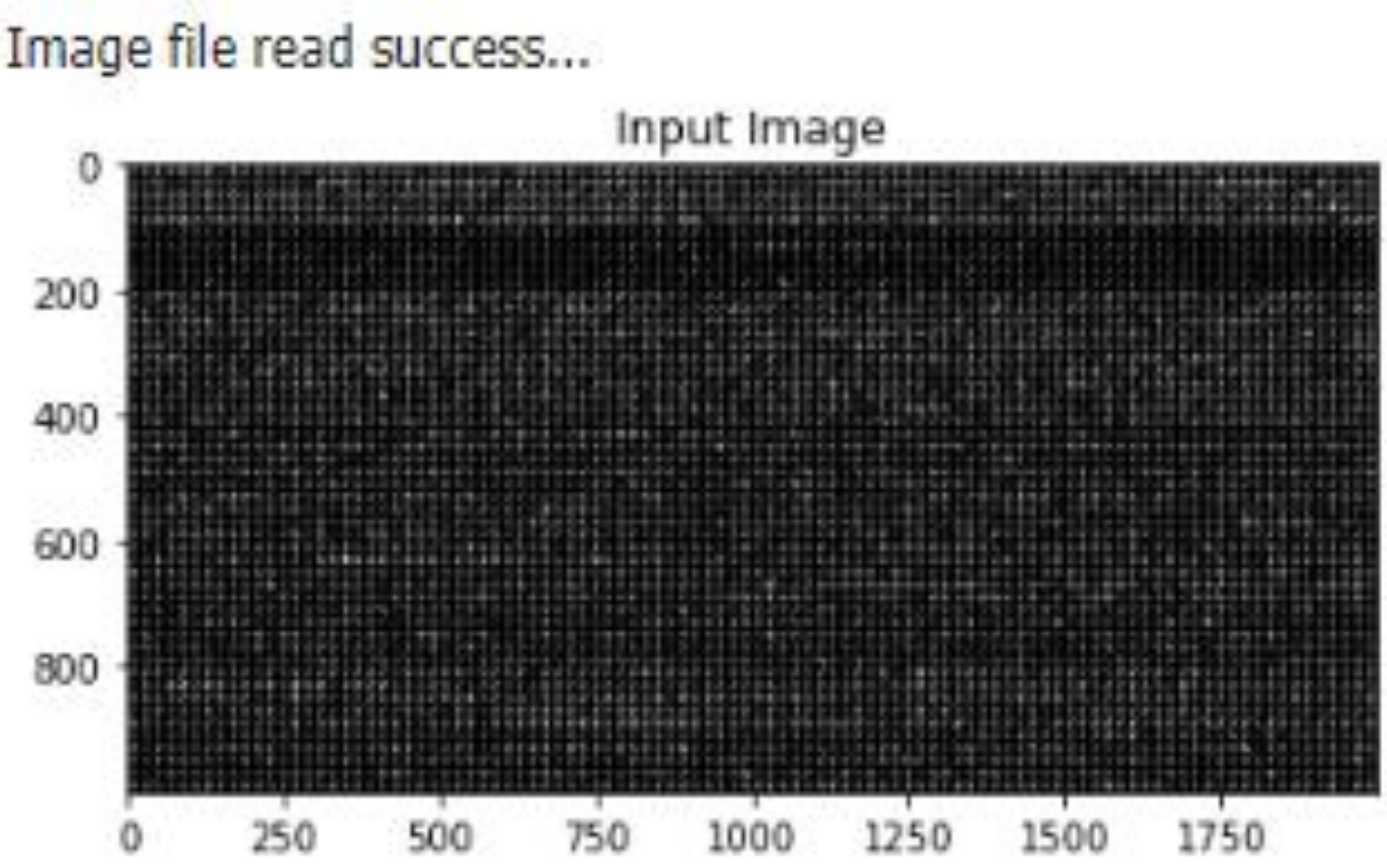
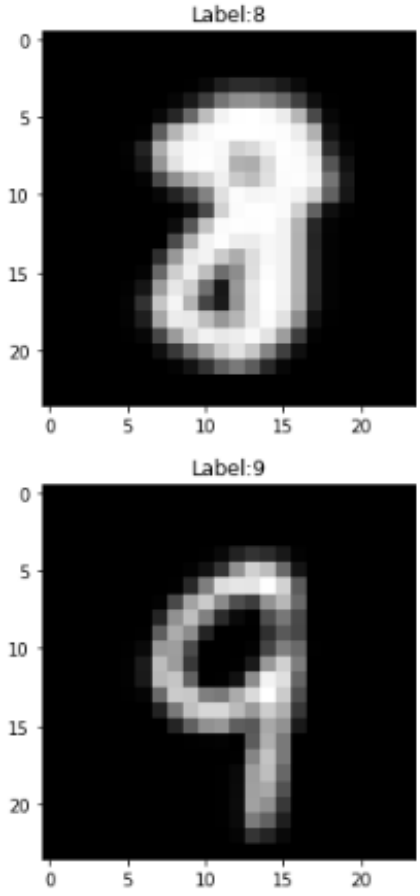


Image file read success...
Input Image
0
200
400
600
800
0
250
500
750
1000
1250
1500
1750

```
[215] # Reshape the image to extract each digit
      digits = np.array([np.hsplit(row, 100) for row in np.vsplit(img, 50)])
      digits = digits.reshape(-1, 20, 20)

      # Rescale  from 20 x 20 to 24 x 24
      digits_rescaled = []
      for digit in digits:
          rescaled_digit = cv2.resize(digit, (24, 24))
          digits_rescaled.append(rescaled_digit)

      # Convert the digit images to a numpy array and normalize the pixel values
      digits_rescaled = np.array(digits_rescaled, dtype=np.float32)

      # Define the labels for each digit
      labels = np.repeat(np.arange(10), len(digits)/10)

      # Split the data into training and testing sets
      training_features, testing_features, training_labels, testing_labels = train_test_split(digits_rescaled, labels, test_size=0.2, random_state=42)

      # Visualization a sample digit
      # Examples can be changed at training_features[i] and training_labels[i], Start i=0
      for i in range(2):
        plt.title("Label:" + str(training_labels[i]))
        plt.imshow(training_features[i], cmap='gray')
        plt.show()
        cv2.imwrite('output{}.png'.format(i+1), training_features[i])
```

Output



a)

```
[216] # Extract features from the images
      def extract_gray_scale_features(X):
          return np.mean(X, axis=(1,2)).reshape(-1, 1)

      def extract_HOG_features(data):
          features = []
          for image in data:
              # convert the image to grayscale
              if len(image.shape) == 3 and image.shape[-1] == 3:
                  gray_image = rgb2gray(image)
              else:
                  gray_image = img_as_float(image)
              # extract the HOG features
              hog_features = hog(gray_image, orientations=20, pixels_per_cell=(4, 4), cells_per_block=(2, 2), block_norm='L2-Hys', visualize=False, feature_vector=True)
              features.append(hog_features)
          return np.array(features)

      X_train_gray = extract_gray_scale_features(training_features)
      X_train_HOG = extract_HOG_features(training_features)
      X_test_gray = extract_gray_scale_features(testing_features)
      X_test_HOG = extract_HOG_features(testing_features)

      # Train KNN classifiers
      knn_5_gray = KNeighborsClassifier(n_neighbors=5).fit(X_train_gray, training_labels)
      knn_5_HOG = KNeighborsClassifier(n_neighbors=5).fit(X_train_HOG, training_labels)
      knn_1_gray = KNeighborsClassifier(n_neighbors=1).fit(X_train_gray, training_labels)
      knn_1_HOG = KNeighborsClassifier(n_neighbors=1).fit(X_train_HOG, training_labels)

      # Test classifiers and report accuracy
      def test_classifier(classifier, X_test, y_test):
          y_pred = classifier.predict(X_test)
          accuracy = np.mean(y_pred == y_test)
          return accuracy

      print("KNN K=5, gray scale features accuracy: %5.2f%%" % (test_classifier(knn_5_gray, X_test_gray, testing_labels)*100))
      print("KNN K=5, HOG features accuracy: %5.2f%%" % (test_classifier(knn_5_HOG, X_test_HOG, testing_labels)*100))
      print("KNN K=1, gray scale features accuracy: %5.2f%%" % (test_classifier(knn_1_gray, X_test_gray, testing_labels)*100))
      print("KNN K=1, HOG features accuracy: %5.2f%%" % (test_classifier(knn_1_HOG, X_test_HOG, testing_labels)*100))
```

Output

```
KNN K=5, gray scale features accuracy: 18.00%
KNN K=5, HOG features accuracy: 94.50%
KNN K=1, gray scale features accuracy: 14.70%
KNN K=1, HOG features accuracy: 96.00%
```

b)

ข้อ b.i

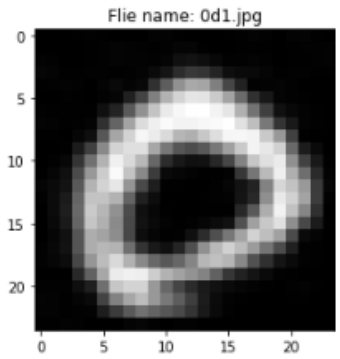
```
[217] # Create a KNN classifier with k = 1 and HOG features
      knn_1_HOG = KNeighborsClassifier(n_neighbors=1).fit(X_train_HOG, training_labels)
      print("KNN K=1, HOG features accuracy: %5.2f%%" % (test_classifier(knn_1_HOG, X_test_HOG, testing_labels)*100))
```

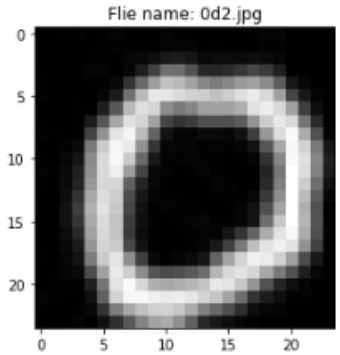
```
[256] # Rescale and compute HOG features for the test images
      # Path for test images
      test_image_path = "/content/drive/MyDrive/KMUTT/CPE393_ML/Quiz3/"

      # Rescale and compute HOG features for the test images
      test_image_files = ["0d1.jpg", "0d2.jpg", "1d1.jpg", "1d2.jpg", "2d1.jpg", "2d2.jpg", "3d1.jpg", "3d2.jpg", "4d1.jpg", "4d2.jpg",
                          "5d1.jpg", "5d2.jpg", "6d1.jpg", "6d2.jpg", "7d1.jpg", "7d2.jpg", "8d1.jpg", "8d2.jpg", "9d1.jpg", "9d2.jpg"]
      test_images = []
      for filename in test_image_files:
        image = cv2.imread(test_image_path + filename, cv2.IMREAD_GRAYSCALE)
        if image is not None:
          resized_image = cv2.resize(image, (24, 24))
          hog_features = hog(resized_image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2))
          test_images.append((resized_image, hog_features))
          plt.title("Flie name: " + filename)
          plt.imshow(resized_image, cmap='gray')
          plt.show()
        else:
          print(f"Could not read file {filename}")
```

Output

```
KNN K=1, HOG features accuracy: 96.00%
```

แสดงเป็นเลข 0-9



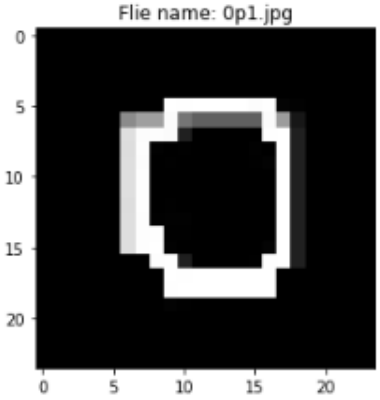


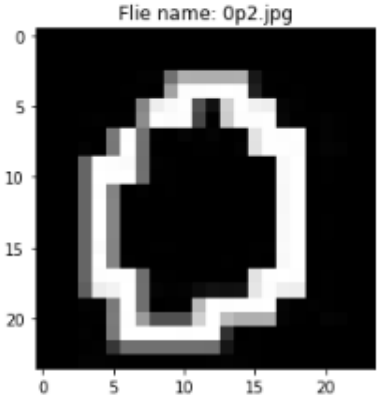
ข้อ b.ii

```
[254] # Rescale and compute HOG features for the test images
      # Path for test images
      test_image_path = "/content/drive/MyDrive/KMUTT/CPE393_ML/Quiz3/"

      # Rescale and compute HOG features for the test images
      test_image_files = ["0p1.jpg", "0p2.jpg", "1p1.jpg", "1p2.jpg", "2p1.jpg", "2p2.jpg", "3p1.jpg", "3p2.jpg", "4p1.jpg", "4p2.jpg",
                          "5p1.jpg", "5p2.jpg", "6p1.jpg", "6p2.jpg", "7p1.jpg", "7p2.jpg", "8p1.jpg", "8p2.jpg", "9p1.jpg", "9p2.jpg"]
      test_images = []
      for filename in test_image_files:
        image = cv2.imread(test_image_path + filename, cv2.IMREAD_GRAYSCALE)
        if image is not None:
          resized_image = cv2.resize(image, (24, 24))
          hog_features = hog(resized_image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2))
          test_images.append((resized_image, hog_features))
          plt.title("Flie name: " + filename)
          plt.imshow(resized_image, cmap='gray')
          plt.show()
        else:
          print(f"Could not read file {filename}")
```

Output

แสดงเป็นเลข 0-9

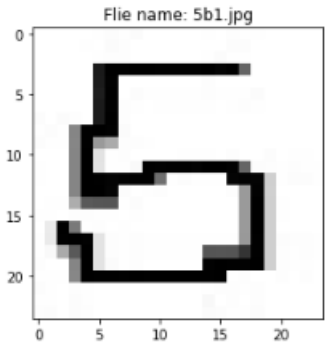


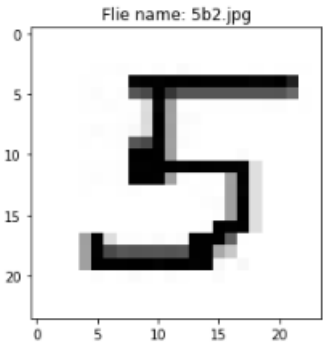


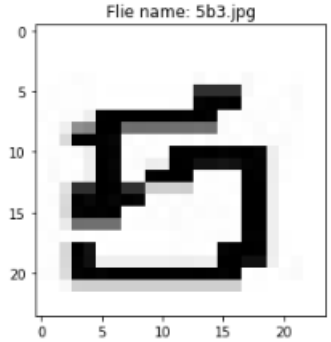
ข้อ b.iii

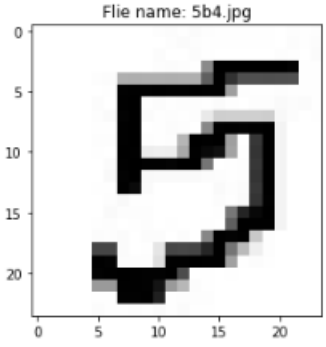
```
[12] true_labels = [5,5,5,5]
     test_digits = []
     test_hog_features = []
     test_labels = []

     # Rescale and compute HOG features for the test images
     # Path for test images
     test_image_path = "/content/drive/MyDrive/KMUTT/CPE393_ML/Quiz3/"

     # Rescale and compute HOG features for the test images
     test_image_files = ["5b1.jpg","5b2.jpg","5b3.jpg","5b4.jpg"]
     for filename, true_label in zip(test_image_files, true_labels):
         image = cv2.imread(test_image_path + filename, cv2.IMREAD_GRAYSCALE)
         if image is not None:
             image_resized = cv2.resize(image, (24, 24))
             test_digits.append(image_resized)
             hog_feature = hog(image_resized, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2))
             test_hog_features.append(hog_feature)
             test_labels.append(true_label)
             plt.title("Flie name: " + filename)
             plt.imshow(image_resized, cmap='gray')
             plt.show()
         else:
             print(f"Could not read file {filename}")
     test_hog_features = np.array(test_hog_features)
     test_labels = np.array(test_labels)
```

Output







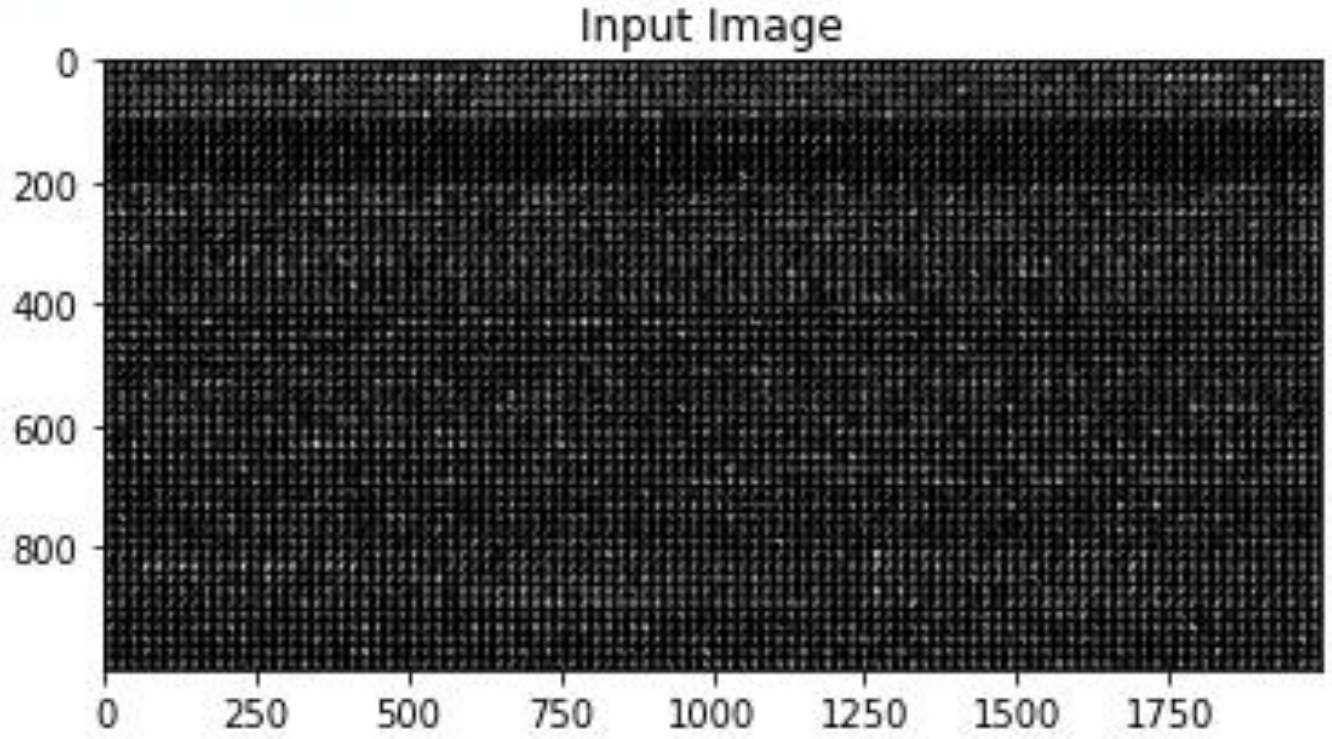


c)

```
[20] # Set the path to the directory containing the digits.png file
     path = ""

     # Load the dataset
     fileName = path + "digits.png"
     img = cv2.imread(fileName, cv2.IMREAD_GRAYSCALE)
     if img is None:
        print("Could not read image")
     else:
        print("Image file read success...")
        plt.title("Input Image")
        plt.imshow(img, cmap='gray')
        plt.show()
        # Resize the image to a uniform size of 5000 pixels per digit
        digit_size = 5000
        digits = cv2.resize(img, ((digit_size + 1) * 10, (digit_size + 1) * 5))

        # Split the resized image into individual digit images and store them in an array
        digits = [digits[y:y+digit_size, x:x+digit_size] for y in range(0, (digit_size + 1) * 5, digit_size + 1) for x in range(0, (digit_size + 1) * 10, digit_size + 1)]
        digits = np.array(digits)

        # Assign labels to each digit image (0 to 9)
        labels = np.repeat(np.arange(10), len(digits) // 10)

        # Apply automatic thresholding to each digit image and find the bounding box of the digit
        digit_images = []
        for digit in digits:
           _, thresh = cv2.threshold(digit, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
           contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
           x, y, w, h = cv2.boundingRect(contours[0])
           digit_image = digit[y:y+h, x:x+w]
           digit_image = cv2.resize(digit_image, (24, 24))
           digit_images.append(digit_image)

        # Compute the HOG features for each digit image
        hog_features = []
        for digit_image in digit_images:
           hog_feature = hog(digit_image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2))
           hog_features.append(hog_feature)
        hog_features = np.array(hog_features)
        labels = np.array(labels)

        # Split the dataset into training and testing sets
        train_hog_features, test_hog_features, train_labels, test_labels = train_test_split(hog_features, labels, test_size=0.2)

        # Train a KNN classifier with K = 1 on the HOG features of the training set
        knn = KNeighborsClassifier(n_neighbors=1)
        knn.fit(train_hog_features, train_labels)

        # Predict the labels for the testing set using the trained KNN classifier
        test_pred_labels = knn.predict(test_hog_features)

        # Compute the accuracy score for the predicted labels
        accuracy = 100*accuracy_score(test_labels, test_pred_labels)
        print("Hog Features:", hog_features.shape)
        print("Labels:", labels.shape)
        print("Accuracy: %5.2f%%" % accuracy)
```

Output

Image file read success...



Hog Features: (50, 144)

Labels: (50,)

Accuracy: 10.00%