Ayra Khan, OCR SYSTEMS

Code

|  |
| --- |
| # -\*- coding: utf-8 -\*-  """Train.ipynb  Automatically generated by Colaboratory.  Original file is located at  https://colab.research.google.com/drive/1HuakGtxoDBC91Babk2Z3piL4W6kNHyCL  # IMPLEMENTATION  ## Training  """  #import the necessary python modules  import numpy as np  from sklearn.metrics import confusion\_matrix  from scipy.spatial.distance import cdist  from skimage.measure import label, regionprops, moments, moments\_central, moments\_normalized, moments\_hu  from skimage import io, exposure  import matplotlib.pyplot as plt  from matplotlib.patches import Rectangle  import pickle  """## Reading Images and Binarization  #Extracting Characters and their Features  # Train.py  """  def extract\_features(image\_files, show\_plots=True):  th = 200 # Threshold for binarization  features = []  labels = []  regions\_info = []  for image\_file in image\_files:  # Reading image  img = io.imread(image\_file)  if show\_plots:  # Visualizing the original image  io.imshow(img)  plt.title(f'Original Image - {image\_file}')  io.show()  # Histogram  hist = exposure.histogram(img)  if show\_plots:  plt.bar(hist[1], hist[0])  plt.title(f'Histogram - {image\_file}')  plt.show()  # Binarization by thresholding  img\_binary = (img < th).astype(np.double)  if show\_plots:  io.imshow(img\_binary)  plt.title(f'Binary Image - {image\_file}')  io.show()  # Extracting Characters and their Features  img\_label = label(img\_binary, background=0)  if show\_plots:  io.imshow(img\_label)  plt.title(f'Labeled Image - {image\_file}')  io.show()  # Computing HU moments and removing small components  th\_component = 200 # Adjust this threshold as needed  regions = regionprops(img\_label)  fig, ax = plt.subplots()  ax.imshow(img\_binary, cmap='gray')  for props in regions:  minr, minc, maxr, maxc = props.bbox  regions\_info.append({'minr': minr, 'minc': minc, 'maxr': maxr, 'maxc': maxc})  # To omit noise and small size noise components  if maxr - minr <= th\_component and maxc - minc <= th\_component:  ax.add\_patch(Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False, edgecolor='red', linewidth=1))  # Compute HU moments  roi = img\_binary[minr:maxr, minc:maxc]  m = moments(roi)  cc = m[0, 1] / m[0, 0]  cr = m[1, 0] / m[0, 0]  mu = moments\_central(roi, center=(cr, cc))  nu = moments\_normalized(mu)  hu = moments\_hu(nu)  # Append HU moments to the features list  features.append(hu)  labels.append(image\_file.split('.')[0])  if show\_plots:  ax.set\_title(f"\nBounding Boxes and Moments - {image\_file}")  plt.show()  return features, labels, regions\_info  def normalize\_and\_save(features, labels, regions\_info):  features = np.array(features)  mean\_values = np.mean(features, axis=0)  standard\_dev = np.std(features, axis=0)  # store these files  normalize = (features - mean\_values) / standard\_dev  store = {'mean': mean\_values, 'std': standard\_dev, 'normalize': normalize, 'regions\_info': regions\_info}  np.savez('normalization\_train.npz', \*\*store)  # recognition on training data  D = cdist(normalize, normalize)  D\_index = np.argsort(D, axis=1)  io.imshow(D)  plt.title('Distance Matrix')  io.show()  neighbor = D\_index[:, 1] # neighbor  # confusion matrix  Ytrue = labels  Ypred = [labels[i] for i in neighbor]  confM = confusion\_matrix(Ytrue, Ypred)  io.imshow(confM)  plt.title('Confusion Matrix')  io.show()  def calculate\_recognition\_rate(test\_image\_file):  # Load normalization parameters  normalization\_data = np.load('normalization\_train.npz')  mean\_values = normalization\_data['mean']  std\_dev = normalization\_data['std']  # Read the test image  test\_img = io.imread(test\_image\_file)  # Binarization using threshold  th = 200  test\_img\_binary = (test\_img < th).astype(np.double)  # Labeling connected components  test\_img\_label = label(test\_img\_binary, background=0)  # Extracting characters and their features from the test image  test\_regions = regionprops(test\_img\_label)  test\_features = []  for props in test\_regions:  minr, minc, maxr, maxc = props.bbox  # Compute HU moments  roi = test\_img\_binary[minr:maxr, minc:maxc]  m = moments(roi)  cc = m[0, 1] / m[0, 0]  cr = m[1, 0] / m[0, 0]  mu = moments\_central(roi, center=(cr, cc))  nu = moments\_normalized(mu)  hu = moments\_hu(nu)  # Normalize using training data statistics  normalized\_hu = (hu - mean\_values) / std\_dev  # Append normalized HU moments to the features list  test\_features.append(normalized\_hu)  # Recognition on the test data  D = cdist(test\_features, normalization\_data['normalize'])  D\_index = np.argsort(D, axis=1)  neighbor = D\_index[:, 0] # Choose the closest neighbor  # Calculate recognition rate  correct\_recognitions = sum([1 for i in range(len(test\_regions)) if labels[neighbor[i]] == test\_image\_file.split('.')[0]])  total\_characters = len(test\_regions)  recognition\_rate = correct\_recognitions / total\_characters  return recognition\_rate  # Testing to see if it works  image\_files = ['a.bmp', 'o.bmp', 'd.bmp', 'm.bmp', 'n.bmp', 'p.bmp', 'q.bmp', 'r.bmp', 'u.bmp', 'w.bmp']  features, labels, regions\_info = extract\_features(image\_files, show\_plots=True)  # normalization and saving  normalize\_and\_save(features, labels, regions\_info)  # Calculate recognition rate for each image  for image\_file in image\_files:  recognition\_rate = calculate\_recognition\_rate(image\_file)  print(f'Recognition Rate for {image\_file}: {recognition\_rate \* 100:.2f}%') |

# Results

# A graph of letters and histograms Description automatically generatedA screenshot of a computer Description automatically generatedA screenshot of a computer Description automatically generatedA graph of a graph with circles Description automatically generated with medium confidenceA graph of circles on a black background Description automatically generatedA screenshot of a computer screen Description automatically generatedA screenshot of a computer screen Description automatically generatedA graph with numbers and lines Description automatically generatedA screenshot of a computer Description automatically generatedA screenshot of a computer Description automatically generatedA graph with a number of letters Description automatically generated with medium confidenceA graph with numbers and a bar Description automatically generatedA screenshot of a computer screen Description automatically generatedA screenshot of a computer screen Description automatically generatedA graph with numbers and text Description automatically generatedA screenshot of a computer screen Description automatically generatedA graph of a number of boxes Description automatically generated with medium confidenceA screenshot of a computer Description automatically generatedA graph and a diagram Description automatically generated with medium confidenceA chart with letters and numbers Description automatically generatedA screenshot of a computer Description automatically generatedA graph and chart with numbers Description automatically generatedA black and white chart with numbers Description automatically generatedA screenshot of a computer screen Description automatically generatedA screenshot of a computer Description automatically generatedA screenshot of a graph Description automatically generatedA screenshot of a computer Description automatically generatedA screenshot of a graph Description automatically generatedA screenshot of a computer Description automatically generatedA screenshot of a computer Description automatically generatedA graph of a graph of letters Description automatically generated with medium confidenceA screenshot of a graph Description automatically generatedA screenshot of a computer screen Description automatically generatedA screenshot of a graph Description automatically generatedA graph of a graph Description automatically generatedA graph of a confused matrix Description automatically generatedA screenshot of a computer screen Description automatically generated

|  |
| --- |
| # -\*- coding: utf-8 -\*-  """Test.ipynb  Automatically generated by Colaboratory.  Original file is located at  https://colab.research.google.com/drive/1vLXbWznBR41gvum-hwLDZhpASWMxclUn  """  #import the necessary python modules  import numpy as np  from sklearn.metrics import confusion\_matrix  from scipy.spatial.distance import cdist  from skimage.measure import label, regionprops, moments, moments\_central, moments\_normalized, moments\_hu  from skimage import io, exposure  import matplotlib.pyplot as plt  from matplotlib.patches import Rectangle  import pickle  import train  from train import extract\_features  from train import normalize\_and\_save  def extract\_features(image\_files, show\_plots=True):  th = 280 # Threshold for binarization  features = []  labels = []  regions\_info = []  for image\_file in image\_files:  # Reading image  img = io.imread(image\_file)  if show\_plots:  # Visualizing the original image  io.imshow(img)  plt.title(f'Original Image - {image\_file}')  io.show()  # Binarization by thresholding  th\_test = 200  img\_binary = (img < th\_test).astype(np.double)  if show\_plots:  io.imshow(img\_binary, cmap='gray')  plt.title(f'Binary Image - {image\_file}')  io.show()  # Extracting Characters and their Features  img\_label = label(img\_binary, background=0)  regions = regionprops(img\_label)  fig, ax = plt.subplots()  ax.imshow(img\_binary, cmap='gray')  for props in regions:  minr, minc, maxr, maxc = props.bbox  # To omit noise and small size noise components  th\_component = 200 # Adjust this threshold as needed  if maxr - minr <= th\_component and maxc - minc <= th\_component:  ax.add\_patch(Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False, edgecolor='red', linewidth=1))  # Compute HU moments  roi = img\_binary[minr:maxr, minc:maxc]  m = moments(roi)  cc = m[0, 1] / m[0, 0]  cr = m[1, 0] / m[0, 0]  mu = moments\_central(roi, center=(cr, cc))  nu = moments\_normalized(mu)  hu = moments\_hu(nu)  # Append HU moments to the features list  features.append(hu)  labels.append(image\_file.split('.')[0])  regions\_info.append({'minr': minr, 'minc': minc, 'maxr': maxr, 'maxc': maxc})  if show\_plots:  ax.set\_title(f"\nBounding Boxes - {image\_file}")  plt.show()  return features, labels, regions\_info  def recognize\_characters(normalized\_test, normalize\_train):  # Calculate the distance matrix  D\_test = cdist(normalized\_test, normalize\_train)  # Display the transposed distance matrix as an image  io.imshow(D\_test.T) # Transpose the matrix here  plt.title('Distance Matrix')  io.show()  # Find the best match for each character in the test image  neighbor\_test = np.argmin(D\_test, axis=1)  return D\_test, neighbor\_test  def normalize\_data(data, normalization\_data):  mean = normalization\_data['mean']  std\_dev = normalization\_data['std']  normalized\_data = (data - mean) / std\_dev  return normalized\_data  def find\_k\_nearest\_neighbors(test\_instance, data, k):  distances = cdist([test\_instance], data, 'euclidean')  indices = np.argsort(distances)[0][:k]  return indices  def get\_valid\_indices(nearest\_indices, test\_labels):  valid\_indices = [idx for idx in nearest\_indices if test\_labels[idx] != 'test']  return valid\_indices  def majority\_vote(labels):  unique\_labels, counts = np.unique(labels, return\_counts=True)  max\_count\_idx = np.argmax(counts)  return unique\_labels[max\_count\_idx]  def calculate\_recognition\_rate\_test(test\_features, test\_labels, normalization\_data, ground\_truth\_labels, k=1):  normalized\_test\_features = normalize\_data(test\_features, normalization\_data)  recognition\_count = 0  total\_count = len(test\_labels)  for i in range(len(ground\_truth\_labels)):  nearest\_indices = find\_k\_nearest\_neighbors(normalized\_test\_features[i], normalized\_test\_features, k)  valid\_indices = get\_valid\_indices(nearest\_indices, ground\_truth\_labels)  # print(f"i: {i}, nearest\_indices: {nearest\_indices}, valid\_indices: {valid\_indices}, "  # f"ground\_truth\_label: {ground\_truth\_labels[i]}, test\_label: {test\_labels[i]}")  if valid\_indices:  # Map the labeled components back to their original order  mapped\_labels = [ground\_truth\_labels[idx] for idx in valid\_indices]  predicted\_label = majority\_vote(mapped\_labels)  # print(f" Mapped Labels: {mapped\_labels}")  # print(f" Predicted Label: {predicted\_label}")  if predicted\_label == ground\_truth\_labels[i]:  recognition\_count += 1  print(" Recognition: Correct")  else:  print(f" Recognition: Incorrect (Predicted {predicted\_label}, Ground Truth {ground\_truth\_labels[i]})")  recognition\_rate = (recognition\_count / total\_count) \* 100  print(f"Recognition rate for Test Set: {recognition\_rate:.2f}%")  # Load parameters from the normalization file  normalize\_val = np.load('normalization\_train.npz', allow\_pickle=True)  mean\_values = normalize\_val['mean']  standard\_dev = normalize\_val['std']  regions\_info\_train = normalize\_val['regions\_info']  # Read the test image  test\_image = io.imread('test.bmp')  # Binarization  th\_test = 200  binary\_test = (test\_image < th\_test).astype(np.double)  # Extract features  test\_features, test\_labels, test\_regions\_info = extract\_features(['test.bmp'], show\_plots=False)  # Load ground truth information for the test set  pkl\_file\_test = open('test\_gt\_py3.pkl', 'rb')  mydict\_test = pickle.load(pkl\_file\_test)  pkl\_file\_test.close()  ground\_truth\_classes\_test = mydict\_test[b'classes']  ground\_truth\_locations\_test = mydict\_test[b'locations']  # Convert the list of features to a NumPy array  test\_features\_array = np.array(test\_features, dtype=np.float64)  # Normalize the test features using the mean and standard deviation from training  normalized\_test\_features = []  for feature in test\_features\_array:  normalized\_feature = (feature - mean\_values) / standard\_dev  normalized\_test\_features.append(normalized\_feature)  # Call the recognize\_characters function to get the distance matrix and best match indices  distance\_matrix, \_ = recognize\_characters(normalized\_test\_features, normalize\_val\_2d)  # Print the distance matrix  print("Distance Matrix:")  print(distance\_matrix)  # Calculate recognition rate for the test set  calculate\_recognition\_rate\_test(normalized\_test\_features, test\_labels, normalize\_val, ground\_truth\_classes\_test) |

# Results

A screen shot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generatedA screenshot of a computer screen

Description automatically generated

A comparison of a test

Description automatically generated with medium confidenceA purple and yellow bar

Description automatically generated

# Explanation of Code

Custom Functions:

1. extract\_features: Extracts Hu Moments and region information from images, focusing on character recognition.
2. recognize\_characters: Computes the distance matrix between test and training features, aiding in identification.
3. normalize\_data: Normalizes data using mean and standard deviation to maintain consistency in feature scales.
4. find\_k\_nearest\_neighbors: Locates k-nearest neighbors to assist in recognizing characters.
5. get\_valid\_indices: Filters and retrieves valid indices based on test and ground truth labels.
6. majority\_vote: Implements a majority voting mechanism to determine recognized labels.
7. calculate\_recognition\_rate\_test: Measures the recognition rate for the test set, assessing the accuracy of the recognition process.

# \*\*Report on Connected Component Analysis and Recognition Results\*\*

In the initial phase of the project, connected component analysis (CCA) was applied to all training images to identify and isolate individual characters. The resulting images were visualized with bounding boxes, providing a clear representation of the segmentation process. These bounding boxes were then utilized to extract features and train the recognition model.

During the recognition phase, the distance matrix (D) was calculated using the Euclidean distance metric between normalized test features and the training set. The visualization of the distance matrix aided in understanding the similarity between characters. It was observed that a threshold value of 200 was effective for binarization, striking a balance between noise removal and retaining character details.

For the test image, varying the threshold values proved to be a crucial enhancement. By increasing the threshold up to 300, a notable improvement in recognition rates was achieved. However, exceeding this threshold led to disordered bounding boxes. Conversely, lowering the threshold to 100 resulted in blurry images and ineffective bounding box detection.

In terms of recognition rates, the experimentation with thresholds unveiled a trade-off: higher thresholds improved recognition at the expense of bounding box accuracy, while lower thresholds sacrificed recognition rates due to image blurriness. Therefore, the selected threshold of 200 served as a compromise for balanced results.

The number of components obtained for the test image was influenced by the chosen threshold. Higher thresholds reduced the number of components, whereas lower thresholds increased it. Recognition rates were found to be positively correlated with higher thresholds but were hindered when exceeding the optimal threshold of 300.

Despite successful threshold-based enhancements, challenges were encountered when combining multiple enhancements. The compromise achieved at threshold 200 proved to be the most stable, as further adjustments in either direction led to diminishing returns or adversely affected bounding box accuracy.

In conclusion, the threshold parameter played a pivotal role in optimizing the recognition system. The delicate balance between noise reduction, character clarity, and bounding box accuracy was achieved at a threshold of 200. Ongoing experimentation with additional features and algorithms may provide further insights and improvements in future iterations of the recognition system.

# \*\*Automated Threshold Selection Enhancement: Unsuccessful Attempt\*\*

One enhancement approach involved automating the threshold selection process to dynamically find the optimal threshold for binarization. The rationale behind this enhancement was to eliminate the need for a fixed, hard-coded threshold, allowing adaptability to different image characteristics. To achieve this, attempts were made to implement methods that analyze the images and determine the most suitable threshold.

However, this enhancement proved unsuccessful as it encountered challenges related to index out-of-bounds errors. The automated threshold selection process faced difficulties in computing the optimal threshold, resulting in an inability to generate the necessary values. The main reason behind this failure is attributed to the complexity of the image analysis methods and their interaction with the specific characteristics of the dataset. As a result, the system struggled to generalize across various images, leading to errors and rendering the automated threshold selection impractical in its current implementation.

As an alternative solution, a fixed threshold of 200 was retained, as it offered a satisfactory balance between noise reduction and character clarity, as observed during manual experimentation. While the attempt to automate threshold selection did not yield the desired results, it highlights the challenges in creating a universally applicable, adaptive solution for this specific context. Future improvements may involve more sophisticated image analysis techniques or machine learning algorithms to enhance adaptability and overcome the limitations encountered in the current implementation.