Department of Computer Science, Rutgers University

CS334 – Imaging and Multimedia

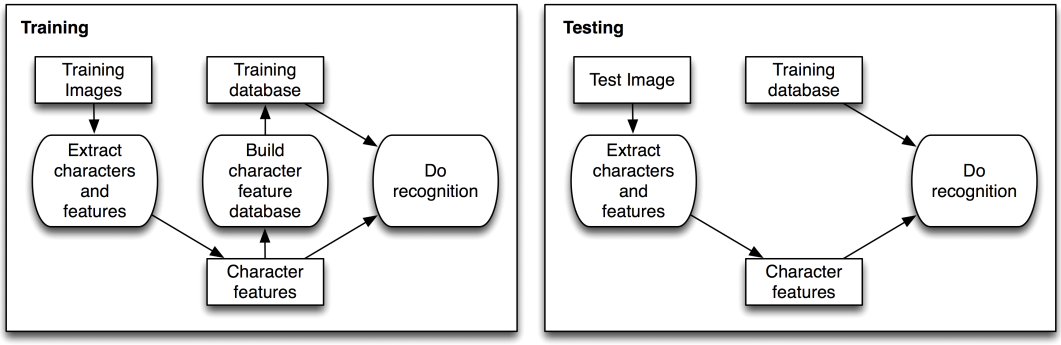
Assignment 4

• Introduction

This assignment is supposed to be a tutorial assignment that will lead you step by step to use Python 3 and built-in functions from several Python libraries to build an optical character recognition (OCR) system for hand-written characters, as a practice on binary image analysis.

Now you are ready to go!

• Problem Overview



In this assignment you will be given grayscale images of hand-written characters, where you will need to **identify** (extract) and **recognize** (classify) each character.

The assignment has two phases: Training and recognition. After completing these, you will be asked to improve your recognition rates by providing your own ideas, which is the enhancement part.

For both training and recognition, you will need to convert the grayscale images to binary images, identify each separate character instance appearing in the image and extract some visual features from these character instances.

In the training phase, you will build (learn) a 'database' of features that you extracted from a given set of character images. These features will be put inside a matrix that will be later used as a recognition database. At this phase you will know what each character is (its label) and you will also store this information along with the extracted features.

In the recognition phase, you will use the built database of features, the associated character labels and the features you extracted from your test image to recognize each character in this test image. Features extracted for each character instance in the test image will be compared to all the features in the database and the label of the 'best' matching character instance in the database will be the recognition result for this test character instance (this is called nearest neighbor classifier).

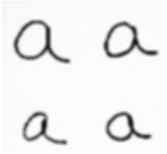
For crosscheck you will also be performing the same recognition step for the training images and measure the how well you have performed. This way you will know if your features are descriptive enough for your characters and you will also have the chance to catch possible errors in your implementation earlier.

• Given Files

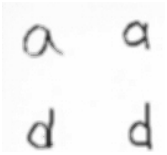
There are several images and code files we supplied with the assignment.

• Images

Training Image Set: You are given a set of 10 images (‘a.bmp’, ‘d.bmp’,… , ‘w.bmp’), each with different instances of one character in it. Images can be of different sizes.


Part of the training image ‘a.jpg’

Test Image: You are also given one test images for evaluation (‘test1’), which contains different instances of the hand-written characters in the training set, combined in one image.


A part of the test image ‘test1’

• Implementation

• Training

• Importing necessary Python modules

In the very beginning of your code, you need to import the following 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
```

These modules contain functions that you need to use for this assignment.

- ## Reading Images and Binarization

- ### Reading an Image File

We can read an image file into a matrix using the function `io.imread()`. For example, to open the image with character 'a' use:

```
img = io.imread('a.bmp');
```

Now, the variable `img` contains the image as a matrix. Check out the size of the image using:

```
print(img.shape)
```

The result will be (image_height, image_weight) (e.g., (750, 600)).

- ### Visualizing an Image/Matrix

We can visualize the loaded image by `io.imshow()` followed by `io.show()`:

```
io.imshow(img)
plt.title('Original Image')
io.show()
```

Note that simply calling `io.imshow(img)` won't show anything, until a call to `io.show()` is made.

- ### Image Histogram

Now we will look into the histogram of the image pixel intensities:

```
hist = exposure.histogram(img)
```

We can visualize the histogram as following:

```
plt.bar(hist[1], hist[0])
plt.title('Histogram')
plt.show()
```

You should see most of the image intensities are concentrated at a large value (~250). This is expected since the image background is mostly white.

- ### Binarization by Thresholding

Given this histogram we can choose a threshold to obtain binary image from the grayscale image (binarize the image). It is up to you to choose a suitable threshold. You can try different values and see the effect on the resulting binary image. To do this, first we define a variable called `th` for the threshold and set it to a certain value, say 200. Then, we use logical operation to find intensity values greater (smaller) than `th` and assign these pixels to 0 (1).

```
th = 200
img_binary = (img < th).astype(np.double)
```

- **Displaying Binary Image**

Similarly with above, we can visualize the binary image by

```
io.imshow(img_binary)
plt.title('Binary Image')
io.show()
```

• Extracting Characters and Their Features

- **Connected Component Analysis**

Given the binary image we have, we can now run connected component analysis to label each character with a unique label. To do this, we can use the `label()` function, which performs connected component analysis on the image and return a labeled image where all the pixels in each component are given an integer label 0, 1, 2,... where 0 is the background.

- `img_label = label(img_binary, background=0)`

You can visualize the resulting component image:

```
io.imshow(img_label)
plt.title('Labeled Image')
io.show()
```

In this figure each component has a different color since it has a different label. To find out how many connected components are in the image, you can find the maximum label value appearing in the labeled image using:

```
print(np.amax(img_label))
```

You can see the number of components is a little more than the number of characters in the page. This is due to small isolated components that are mainly noise. Usually this is called salt and pepper noise. This can be removed using mathematical morphology. Or you can try to omit small size components from further analysis by simply comparing their height and width to certain threshold.

A sample character instance from character ‘o’ and its thresholded version. You can see the small component that will be identified as another letter instance in the labeling process and thus has to be removed before feature extraction.

- **Displaying Component Bounding Boxes**

For each component you can find out and visualize the bounding box containing it using the following piece of code that loops through the components and find the maximum and minimum of their coordinates:

```
regions = regionprops(img_label)
io.imshow(img_binary)
ax = plt.gca()
```

```
for props in regions:
    minr, minc, maxr, maxc = props.bbox
    ax.add_patch(Rectangle((minc, minr), maxc - minc, maxr - minr,
fill=False, edgecolor='red', linewidth=1))
ax.set_title('Bounding Boxes')
io.show()
```

Note that `regionprops()` returns all connected components in a list, where each element contains a set of properties for each component. Details about this function can be found here:

http://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.regionprops.

- **Computing Hu Moments and Removing Small Components**

In this step we will compute the Hu moments and other statistical measures for each character. For each bounding box `props.bbox`, we get the corner coordinates `(minr, minc, maxr, maxc)` as shown above, then we do this:

```
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)
```

All you need to do is to insert these lines into the previous loop at a proper position.

It would be useful to omit small size noise components as mentioned above before calling the moment function. Just add an 'if' statement to compare components height and width with a given size threshold, and ignore those components that are too small. Experiment and visualize your bounding boxes until you find a good size threshold.

- **Storing Features**

The next step is to modify the above code in order to store the resulting features for each character to be used in recognition. To do this, simply create an empty list before the loop, let's call it '`Features`' using:

```
Features=[]
```

Then, inside the loop you need to concatenate each character features to the existing feature list. At the end, '`Features`' will contain one row for each character with 7 features in each row (7 dimensions of Hu moments). The concatenation can be done using:

```
Features.append(hu)
```

- ## Building Character Features Database for Recognition

- ### Creating a File to Process Each Image

Now that we have extracted features for one image, the final part of training phase is to extract the features for all the characters in all the images given to you to create a database of character features to be used in recognition.

You will need to use the above steps to process all the character images and to extract features for all characters and put them into one big features list as above. Modify the above code by adding all the necessary steps from above for reading the image, binarizing, extracting components, etc. into one .py file where you can call it for different images. Make this file a function file with the name **train.py**. The file should include a function definition and would need to get all the necessary information as input parameters and should use output parameters to return the outputs. It would also be nice to have a parameter that controls whether all the plots are displayed or not.

Of course, you need a way to remember what is the character class for each row in the Features list, since there may not be equal number of character instance for each character. One way to do that is to use another array with the same number of rows as Features, where in each row you keep the corresponding class label, i.e., 1 for 'a', 2 for 'd', 3 for 'm' etc. Another way is to keep the labels in the Features list as the first or the last column. If you follow this way, you will need to pay a lot of attention not to include the labels column in any of the normalization, training or recognition steps. You can also come up with your own method to associate the labels to the features.

- ### Normalization

Once you create the big Features list and the corresponding class labels you are ready to do recognition. In this project we will just use a simple nearest neighbor approach to find the closest character in the database for a given test character.

One problem is that different features have different ranges, so that a large numerical difference in a feature with high variability may swamp the effects of smaller, but perhaps more significant differences in features that have very small variability. The standard solution is to transform all of the feature distributions to a standard distribution with **0 mean** and **variance of 1.0**. This is done by first computing the mean and standard deviation of all the features (this is done over the entire set of training characters and not for one character type at a time), and then normalizing the features by subtracting the mean and dividing by the standard deviation on each feature. It would be nice to store these means and variances so that you can reuse them in the testing (recognition) phase.

- ### Recognition on Training Data

To evaluate the performance on the training data you can find the nearest neighbor for each character in the training data and check if its class matches the correct class. Since the particular character instance itself was in the training, its feature vector will also be in the feature database and definitely this will be the closest feature vector (with distance zero). For the case of recognition on the training data we will need to find the second closest row to the feature vector we are looking for. Hopefully, the next best match will correspond to another instance of the same letter, but sometimes other letter instances can be closer.

We will use a function provided with this assignment called `cdist()` which returns the **Euclidean distance** between all pairs of points. We will use it to evaluate the distance between each character and all other characters. The distance between the row vectors in the Normalized Features matrix can be found using:

```
D = cdist(Features, Features)
```

The resulting `D` is an NxN matrix where N is the number of characters (number of rows of Features). You can visualize `D` as an image using:

```
io.imshow(D)
plt.title('Distance Matrix')
io.show()
```

For example `D(1, 5)` will give you the distance between character instance 1 and character instance 5. Obviously `D` will have zeros on the diagonal since the distance between each character and itself is 0. To find the nearest neighbor for each character (excluding itself) you need to find the second smallest distance in each row in the `D` matrix. One way to do this is to sort the columns of `D` along each row and to find the index of the second smallest distance in each row:

```
D_index = np.argsort(dist, axis=1)
```

The `D_index` matrix contains the index of the columns in `D` sorted according to the. The second column of `D_index` will contain the index of the closest match to each character (excluding itself). Find the class for this closest match by looking at the label corresponding to this instance in the label vector you constructed earlier.

- **Confusion Matrix**

You can compute the confusion matrix between character classes given the built-in function `confusion_matrix(Ytrue, Ypred)`, which takes as input, the correct classes Ytrue (as a vector) and the output classes Ypred (as a vector). The confusion matrix elements will contain a normalized measure of how much each class is confused (recognized wrongly as) with another. This matrix is not necessarily symmetric, since i.e. 'a' can be recognized as 'o', but 'o' is recognized as 'u'. The diagonal elements in the matrix are the cases where character classes are recognized correctly. Visualize this matrix and try to understand the reasons of very obvious confusions between several classes:

```
io.imshow(confM)
plt.title('Confusion Matrix')
io.show()
```

• Testing (Recognition)

For evaluation you are given a test image (test.jpg) with instances from all characters. You will need to do the whole processing for this image and extract the features for each character. You will need to normalize the extracted features using the same means and standard deviations, which were computed from the training data. Then using the character features database obtained above and the function `cdist()`, find the nearest neighbor match for each character in the test image. As opposed to training phase, this time find the best match instead of the second best. You can create a file named **test.py** and do the following:

• Reading Image and Binarization

Read the test image and binarize it for feature extraction. You should be using exactly the same process you followed in the training phase; you cannot change any of the processing steps or any of the parameters specifically for testing. Normally, you should not be using the test data to improve your algorithms, which may cause over tailoring of your algorithms for this particular test data; but for this assignment you can change your algorithms by looking at how they perform on your test data.

• Extracting Characters and Their Features

At this step make sure you fix all the problems you can fix related to small pieces of characters that may appear as different components.

When storing extracted features from the test image, this time you will not be saving any label for the characters since this is the information you want to obtain from the our recognition system.

Make sure you performed the normalization using the mean and variance stored for each feature dimension in the training phase. Do NOT normalize test data using mean and variance of the test data features themselves.

In the testing (recognition) phase, you will be calculating a distance matrix, but not a confusion matrix.

• Enhancements

After fully completing the training and testing (recognition) parts, you can experiment with different ideas in order to improve your results. Try to fix any problems associated with the previous parts of your code before attempting to enhance it. If an enhancement is a successful one, it should increase recognition rate of the test images. An enhancement that increases one recognition rate, but decreases the other is not a good one.

Improving your results using different enhancements will contribute up to 20% of the assignment grade.

An enhancement can be completely experimental or it can be a remedy for a shortcoming that you observed in the current setting. If you are specifically testing an enhancement that should fix a problem in the current setting, try to observe/measure the resulting improvement on the problem you were targeting, instead of only measuring the recognition rate improvement. It may be a good idea to test improvements independently first, then combined together. You would not want to test ten different enhancements and be not sure which one made an improvement and which made it worse.

Make sure you document every thing in your report, even the unsuccessful ideas and trials.

In your report we would like to see:

- What you observed: A problem or an unsatisfactory point.
- What you deducted from it: Why this is happening, what is the main reason of this.
- What solution you came up with: A candidate solution designed to solve this particular problem.
- How this solution worked: Improvements in recognition rate from this specific enhancement and how the problem is solved, maybe with sample images or measurements. If failed to fix the problem or it introduced another problem, why do you think it failed.

• Enhancement Ideas

- Automate the threshold selection process: instead of a fixed hard-coded threshold, find methods that analyze the images and find the best threshold.
- Use binary morphology: Using binary morphology might be useful if you have fragmented characters that you cannot get rid of solely by improving the thresholding. There could be many other uses for binary morphology.
- Investigate different shape and region descriptors to extract invariant features to be used in the recognition: More invariant moments, shape profiles, contour descriptors, etc. Make sure you perform a meaningful normalization on each new feature you included. Sometimes you will not need to normalize a feature using mean and variance, but maybe using its known bounds.

- Use a better classifier: Instead of the nearest neighbor (closest distance) from the features' database for recognition, you can find the k-nearest neighbors (k is small number 3, 5, 7) and do a majority vote. If you have a Machine Learning background and you think another classifier will perform better, go ahead and try it. Document everything you do and make sure you do not concentrate only on the classification problem, since this is a Computer Vision assignment.

You don't have to investigate all the above possibilities. These are just some ideas. You need to come up with a way to enhance the recognition rate. You should use the training data for validating your proposed approach and the test image for testing only (no training done on it). Keep in mind that your final code will be evaluated with a similar image, which will not be given to you. Your grade in this part will be based on how much improvement you can do over the base line established in previous parts.

• Expected Deliverables

• Report Document

Put all your deliverables in the report with the code and **submit a soft copy to Sakai**. The document must be either in **PDF**, **DOC** or **DOCX** format; reports in any other format **will NOT be graded**. Embed all figures in your report in correct order and properly label them. You do NOT need to submit your figures as separate image files.

The report should contain the following.

• Code Files

You should submit all the code that you have written. Do NOT use other people's code or code files you found from Internet; however if it is absolutely necessary to use some external code, make sure you submit these files too. Submitting a non-working assignment may affect your score. You can use any additional Python modules if necessary.

• RunMyOCR.py

Submit your final version of your code so that the TA can run it on some other images to evaluate performance. Create a function that takes an image filename as an input, runs your training code on the training data, runs your testing (recognition) code on the given test image and reports testing recognition rates. The function should also create all the images appearing in the report, but the most important figure is the final recognition figure on the given test image. This figure should be the connected component test image with detected bounding boxes and corresponding recognized character classes.

To be able to generate recognition rates for any given test image, you need to have access to the groundtruth character locations and their correct classes. Your function needs to have this information as two separate matrices: **Locations** and **classes**, which are given in this assignment. Locations will be an Nx2 matrix of rough center coordinates of characters and classes will be an Nx1 matrix of the corresponding class label for each character, where N is the number of characters in the image. You will need to understand that labeling process may change the order of how the components are labeled and because of this, you cannot just feed class label corresponding to each component number directly as groundtruth. To compare to this groundtruth, you can go over your detected and recognized components and see if they contain any one of the given groundtruth character centers. If a center is in a detected component, then you can compare its groundtruth label and the recognized label. This way you can create a recognition rate finding mechanism that does not require that you have the same number of components as your actual characters.

In order to load the ground truth locations and classes, you can do this:

```
pkl_file = open('test2_gt.pkl', 'rb')
mydict = pickle.load(pkl_file)
pkl_file.close()
classes = mydict['classes']
locations = mydict['locations']
```

Now `classes` contain the ground truth labels for all characters and `locations` contain their center coordinates.

• Results to report

For All Training Images

- Connected component image with bounding boxes and recognition results

For Recognition Phase

- Distance matrix (`D`)
- Test image connected components with bounding boxes and recognition results
- Test image connected components with bounding boxes and recognition results - For every (successful) enhancement
- Test image connected components with bounding boxes and recognition results - For all enhancements combined.

• Recognition Rates and Other Values

In your report list the following values:

- Threshold value you have picked, or any algorithm you used to find a threshold
- Number of components you obtained for test image
- Recognition rate for the test image
- Recognition rate for test image after each enhancement
- Recognition rate for test image after all enhancement combined

• Tested/Used Features

List every different feature that you have used in your code. Try to justify why you think they will work. Document your experiments in the report.

• Grading

• Base Recognition Rates

Without any enhancements, the base recognition rate for the given procedure is limited. You still can improve it by picking a better threshold. You are expected to achieve close to this recognition rate before applying any enhancements. This part will be up to **60%** of your grade.

- **Enhancements**

After obtaining a recognition rate close to the base recognition rate, you will try different methods to improve this result. This part will be up to **20%** of your grade.

- **Report**

Reporting everything clearly and putting all necessary figures in to your report will be up to **20%** of your grade.

- **Grade Penalties**
  - **10%** for late submission up to two days.
  - Up to **10%** if the code is not running for some reason and/or not giving significantly different results than the reported ones. Do not forget to submit all the files you have written and any external files that you have used.
  - Up to **50%** for group work, downloading codes from the internet, and/or code sharing. See Academic Integrity section.

• Academic Integrity

This is an **individual assignment** and it has to be completed by each person independently and **not with groups**. Sharing code or/and specific ideas is not allowed. The code, results and report that you have submitted will be compared with other people's submissions and your grade will be severely affected if a clear resemblance exists. Instead of consulting to your classmates, please contact the TA or the Professor to clear out any confusing points about the assignment.