

OCR Recognition

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

The assignment has two phases: Training and recognition

Initial Training: As per assignment instructions all image files except test1.bmp and test2.bmp are used to extract important features of individual letters.

Python Modules used:

```
NumPy
sklearn
SciPy
skimage
matplotlib
seaborn
math
pickle
os
csv
```

We start with reading the image, `skimage.io.imread(filename)`, and then binarizing the image using threshold value of 200.

Once the image is binarized, we then run connected component analysis to label each character with a unique label, `skimage.measure.label(binary_img, background=0)`.

After labelling the different components of an image file, we observe few extra components being labelled than expected, these are noise and during training phase we try to omit these by using certain threshold for height and width. We do this thresholding for height and width by calculating and comparing each component's height/width using their *minr*, *minc*, *maxr*, *maxc* values which is returned for each labelled component, `skimage.measure.regionprops(img_label)`.

The components that pass these threshold values are then processed for different feature computations. We store each image files' component's hu moments (7 hu moments) as their feature. As we store the features, we also store the labels of each image component.

Once, we have the complete feature list for all the image file components, we normalize the data matrix using mean and standard deviation of each feature.

At this point, we are ready to do the recognition for trained data. We use the normalized matrix to calculate each component's feature distance with every other component, which gives us N x N matrix of distances (N being number of components identified), `D=scipy.spatial.distance.cdist(Features, Features)`. We will observe that diagonal of this distance matrix is 0, this is because distance of each component's feature vector with itself will be 0. We will sort this distance matrix in ascending order to get least distanced index in the first column, (this is the component nearest to the component under investigation), `D_index=numpy.argsort(D, axis=1)`. However, during training we don't take the least distant index, as this will be with the component itself. So, we take the index of the second least distant column and predict the component under investigation as component label value at the identified index in labels stored for each component during feature extraction/storing, this will be our predicted value for this component.

Once we are done predicting labels for each image component, we will calculate the accuracy of our prediction during training phase. We also visualize our prediction using confusion matrix, which gives us a clear picture of actual vs. predicted label differences, `sklearn.metrics.confusion_matrix(Ytrue, Ypred)`.

In the next sections, we will look at the snapshots of each read image and result of different image processing operations on them.

OCRRRecognition.ipynb

Folder structure for the file is as below:

- *Assignment1materials*
 - *OCRRRecognition.ipynb*
 - test1_gk.pkl
 - test2_gk.pkl
- *H1-16images*
 - a-z.bmp
 - test1.bmp
 - test2.bmp

Code is written in a way to read the files in above folder structure.

```
train(dir_list, path, False)
```

The parameters passed to the train function are as below:

dir_list: list of all training images files to be processed, files are read in above folder structure

```
dir_list = [ i for i in os.listdir(path) if 'test' not in i ]
```

path: concatenation of current working directory and H1-16images folder

```
path = cwd + '/' + img_folder
```

third argument to train gives control to display all image plots. If False, only distance matrix and confusion matrix is displayed. If True, all plots are displayed.

In Testing block

test_mode = 'default', will run the code in usual manner, with above folder structure.

In order to test new files, make below changes:

```
test_mode = 'new'
```

```
test_dir = [<image file name with extension >]
```

```
path = [<Image Folder Path>]
```

```
pkl = <pkl file path>, if no pickle file to add, ignore this or leave commented
```

Dimension of training Feature list that will be created (Features): (1281, 16)

Output Labels dimension (y): (1281,)

Original Image and Connected Component images with bounding boxes:

- **Base Training**

Parameters used during initial training:

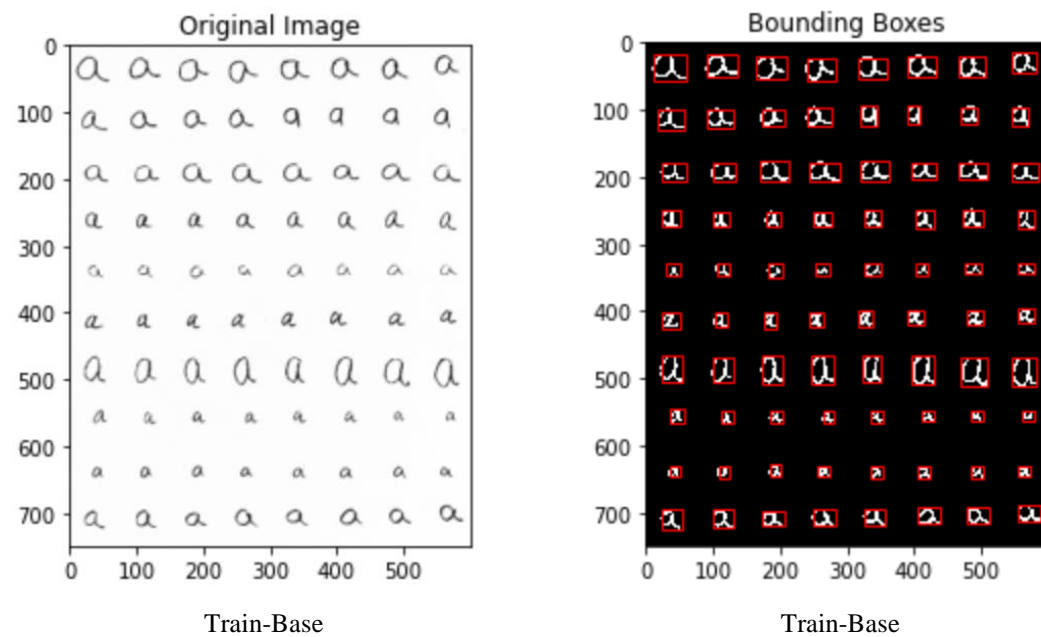
→ *Threshold = 200*

→ *Height and width threshold for image components:*

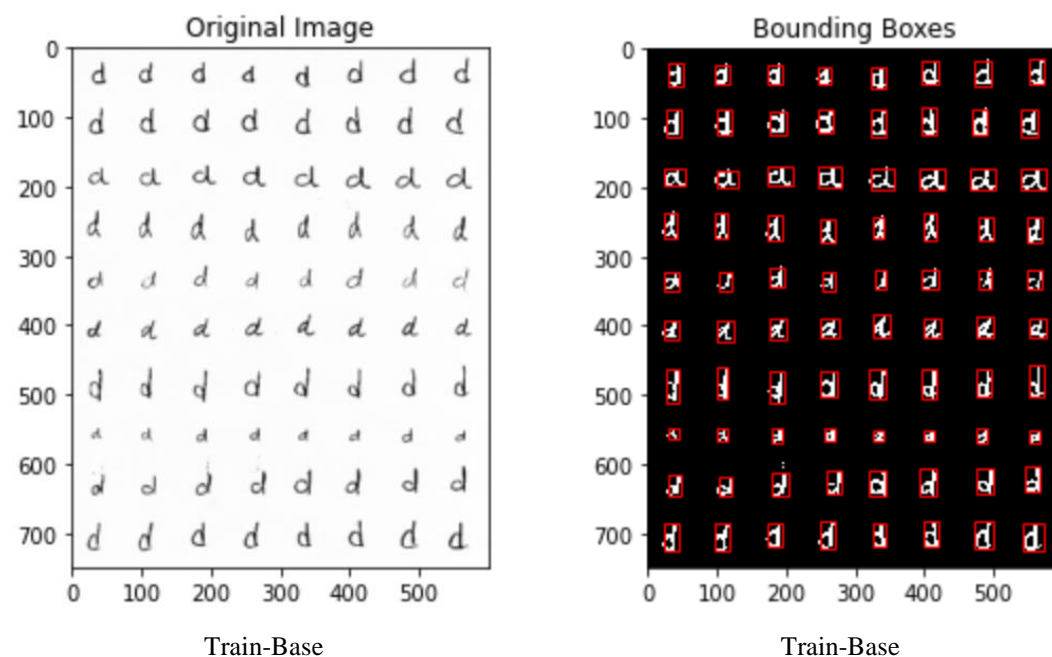
```
if (maxr-minr >= 14 and maxr-minr <= 80) and (maxc-minc >= 14 and maxc-minc <= 80):
```

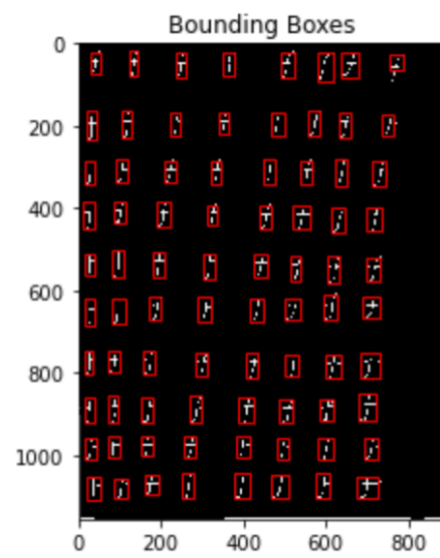
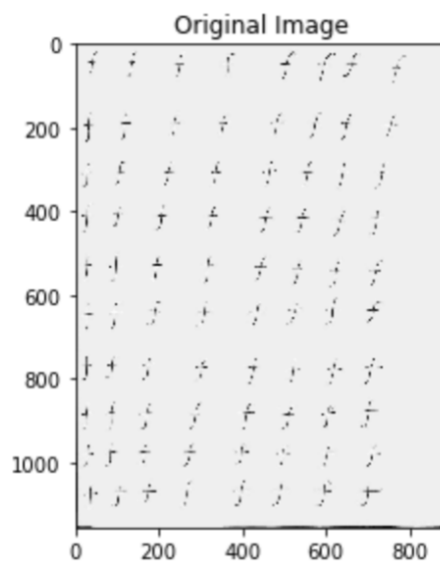
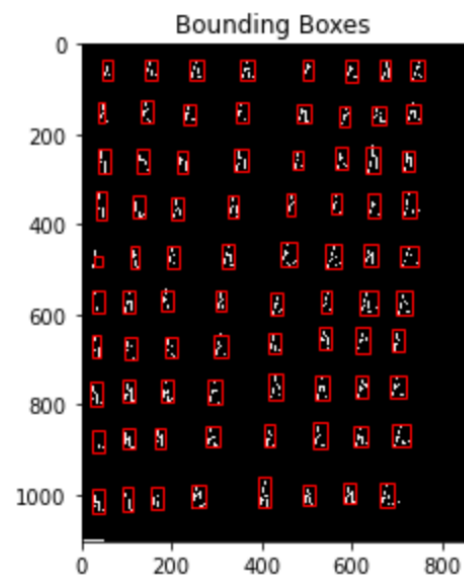
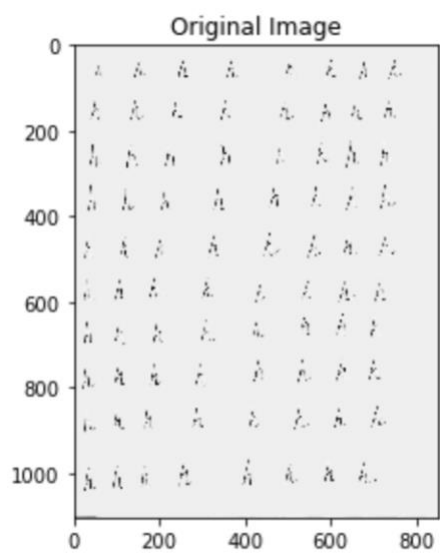
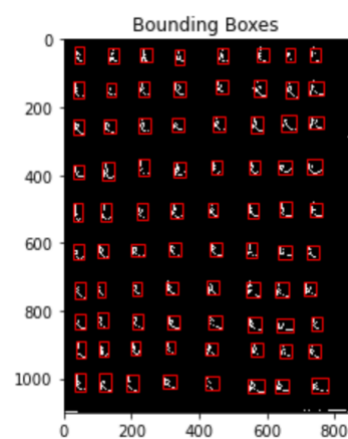
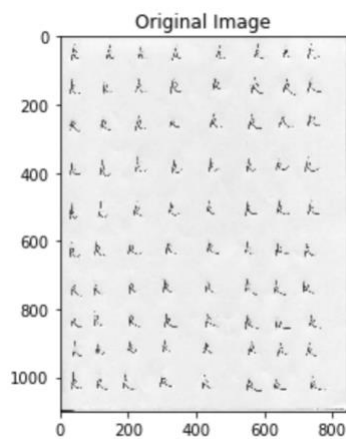
→ *Features list: 7 hu moments*

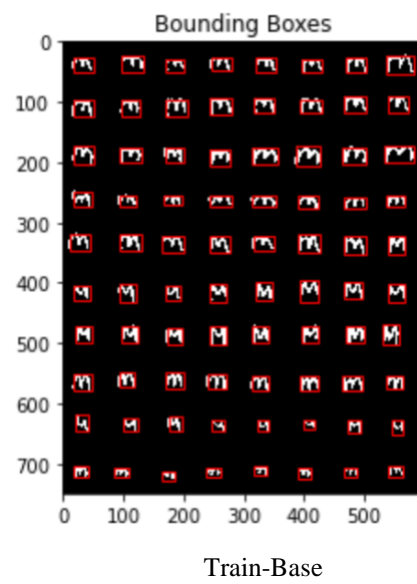
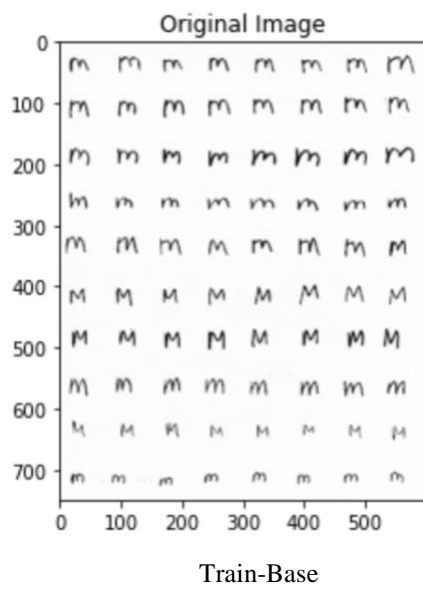
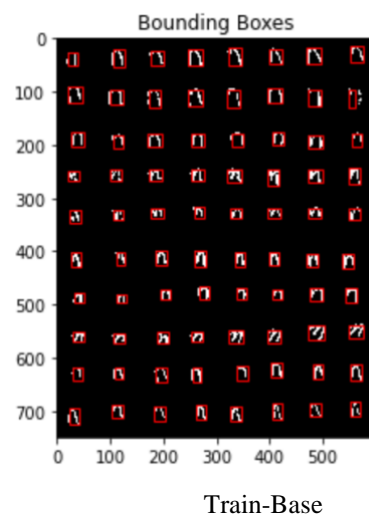
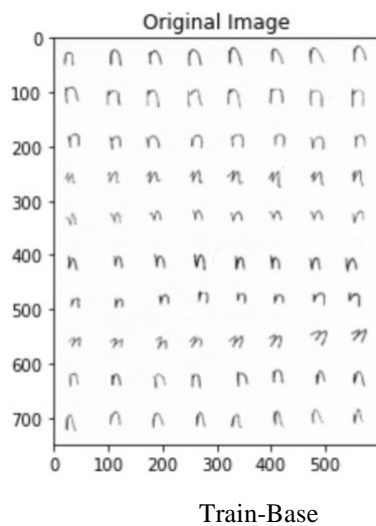
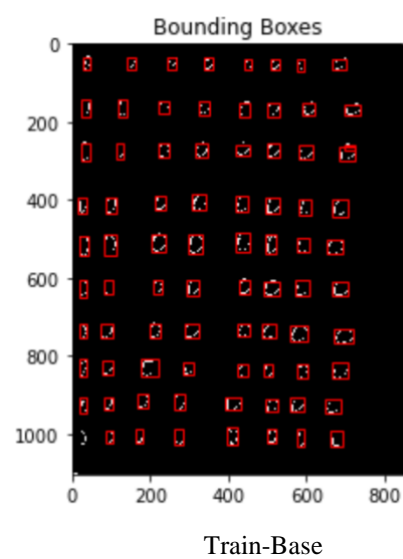
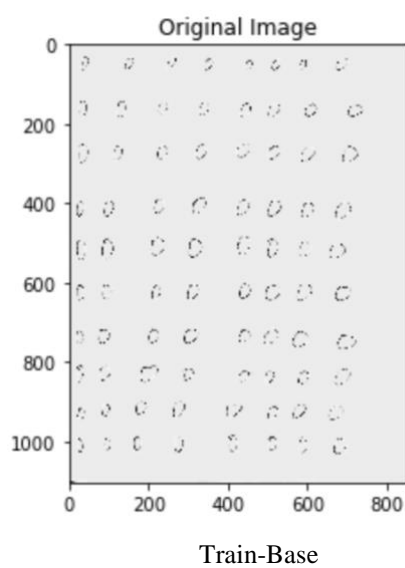
a.bmp

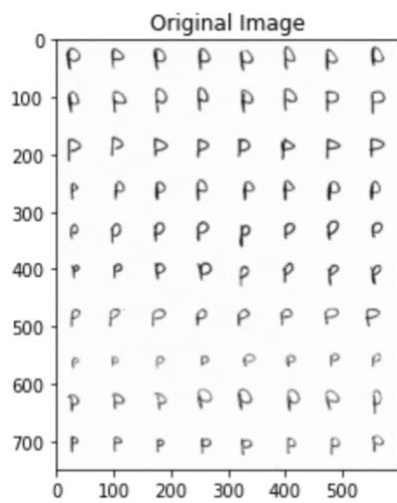


d.bmp

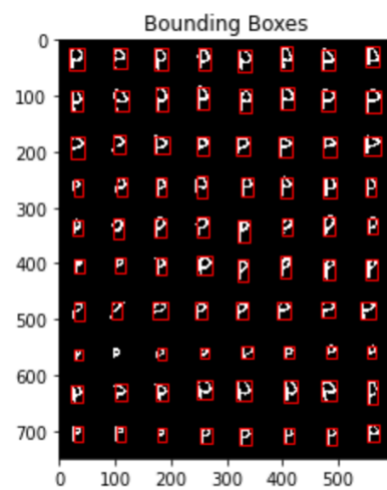


f.bmp*h.bmp**k.bmp*

m.bmp*n.bmp**o.bmp*

p.bmp

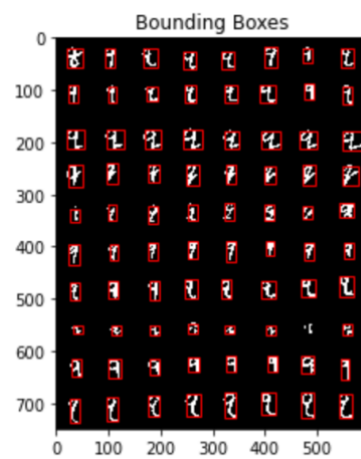
Train-Base



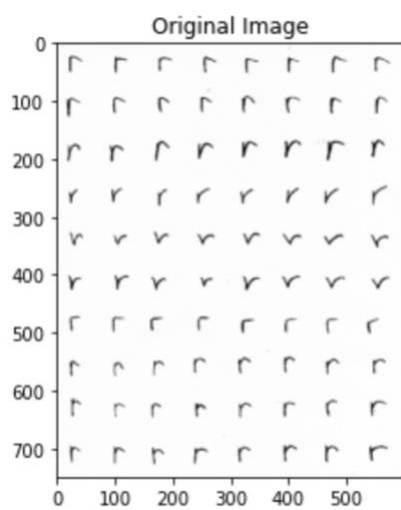
Train-Base

q.bmp

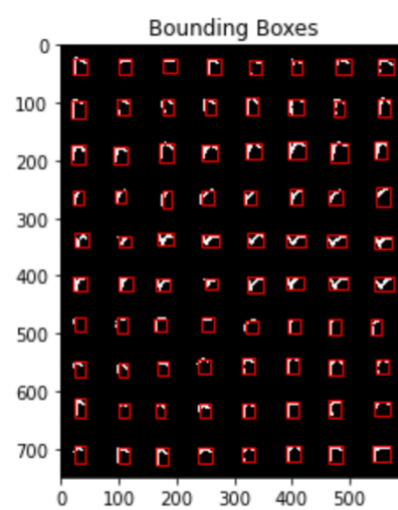
Train-Base



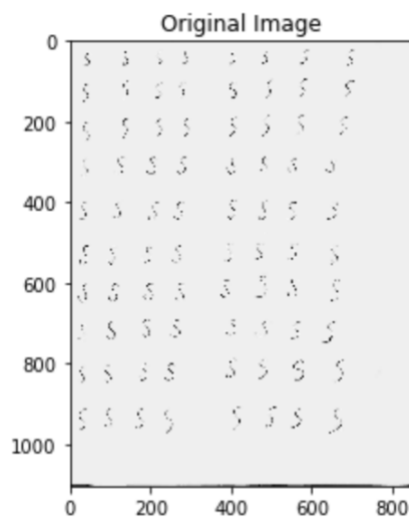
Train-Base

r.bmp

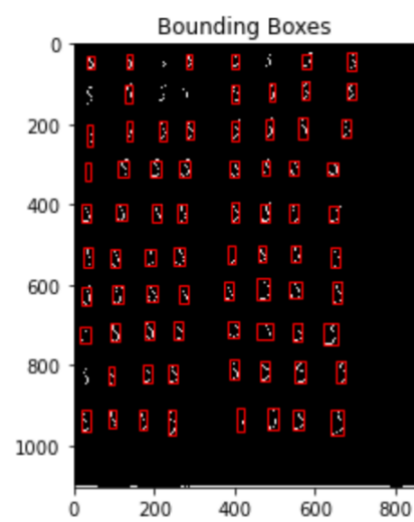
Train-Base



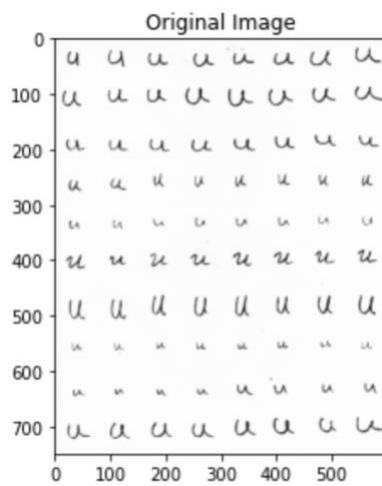
Train-Base

s.bmp

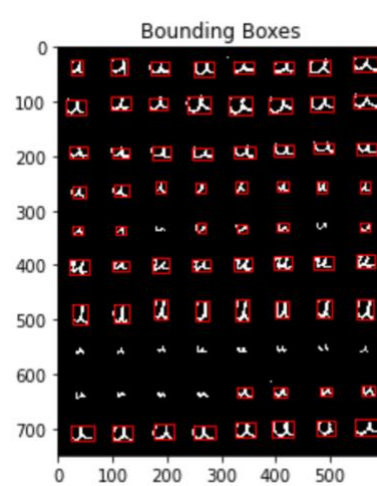
Train-Base



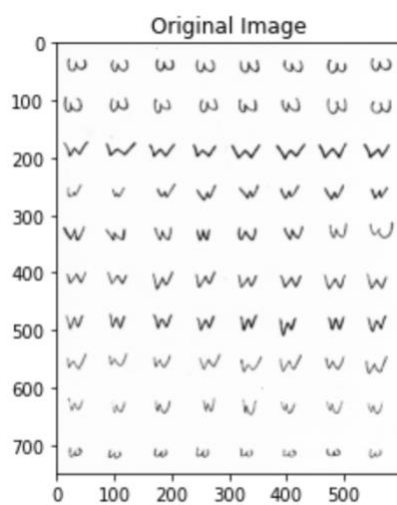
Train-Base

u.bmp

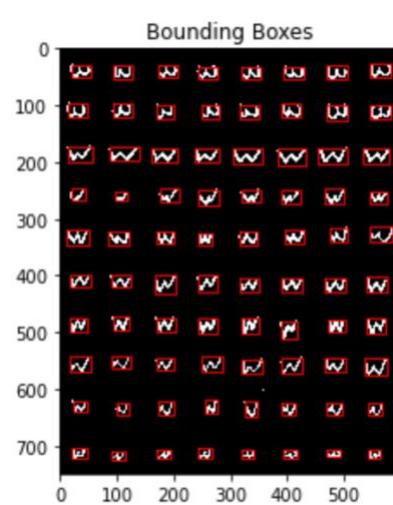
Train-Base



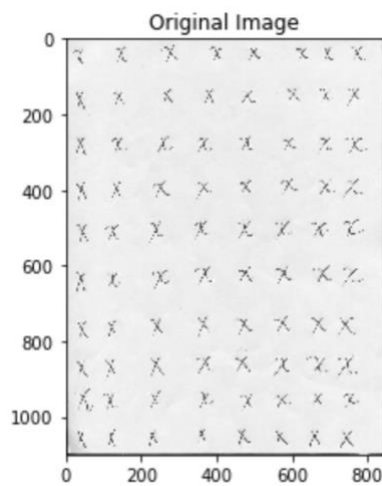
Train-Base

w.bmp

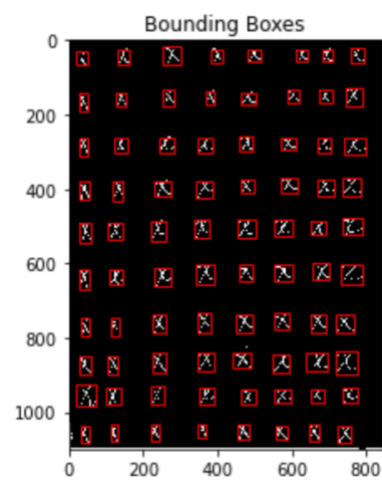
Train-Base



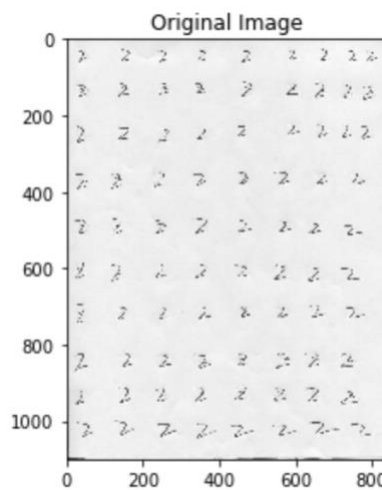
Train-Base

x.bmp

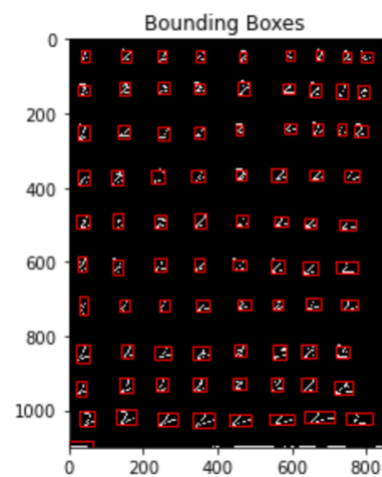
Train-Base



Train-Base

z.bmp

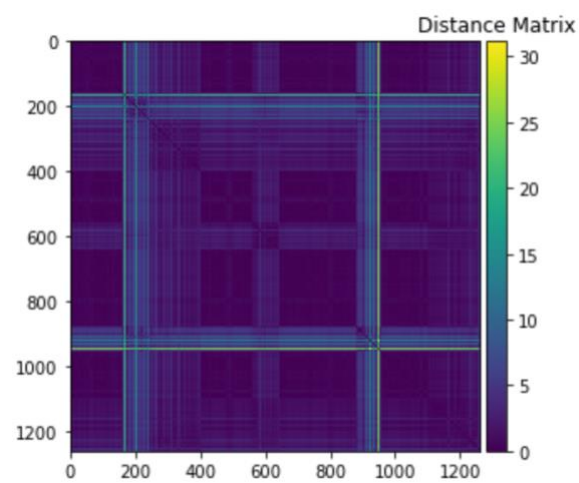
Train-Base



Train-Base

Recognition during initial training:

Distance Matrix – Diagonal of distance matrix would be 0 during training as it is distance with itself



Train-Base (Distance Matrix)

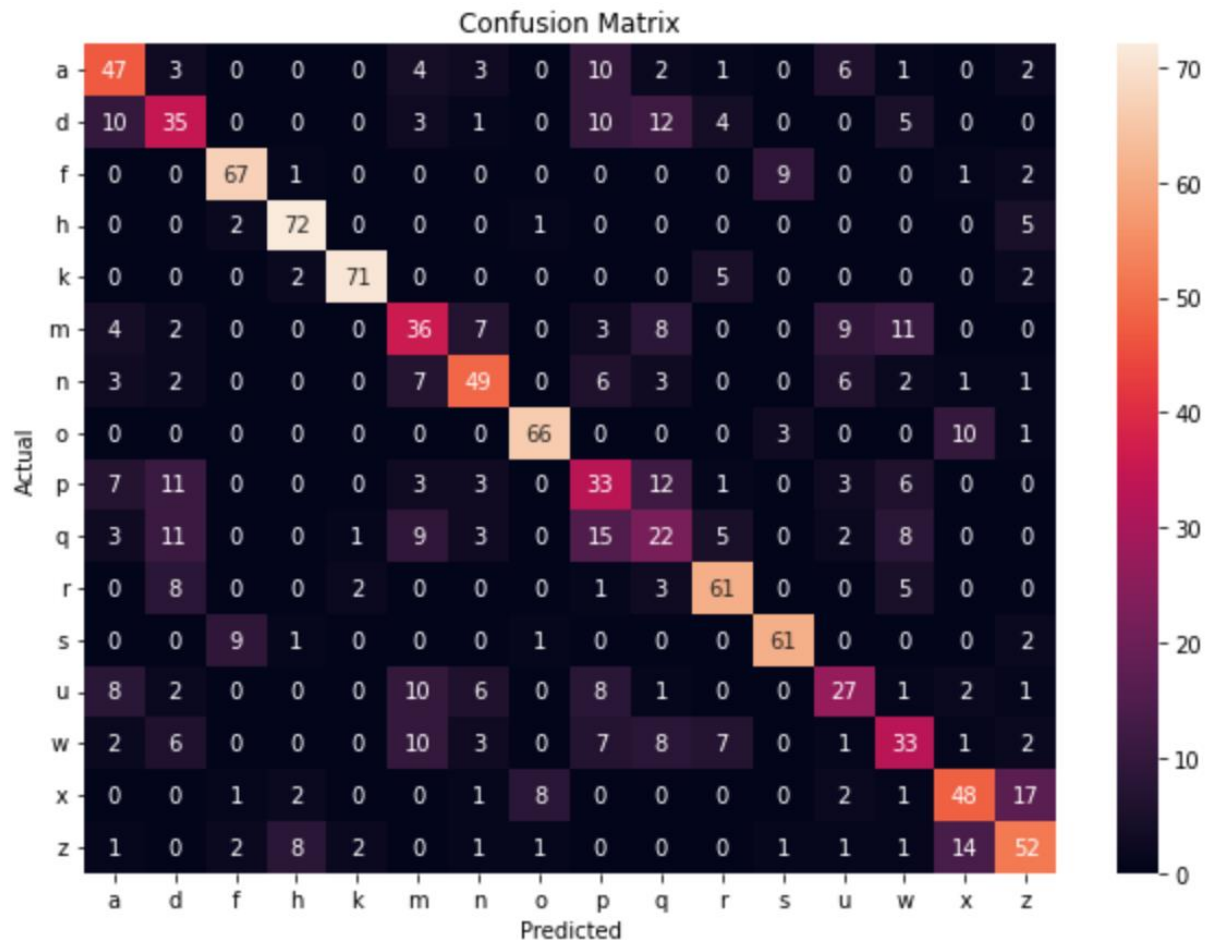
Labels after feature extraction of each component:

{0: 'a', 1: 'd', 2: 'f', 3: 'h', 4: 'k', 5: 'm', 6: 'n', 7: 'o',
8: 'p', 9: 'q', 10: 'r', 11: 's', 12: 'u', 13: 'w', 14: 'x', 15: 'z'}

Components obtained for base training: 1261

Recognition Rate on training data: 61.85%

Confusion Matrix of initial training:



We observe here that pixels of images are broken into small components and have some noise. This is resulting in few image components not being detected and some noise being detected as image components. We will try to resolve this in Enhancements section using different enhancement techniques.

- **Base Testing**

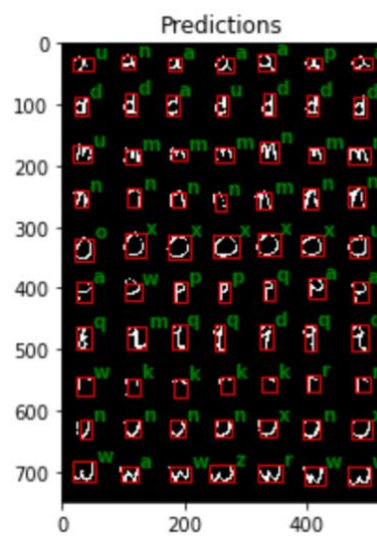
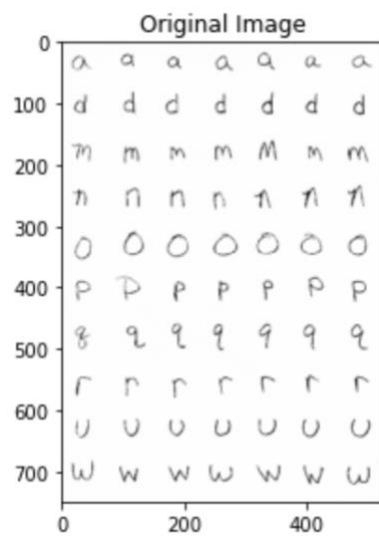
Parameters used during initial training:

→ *Threshold* = 200

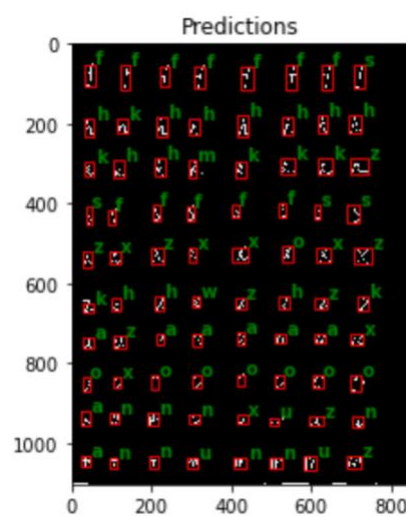
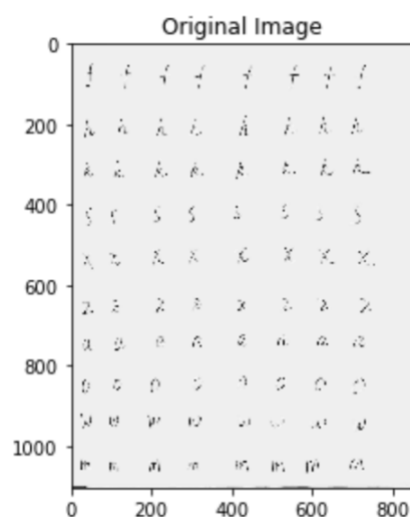
→ *Height and width threshold for image components:*

```
if (maxr-minr >= 14 and maxr-minr <= 80) and (maxc-minc >= 14 and maxc-minc <= 80):
```

→ *Features list:* 7 hu moments

Test1.bmp

Components obtained for Test1.bmp = 70

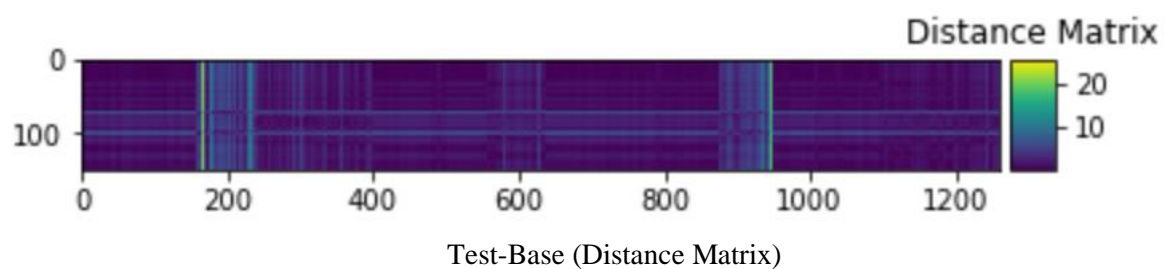
Test2.bmp

Components obtained for Test2.bmp = 80

Total components obtained for testing: $70 + 80 = 150$

Recognition during base testing:

Distance Matrix



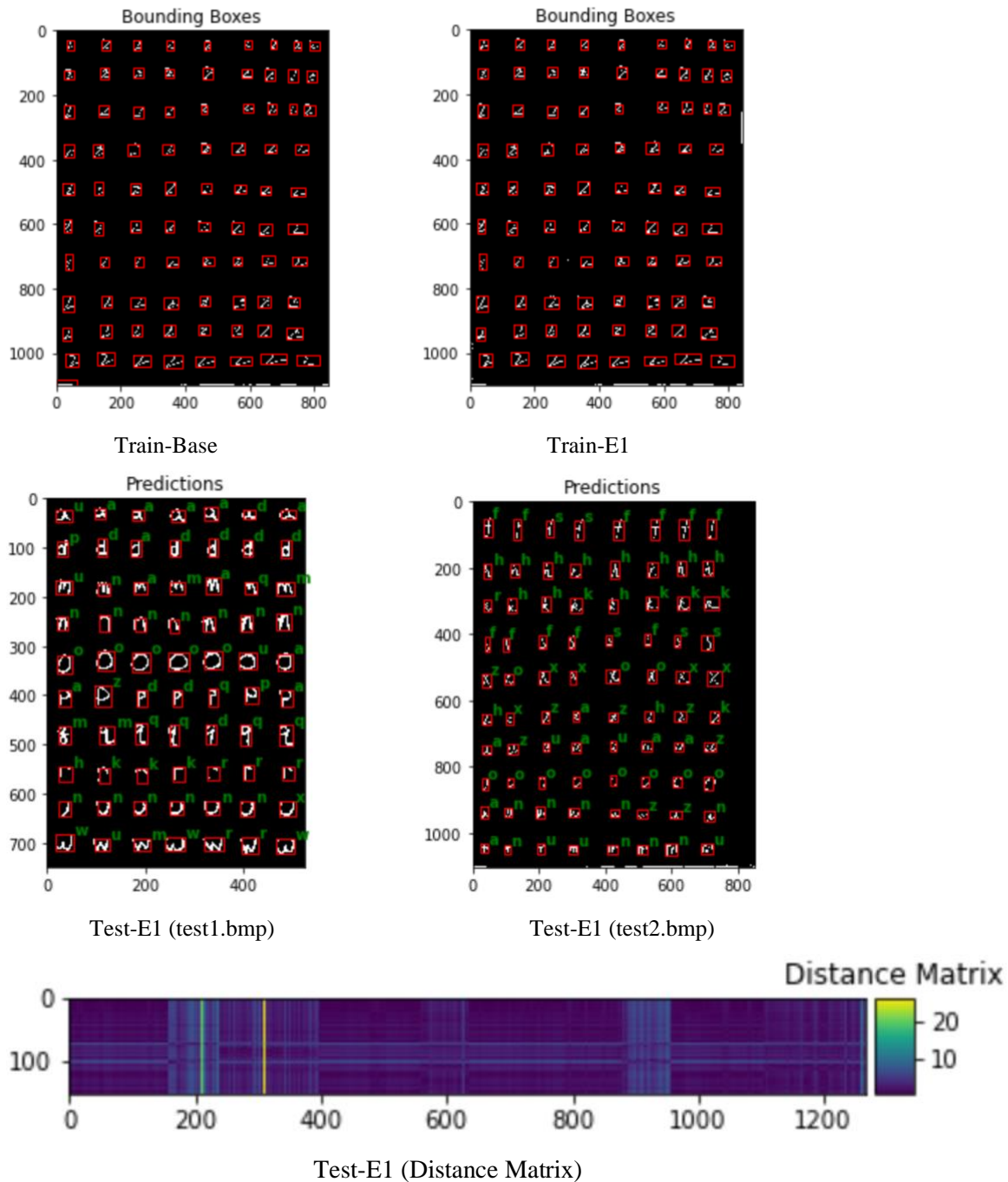
Recognition Rate on testing data (Test1.bmp and Test2.bmp): 47.33%

- **Enhancements**

1) Automate threshold (E1)

The first enhancement is to automate the threshold rather than using a hard-coded threshold for every image. To automate threshold, we use `skimage.filters.threshold_yen()`. It takes image or histogram of image as input and gives an optimal threshold value.

We observe that some noise that was being detected earlier, is not detected now. Observe the little noise in Train-Base at the bottom left corner.



Recognition Rate Train-E1: 60.61%

Components obtained for Train-E1 = 1267

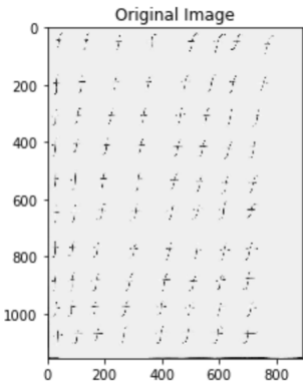
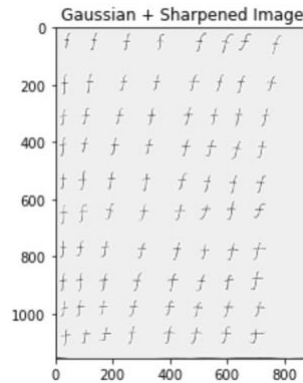
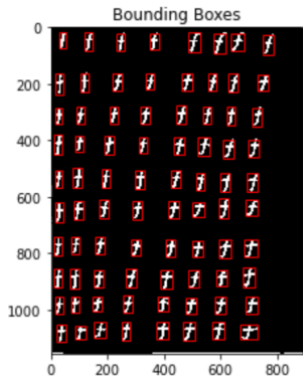
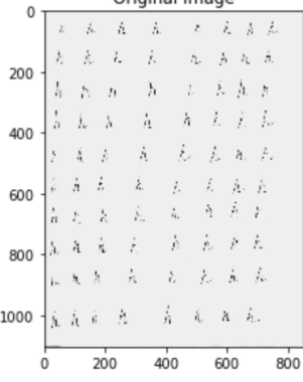
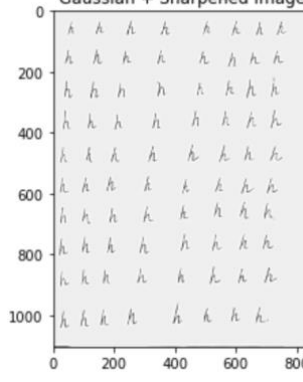
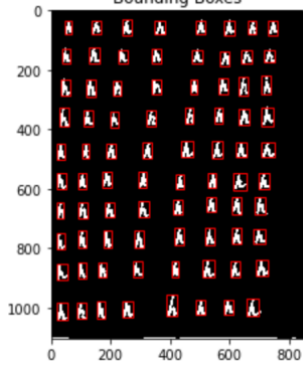
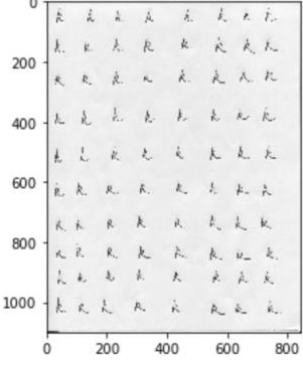
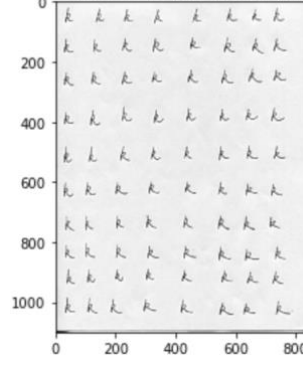
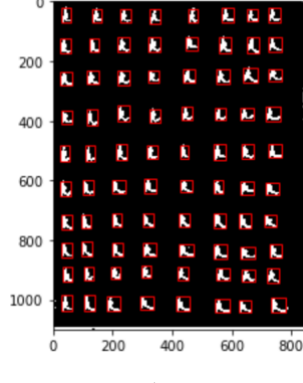
Recognition Rate Test-E1: 50.66%

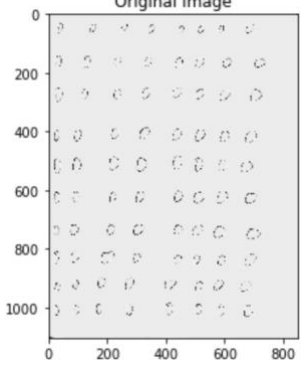
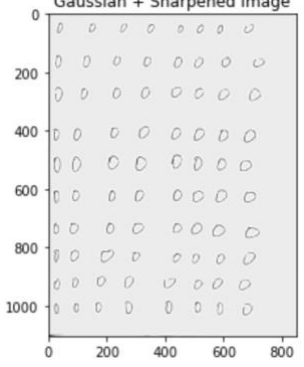
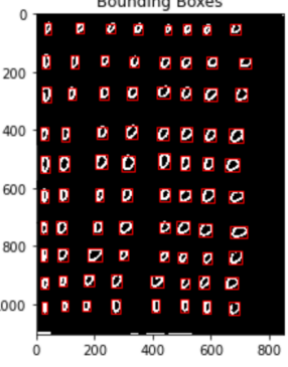
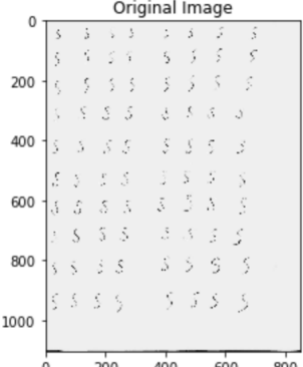
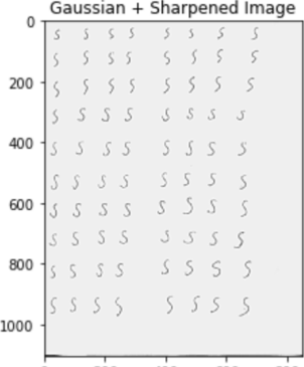
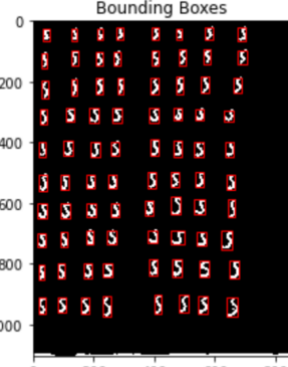
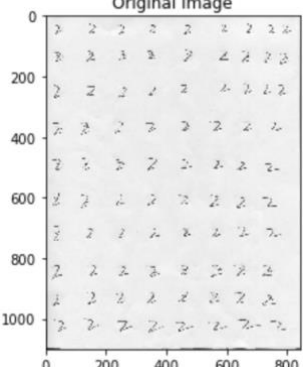
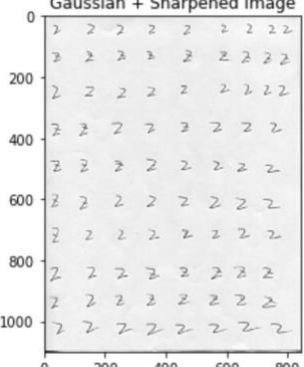
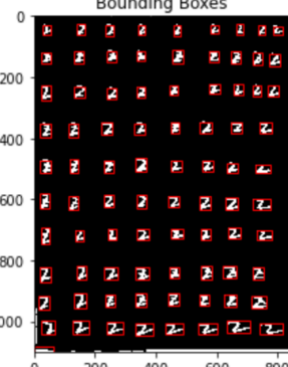
Components obtained for Test-E1 = 150

2) Filters (E2)

We observe that when the image file is read, the pixels are fragmented and distorted which results in unwanted noise and the image features cannot be captured properly. As a solution to this problem, we apply gaussian filter to the read image file, `skimage.filters.gaussian()`. Gaussian filter is a smoothing filter which reduces image noise to a great extent. To heal the image sharpness after gaussian blur effect, perform image sharpening, `skimage.filters.unsharp_mask()`. This combination of gaussian and unsharp_mask does an excellent job of removing noise and intensifying the image components.

Training:

| Image | Read Image | Image After Gaussian and Sharpening | Components with Bounding Boxes |
|-------|---|--|---|
| f.bmp |  <p>Original Image</p> <p>Train-E2</p> |  <p>Gaussian + Sharpened Image</p> <p>Train-E2</p> |  <p>Bounding Boxes</p> <p>Train-E2</p> |
| h.bmp |  <p>Original Image</p> <p>Train-E2</p> |  <p>Gaussian + Sharpened Image</p> <p>Train-E2</p> |  <p>Bounding Boxes</p> <p>Train-E2</p> |
| k.bmp |  <p>Original Image</p> <p>Train-E2</p> |  <p>Gaussian + Sharpened Image</p> <p>Train-E2</p> |  <p>Bounding Boxes</p> <p>Train-E2</p> |

| | | | |
|-------|---|--|---|
| o.bmp |  <p>Original Image</p> <p>Train-E2</p> |  <p>Gaussian + Sharpened Image</p> <p>Train-E2</p> |  <p>Bounding Boxes</p> <p>Train-E2</p> |
| s.bmp |  <p>Original Image</p> <p>Train-E2</p> |  <p>Gaussian + Sharpened Image</p> <p>Train-E2</p> |  <p>Bounding Boxes</p> <p>Train-E2</p> |
| z.bmp |  <p>Original Image</p> <p>Train-E2</p> |  <p>Gaussian + Sharpened Image</p> <p>Train-E2</p> |  <p>Bounding Boxes</p> <p>Train-E2</p> |

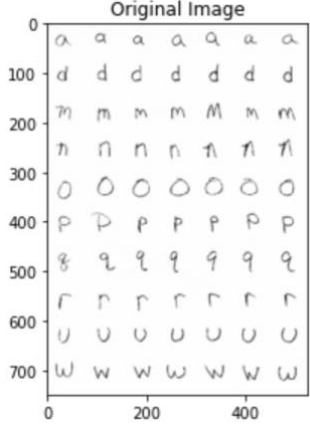
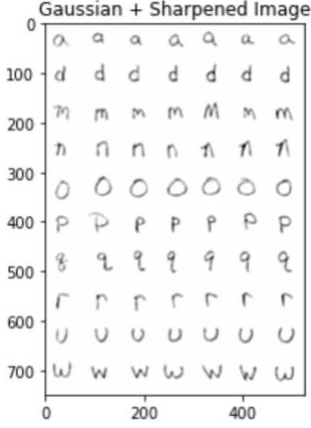
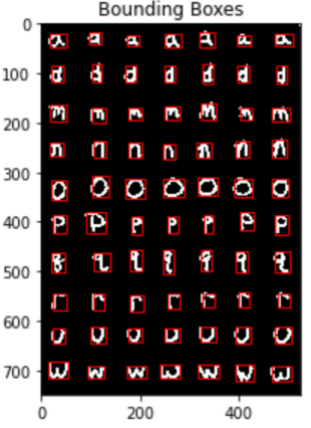
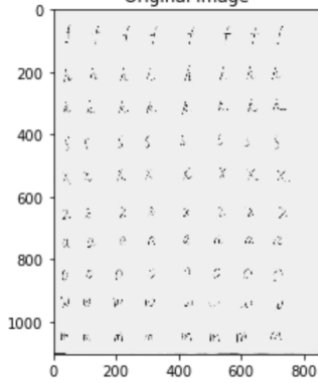
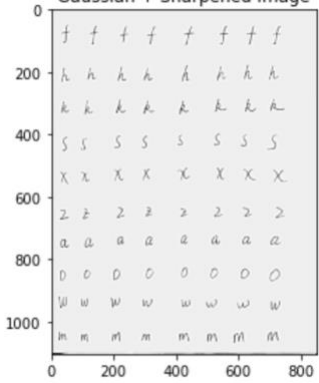
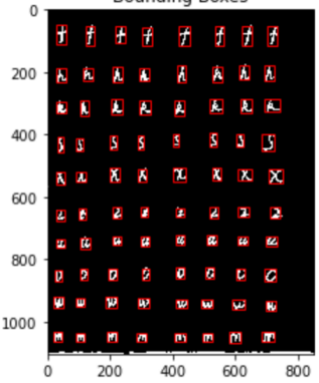
Components obtained for Train-E2: 1281

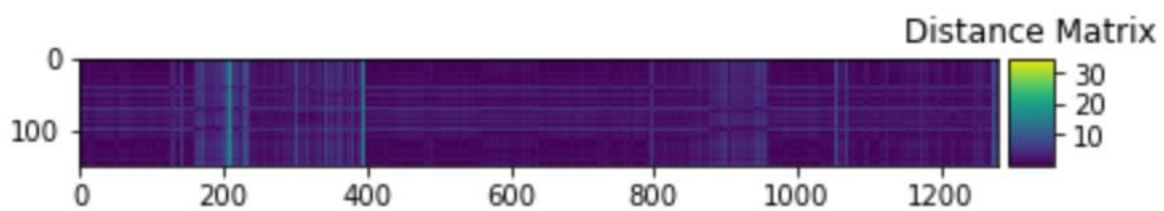
Recognition Rate Train-E2: 58.20%

We would expect recognition-rate / accuracy to increase after we have all the components image clear, however it went down. One thing to note here is we have identified more valid components this time as compared to Train-E1. We can further add more features to this cleared image to have a better prediction.

As the image is better processed now, we can extract more accurate features.

Testing:

| Image | Read Image | Image After Gaussian and Sharpening | Components with Bounding Boxes |
|-----------|---|--|---|
| test1.bmp |  <p>Original Image</p> <p>Test-E2</p> |  <p>Gaussian + Sharpened Image</p> <p>Test-E2</p> |  <p>Bounding Boxes</p> <p>Test-E2</p> |
| test2.bmp |  <p>Original Image</p> <p>Test-E2</p> |  <p>Gaussian + Sharpened Image</p> <p>Test-E2</p> |  <p>Bounding Boxes</p> <p>Test-E2</p> |

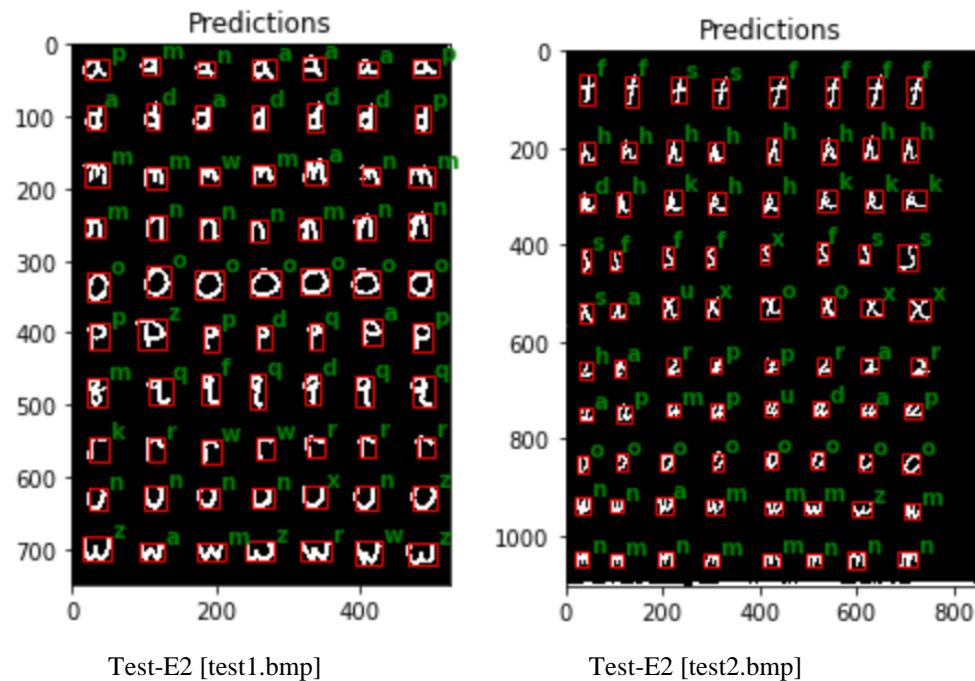


Test-E2 (Distance Matrix)

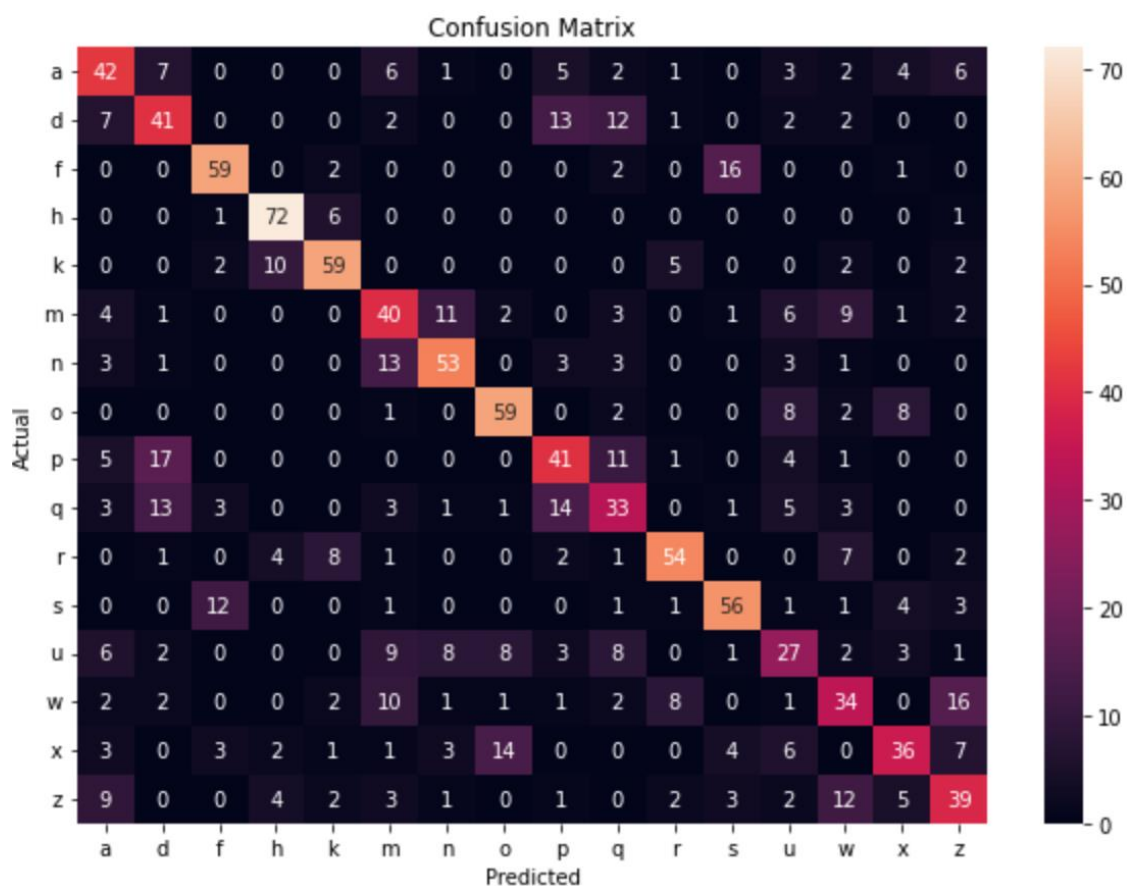
Components obtained for Test-E2: 150

Recognition Rate Test-E2: 46%

Although, we have a low recognition rate for testing here, we know we were able to detect more valid components during training and were able to process the image better, hence, we can add more features to our feature list in next enhancement to get a better accuracy rate.



Confusion matrix of Train-E2:



Train-E2 (Confusion Matrix)

From confusion matrix of Train-E2, it is evident that we have more problem predicting letters that are somewhat rotational variant of each other, like $p / q / d$ also, n / u and m / w .

3) Additional Features (E3)

As during previous enhancement, we figured that the problem now lies with the letters that are rotational look-alike of each other, we need more features to strike difference between them and increase the recognition rate.

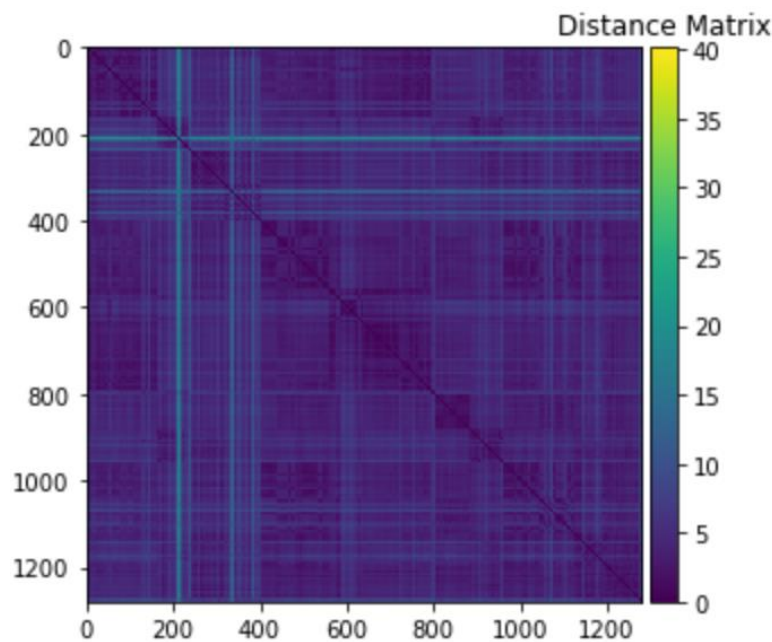
At this point, let's add **orientation** and two third order moments, **$\mu[3,0]$ and $\mu[0,3]$** . These two third-order moments can be used to determine the degree of skewness about the mean. Two normalized moments, **$\nu[1,2]$ and $\nu[2,1]$** , sign and magnitude of centralized moments tell us about the symmetrical properties of these characters.

If we observe the Train-E2 confusion matrix further, we will see that there is a confusion between letters that are almost similar, like **m / n and $h / k / x$** . To further find a difference between them, let's add few more features that could give some area or volumetric difference between them and also point towards their major axis and topology to have a better information about their shape. We'll add **$perimeter_crofton$, $euler_number$** and ratio of **$area_filled/perimeter$** into our feature list.

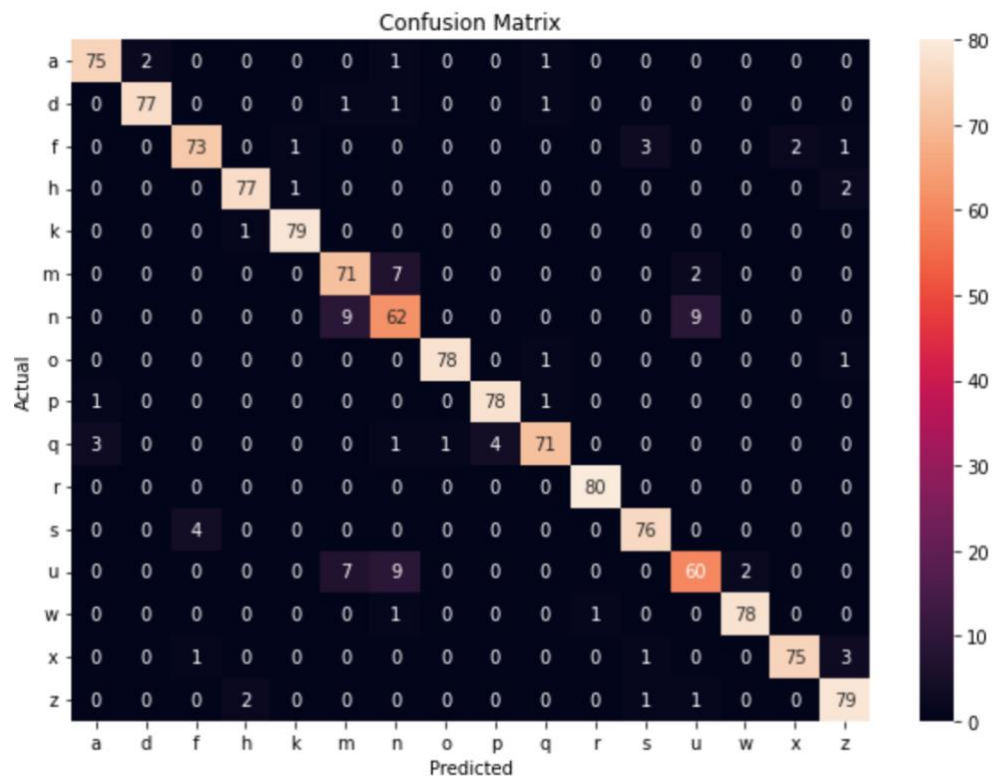
We are adding following additional features to our feature list for each component:

| | | | | | | | | |
|-------------|------------|------------|------------|------------|------------|----------------------|--------------------------|-----------------|
| Orientation | $\mu[3,0]$ | $\mu[0,3]$ | $\nu[1,2]$ | $\nu[2,1]$ | $\nu[1,1]$ | $perimeter_crofton$ | $area_filled/perimeter$ | $euler_number$ |
|-------------|------------|------------|------------|------------|------------|----------------------|--------------------------|-----------------|

Training:



Train-E3 (Distance Matrix)

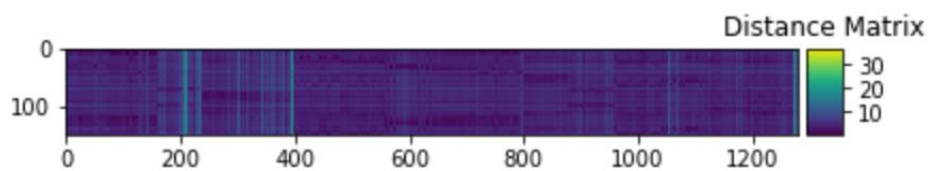


Train-E3 (Confusion Matrix)

Components obtained for Train-E3: 1281

Recognition Rate Train-E3: 92.89%

Testing:



Test-E3 (Distance Matrix)

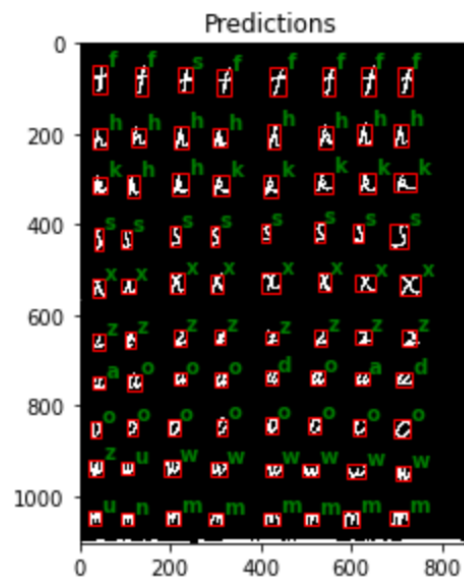
Components obtained for Test-E3: 150

Recognition Rate Test-E3: 81.33%

We can deduce from confusion matrix that majority problem now lies with the recognition of **u/ w/ m/ n** and **a/ d/ o**.



Test-E3 [test1.bmp]

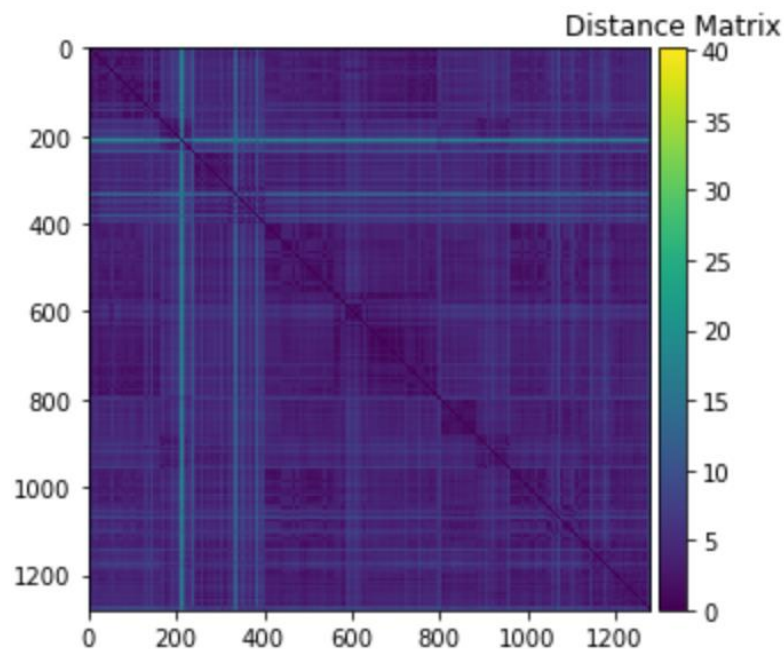


Test-E3 [test2.bmp]

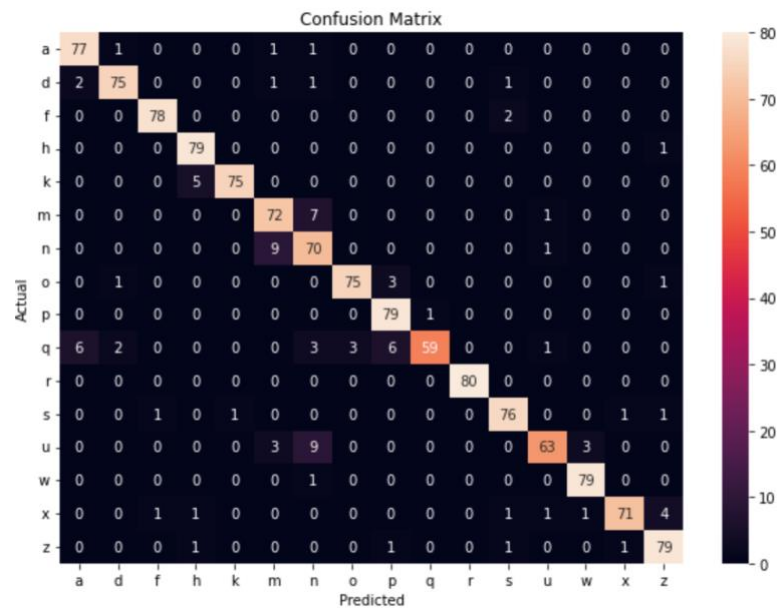
4) Classifier (E4)

Now, we try to use a different classifier, k-nearest neighbors (k is small number 3, 5, 7). Instead of predicting the class of letters based on nearest neighbor, we now k-nearest neighbor of each component and do a majority vote on that to predict the final class of component under investigation. Let's take k as 5.

Training:



Train-E4 / Final (Distance Matrix)

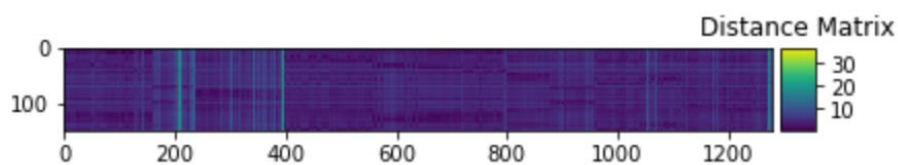


Train-E4 / Final (Confusion Matrix)

Components obtained for Final / Train-E4: 1281

Recognition Rate Final / Train-E4: 92.66%

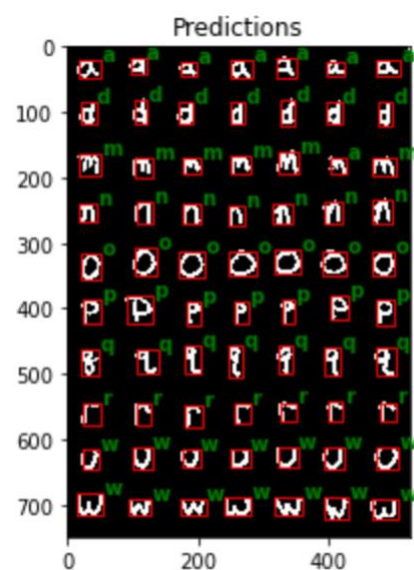
Testing:



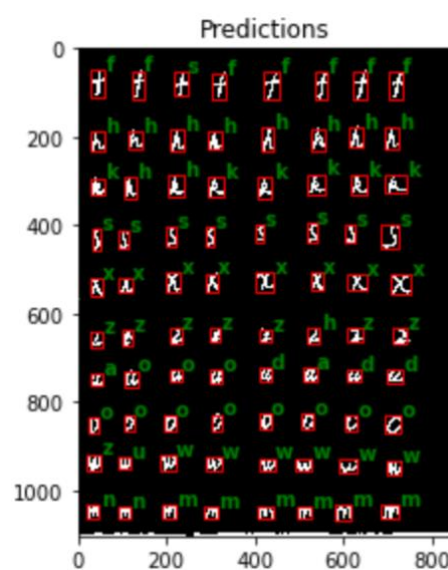
Test-E4 / Final (Distance Matrix)

Components obtained for Final / Test-E4: 150

Recognition Rate Final / Test-E4: 83.33%



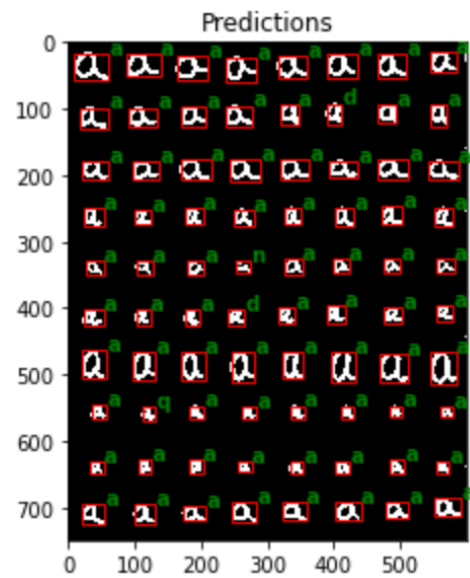
Test-E4 / Final [test1.bmp]



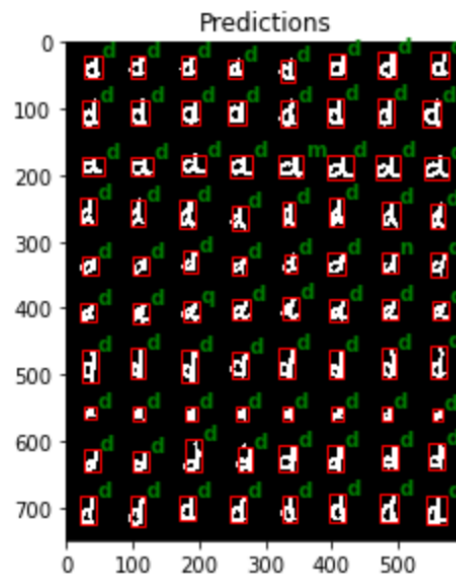
Test-E4 / Final [test2.bmp]

- Final Recognition Results:

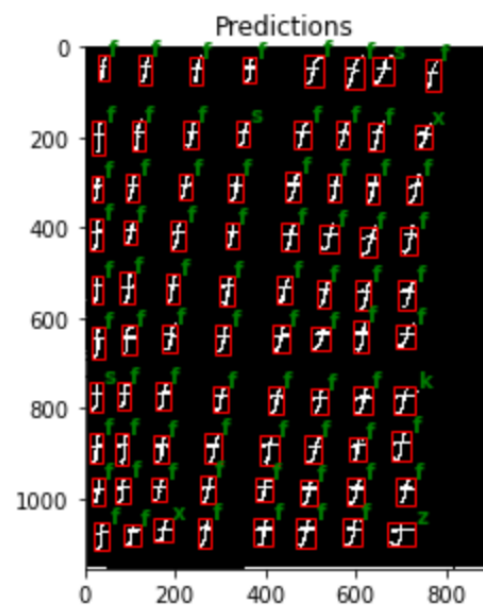
Training:



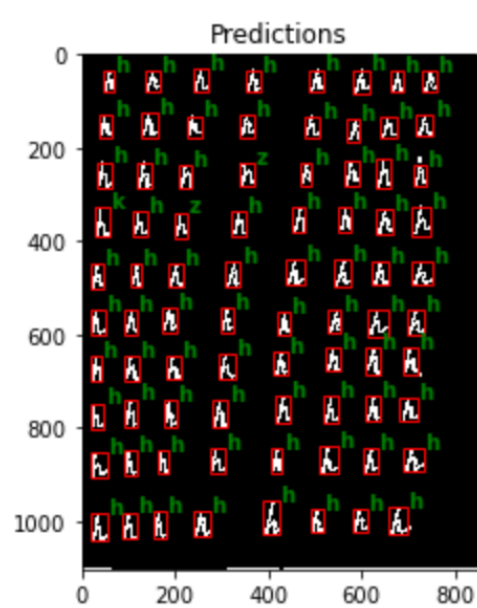
Train-F



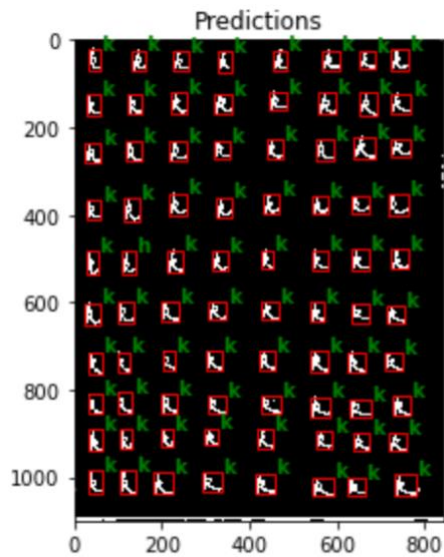
Train-F



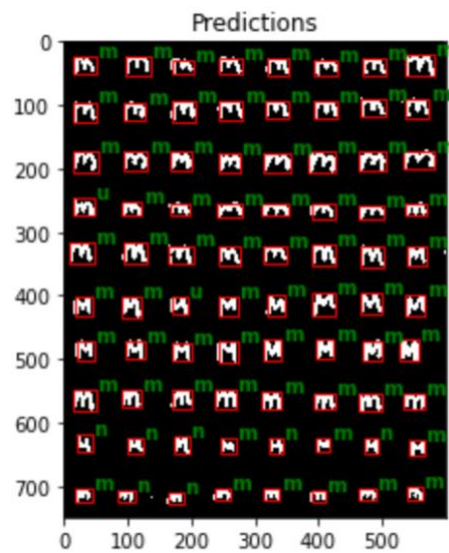
Train-F



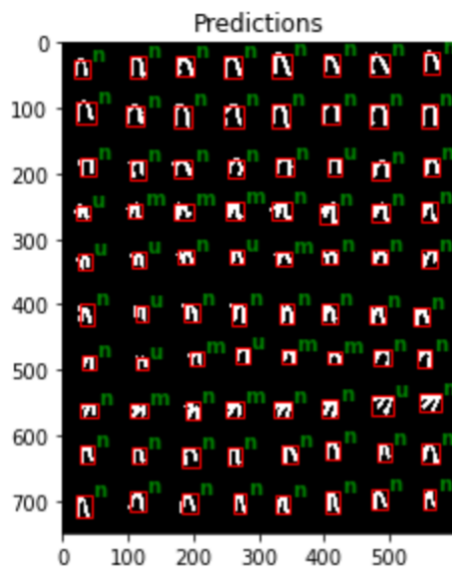
Train-F



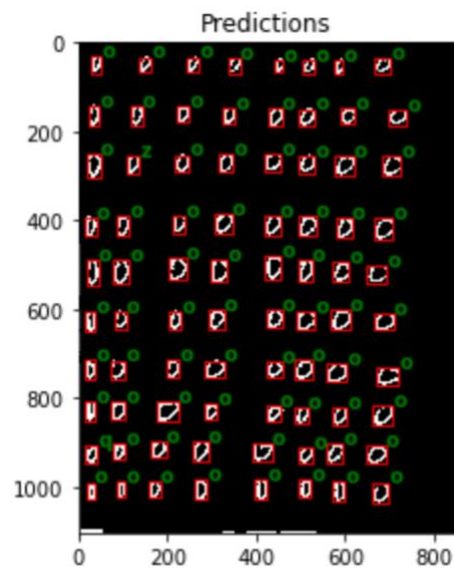
Train-F



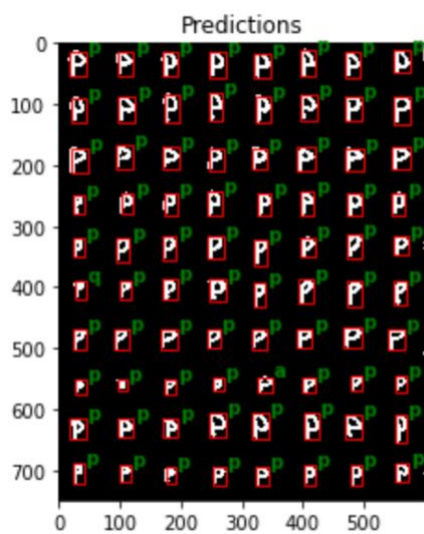
Train-F



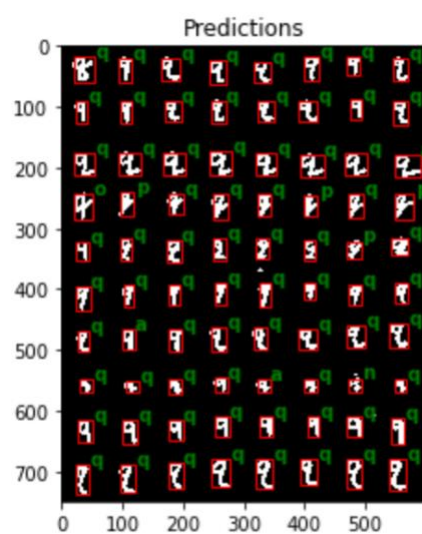
Train-F



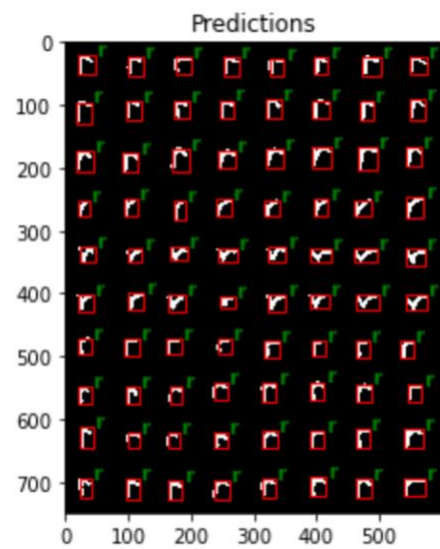
Train-F



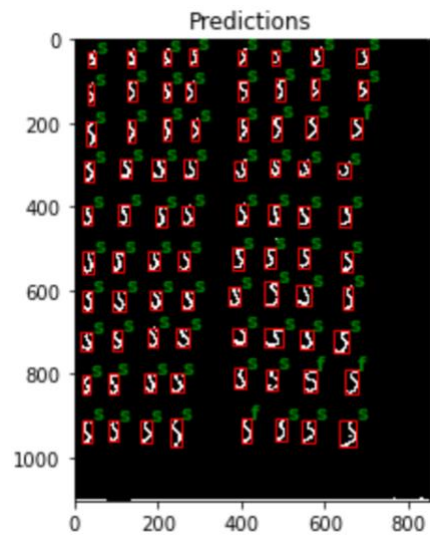
Train-F



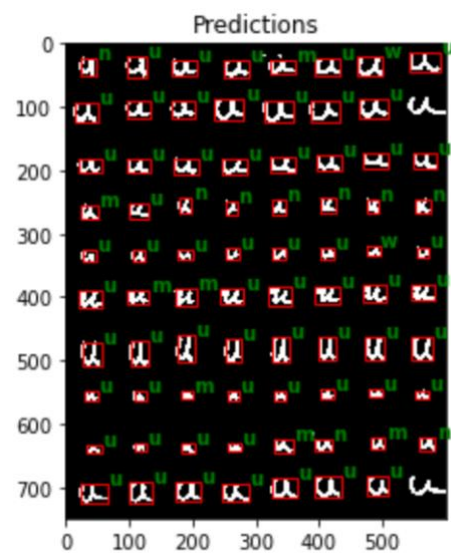
Train-F



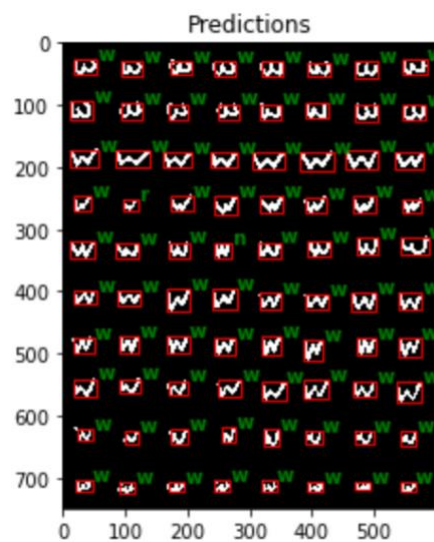
Train-F



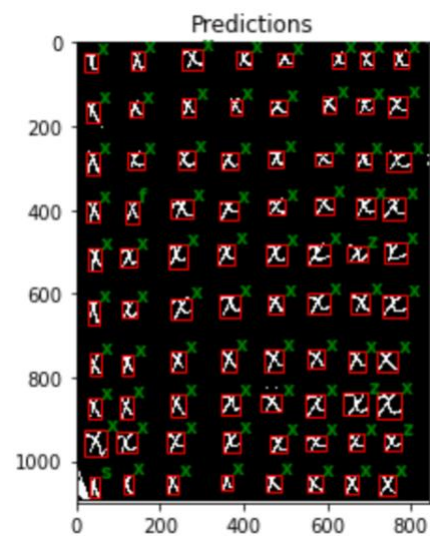
Train-F



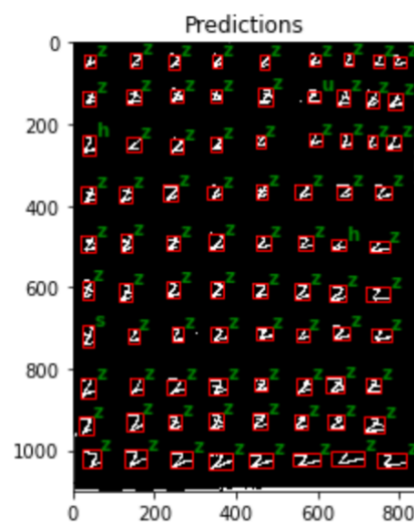
Train-F



Train-F



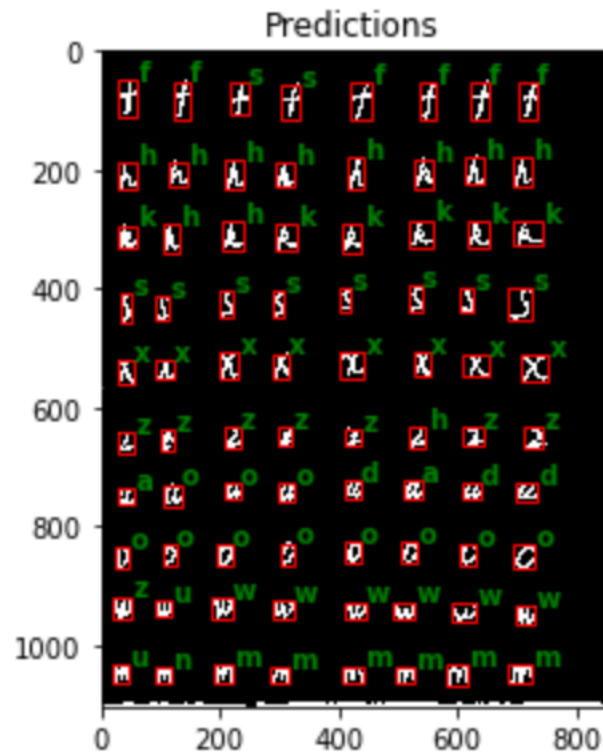
Train-F



Train-F

Testing:

Test-F [test1.bmp]



Test-F [test2.bmp]

- Results

| Methods | | Train-rate | Figure label | Test-rate | Figure label |
|------------------|---|------------|--------------|-----------|--------------|
| Base Recognition | | 61.85 | [Train-Base] | 47.33 | [Test-Base] |
| Enhancement [1] | What: Automating threshold | 60.61 | [Train-E1] | 50.66 | [Test-E1] |
| | How: Using skimage module's threshold_yen() function to calculate threshold for each input image. | | | | |
| | Why: In previous train results, we saw that there was some noise also getting detected as image components. Automating the threshold for each input image will give us a better threshold value and will help in removing this noise. | | | | |
| | What you deducted from this trial: It reduced the train accuracy, as it also stops recognizing some smaller components of image along with noise. However, it increases test accuracy. | | | | |
| Enhancement [2] | What: Filters | 58.20 | [Train-E2] | 46 | [Test-E2] |
| | How: Use filters like gaussian to and unsharp_mask from skimage module. | | | | |
| | Why: When we read images, we observe that pixels get fragmented and are main reason for noise. With a combination of gaussian and sharpening, we intend to get a smoother and denoised image for better processing. | | | | |
| | What you deducted from this trial: We saw that although the training and testing recognition rate went down with this enhancement, we now have a more descriptive image of all components to extract | | | | |

| | | | | | |
|----------------------|---|-------|------------|-------|-----------|
| | some additional features for recognition. | | | | |
| Enhancement [3] | What: Additional Features | 92.89 | [Train-E3] | 81.33 | [Test-E3] |
| | How: Add additional features like, mu moments, nu moments, orientation, euler number, perimeter, area etc. | | | | |
| | Why: As we obtained a cleaner and smoother image in previous enhancement, we now take additional features to know more about each component's shape and orientation. This will help us to predict each component with even more accuracy. | | | | |
| | What you deducted from this trial: As expected, adding more features increases the recognition rate for both training and testing significantly. | | | | |
| Enhancement [4] | What: Classifier | 92.66 | [Train-E4] | 83.33 | [Test-E4] |
| | How: Take k-nearest neighbors (5) and do a majority vote | | | | |
| | Why: We apply k-nearest neighbors to potentially break the ambiguity of similar letters. | | | | |
| | What you deducted from this trial: It does increase the test recognition rate slightly. | | | | |
| Final Configurations | Added threshold function to calculate an optimal value to discard noise and help in binarization. Added gaussian and unsharp_mask filters to smoothen and further denoise the input image. Added additional features to help understand characteristic of each letter component even better. Added k-nearest neighbor classifier to help improve | 92.66 | [Train-F] | 83.33 | [Test-F] |

| | | | | | |
|--|---------------------------------------|--|--|--|--|
| | accuracy for similar looking letters. | | | | |
|--|---------------------------------------|--|--|--|--|