**TABLE OF CONTENTS**

**1. Introduction**

1.1 Abstract

1.2 Background

**2. Overview and Planning**

2.1 Proposed Work

2.2 Hardware Requirements

2.3 Software Requirements

**3. Literature Survey and Review**

3.1 Literature Summary

**4. Methodology**

4.1 Method Used

4.2 Applications

**5. System Implementation**

5.1 Code

5.2 Results and discussion

**6. Conclusion**

6.1 Conclusion

6.2 Future Work

**7. References**

1. **INTRODUCTION**

**1.1 Abstract:**

Text digitalization is an important process in order to achieve cheap digitalized books for distribution and preservation. Document digitalization is an essential in enhancing the working of government institutions. This may include digitization of previous health records, tax and payment statements of employees etc. Digitizing such documents helps in increasing the efficiency of executing processing and requests in such institutions. Looking into hundreds of health records to find the desired report is far more difficult than just typing the name of the patient on the keyboard to achieve the same. In this project we aim to develop a text digitalization software that takes an image as an input and gives a text document containing the digitalized text as the output.

**1.2 Background:**

Optical character recognition has been a popular area in research for many years. Optical character recognition systems in the recent years have grown more powerful and accurate due to its integration with neural networks and other machine learning algorithms. Optical recognition software have been tuned to meet the requirements of the field it is used in. However the basic principles and workings of all OCR software boils down to the same principles. **Figure 1** denotes the working pipeline of an optical recognition software.

**Figure 1: OCR PIPELINE**

1. **OVERVIEW AND PLANNING**

**2.1 Proposed Work:**

We propose to build a text digitizing software that follows the pipeline described in figure 1 to implement the OCR process. Our project contains the following primary workflow and files:

**Figure 2: Primary workflow**

Apart from the primary workflow the recognition sub process had to be carried out separately and hence it has its own secondary workflow as follows:

**Figure 3: Secondary Workflow**

**2.2 Hardware Requirements:**

* Laptop/Pc with at least 8 GB of ram.
* Laptop/Pc with 6th generation processor or above

**2.3 Software Requirements:**

* Text editor: Pycharm preferred due to friendly gui and easy library management
* Python: Version 3.5 or above for libraries compatibility
* Windows 10 / Ubuntu 16.04 or better

**2. 4 Libraries Requirements:**

* **opencv-python** for image processing tools and functions
* **numpy** for image manipulation as an array
* **sklearn** for svm classifiers
* **os** for directory manipulation
* **pickle** for saving and loading the svm model
* **scipy** for loading the emnist-lettersdataset from a mat format file
* **subprocess** to open the text file into a notepad.

**2.5 Dataset Requirement:**

* **emnist-letters** dataset for training on 26 unbalanced classes
* **char74k** dataset for training on 62 balanced classes

1. **Literature Survey and Review**

**3.1 Literature Summary:**

Character recognition is better known as optical character recognition (OCR) since it deals with recognition of optically processed-characters rather than magnetically processed. Though the origin of character recognition can be found as early as 1870, it first appeared as an aid to visually handicapped, and the first successful attempt was made by the Russian Scientist Tyurin in 1900. The modern version of OCR appeared in the middle of 1940s with the development of digital computers. The principal motivation for the development of OCR systems is the need to cope with the enormous flood of paper like bank cheques, commercial forms, government records, credit card imprints and mail sorting.

**Character recognition using Support Vector Machines**

Pattern recognition is formally defined as the process whereby a received pattern is assigned to one of a prescribed number of classes. Pattern recognition performed by a support vector machine is statistical in nature, with the patterns being represented by points in a multidimensional decision space. The decision space is divided into regions, each of which is associated with a class. The decision boundaries are determined by the training process. The construction of these boundaries is made statistical by the inherent variability that exists within and between classes. The ultimate success of SVM depends on their effectiveness in solving a variety of real life pattern recognition problems that are more demanding and more difficult. Handwritten character recognition represents an important example of realistic, yet difficult benchmark pattern recognition task. Analysis shows that many difficult pattern recognition problems can be formulated as multidimensional curve fitting using the methods of approximate learning. In handwritten character recognition, however, the general rules for distinguishing between characters are neither known, nor have they been formulated. The best solution is to examine a large cross section of the character population with the goal of finding a function that adequately generalizes the exemplars from the training set. This would possibly allow for recognition of the test characters not contained in the original training set.

1. **Methodology**

**4.1 Method Used:**

In this project we use a pipelined approach to implement the OCR process. We first start with denoising the image and then applying a deskewing algorithm to obtain straight text. We then segment all the characters from the textual regions using line detection, word detection and character detection modules. The detected characters are then sent for recognition. The recognised characters are then used to reconstruct the text back. Finally the reconstructed text is saved in a text file.

Apart from recognition one important aspect of this project was to process out the data obtained from emnist-letters and char74k dataset.

Since the char74k dataset was just images in separate folders, it had to be resized to 28\*28 pixel size and then converted into a numpy array of 784 elements. Then it was normalised to represent a value between 0 and 1. Finally it was combined with its corresponding class vectors and saved as an npy file to be used for training by the svm.

The svm used here is a basic svm which uses a linear kernel without cross validation to classify the characters into one of the 62 classes.

**4.2 Applications:**

This project can be used to digitise the following types of documents:

1. Typewritten documents with less noise
2. Screenshots of textual matter without any separators such as boxes and lines in the images
3. **System Implementation**

**5.1 Code:**

The project is written in python and contains the following:

**Module Interface: Combines all the modules in the OCR pipeline**

**import** noise\_removal **as** nr  
**import** skew\_correction **as** sc  
**import** line\_detector **as** ldr  
**import** word\_detector **as** wdr  
**import** character\_detector **as** chardet  
**import** pre\_recognition\_processing **as** prp  
**import** character\_recognition **as** recognise  
**import** text\_generator **as** tg  
**import** output\_file\_opener **as** opf  
noise\_removed\_image = nr.image\_noise\_removal(**r'C:\Users\Acer\PycharmProjects\Image\_processing-1\clear.jpg'**)  
skew\_corrected = sc.skew\_corrector(noise\_removed\_image)  
lines = ldr.line\_extractor(skew\_corrected)  
words = wdr.word\_extractor(lines)  
segmented\_characters = chardet.character\_detector(words)  
test\_data = prp.pre\_recognition(segmented\_characters)  
prediction\_matrix = recognise.character\_prediction(test\_data)  
fileName = tg.text\_generation(segmented\_characters,prediction\_matrix)  
opf.open\_file(fileName)

**noise\_removal: Used for removing noise from the image**

**import** cv2  
**def** image\_noise\_removal(path):  
raw\_image = cv2.imread(path)  
raw\_gray = cv2.cvtColor(raw\_image, cv2.COLOR\_BGR2GRAY)  
raw\_gray = cv2.bitwise\_not(raw\_gray)  
cv2.imshow(**"Original Image"**, raw\_image)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()  
canny = cv2.Canny(raw\_gray,100,200)  
smoothened\_gray\_image = raw\_gray-canny  
smoothened\_gray\_image= cv2.resize(smoothened\_gray\_image,(600,600))  
 **return** smoothened\_gray\_image

**skew\_correction: Used to deskew text images**

**import** numpy **as** np  
**import** cv2  
**def** skew\_corrector(image):  
thresh = cv2.threshold(image, 0, 255,  
 cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)[1]  
white\_coordinates = np.column\_stack(np.where(thresh > 0))  
rectangle = cv2.minAreaRect(white\_coordinates)  
angle = rectangle[-1]  
**if** angle< -45:  
 angle = -(90 + angle)  
 **else**:  
 angle = -angle  
(h, w) = image.shape[:2]  
center = (w // 2, h // 2)  
affine\_matrix = cv2.getRotationMatrix2D(center, angle, 1.0)  
rotated = cv2.warpAffine(thresh, affine\_matrix, (w,h),  
 flags=cv2.INTER\_CUBIC, borderMode=cv2.BORDER\_REPLICATE)  
 rotated = cv2.threshold(rotated, 0, 255,cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)[1]  
cv2.imshow(**'Deskewed Image'**,rotated)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()  
**return** rotated

**line\_detector: used to detect lines in textual regions**

**import** cv2  
**def** line\_extractor(deskewed\_image):  
kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (1,2))  
eroded\_image = cv2.erode(deskewed\_image, kernel, iterations=0)  
x\_sum = cv2.reduce(eroded\_image, 1, cv2.REDUCE\_AVG)  
 hist = []  
height = eroded\_image.shape[:2][0]  
 white\_pixel\_threshold = 20  
 transition\_count\_threshold = 4  
 line\_thickness\_threshold = 10  
p = int(eroded\_image.shape[1] \* 0.01)  
 eroded\_image[:, 0:p] = 0  
 eroded\_image[:, eroded\_image.shape[1] - p:] = 0  
 **for** i **in** range(0,height):  
 **if** x\_sum[i][0]==0:  
 hist.append(**False**)  
 **elif** x\_sum[i][0]>0 **and** x\_sum[i][0]<=white\_pixel\_threshold: l = eroded\_image[i]  
 transition\_count = 0  
 **for** j **in** range(1,len(l)):  
 **if** l[j]!=l[j-1]:  
 transition\_count+=1  
 **if** transition\_count>=transition\_count\_threshold:  
 hist.append(**True**)  
 **else**:  
 hist.append(**False**)  
 **else**:  
 hist.append(**True**)

j=1  
 y\_start=0

y\_end = 0

y\_coord = []

i=0

**while** i<height:  
 j=1  
 **if not**(hist[i]): *#If it is a space row* i=i+1  
 **continue  
 else**:  
 y\_start = i

k = i *# second counter* j=0  
 **while** k<height **and** hist[k]: k+=1  
 j+=1  
 i = i+j

y\_end = y\_start+j

**if** y\_end-y\_start>line\_thickness\_threshold:**if** height-y\_end>2: y\_coord.append((y\_start-2,y\_end+2)) **else**:  
 y\_coord.append((y\_start-2,y\_end)) lines = []  
 **for** i **in** range (0,len(y\_coord),1):  
 length = y\_coord[i][1]-y\_coord[i][0]  
 roi = eroded\_image[y\_coord[i][0]:y\_coord[i][0]+length,0:height]  
 lines.append(roi)display\_image=eroded\_image.copy()  
 **for** i **in** range(0,len(y\_coord)):  
 cv2.rectangle(display\_image, (0, y\_coord[i][0]), (600, y\_coord[i][1]), (255, 255, 255), 1)  
 cv2.imshow(**'Detected Lines'**, display\_image)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()  
**return** lines

**word\_detector: used to detect words in textual lines**

**import** cv2  
**import** numpy **as** np  
**def** word\_extractor(lines):words = [] **for** i **in** range(0,len(lines)):  
 raw\_line = lines[i].copy()  
 raw\_line = cv2.resize(raw\_line,(900,300))kernel=cv2.getStructuringElement(cv2.MORPH\_RECT,(2,2)) copy\_line=cv2.dilate(raw\_line,kernel,iterations=2) y\_sum = cv2.reduce(copy\_line, 0, cv2.REDUCE\_AVG)

y\_sum=y\_sum[0] pass\_1\_count = [] *1* j=1  
 i=0  
 x\_start=0  
 x\_end=0  
 x\_coord=[] **while** i<len(y\_sum):  
 j=1  
 **if** y\_sum[i]!=0:  
 i=i+1  
 **continue  
 else**: x\_start=i  
 k=i  
 j=0  
 **while** k<len(y\_sum) **and** y\_sum[k]==0: k+=1  
 j+=1  
 i=i+j  
 x\_end=x\_start+j x\_coord.append([x\_start,x\_end]) pass\_1\_count.append(j) pass\_1\_count=pass\_1\_count[1:-1] x\_coord=x\_coord[1:-1] average=5  
 **if**(len(pass\_1\_count)!=0):  
 average = sum(pass\_1\_count)//len(pass\_1\_count) loc=[]

i=0  
 threshold = 0.4 **for** i **in** range(0,len(x\_coord)):  
 difference = x\_coord[i][1]-x\_coord[i][0] **if** difference/average<threshold:  
 loc.append(x\_coord[i])  
 **for** i **in** range(0,len(loc)):  
 x\_coord.remove(loc[i]) word\_line=[]

height = raw\_line.shape[:2][0]

roi = raw\_line[0:height,0:x\_coord[0][0]] y\_sum = cv2.reduce(roi, 0, cv2.REDUCE\_AVG)  
 y\_sum=y\_sum[0]  
 i=0  
 **while**(y\_sum[i]<=0

i+=1  
 roi = raw\_line[0:height,i:x\_coord[0][0]] roi = cv2.resize(roi,(500,500)) word\_line.append(roi) **for** i **in** range(1,len(x\_coord)):  
 roi=raw\_line[0:height,x\_coord[i-1][1]:x\_coord[i][0]]  
 roi = cv2.resize(roi,(500,500))  
 word\_line.append(roi)  
roi = raw\_line[0:height,x\_coord[len(x\_coord)-1][1]:900]  
 y\_sum = cv2.reduce(roi, 0, cv2.REDUCE\_AVG)  
 y\_sum=y\_sum[0]  
 i=len(y\_sum)-1  
 **while**(y\_sum[i]<=0):  
 i-=1  
 roi = raw\_line[0:height,x\_coord[len(x\_coord)-1][1]:x\_coord[len(x\_coord)-1][1]+i]  
 roi = cv2.resize(roi,(500,500))  
 word\_line.append(roi)  
words.append(word\_line)  
 **return** words

**character\_detector: used to detect words in textual words**

**import** cv2  
**def** character\_detector(words):  
all\_characters=[]  
 kernel = cv2.getStructuringElement(cv2.MORPH\_RECT,(3,3))  
 **for** i **in** words:  
 line\_character=[]  
 j=i  
 **for** p **in** j:  
 word\_character=[]  
 s=p  
 s = cv2.threshold(p,100,255,cv2.THRESH\_OTSU+cv2.THRESH\_BINARY)[1]  
 s = cv2.erode(s, kernel, iterations=2)  
 x\_sum = cv2.reduce(s, 0, cv2.REDUCE\_AVG) x\_sum=x\_sum[0]  
 x\_avg = sum(x\_sum)//500  
 hist =[]  
 **for** i **in** range(0,500):  
 **if**(x\_sum[i]==0):  
 hist.append(**False**)  
 **else**:  
 hist.append(**True**)  
 j=1  
 x\_start=0

x\_end = 0 x\_coord=[]

i=0 **while** i<500:  
 j=1  
 **if not**(hist[i]): i=i+1  
 **continue  
 else**:  
 x\_start = i temp = i j=0  
 **while** temp<500 **and** hist[temp]: temp+=1  
 j+=1  
 i= i+j

x\_end = x\_start+j x\_coord.append((x\_start,x\_end))  
 **for** i **in** range (0,len(x\_coord),1):  
 roi = s[0:500,x\_coord[i][0]:x\_coord[i][1]]  
 word\_character.append(roi)  
 line\_character.append(word\_character)  
 all\_characters.append(line\_character)  
 **return** all\_characters

**pre\_recognition\_processing: used to process characters for recognition**

**import** cv2  
**import** numpy **as** np  
**def** pre\_recognition(all\_characters):  
modified\_characters=[]  
 i=0  
 **for** lines **in** all\_characters:  
 **for** words **in** lines:  
 **for** characters **in** words:  
 copy = characters  
copy = cv2.resize(copy,(24,24))  
 copy= cv2.copyMakeBorder(copy,2,2,2,2,cv2.BORDER\_CONSTANT,value=[0,0,0])  
 copy = cv2.bitwise\_not(copy)  
 copy = copy/ 255  
 **if**(i<11):  
 cv2.imshow(**'Copy'**,copy)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 i+=1  
 copy = copy.flatten(**'F'**)  
 modified\_characters.append(copy)  
 modified\_characters = np.array(modified\_characters)  
 print(**'No of characters detected:'** +str(modified\_characters.shape[0]))  
 **return** modified\_characters

**character\_recognition: used to recognise characters from the svm model**

**import** cv2  
**import** numpy **as** np  
**from** sklearn **import** svm  
**import** pickle  
**def** character\_prediction(test\_data):  
 filename = **'finalized\_model.sav'** loaded\_model = pickle.load(open(filename, **'rb'**))  
 predictions = loaded\_model.predict(test\_data)  
 print(**'Prediction Matrix is:'**)  
 print(predictions)  
 **return** predictions

**text\_generator: used to generate text from the predicted characters**

**def** text\_generation(list1,output\_list):  
 ref\_list = [**'0'**,**'1'**,**'2'**,**'3'**,**'4'**,**'5'**,**'6'**,**'7'**,**'8'**,**'9'**,**'A'**,**'B'**,**'C'**,**'D'**,**'E'**,**'F'**,**'G'**,**'H'**,**'I'**,**'J'**,**'K'**,**'L'**,**'M'**,**'N'**,**'O'**,**'P'**,**'Q'**,**'R'**,**'S'**,**'T'**,**'U'**,**'V'**,**'W'**,**'X'**,**'Y'**,**'Z'**,**'a'**,**'b'**,**'c'**,**'d'**,**'e'**,**'f'**,**'g'**,**'h'**,**'i'**,**'j'**,**'k'**,**'l'**,**'m'**,**'n'**,**'o'**,**'p'**,**'q'**,**'r'**,**'s'**,**'t'**,**'u'**,**'v'**,**'w'**,**'x'**,**'y'**,**'z'**]  
 output = []  
 c = 0  
 **for** k **in** range(len(list1)):  
 t1 = list1[k]  
 **for** j **in** range(len(t1)):  
 t2 = t1[j]  
 t = []  
 **for** i **in** range(len(t2)):  
 t.append(ref\_list[output\_list[c]])  
 c = c + 1  
 t = **""**.join(t)  
 t=t.lower()  
 output.append(t)  
 output.append(**" "**)  
 output.append(**"\n"**)  
 output = **""**.join(output)  
 fileName = **"recognised\_text.txt"** f = open(**"recognised\_text.txt"**,**"w+"**)  
 f.write(output)  
 f.close()  
 print(**'Recognition done!! Opening file:'**)  
 **return** fileName

**output\_file\_opener: used to open output file in notepad**

**import** subprocess **as** sp  
**def** open\_file(fileName):  
 programName = **"notepad.exe"** sp.Popen([programName, fileName])

**mnist\_loader: used to load char74k dataset in mnist format**

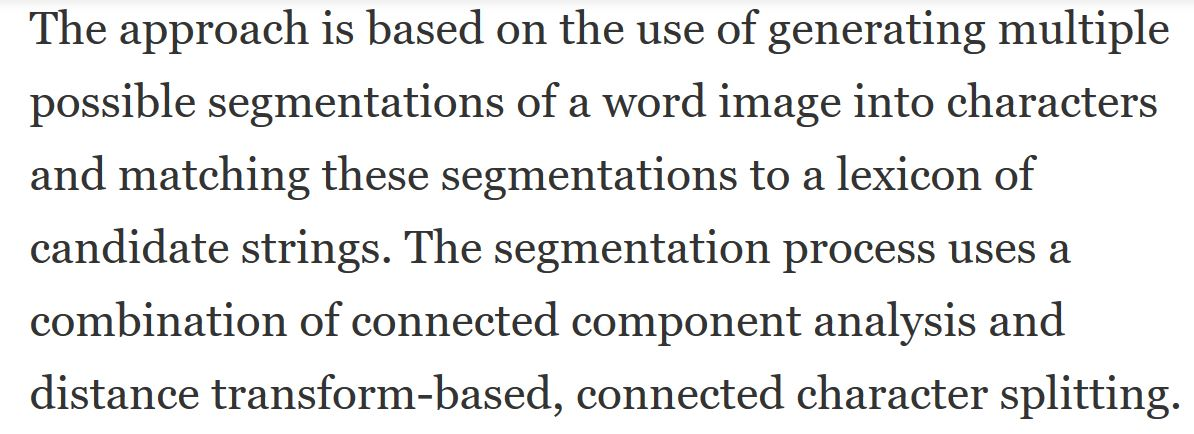
**import** numpy **as** np  
**def** load\_data(name):  
 data = np.load(name)  
 **return** data  
**def** load\_data\_wrapper():  
 name = **'training\_data.npy'** tr\_d = load\_data(name)  
 training\_inputs = [np.reshape(x, (784, )) **for** x **in** tr\_d[0]]  
 training\_results\_vectors = [np.reshape(y,(62,)) **for** y **in** tr\_d[1]]name = **'testing\_data.npy'** te\_d = load\_data(name)  
 test\_inputs = [np.reshape(x, (784, )) **for** x **in** te\_d[0]]  
 test\_result\_vector = [np.reshape(y,(62,)) **for** y **in** te\_d[1]]  
 test\_results =[]  
 training\_results=[]  
 **for** a **in** training\_results\_vectors:  
 i = a.tolist().index(1.)  
 training\_results.append(i)  
 **for** a **in** test\_result\_vector:  
 i = a.tolist().index(1.)  
 test\_results.append(i)  
 test\_results = np.array(test\_results)  
 **return** training\_inputs , training\_results, test\_inputs,test\_results

**svm: used to load and train on formatted char74k dataset**

**import** mnist\_loader  
**from** sklearn **import** svm  
**import** numpy **as** np  
**import** pickle  
  
**def** svm\_baseline():  
 training\_input, training\_results, test\_input,test\_results = mnist\_loader.load\_data\_wrapper()training\_input = np.array(training\_input)  
 training\_results = np.array(training\_results)  
 test\_input = np.array(test\_input)  
 test\_results = np.array(test\_results)  
 clf = svm.SVC()  
 clf.fit(training\_input, training\_results)  
 filename = **'finalized\_model2.sav'** pickle.dump(clf, open(filename, **'wb'**))  
 predictions = [ int(a) **for** a **in** clf.predict(test\_input)]  
 num\_correct = sum(int(a == y) **for** a, y **in** zip(predictions, test\_results))  
 print(**"Baseline classifier using an SVM."**)  
 print(str(num\_correct) + **" of "** + str(len(test\_input)) + **" values correct."**)**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 svm\_baseline()

**5.2 Results and discussion:**

The image processing part consists of segmenting the textual image into its constituent characters and this task was performed fairly accurately for an image with less noise.



**Figure 4: Sample text image**

The algorithm successfully segmented the image into its 272 characters ignoring ‘.’ The first three segmented characters from figure 4 are as follows:



**Figure 5: First 3 segmented characters from the text image**

Once characters were segmented they were sent for recognition. The following datasets were used to train the svm classifier:

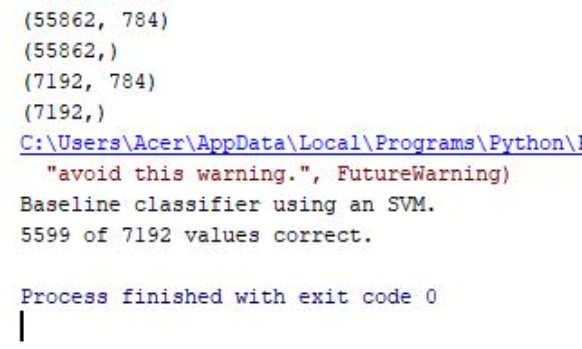
**Char74k dataset:**

In the **English** language, Latin script (excluding accents) and Hindu-Arabic numerals are used. The dataset consists of:

* 62 classes (0-9, A-Z, a-z)
* 7705 characters obtained from natural images
* 3410 hand drawn characters using a tablet PC
* 62992 synthesised characters from computer fonts

This dataset was divided into a training set with 55,862 samples and a testing set of 7,912 images.

The classifier achieved an accuracy of 77.85% and was able to classify 5599 out of 7912 test samples correctly.



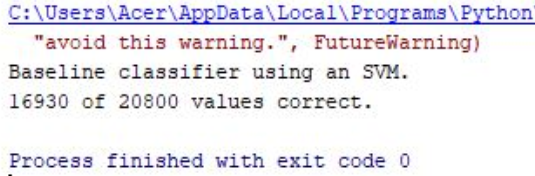
**Figure 5: SVM baseline classifier results on char74k dataset**

**Emnist-letters dataset:**

EMNIST Letters: 145,600 characters. 26 balanced classes.

The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class task.

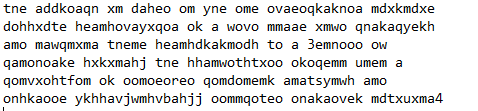
The svm classifier performed better on this dataset classifying 16930 samples correctly out of 20800 samples to achieve an accuracy of 81.39%.



**Figure 5: SVM baseline classifier results on emnist-letters dataset**

However since the emnist dataset did not have digits it could not be used as the final svm model. The char74k did not perform well while recognising the segmented characters and often ended up recognising ‘h’ as ‘n’ and ‘p’ as ‘d’.

This is because the characters segmented from the image had some distortions and the char74k contains perfect fonts and does not contain any distortion. This gave poor recognition results even though the characters in the segmented image were fairly recognisable. Thus lack of a proper dataset has made the recognition process tough.



**Figure 6: Recognised text using svm trained on chars74k dataset**

1. **Conclusion And Future Work**

**6.1 Conclusion:**

The OCR software developed here has a decent image processing module and segments the characters fairly accurately. However due to the lack of proper dataset the svm classifier misclassified the segmented characters making it difficult to use. The project however, successfully demonstrates the OCR pipeline and how it can be implemented. It also brings forth the importance of a good dataset for reliable recognition results.

**6.2 Future Work:**

We aim to make the following changes to the project to enhance its usability and reliability:

* Augmenting Dataset: Augmenting chars74k dataset to achieve distorted fonts can be done before training a svm to achieve better results.
* Using an autocorrect module to correct recognised words which are one to two edit distances away from actual words.
* Using deep neural network to train the statistical model to achieve better recognition results.

1. **References**

* Open cv documentation
* https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
* http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/
* https://www.nist.gov/itl/iad/image-group/emnist-dataset
* https://www.pyimagesearch.com/2017/02/20/text-skew-correction-opencv-python/
* https://stackoverflow.com/questions/14211413/segmentation-for-connected-characters
* https://docs.opencv.org/3.0-beta/doc/py\_tutorials/py\_imgproc/py\_morphological\_ops/py\_morphological\_ops.html
* http://www.imagemagick.org/discourse-server/viewtopic.php?t=26571
* https://www.danvk.org/2015/01/07/finding-blocks-of-text-in-an-image-using-python-opencv-and-numpy.html
* https://www.pyimagesearch.com/2015/11/30/detecting-machine-readable-zones-in-passport-images/