Project Report

Handwritten Character Recognition System Using Convolution Neural Networks

ECE 569A

University of Victoria

Submitted to:

Dr. Deepali Arora

Submitted by:

Sanjana Arora (V00966221)

Siddharth Chadda (V00947906)

Tavanpreet S. Oberoi (V00963163)

TABLE OF CONTENTS

1. Introduction

For the ECE569A course project we have implemented a handwritten character recognition system using Convolution Neural Networks that facilitates users to interpret handwritten characters images.

For this project, we will use the EMNIST [3] balanced dataset of 131,600 characters, 47 balanced classes for training and testing our model. EMNIST [3] is a large dataset that provides a good combination of handwritten lower/upper case characters and digits converted into 28x28 pixel image format required to train our convolution neural network model [4].

Problem Statement and Motivation

In India, medical prescriptions are still written by hand and most of the times it's challenging to recognize what's written on the prescription due to poor handwriting. This problem poses a risk of buying wrong drugs from the pharmacist and that could eventually lead to the patient consuming drugs that could prove detrimental for their health. Motivated by these issues, we propose a python-based solution that allows a user to input their medical prescription into our system and the model would recognize the characters for display. Thus, a user can easily read and understand what's written on the medical prescription, eventually, consuming the drugs that are actually prescribed by the doctor. The proposed system can also be implemented in purchasing the medicines online, wherein, the user can simply upload the prescription and the system would identify the medicines to be ordered on its own. This makes the online purchase of medicines convenient for both the consumer and pharmacists.

Furthermore, this project may find many applications in the legal and banking industries that still largely depend on paper work. This AI based solution can assist in converting and digitizing those bulks of handwritten documents that would have otherwise been done manually.

2. Related Work

For approaching this problem, we are using the ideas proposed by R.Manmatha and N.Srimal in their work [1] and Avula Manisree, Rasakatla Rushmitha and Madduri Divya in their work [2]. R.Manmatha and N.Srimal propose a method of performing line segmentation that allows separating consecutive lines. Word segmentation is followed by line segmentation, wherein, each line is examined to extract words. Afterwards character segmentation is done on the extracted words to break the words down to their constituent characters. To recognize the characters from the input words we plan to build a Convolution Neural Network using the TensorFlow library.

3. Problem Formulation

Doctor's prescriptions are written in single/multiple or cursive/non-cursive stroke handwriting which makes it difficult to distinguish between the characters. It is a complicated task to

recognize handwritten characters especially when they are skewed or irregular or slanted. To simplify the objective, the task is majorly divided into 4 sections as:

- Line Segmentation
- Word Segmentation
- Character Segmentation
- Digitizing based on Convolutional Neural Network Training and Prediction

The flow graph of the problem statement has been shared in the below figure.

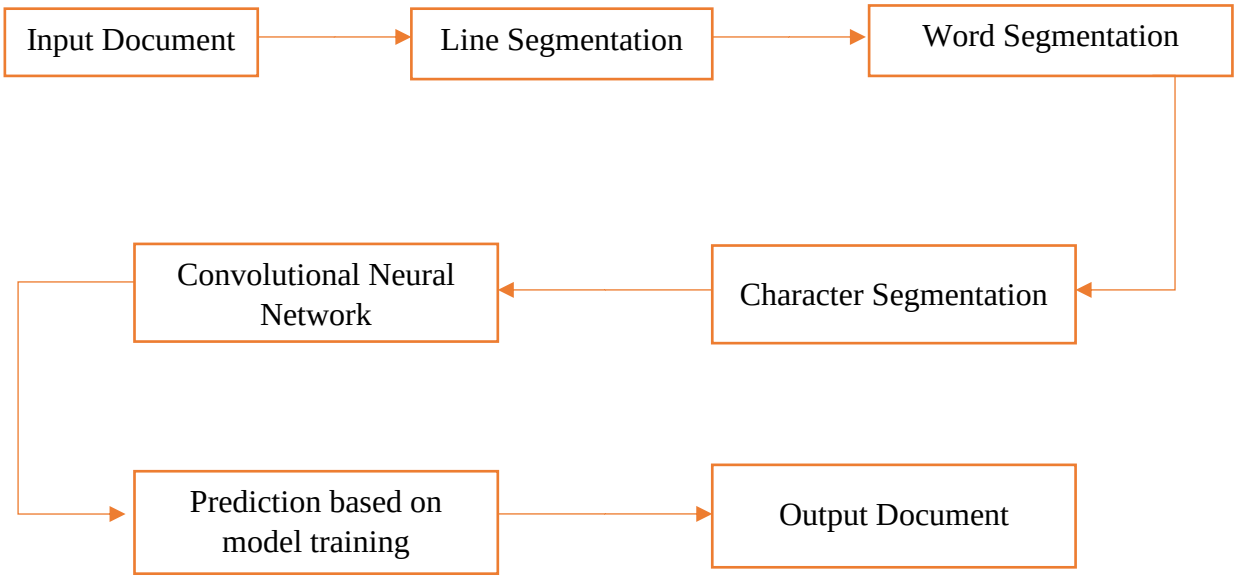


Figure 1. Flow Graph of Problem Statement

Algorithm for Line Segmentation

This functionality of this algorithm is used to convert the scanned prescription images into the BMP format, which further is converted into the 2-Dimensional binary matrix. It helps in finding the different lines in the document. The following steps are followed for segmenting lines from the document given in the research paper:

- Converting the scanned document into a 2-D array.
- Finding the bounding box along with the image by locating the topmost and bottommost rows of the text in the image.
- The topmost row is located by finding the black spot from the top.
- Similarly, the bottommost row is located by finding the black spot from the bottom.
- Rotate the image, so that lines can be split based on columns.
- Now to split into the lines, a 2-Dimensional array is split based on columns, as 1 column will contain 1 line.
- After the split, multiple row images are displayed and stored in the desired folder.

Algorithm for Word Segmentation

This functionality of the algorithm is used to convert the single line images to multiple word images. It helps in finding the words present in the one line of the given image. The following steps are followed for segmenting words from the document given in the research paper:

- The images of the lines are being read from the folder.
- For each line image, the below actions are performed.
    - Normalization is done to pass the image for segmentation.
    - In segmentation, a rectangular box is created along with each word, if the gap between characters is more than the minimum area, then it split and creates a new rectangular box from that point.
    - Continuous characters comprising the words are calculated based on the presence of black pixels.
- Now each rectangular box area is converted to an image and stored in the desired folder.

Algorithm for Character Segmentation

This functionality of the algorithm is used to convert the single word images to multiple character images. It helps in finding all the characters present in the word image. The following steps are followed for segmenting characters from the document given in the research paper:

- The images of the words are being read from the folder.
- For each word image, the below actions are performed.
    - Rotate the image, so that characters can be split based on columns.
    - Now to split on the characters, a 2-Dimensional array is split based on columns, as 1 column will contain 1 character based on consecutive black pixels.
- After the split, all the character images are stored in the desired folder.

The segmentation of the handwritten characters forms a major part of this project. Now, these character images will be fed to a trained Convolutional Neural Network and the result will be predicted.

The further process of **Digitizing based on Convolutional Neural Network Training and Prediction** will be discussed in detail under the Methodology section of the report for ease of understanding.

4. Methodology

The project is divided into two parts: -

- **First part**
  The first part of the project comprises of image processing steps comprising of receiving an input image, breaking the image into lines, words and characters using bounding boxes, image segmentation. Single character images are then used as input into the convolution neural networks to predict the digitized character.

- **Second part**
  Second part of the project comprises of training of Neural Networks with all 62 classes using convolutional layers, ReLu activation, Maxpooling layers and fully connected layers. We have saved the convolutional Neural Network models so as to re-use the same later.

Data Pre-processing

As part of the data pre-processing, we have flipped and rotated the EMNIST images. The images have also been reshaped between (-1,1). Further, we have rescaled the image intensity between 0 to 1 by dividing the data with 255 (maximum pixel intensity). As the EMNIST provides data labels in form of alphabetical and numerical format, we have one-hot encoded the target data labels as the Convolutional Neural Networks are capable of interpreting Boolean target labels.

Building the Neural Network models

We have used **two** Convolutional Neural Network models: -

The **first** convolutional network model comprising of eight layers including three convolutional layers, three maxpooling layers, two fully connected layers and one flatten layer. Below is the brief description about the architecture of the model:

Input --> 2 D Convolution layer --> Maxpool 2 D --> 2 D Convolution layer --> Maxpool 2 D --> 2 D Convolution layer --> Flatten --> Dense -1 --> Dropout -->Dense -2

- Input with (28,28) dimensions is fed into layer C1 2D convolutional layer having Rectifier Linear Unit (RELU) activation function, kernel size 3, filter of size 128 and "Same" padding.
- Then layer C2 is Maxpool 2D layer with pool size of 2 and strides of 2.
- Thereafter, we have used the second 2D convolutional layer with filter of size 64, Rectifier Linear Unit (RELU) activation function, kernel size 3 and "Same" padding.
- Layer C4 is Maxpool 2D layer with pool size of 2 and strides of 2.
- Layer C5 is third 2D convolutional layer with filter size 32, Rectifier Linear Unit (RELU) activation function, kernel size 3 and "Same" padding.
- Layer C6 is Maxpool 2D layer with pool size of 2 and strides of 2.
- Then we have used a dense layer of 128 neurons with Rectifier Linear Unit (RELU) activation function and a dropout of 0.5.
- Layer C8 is dense layer that has 10 dimensions and outputs the predicted value.

We ran the model for 20 epochs, however, we set an early stopping criterion such that the training stops as soon as the testing accuracy starting falling down after reaching the highest value. Below screenshot represents the results of the above-described sequential neural network:

```
Epoch 1/10
794/794 [==============================] - 307s 386ms/step - loss: 1.3089 - accuracy: 0.6145 - val_loss: 0.4959 - val_accuracy: 0.8311
Epoch 2/10
794/794 [==============================] - 307s 387ms/step - loss: 0.6498 - accuracy: 0.7876 - val_loss: 0.4234 - val_accuracy: 0.8460
Epoch 3/10
794/794 [==============================] - 309s 389ms/step - loss: 0.5594 - accuracy: 0.8146 - val_loss: 0.4123 - val_accuracy: 0.8542
Epoch 4/10
794/794 [==============================] - 307s 386ms/step - loss: 0.5043 - accuracy: 0.8306 - val_loss: 0.3797 - val_accuracy: 0.8609
Epoch 5/10
794/794 [==============================] - 306s 385ms/step - loss: 0.4776 - accuracy: 0.8373 - val_loss: 0.3586 - val_accuracy: 0.8693
Epoch 6/10
794/794 [==============================] - 304s 383ms/step - loss: 0.4493 - accuracy: 0.8462 - val_loss: 0.3435 - val_accuracy: 0.8768
Epoch 7/10
794/794 [==============================] - 306s 385ms/step - loss: 0.4356 - accuracy: 0.8498 - val_loss: 0.3453 - val_accuracy: 0.8767
Epoch 8/10
794/794 [==============================] - 318s 400ms/step - loss: 0.4141 - accuracy: 0.8556 - val_loss: 0.3387 - val_accuracy: 0.8738
Epoch 9/10
794/794 [==============================] - 322s 405ms/step - loss: 0.4012 - accuracy: 0.8601 - val_loss: 0.3375 - val_accuracy: 0.8768
Epoch 10/10
794/794 [==============================] - 317s 399ms/step - loss: 0.3885 - accuracy: 0.8631 - val_loss: 0.3318 - val_accuracy: 0.8799
<tensorflow.python.keras.callbacks.History at 0x7fd3bbcf8e90>
```

Figure 2. Accuracy/Loss of training data based on Epochs

Finally, the model is compiled, “Adam” optimizer and accuracy metrics are used. The “categorical_crossentropy” loss function is used to train the machine learning model. The accuracy metric is used to evaluate the performance of training.

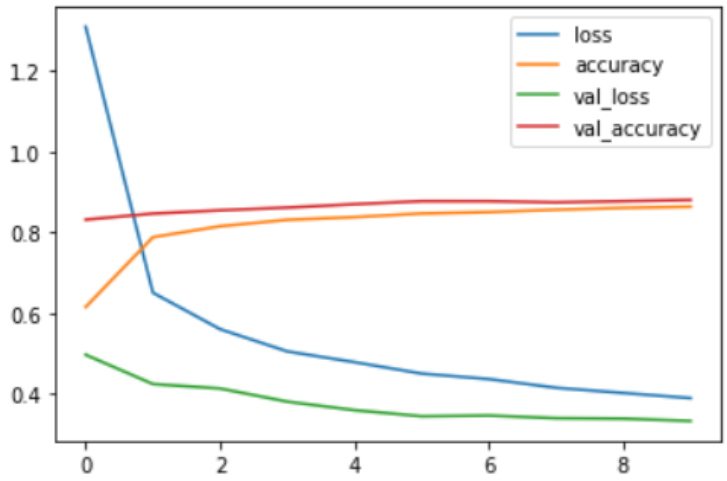


Figure 3. Accuracy/Loss of Training Data Plot

The Figure 3 screenshot captures the training accuracy, training loss, testing accuracy and testing loss.

The **second** model that we have implemented follows LeNet-5 architecture [5] that has 7 layers, excluding input layers.

- Layer C1 is the first convolutional layer with stride of 1, kernel size 5, relu activation.
- Layer C2 is max pooling layer with stride 2.
- Layer C3 is the second convolutional layer with ReLu activation function, kernel size of 5.
- Layer C4 is max pooling layer with stride 2.

- The next layer is responsible for flattening the output of the previous layer into one dimensional array.
- Layer C5 is a dense block with ReLu activation function.
- Layer C6 is another dense block with ReLu activation function.
- Output has 10 dimensions (equals number of classes in the database).

The Figure 4 screenshot captures the LeNet-5 model summary: -

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 28, 28, 32)        832
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 10, 10, 48)        38448
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 5, 5, 48)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 1200)              0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               307456
_________________________________________________________________
dense_11 (Dense)             (None, 84)                21588
_________________________________________________________________
dense_12 (Dense)             (None, 47)                3995
=================================================================
Total params: 372,319
Trainable params: 372,319
Non-trainable params: 0
_________________________________________________________________
```

Figure 4. Summary for LeNet-5 Model

Finally, the model is compiled, “Adam” optimizer and accuracy metrics are used. The “categorical_crossentropy” loss function is used to train the machine learning model. The accuracy metric is used to evaluate the performance of training.

We ran the model for 20 epochs, however, we set an early stopping criterion such that the training stops as soon as the testing accuracy starting falling down after reaching the highest value. Below screenshot represents the results of the LeNet-5 neural network model:

```
Epoch 1/20
3173/3173 [==============================] - 118s 37ms/step - loss: 0.6346 - accuracy: 0.7943 - val_loss: 0.4049 - val_accuracy: 0.8568

Epoch 00001: val_loss improved from inf to 0.40493, saving model to LesNetmodel.h5
Epoch 2/20
3173/3173 [==============================] - 118s 37ms/step - loss: 0.3715 - accuracy: 0.8670 - val_loss: 0.3645 - val_accuracy: 0.8677

Epoch 00002: val_loss improved from 0.40493 to 0.36452, saving model to LesNetmodel.h5
Epoch 3/20
3173/3173 [==============================] - 117s 37ms/step - loss: 0.3194 - accuracy: 0.8817 - val_loss: 0.3413 - val_accuracy: 0.8728

Epoch 00003: val_loss improved from 0.36452 to 0.34126, saving model to LesNetmodel.h5
Epoch 4/20
3173/3173 [==============================] - 116s 37ms/step - loss: 0.2829 - accuracy: 0.8931 - val_loss: 0.3465 - val_accuracy: 0.8775

Epoch 00004: val_loss did not improve from 0.34126
Epoch 5/20
3173/3173 [==============================] - 117s 37ms/step - loss: 0.2563 - accuracy: 0.9003 - val_loss: 0.3454 - val_accuracy: 0.8784

Epoch 00005: val_loss did not improve from 0.34126
Epoch 6/20
3173/3173 [==============================] - 116s 36ms/step - loss: 0.2347 - accuracy: 0.9069 - val_loss: 0.3597 - val_accuracy: 0.8773

Epoch 00006: val_loss did not improve from 0.34126
Epoch 7/20
3173/3173 [==============================] - 116s 37ms/step - loss: 0.2156 - accuracy: 0.9126 - val_loss: 0.3779 - val_accuracy: 0.8761

Epoch 00007: val_loss did not improve from 0.34126
Epoch 8/20
3173/3173 [==============================] - 115s 36ms/step - loss: 0.2003 - accuracy: 0.9174 - val_loss: 0.3936 - val_accuracy: 0.8693

Epoch 00008: val_loss did not improve from 0.34126
Epoch 00008: early stopping
<tensorflow.python.keras.callbacks.History at 0x7fd3b8570910>
```

Figure 5. Training Accuracy/Loss based on LeNet-5 Model

LeNet-5 architecture provides much better training accuracy (91.74%) as compared to the training accuracy (86.31%) achieved using previous Convolutional Neural Network Architecture. Both the models provide approximately 87% testing accuracy. Therefore, we have used the LeNet-5 architecture below for the digitized predictions for the input.

Execution of the Code:

- Open ‘line_segmentation.ipynb’ and give path of the input image to segment the image based on lines. Execute the file and images will be stored in folder ‘data’ placed at same directory as ‘ipynb’ file.
- Now open ‘word_segmentation.ipynb’ and input files will be selected automatically. Execute the file and images will be stored in folder ‘out’ placed at same directory as ‘ipynb’ file in different folders based on lines.
- Now open ‘char_segmentation.ipynb’ and input files will be selected automatically. Execute the file and images will be stored in folder ‘out’ placed at same directory as ‘ipynb’ file in different folders based on lines and word.
- After segmentation, open the file ‘AI_Project.ipynb’ to train the model. Make sure to give correct path for EMNIST dataset. Execute the file and training will be saved in file ‘.h5’.
- Now open file ‘AI_Prediction.ipynb’ to predict the handwritten characters by feeding to the trained Convolutional Neural Network. Execute the file and select path to the input image. Result or predicted image will be displayed.

5. Results and Discussions

Input Image:

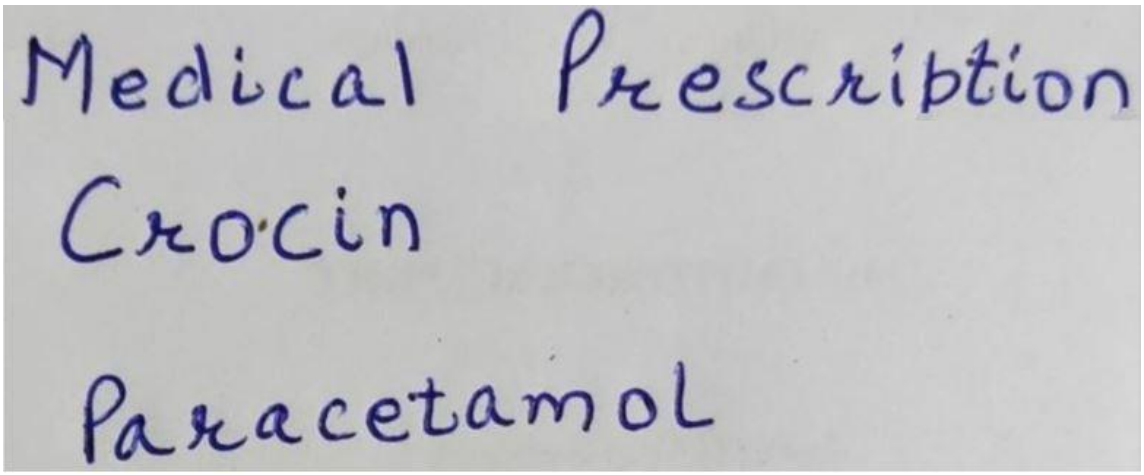
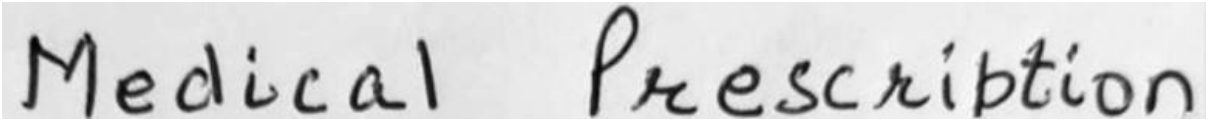


Figure 6. Input Image to the Model

Output Result from Line Segmentation:

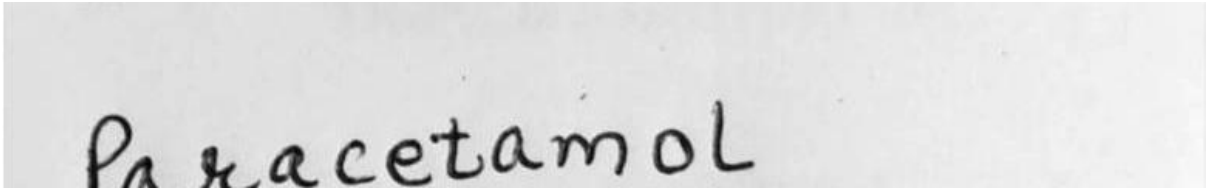
All line images will be stored as ‘PNG’ in folder ‘data’ placed at the same directory where python notebook is placed.

Line 1:



Line 2:

Line 3:

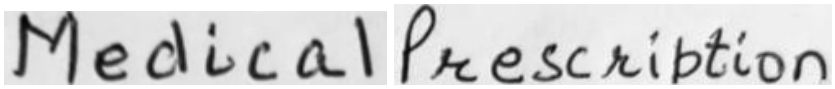


Output Result from Word Segmentation:

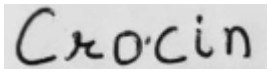
All word images will be stored as ‘PNG’ in folder ‘out’ under folder name ‘line number’ placed at the same directory where python notebook is placed.

Figure 7. Folder Creation for Word Segmentation

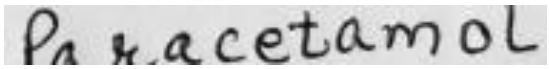
Line 1:



Line 2:



Line 3:



Output Result from Character Segmentation:

All character images will be stored as 'PNG' in folder 'out' under folder name 'line number' and 'word number' placed at the same directory where python notebook is placed.

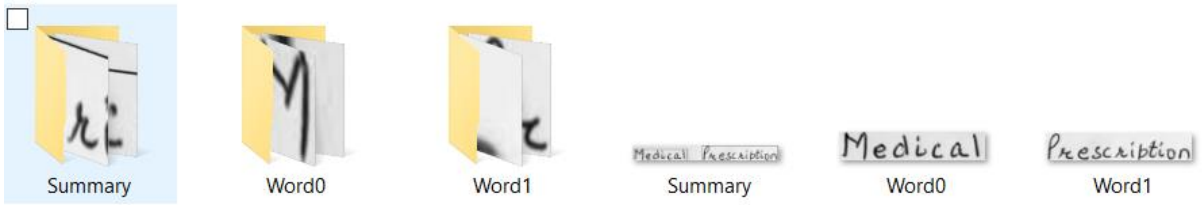
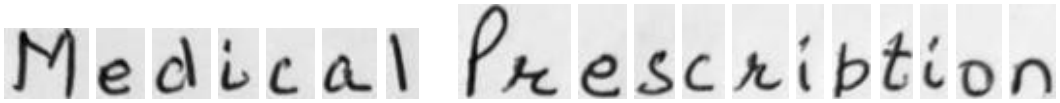
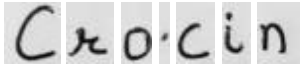


Figure 8. Folder creation for Character Segmentation

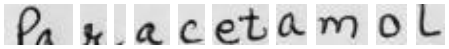
Line 1:



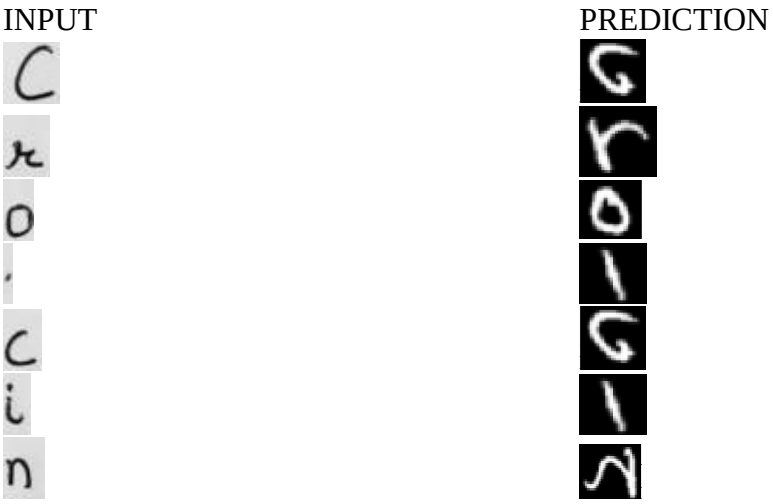
Line 2:



Line 3:



Results from Neural Network Prediction



It has been observed that after segmentation, some characters are combined which resulted in an incorrect output as model always return single character per image. Also, it can be seen a smaller ink has always been able to separate out in word crocin which gives confidence of correct segmentation, but not contribute well to prediction. There is much scope of improvements and will be done under future work.

6. Conclusion

We have successfully executed the concepts of computer vision to receive an input image, segment the same into lines, words and characters. Further, we have successfully trained Convolution Neural Network model and LeNet-5 architecture. The report also focuses on the evaluation metrics used for evaluating the convolution neural networks. We have demonstrated and presented the results of the Image segmentation and Convolutional Neural Network that can be used behind any application to digitize handwritten documents.

7. Future Work

The project is limited to only English language words and can be expanded to other languages as well. Further, the project is limited to single stoke handwritten notes, therefore, the project can be expanded to work for all types of handwritten notes including free hand cursive handwriting. A spell checker can be added as a full proof method to cover up for the approximately 13% inaccuracy of the model.

8. References

[1] Manmatha R, Srimal N. Scale space technique for word segmentation in handwritten documents. International conference on scale-space theories in computer vision 1999 Sep 26 (pp. 22-33). Springer, Berlin, Heidelberg.

[2] Manisree A, Rushmitha R, Divya M. Medical prescription generator using python. EasyChair; 2020 Dec 29.

[3] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373

[4] EMNIST: an extension of MNIST to handwritten letters, 17 Feb 2017·Gregory Cohen, Saeed Afshar, Jonathan Tapson, André van Schaik.

[5] https://towardsdatascience.com/convolutional-neural-network-champions-part-1-lenet-5-7a8d6eb98df6