Single-Line Handwritten Sentence Recognition

Shivam Pandey | Shivam Kumar | Abhishek Varghese

November 25, 2018

1 Introduction and Overview

Over the last few decades, researches on handwritten text recognition have made impressive progress. They are to a large extent motivated by many application areas, such as automated postal address, data acquisition in banks, image translation, etc. The difficulties encountered in this are mainly caused by huge variations in writing styles and the overlapping and the interconnection of neighboring characters. We implemented an amalgam of machine learning (ML) and non-ML algorithms to create a model to recognize single-line handwritten sentences. Although the accuracy of our model is not enough to be used in mission-critical or fully automated systems, it could assist us by generating a fairly accurate text from a handwritten document. The method section describes the architecture and design of our model, while the experimental analysis section describes the datasets used and showcases the results of our model on test sentences. We conclude by giving some future directions to improve over the performance of our model.

2 Methods

We have exploited techniques such as *Kadane's Algorithm, K-means Clustering, Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short Term Memory (LSTM) units, Connectionist Temporal Classification (CTC), Language Model (LM)* and *Maximum Likelihood Estimation (MLE)*. The input given to the recognition system is an image of a single-line handwritten English sentence, which is processed through a series of aforementioned techniques to finally output the sentence in machine-printed format. The proposed model will be substantiated by the subsequent details.

Line Segmentation into Words

Commencement of our model entails segmenting the input sentence image into a sequence of words' images. Firstly, our image is smoothed out to remove noise and then, converted into binary image. After that, we associate a large positive weight to black pixels and a small negative weight to white pixels. Then, we project the image on horizontal and vertical axes, so that the index of horizontal projection associated with the column having all pixels as *white* will have negative value, and the index of horizontal projection associated with the columns having some pixels as *black* will have positive value. After finding maximum sum subarray of horizontal projection using Kadane's algorithm, we know where our sentence is located in picture. Using this, we remove the spaces from start and end of the sentence present in picture, and do the same for top and bottom of the sentence. Our horizontal projection has one more property that, when there are spaces in the sentence, the values of horizontal projection in those areas are very low. Using this property, we locate all the spaces present in the sentence. But these spaces could be inter-character or inter-word space. Here, we assume that gaps between words are larger than the gaps between characters. We use k-means algorithm to cluster the inter-character spaces and inter-word spaces. After getting inter-word spaces, we split the sentence at the bisector lines of these inter-word spaces. After that, we remove the spaces around words using the same method which was used to remove spaces around sentence.

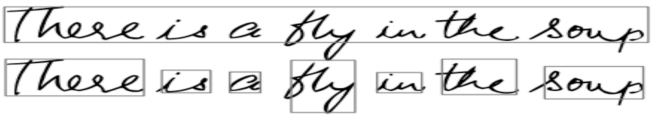

Figure 1: Line Segmentation into Words

Handwritten Word Recognition

After line segmentation step, we obtain a sequence of images of words extracted from the image of the sentence. Next step is to recognize each of the word from the image. We will build a Neural Network (NN) as shown in figure-2 which is trained on word-images from the IAM dataset. This NN consists of CNN layers, RNN layers (LSTM-RNN) and a final CTC layer.

CNN : The input image is fed into the CNN layers. These layers are trained to extract relevant features from the image. Each layer consists of three operations. First, the convolution operation, which applies a filter kernel of size 5x5 in the first two layers and 3x3 in the last three layers to the input. Then, the non-linear RELU function is applied. Finally, a pooling layer summarizes image regions and outputs a downsized version of the input. While the image height is downsized by 2 in each layer, feature maps (channels) are added, so that the output feature map (or sequence) has a size of 32x256.

RNN : The feature sequence contains 256 features per time-step, the RNN propagates relevant information through this sequence. The LSTM implementation of RNNs is used, as it is able to propagate information through longer distances and provides more robust training-characteristics than normal RNN. The RNN output sequence is mapped to a matrix of size 32x80. The IAM dataset consists of 79 different characters, further one additional character is needed for the CTC operation (CTC blank label), therefore there are 80 entries for each of the 32 time-steps.

CTC: While training the NN, the CTC is given the LSTM-RNN output matrix and the ground truth text and it computes the loss value. While inferring, the CTC is only given the matrix

and it decodes it into the final text.

The NN outputs a character-probability matrix. This matrix is either used for CTC loss calculation or for CTC decoding. Ultimately, after performing this second step of the algorithm, we will obtain the machine-printed words out of the images provided as input.

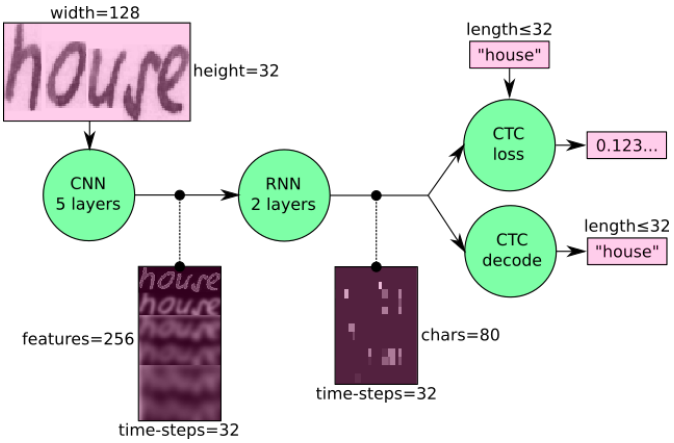

Figure 2: Convolutional-Recurrent Neural Network (CRNN) Architecture and Design

Improving Accuracy of Recognized Sentence

This section deals with improving the accuracy of the output procured from neural network using techniques such as LM and MLE. To build a LM, we collect the text from “All News” dataset, clean it of other symbols except characters and spaces, and tokenize it. From the resulting data, we create a set, each for unigrams and bigrams. We will denote a “word” as “w” from here. We calculate probabilities (P(w)) of unigrams, and construct a conditional probability table (CPT) from the bigrams. Next, a tree is generated for implementation of tree search algorithms, where each node at any level represents a word from the set of words, and the number of nodes in each level equals the cardinality of this set. The root node represents the empty word. A node at a given level is connected to each node of the next level. An edge connecting an i^{th} level node (w_i) to an $(i+1)^{th}$ level node (w_{i+1}) has a weight given by:-

Levenshtein_Distance(word appearing in $(i+1)^{th}$ level, $(i+1)^{th}$ word in current sentence) $*$ (1 - $P(w_{i+1}|w_i)$)

where $P(w_{i+1}|w_i)$ is computed from the CPT. A path from the root node to any leaf node describes a sentence, and the length of the path reveals how likely it is that the sentence made by nodes occurring in the path is actually the one given to us as input. Uniform cost search is applied to the this tree and the shortest path gives the correctly determined sentence.

3 Experimental Analysis

Datasets

The IAM Handwriting Database contains many forms of handwritten English text like words and sentences with labels. We have used this dataset for training and testing the word recognition part of our model. We also used this to test our sentence segmentation method.
We also used "All the news" dataset from kaggle to train our sentence correction part of our model. This dataset contains the publications of the New York Times, Breitbart, CNN, Business Insider, the Atlantic, Fox News, Talking Points Memo, Buzzfeed News, National Review, New York Post, the Guardian, NPR, Reuters, Vox, and the Washington Post between 2015 to 2017.

Results

The method by which we had split sentences into words has an accuracy of more than 85 percent. We calculated error by counting the number of inter-word spaces recognized wrongly. And we build our accuracy matrix as sum of error in all sentences divided by total number of words in those sentences.

Accuracy of the NN model of recognizing the word is 66%. This is calculated as number of correctly recognized words divided by the total number of words.
Currently there does not exist any standard performance metric for LM.

4 Discussion and Future Directions

For splitting the sentence’s image into words’ images, we assumed that gaps between the words are larger than the gaps between characters. We also assumed that in between words, there will be a horizontal space. However, in slant hand-writing, finding the horizontal space will not give us desired result. In that case, we can take projection of sentence on a new axis created by tilting our horizontal axis a bit. Tilting the axis by right amount, spaces would be clearly visible on projection of sentence on the new axis. If we create groups based on these spaces, the intra-group variance can be minimized. It may also be possible that inter-character spaces are almost equal to inter-word spaces; in such situations, problem would be much difficult to solve and we will need to incorporate the information of characters to find inter-word spaces.

To improve the recognition accuracy we can do several things such as data augmentation, which involves increasing the dataset size by applying further (random) transformations to the input images. We can remove cursive writing style in the input images, i.e. deslanting. More CNN layers could be added. LSTM can be replaced by 2D-LSTM. Word beam search decoding (CTC word beam search) can be incorporated to constrain the output to dictionary words. We could also use a better dataset for constructing LM. Using trigrams and ngrams while creating LM would also help in improving the results.

References

[1] "All the news" dataset at kaggle - https://www.kaggle.com/snapcrack/all-the-news

[2] IAM Handwriting Database - https://www.kaggle.com/snapcrack/all-the-news

[3] https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5