Handwritten Text Recognition System

Tushar Srivastava

Department of Computer Science and Engineering, SRM Institute of Science and Technology
Modinagar, Uttar Pradesh, India - 201204

***Abstract*— Handwritten text recognition is one of the most challenging and active research areas in the field of image processing and pattern recognition. It has numerous applications including processing bank and conversion of any hand written document into digitized text. This paper makes the use of an Artificial Neural Network with feature extraction to recognize patterns in a handwriting using the IAM data set containing articles from numerous different writers.**

I. Introduction

Handwritten Text Recognition System is a software that converts the text present in a scanned image into digitized text. There are two types of handwritten text recognition (HTR) systems Online and Offline. The online system works while the text is being written, whereas the offline system works on already written text which is more challenging than the online HTR system. Challenges in the HTR include the cursive nature of the writing, variety of each character in size and shape and large vocabularies.

A. Scope of Discussion

This paper focuses on the Offline HTR system and the classifier, it's parameters, feature extraction methods. The classifier is build on Artificial Neural Networks(ANN) which is a combination of Neural Network (CNN) and a Recurrent Neural Network (RNN).

B. Methodology

The system makes use of Artificial Neural Networks (ANN). The ANN is a combination of multiple layers of CNN trained to extract relevant features from the images, which output a 2D feature matrix which is fed to the following RNN. The RNN uses the Long-Short-Term implementation to propagate the information through the sequence. Then the output of the RNN is mapped into a matrix which contains a score for each character per sequence element. The following CTC (Temporal Classification) operation decode the score matrix and gives the final output.

II. Objectives

The main objectives of this system are:

1. Recognize text present in various documents which include characters, words, and digits.

2. Help banks to be able to recognize the text present in documents such as cheques.

3. To be able to recognize different cursive styles of handwriting.

4. To help in the industries of healthcare and pharmaceutical in the digitization of patient prescriptions.

5. Be able to recognize the text present in old documents in bad conditions.

III. Model Architecture

A. Data set

The model is trained using the IAM data set which is a collection of handwritten passages from several writers. This data set contains images of each handwritten sentence with a dash separated file format.

B. Preprocessing

Includes to ensure that the text is set upright and modifications to ensure uniform size of the texts mainly to the size 128 x 32.

C. Classification

We use an Artificial Neural Network. A 7 layered CNN helps in feature extraction, then a 2 layered RNN of Long-Short-Term-Memory type is integrated to propagate information through the feature sequence and output a probability matrix.

D. Postprocessing

CTC layer decodes the probability matrix from the RNN and scores to the corresponding letters and the CTC loss function is used to calculate the accuracy of the model.

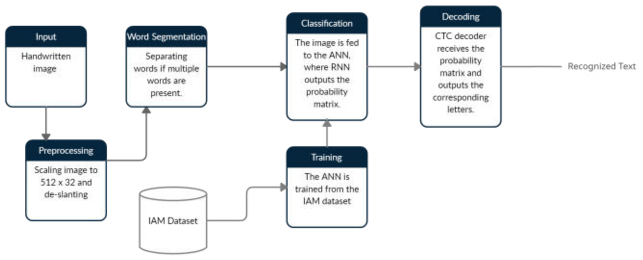

Input
Handwritten image
Word Segmentation
Separating words if multiple words are present.
Classification
The image is fed to the ANN, where RNN outputs the probability matrix.
Decoding
CTC decoder receives the probability matrix and outputs the corresponding letters.
Recognized Text
Preprocessing
Scaling image to 512 x 32 and de-slanting
IAM Dataset
Training
The ANN is trained from the IAM dataset

Fig 1. Architectural Design of the system.

IV. ARTIFICIAL NEURAL NETWORKS

This section introduces with a focus on topics necessary to understand the working and application of the HTR.

A. Definition

ANN's basic structure is composed of neurons. These neurons are loosely based on the biological neurons that propagate signals in the nervous system in out body.

B. Perceptron

A perceptron is a term used for supervised learning of binary classifiers. They decide whether an input, usually represented by a series of vectors, belongs to a specific class. In short, a is a single-layer neural network as shown in Figure 2.

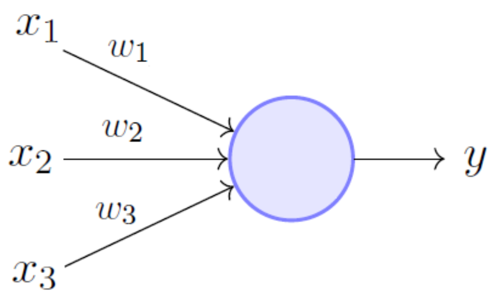


Figure 2. A Perceptron

C. Multi-Layer Perceptron

An idea to improve perceptrons was to combine multiple neurons into a neural network as shown in Figure 3.

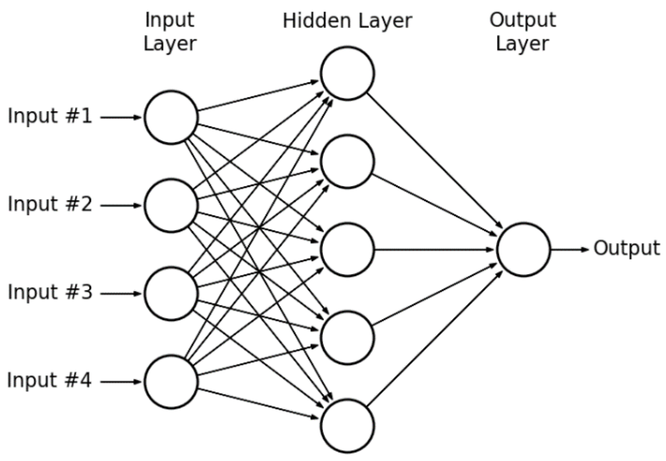


Figure 3. A Neural Network

D. Activation Function

Each input into a neuron is assigned a weight *w* which is assigned on the basis of it's relevant importance to other inputs. The node applies the below defined function *f* to it's weighted sum of inputs as shown in Figure 4. This is called as **Activation Function** of the proceeding neuron. Every activation function takes a single number and performs a certain fixed mathematical operation on it. I have used a ReLU (Rectified Linear Unit) activation function in this paper.

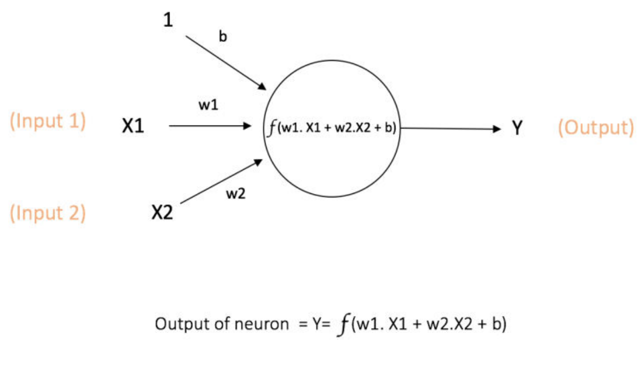
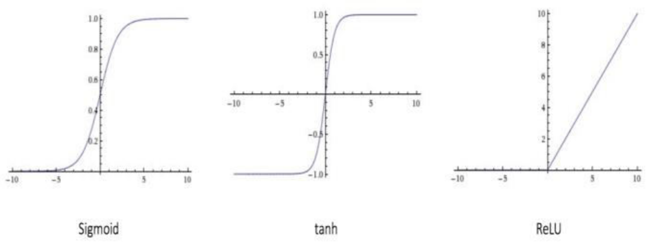


Figure 4. Activation Function

E. ReLU Function

ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero). The following are the different types of activation functions in Figure 5.



F. Layers

1. Input layer - The Input layer has four nodes (refer to figure 3). The Bias node has a value of 1. The other three nodes take X1, X2, and X3 as external inputs (which are numerical values depending upon the input dataset). As discussed above, no computation is performed in the Input layer, so the outputs from nodes in the Input layer are 1, X1, X2, and X3 respectively, which are fed into the Hidden Layer.

2. Hidden Layer - The Hidden layer has five nodes with the Bias node having an output of 1. The output of the other four nodes in the Hidden layer depends on the outputs from the Input layer (1, X1, X2, X3) as well as the weights associated with the connections (edges).Similarly, the output from other hidden node can be calculated. Remember that f refers to the activation function. These outputs are then fed to the nodes in the Output layer.

V. Convolutional Neural Networks

The traditional image classification is to extract features from the image which then serve as an input to the classifier. Convolution is an operation on two functions and as such outputs a third function.

Every layer of the CNN consists of three operations. First, the convolution operation. A filter kernel of size 5×5 is applied in the first three layers and a filter kernel of size 3×3 in the last four layers to the input. Then, the ReLU function is applied. Finally, a pooling layer is used to summarize the image regions which outputs a downsized version of the input. The image height is downsized by 2 in each layer, and to ensure the output feature map has a size of 32 x 256, feature maps (channels) are added

A. CNN Input.

The input is a gray-scale image of size 128×32. Mostly, the images from the IAM dataset do not have exactly this size, therefore we resize it (without distortion) until it either has a width of 128 or a height of 32. We copy the image into a (white) target image of size 128×32. This process is shown in Fig. 6. To wrap things up we normalize the grayvalues of the image which simplifies the task for the NN. Data augmentation can easily be integrated by copying the image to random positions instead of aligning it to the left or by randomly resizing the image.

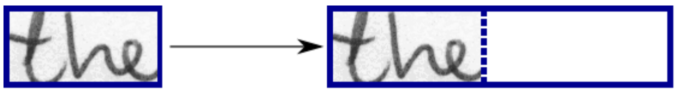


Figure 6.Left: A sample from the data set. Right: The resized version.

B. CNN Output

The output of the CNN layers is a sequence of length 32 as shown in the Figure 7. Each and every entry contains 256 features. All of the features are further processed by the RNN but still some features already show a high correlation with certain highlevel properties of the input image: there are features which have a high correlation with characters such as *e*, or with duplicate characters *tt*, or with character-properties such as loops (as contained in handwritten*l* and *e*).

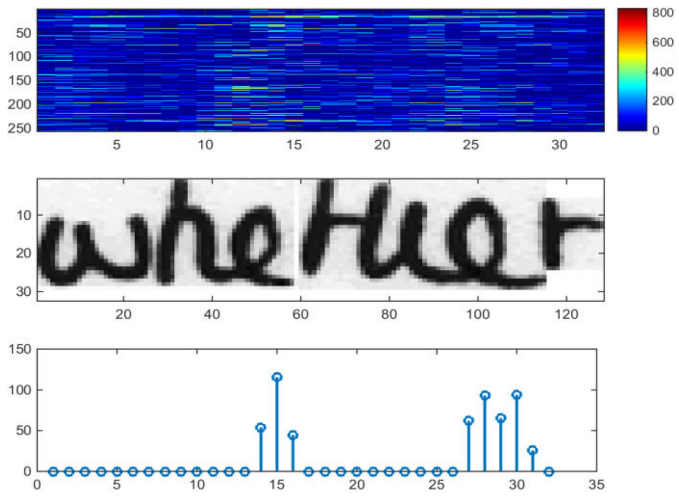


Fig. 7: Top: 256 feature per time-step are computed by the CNN layers. Middle: input image. Bottom: plot of the 32nd feature, which has a high correlation with the occurrence of the character "e" in the image.

VI. Recurrent Neural Network

As compared to vanilla Neural Networks, RNNs are designed to take a series of inputs with no limit on it's size. RNNs are mostly used for text summarization therefore have been used in this model.

A. RNN Input

The RNN receives the input from the CNN and the feature sequence contains 256 features per time-step, the RNN propagates relevant information through this sequence using the LSTM (Long Short-Term Memory) implementation, because it is able to propagate information through long distances and provide more precise trainingcharacteristics than the normal RNN. The RNN's outputis mapped to a probability matrix of size 32×80. There are 79 different characters in the IAM dataset, further one additional character is needed for the CTC operation (CTC blank label), this is why there are 80 entries for each of the 32 time-steps.

B. RNN Output

A visualization of the RNN output matrix is show in Figure 8. for an image containing the text "little". The matrix contained in the top graph shows the scores for the characters including the CTC blank label as its last (80th) entry.

The rest matrixentries, from top to bottom, correspond to the following characters: " !"#&'()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz".

It is noticed that most of the time, the characters appear exactly at the position they are predicted to be appearing. Only the last character *e* is not aligned. But this is fine, as the next CTC operation is segmentation-free and doesn't care about absolute positions. The text in the bottom-most graph showing the scores for the characters *l, i, t, e* and the CTC blank labelcan be decoded easily: so we just take the most probable character from each time-step, this forms the so called best path, then we throw away repeated characters and finally all blanks: *l---ii--t-t--l-…-e* → *l---i--t-t--l-…-e* → *little*.

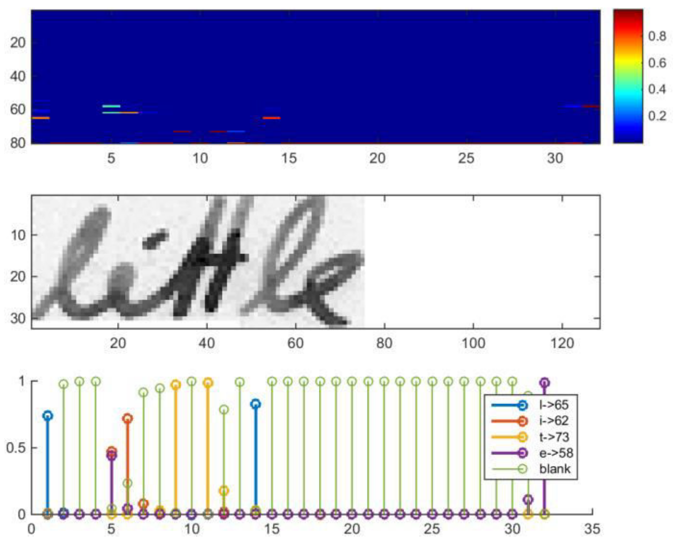


Fig. 8: Top: output matrix of the RNN layers. Middle: input image. Bottom: Probabilities for the characters *l, i, t, e* and the CTC blank label.

VII. CONNECTIONIST TEMPORAL CLASSIFICATION

CTC (Connectionist temporal classification) is a type of scoring function for neural networks, most commonly used for training RNNs such as the Long-Short-Term-Memory implementation used here. They also tackle problems where the timing is variable.

A. CTC Input

The RNN outputs a sequence of length T with C + 1 number of character probabilities per element in the sequence, where C is the number of characters. The RNN also adds an additional character called *blank* which is used for character loss calculation per epoch. The CTC receives the RNN output matrix as the input in the neural network and ground truth text which is used to compute the loss value and therefore the accuracy. While deducing, the CTC is only given the probability matrix for the characters present in the image and it decodes it into the final text. Both therecognized text and the ground truth text can be at most 32 characters long.

B. Limitations of CTC

The vanilla CTC simply outputs the characters it sees int the image without regard for errors due to similar looking characters such as *a* and *o* . The figure 9. shows one such example of the problem where the NN recognizes *a* as *oi* . Therefore to tackle such situations there are different decoding algorithms such as *Best path decoding* which only uses the output of the NN as it is and just computes the most likely character in the sequence, *Beam search with character-LM* which additionally scores character sequences for improving the results even further, *Token Passing*which uses a dictionary and word-LM. The most probable sequence of dictionary words is searched for in the NN output, but arbitrary character sequences cannot be handled(numbers, punctuation marks) like ": 1234.".None of the above mentioned algorithms get the work done correctly but we observe the good properties of *Beam Search* and *Token Passing*, when we see a word we only allow words from a dictionary. Therefore I have combined the algorithms to propose a ***Word Beam Search*** algorithm.

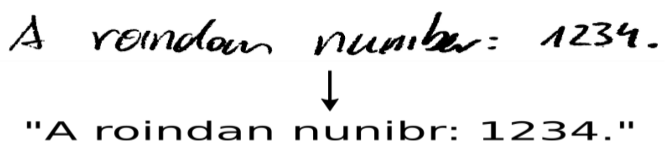


Figure 9. The output of a vanilla CTC mistaking *a* as *oi*.

C. Word Beam Search

We will use the Beam Search as a starting point. This algorithm will iterate through the ANN output and create text candidates (called beams) which are scored. In figure 10.an illustration of the evolution of beams is shown: we start with the empty beam, add all possible characters (we only have 'a' and 'b' in this example) to it in the first iteration and we only keep the best scoring characters. The beam width controls the number of surviving beams. The algorithm is repeated until the complete NN output is processed.

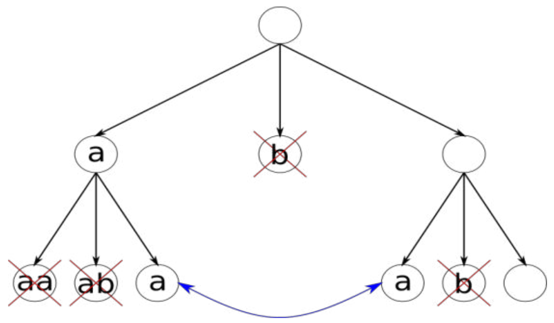


Fig10. Beams are created iteratively, equal beams merged and 2 beams kept.

For the algorithm to behave differently when it recognizes a word and when it recognizes a number or punctuation marks, we add a state variable to each beam. A beam is either in wordstate or in nonwordstate. If the beamtext currently is "Hel", then we are in wordstate and only allow adding wordcharacters until we have a complete word like "Hell" or "Hello". Going from wordstate to nonwordstate is allowed when a word is completed: then we can add a nonwordcharacter to it, like if we have "Hello" and add " " to it we get "Hello ". In word state we only allow adding characters that will eventually form words. A prefix tree is shown in Figure 11.

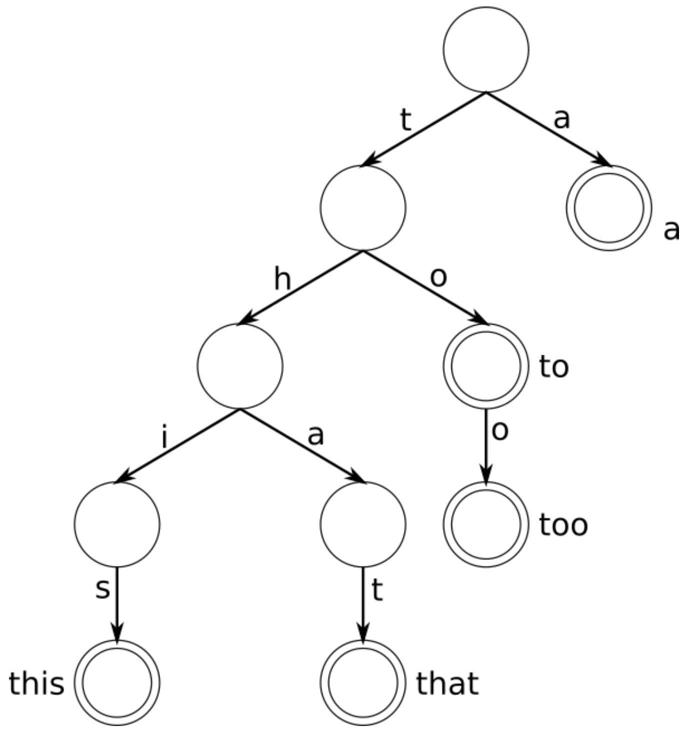


Figure 11. A prefix tree containing the words *a, to, too, this,* and *that.*

D. Word Beam Search Comparison

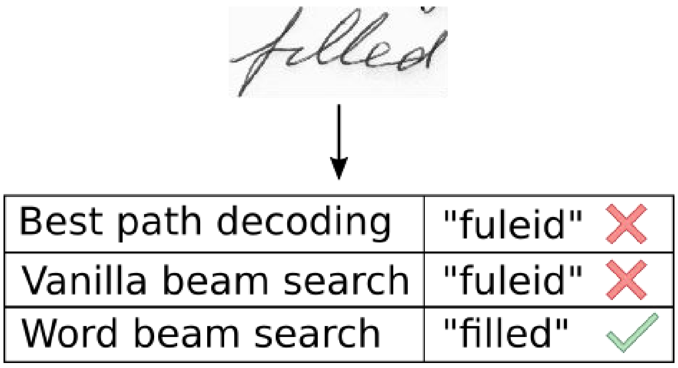
In our algorithm we keep only the best-scoring beams per iteration. The final score depends on the ANN output. Therefore, now we have an algorithm which is able to recognize the correct text from Figure 9. - "A random number: 1234". The Figure 12. shows the different results obtained from the different CTC algorithms.

	Perfect LM	Rudimentary LM
Best Path	5.60 / 17.06	5.60 / 17.06
Token Passing	8.16 / 9.24	(not feasible)
VBS	5.35 / 16.02	5.55 / 16.39
WBS W	4.22 / 7.90	**5.47** / 14.09
WBS N	4.07 / **7.08**	6.15 / **13.90**

Figure 12. Given as CER(Character Error Rate) /WER(Word Error Rate), VBS - Vanilla Beam Search, WBS W - Word Beam Search without LM, WBS N - Word Beam Search with LM.

VIII. CONCLUSIONS

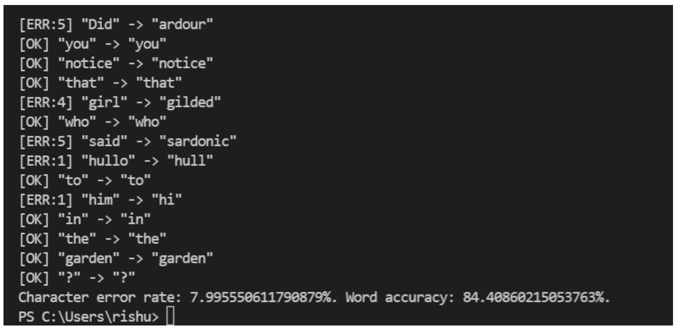
The proposed model performs with an accuracy of ~84 - 85% by using the Word Beam Search algorithm as compared to ~74% using the vanilla CTC decoder. The decoder output is shown in the Figure 13.



Best path decoding	"fuleid" ✗
Vanilla beam search	"fuleid" ✗
Word beam search	"filled" ✓

Figure 13. The Best Path and VBS give predictions whereas WBS doesn't.

The accuracy is as shown below in the Figure 14.



```
[ERR:5] "Did" -> "ardour"
[OK] "you" -> "you"
[OK] "notice" -> "notice"
[OK] "that" -> "that"
[ERR:4] "girl" -> "gilded"
[OK] "who" -> "who"
[ERR:5] "said" -> "sardonic"
[ERR:1] "hullo" -> "hull"
[OK] "to" -> "to"
[ERR:1] "him" -> "hi"
[OK] "in" -> "in"
[OK] "the" -> "the"
[OK] "garden" -> "garden"
[OK] "?" -> "?"
Character error rate: 7.995550611790879%. Word accuracy: 84.40860215053763%.
PS C:\Users\rishu>
```

Figure 14. The Word Beam Search providing word accuracy of ~85%.

IX. FUTURE WORKS

For making the tasks easier for the classifier further preprocessing methods such as deslanting for cursive writing, which results in text lying in approximately horizontal direction will be tested. Further experiments will also be conducted to identify intermediate outputs which can be ignored or downsampled without affecting the accuracy.

REFERENCES

[1] Harald Scheidl, Stefan Fiel, Robert Sablatnig, Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm, Conference: 2018 16th International Conference on Frontiers in Handwriting Recognition(ICFHR)

[2] Harald Scheidl, Robert Sablatnig, Handwritten Text Recognition in Historial Documents, 2018 Technical University Vienna.

[3] Dr. Yassine Ghouzam, Introduction to CNN keras, 2017, Paris Diderot University.

[4] A. Graves, N. Jaitly, and A. Mohamed, "Hybrid speech recognition with deepbidirectional LSTM," in Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on. IEEE, 2013, pp. 273–278.

[5] Hasim Sak, Andrew Senior, Francoise Beaufays, Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling, Google, USA.

[6] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in Advances in Neural Information Processing Systems, 2013, pp. 190–198.

[7] Sandip A Kale, M. Uday, Research study on applications of artificial neural networks and e-learning personalization, 2017 International Journal of Civil Engineering and Technology.

[8] Martin Abadi andPaul Barham, Tensorflow: A system for large-scale machine learning, 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX Association (2016), pp. 265-283

[9] U-V Marti and H. Bunke, The IAM-database: an English sentence database for offline handwriting recognition, International Journal on Document Analysis and Recognition (39 - 46) 2002.