International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 05 | May 2020 www.irjet.net p-ISSN: 2395-0072

Handwritten Text Classification using Deep Learning

Vidushi Garg

Student, Department of Information Technology, Maharaja Agrasen Institute of Technology, New Delhi, India

---

Abstract - *In this paper, an innovative method is presented for offline handwritten character detection using deep neural networks. In today's world, it has become easier to train deep neural networks because of availability of huge amount of data and various Algorithmic innovations which are taking place. Handwritten Text Recognition (HTR) is an automatic way to transcribe documents by a computer. There are two main approaches for HTR, namely hidden Markov models and Artificial Neural Networks (ANNs). The proposed HTR system is based on ANNs. Preprocessing methods enhance the input images and therefore simplify the problem for the classifier. These methods include contrast normalization as well as data augmentation to increase the size of the dataset. The classifier has Convolutional Neural Network (CNN) layers to extract features from the input image and Recurrent Neural Network (RNN) layers to propagate information through the image. The RNN outputs a matrix which contains a probability distribution over the characters at each image position. Decoding this matrix by the connectionist temporal classification operation yields the final text.*

Key Words: **Convolutional and Recurrent Neural Network (CRNN), Long short-term memory (LSTM), Connectionist temporal classification (CTC), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Artificial Neural Network (ANN), Handwritten Text Recognition model (HTR)**

1. INTRODUCTION

Handwritten Text Recognition (HTR) is the task of transcribing handwritten text into digital text. It is a technology that is much needed in this world as of today. Before proper implementation of this technology we have relied on writing texts with our own hands which can result in errors. It's difficult to store and access physical data with efficiency. Manual labor is required in order to maintain proper organization of the data. Throughout history, there has been severe loss of data because of the traditional method of storing data. Modern day technology is letting people store the data over machines, where the storage, organization and accessing of data is relatively easier. [2] Adopting the use of Handwritten Text Recognition software, it's easier to store and access data that was traditionally stored. Furthermore, it provides more security to the data.

Handwriting recognition is a challenging task because of many reasons. The primary reason is that different people have different styles of writing. The secondary reason is there are lot of characters like Capital letters, Small letters, Digits and Special symbols. Thus a large dataset is required to train a near-accurate neural network model. Handwritten Text Recognition (HTR) is divided into online and offline recognition. Online recognition is performed while the text to be recognized is written (e.g. by a pressure sensitivity device), therefore geometric and temporal information is available. Offline recognition, on the other hand, is performed after the text has been written. The text is captured (e.g. by a scanner) and the resulting images are processed.

To build the Handwritten Text Recognition model (HTR), the neural network is trained on word-images from the IAM dataset. The proposed system makes use of Artificial Neural Networks (ANNs). Multiple Convolutional Neural Network (CNN) layers are trained to extract relevant features from the input image. These layers output a 1D or 2D feature map (or sequence) which is handed over to the Recurrent Neural Network (RNN) layers. The RNN propagates information through the sequence. Afterwards, the output of the RNN is mapped onto a matrix which contains a score for each character per sequence element. As the ANN is trained using a specific coding scheme, a decoding algorithm must be applied to the RNN output to get the final text. Training and decoding from this matrix is done by the Connectionist Temporal Classification (CTC) operation.

2. RESEARCH METHODOLGY

2.1. Description of the dataset

To build the Handwritten Text Recognition model (HTR), the neural network is trained on word-images from the IAM dataset. The IAM Handwriting Database contains forms of handwritten English text which can be used to train and test handwritten text recognizers and to perform writer identification and verification experiments. The database contains forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels.

The IAM Handwriting Database 3.0 is structured as follows:

- 657 writers contributed samples of their handwriting
- 1'539 pages of scanned text
- 5'685 isolated and labeled sentences
- 13'353 isolated and labeled text lines
- 115'320 isolated and labeled words

The words have been extracted from pages of scanned text using an automatic segmentation scheme and were verified manually.

Fig 1: Dataset

2.2. Data Preprocessing

The input is a gray-value image of size 128×32. Usually, the images from the dataset do not have exactly this size, therefore resize it (without distortion) until it either has a width of 128 or a height of 32. Then, copy the image into a (white) target image of size 128×32. This process is shown in Figure 2. Finally, normalize the gray-values of the image which simplifies the task for the NN. Normalization is done to change the range of pixel intensity values.

Data augmentation is a strategy to significantly increase the diversity of data available for training models, without actually collecting new data. Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks. 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.

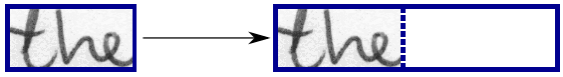


Fig 2: Left: an image from the dataset with an arbitrary size. It is called to fit the target image of size 128 x 32, the empty part of the target image is filled with white color.

The model not only learns how to read text, but it also learns how the samples look like. There are some noticeable patterns in the IAM dataset as shown in Figure 3:

- Images have high contrast
- Words are tightly cropped
- Bold writing style

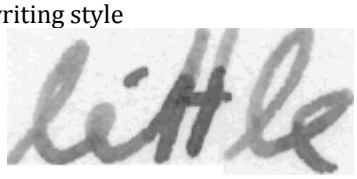


Fig 3: A sample from the IAM dataset

If an image with a very different style is feed to the model, the model might predict a bad result. The reason is that the model has never seen images like this:

- Low-contrast
- Much space around the word
- Lines very thin

Hence, the input image is pre-processed such that it looks like a sample from IAM (see Figure 4). First, crop the image: The model still recognizes ".". Then, increase the contrast: Now, the model gives a much better result: "tello". This is almost correct. If the lines are thickened by applying a morphological operation, the model is finally able to recognize the correct text: "Hello".

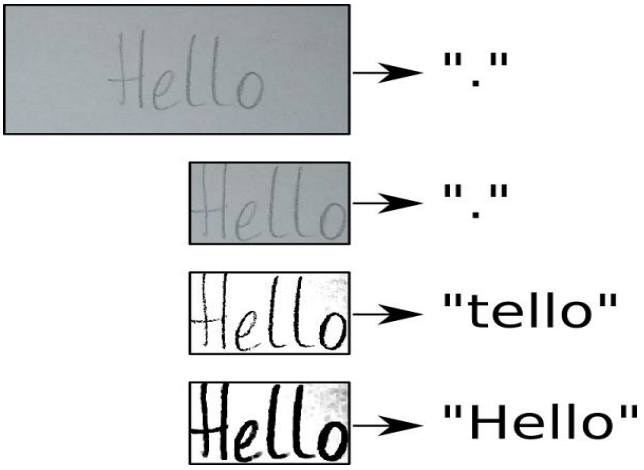


Fig 4: Pre-processing steps and the recognized text for each of them

2.3. Implementation

2.3.1. Methodology

The end-to-end trainable Artificial Neural Network (ANN) contains 5 convolutional layers and 2 layers of bidirectional LSTM units with 256 hidden cells. The input images only contain single words. The input to the ANN is a gray-value image containing text. A stack of convolutional layers maps the input image onto feature maps. The output of the final layer of the CNN can be regarded as a sequence of length *T* with *F* features. Information is then propagated along this sequence with a stack of RNNs. [3] The RNNs map the sequence to another sequence of same length *T*, assigning probabilities to each of the *C* different classes. Finding the most probable labeling in this C x T sized matrix is called decoding and is done with a CTC output layer. While training, the CTC loss is used to calculate a loss value for a training batch which is then backpropagated to the output layer of the RNN. The usage of the CTC operation is twofold: it serves as a loss function and as a decoding function.

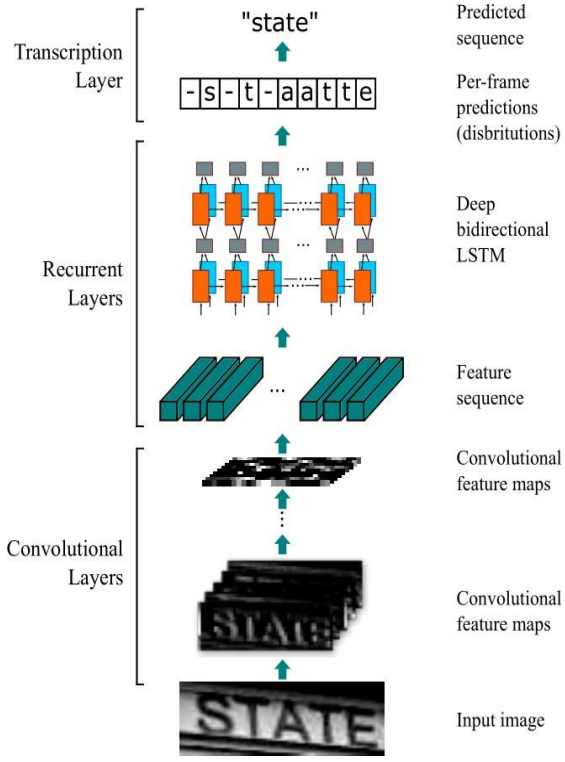


Fig 5: The network architecture called Convolutional and Recurrent Neural Network (CRNN). It uses CNN layers, followed by LSTM layers and a final CTC layer.

2.3.2. Convolutional Neural Network

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that *convolve* with a multiplication or other dot product. The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution. The final convolution, in turn, often involves backpropagation in order to more accurately weight the end product.

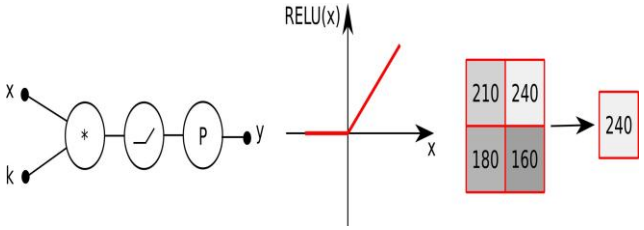


Fig 6: Left: Network diagram of a convolutional layer. An image patch *x* is convolved by a kernel *k*. As a non-linearity the RELU function is applied. Finally, pooling generates a summary statistic for small regions. Middle: plot of the RELU function. Right: max-pooling of a 2 × 2 region.

2.3.3. Long Short-Term Memory (LSTM) Neural Network

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) used in the field of deep learning. Unlike standard feed-forward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three *gates* regulate the flow of information into and out of the cell. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the exploding and vanishing gradient problems that can be encountered when training traditional RNNs.

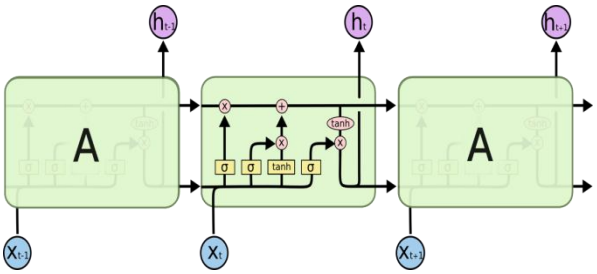


Fig 7: Architecture of Long Short-Term Memory (LSTM) Neural Network

2.3.4. Connectionist Temporal Classification (CTC) Decoder

Connectionist temporal classification (CTC) is a type of neural network to produce output using associated scoring function, for training recurrent neural networks (RNNs) such as LSTM networks to tackle sequence problems where the timing is variable. It can be used for tasks like on-line handwriting recognition or recognizing phonemes in speech audio.

The NN for such use-cases usually consists of convolutional layers (CNN) to extract a sequence of features and recurrent layers (RNN) to propagate information through this sequence. It outputs character-scores for each sequence-element, which simply is represented by a matrix. Now, there are two things we want to do with this matrix:

1. train: calculate the loss value to train the NN

2. infer: decode the matrix to get the text contained in the input image

Both tasks are achieved by the CTC operation. The NN-training will be guided by the CTC loss function. We only feed the output matrix of the NN and the corresponding ground-truth (GT) text to the CTC loss function. It tries all possible alignments of the GT text in the image and takes the sum of all scores. This way, the score of a GT text is high if the sum over the alignment-scores has a high value. There was the issue of how to encode duplicate characters. It is solved by introducing a pseudo-character (called blank, not a white-space character).

There are the following different decoding algorithms available, some also include a language model (LM):

a. **Best path decoding:** Best path decoding only uses the output of the NN (i.e. no language model) and computes an approximation by taking the most likely character at each position. It calculates the best path by taking the most likely character per time-step. It undoes the encoding by first removing duplicate characters and then removing all blanks from the path what remains represents the recognized text.

b. **Beam search:** It also only uses the NN output, but it uses more information from it and therefore produces a more accurate result. Beam search decoding iteratively creates text candidates (beams) and scores them. At each time-step, only the best scoring beams from the previous time-step are kept.

The RNN output matrix of the given figure 8 contains 2 time-steps (t0 and t1) and 3 labels (a, b and - representing the CTC-blank). Best path decoding (see left figure) takes the most probable label per time-step which gives the path "--" and therefore the recognized text "" with probability 0.6*0.6=0.36. Beam search calculates the probability of labelings. For the labeling "a" the algorithm sum over the paths "-a", "a-" and "aa" (see right figure) with probability 0.6*0.4+0.4*0.6+0.4*0.4=0.64. The only path which gives "" still has probability 0.36, therefore "a" is the result returned by beam search.

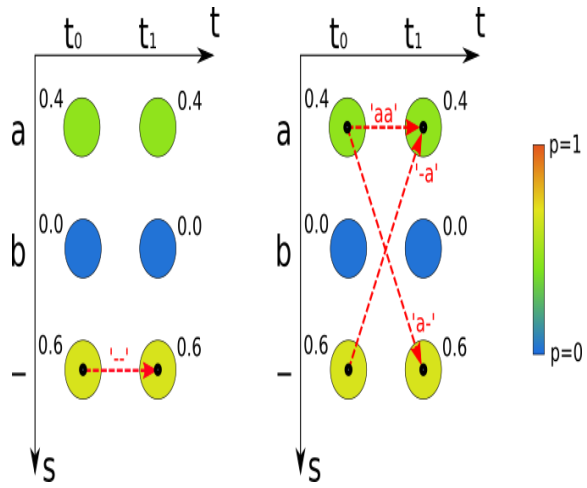


Fig 8: Left: Concatenate most probable characters per time-step to get best path, Right: All paths corresponding to text "a"

2.4. Evaluation Metric

The probability of the recognized text is calculated by using the CTC loss function. The loss function takes the character-probability matrix and the text as input and outputs the loss value L.[3] The loss value L is the negative log-likelihood of seeing the given text, i.e. L = -log(P). If we feed the character-probability matrix and the recognized text to the loss function and afterwards undo the log and the minus, we get the probability P of the recognized text: P = exp(-L).

2.5. Results and Declaration

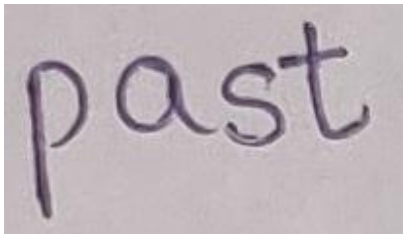
- python main.py –train

```
INFO:tensorflow:Restoring parameters from ../
model/snapshot-2
Epoch: 1
Train NN
Batch: 1 / 431 Loss: 1.2533284
Batch: 2 / 431 Loss: 22.58075
Batch: 3 / 431 Loss: 30.367832
Batch: 4 / 431 Loss: 87.433395
Batch: 5 / 431 Loss: 62.855892
Batch: 6 / 431 Loss: 164.98897
Batch: 7 / 431 Loss: 63.118935
Batch: 8 / 431 Loss: 23.982952
Batch: 9 / 431 Loss: 25.498964
Batch: 10 / 431 Loss: 27.86906
Batch: 11 / 431 Loss: 15.101894
Batch: 12 / 431 Loss: 15.714156
```

- python main.py –validate

```
INFO:tensorflow:Restoring parameters from ../
model/snapshot-2
Validate NN
Batch: 1 / 22
Ground truth -> Recognized
[OK] "," -> ","
[OK] "a" -> "a"
[OK] "demand" -> "demand"
[OK] "that" -> "that"
[OK] "the" -> "the"
[OK] "movement" -> "movement"
[ERR:3] "unite" -> "mike"
[OK] "with" -> "with"
[OK] "the" -> "the"
[ERR:1] "Gaitskellites" -> "Gaiskellites"
[OK] "on" -> "on"
[OK] "their" -> "their"
[OK] "policy" -> "policy"
[OK] "and" -> "and"
[OK] "no" -> "no"
[OK] "other" -> "other"
[OK] "seems" -> "seems"
[OK] "to" -> "to"
[ERR:1] "have" -> "hove"
[OK] "done" -> "done"
```

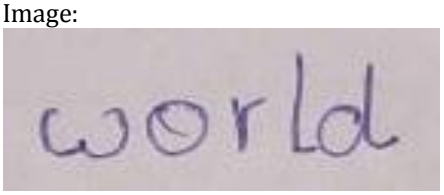
- python main.py
  - Input Image:



Output Image:

```
INFO:tensorflow:Restoring parameters from ../
model/snapshot-1
Recognized: "past"
Probability: 0.9765697
```

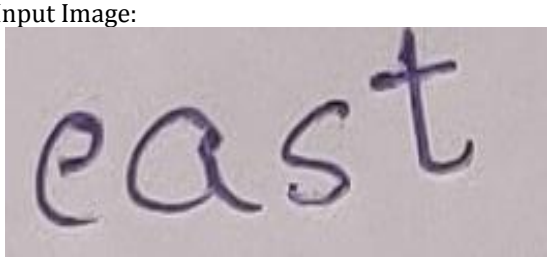
- Input Image:



Output Image:

```
INFO:tensorflow:Restoring parameters from ../
model/snapshot-1
Recognized: "world"
Probability: 0.77936614
```

- Input Image:



Output Image:

```
INFO:tensorflow:Restoring parameters from ../
model/snapshot-1
Recognized: "east"
Probability: 0.99353516
```

3. CONCLUSION

An efficient handwritten character recognition approach is proposed in this work. The proposed work employed a deep network model to recognize the handwritten texts. Preprocessing of dataset is also considered as a major factor behind the accuracy of the models. Training of network requires a supportive hardware to implement large dataset efficiently. It is necessary to train the network with large amount of dataset so that the network can easily recognize the character. Hence there is a requirement of high memory and high processing speed to achieve efficient network.

In this project, the model consists of 5 layers of Convolutional Neural Network (CNN), 2 layers of Recurrent Neural Network (RNN) and outputs a character-probability

matrix. This matrix is either used for CTC loss calculation or for CTC decoding.

The accuracy can be further improved by applying additional preprocessing techniques such as deslanting, word segmentation, removing background noises etc. [3] Word beam search decoder can be used to decode the text from the RNN output matrix instead of best beam search or vanilla beam search decoder. 7 layers of Convolutional Neural Network can be used instead of 5 layers. Finally, the LSTM layers can be replaced by a MDLSTM layer to also propagate information along the vertical image axis.

REFERENCES

[1] S. Mori, C. Y. Suen, and K. Yammamoto, "historical review of OCR research and development," Proc. IEEE, vol. 80, no. 7, pp. 1029-1058, 1992.

[2] Shubham Sanjay Mor, Shivam Solanki, Saransh Gupta, Sayam Dhingra, Monika Jain, Rahul Saxena "HANDWRITTEN TEXT RECOGNITION: with Deep Learning and Android"

[3] Diploma Thesis by Harald Scheidl "Handwritten Text Recognition in Historical Documents"

[4] Batuhan Balci, Dan Saadati, Dan Shiferaw "Handwritten Text Recognition using Deep Learning"

[5] T. Tieleman and G. Hinton, "Lecture 6.5–rmsprop: Divide the gradient by a running average of its recent magnitude", COURSERA: Neural Networks for Machine Learning, 2012.

[6] Bishwajit Purkaystha, Tapos Datta, Md Saiful Islam, "Bengali Handwritten Character Recognition Using Deep Convolutional Neural Network", 2017 20th International Conference of Computer and Information technology(ICCIT), 22-24 December, 2017

[7] Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.