Calligraphy Word Identification Using Recurrent Neural Networks

A. Helana
Assistant professor, Department of CSE, JNN Arts and Science Women's College

Abstract - **Calligraphy Word identification is a process of pattern recognition which defines ability of a system to identify word. There are many applications of Calligraphy Word identification (CWI) system such as reading postal addresses, bank check amounts, mail sorting and many more. CWI systems transfer human written text into digital text. Plenty of research done in the field of recognizing calligraphy words but lacking in best accuracy is a challenge. In this proposed technique, offline CWI is done using Recurrent Neural networks (RNN) and Tensor flow is proposed. RNNs have connections that form directed cycles, which allow the outputs from the LSTM to be fed as inputs to the current phase. The output from the LSTM becomes an input to the current phase and can memorize previous inputs due to its internal memory. RNNs are commonly used for image captioning, time-series analysis, natural-language processing, handwriting recognition, and machine translation. we were able to recognize the calligraphy word with highest accuracy. The experiment is performed on proposed technique with accuracy of 80.5% compared to the state-of-the-art.**

Index Terms— **Calligraphy Word identification, LSTM dataset, Recurrent Neural networks, Tensor flow.**

1. INTRODUCTION

Calligraphy Word Identification identifies the characters or words written by a user handwriting and converts into a format that the system understands. Most of the Calligraphy recognizers are now using Recurrent Neural Networks and Tensor flow. RNN performs a initial analyzing and categorizing the hand written input. Recurrent Neural network must be trained repetitively by providing samples of handwritten words as input and given feedback as to whether the Recurrent Neural network guesses correctly or not at the interpretation of handwritten input. Recurrent Neural networks consists of 3 layers namely Input layers, Hidden layers and Output layer. Tensor flow is a open-source software library. It is used for applications in ML such as Recurrent Neural networks. It is symbolic math library which is used for both research and production. CWI is divided into two types, as offline and online. In online CWI method the 2D coordinates of successive points are represented as a function of time, whereas in offline CWI method the writing is usually captured optically by a scanner and is stored as an image. Offline CWI method is more familiar than online CWI because of the temporal information available within the former. Many applications like bank processing, document reading, and postal address recognition require offline CWI system.

In this paper, an offline CWI system using Recurrent neural networks and tensor flow is proposed aiming to provide best accuracy.

2. CALLIGRAPHY WORD IDENTIFICATION

Calligraphy word identification is the process to identify the word accuracy with the help of LSTM dataset. LSTM is a rule of thumb in statistics that the minimum number of data samples is 30 times of the number of your parameters (weights and biases) of your model. As an example, if you have 100 neurons and each has 100 weights, then you need 300000 data samples. In case of time series, then you need 300000 time points.

LSTMs are a type of Recurrent Neural Network (RNN) that can learn and memorize long-term dependencies. Recalling past information for long periods is the default behavior. LSTMs retain information over time. They are useful in time-series prediction because they remember previous inputs. LSTMs have a chain-like structure where four interacting layers communicate in a unique way. It can store the data long – term dependencies.

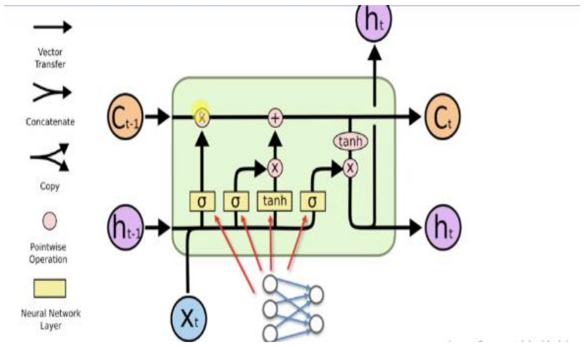


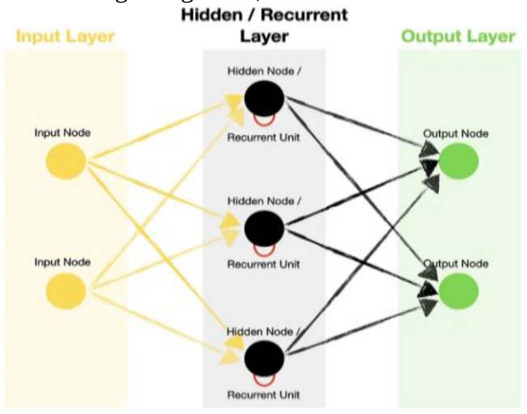
Figure1: Overview of Long Short-Term Memory Networks dataset

LSTM RNNs [10] are comprised of memory blocks usually containing one memory cell each of them, a forget gate f , an input gate i , and an output gate o . For a time step t :

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ \widetilde{C_t} &= \tanh(W_C x_t + U_C h_{t-1} + b_C) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \widetilde{C_t} \\ h_t &= o_t \odot \tanh(C_t) \end{aligned}$$

3. RECURRENT NEURAL NETWORKS (RNNS)

RNNs have connections that form directed cycles, which allow the outputs from the LSTM to be fed as inputs to the current phase. The output from the LSTM becomes an input to the current phase and can memorize previous inputs due to its internal memory. RNNs are commonly used for image captioning, time-series analysis, natural-language processing, handwriting recognition, and machine translation.



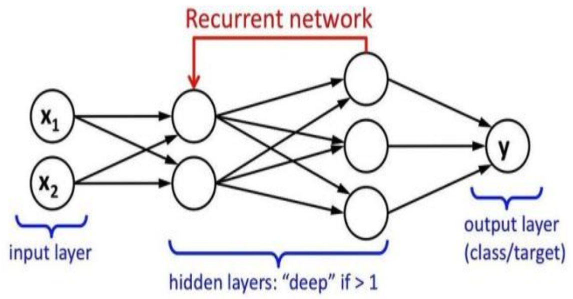


Figure2: Overview of Recurrent Neural network layers in predicting the words.

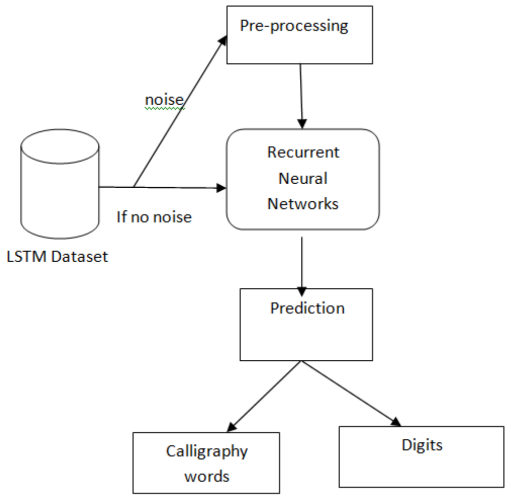
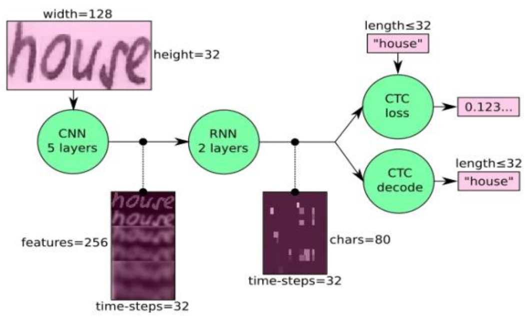


Figure2: System Architecture

The above figure shows the system architecture of our proposed model. In which firstly, dataset will be provided to the model. If the dataset contain noise preprocessing techniques will be applied, otherwise directly given to the model so that it can predict the words or Digits.

4. MODEL OVERVIEW

We use a NN for our task. It consists of convolutional NN (CNN) layers, recurrent NN (RNN) layers and a final Connectionist Temporal Classification (CTC) layer.



We can also view the RNN in a more formal way as a function (see Eq. 1) which maps an image (or matrix)

M of size W×H to a character sequence (c1, c2, …) with a length between 0 and L. As you can see, the text is recognized on character-level, therefore words or texts not contained in the training data can be recognized too .

RNN : M $\rightarrow$ (C1, C2, ….., Cn)

Data

Input: it is a gray-value image of size 128×32. Usually, the images from the 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. Then, we copy the image into a (white) target image of size 128×32. This process is shown in Fig. 3. Finally, we normalize the gray-values 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.

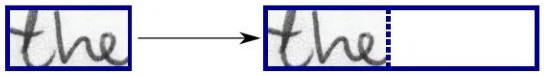


Fig. 3: Left: an image from the dataset with an arbitrary size. It is scaled to fit the target image of size 128×32, the empty part of the target image is filled with white color.

5. RNN OUTPUT

Fig. 4 shows a visualization of the RNN output matrix for an image containing the text “little”. The matrix shown in the top-most graph contains the scores for the characters including the CTC blank label as its last (80th) entry. The other matrix-entries, from top to bottom, correspond to the following characters:“!”#&’()*+,./0123456789:;?ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz”. It can be seen that most of the time, the characters are predicted exactly at the position they appear in the image (e.g. compare the position of the “i” in the image and in the graph). Only the last character “e” is not aligned. But this is OK, as the CTC operation is segmentation-free and does not care about absolute positions. From the bottom-most graph showing the scores for the characters “l”, “i”, “t”, “e” and the CTC blank label, the text can easily be decoded: 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---i--t-t--l-…-e” → “l---i--t-t--l-…-e” → “little”.

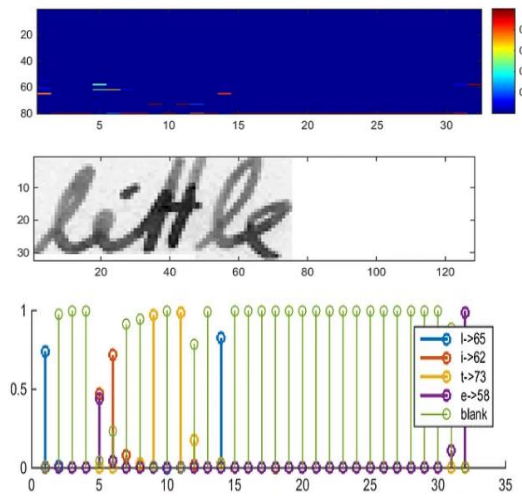


Fig. 5: Top: output matrix of the RNN layers. Middle: input image. Bottom: Probabilities for the characters “l”, “i”, “t”, “e”.

6. TENSOR FLOW

Tensor Flow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art.

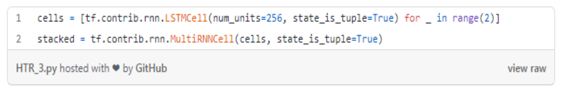
Implementation using TF

The implementation consists of 4 modules:

1. SamplePreprocessor.py: prepares the images from the IAM dataset for the RNN
2. DataLoader.py: reads samples, puts them into batches and provides an iterator-interface to go through the data
3. Model.py: creates the model as described above, loads and saves models, manages the TF sessions and provides an interface for training and inference
4. main.py: puts all previously mentioned modules together.

7. MATH

Create and stack two RNN layers with 256 units each.



```
cells = [tf.contrib.rnn.LSTMCell(num_units=256, state_is_tuple=True) for _ in range(2)]
stacked = tf.contrib.rnn.MultiRNNCell(cells, state_is_tuple=True)
```

Then, create a bidirectional RNN from it, such that the input sequence is traversed from front to back and the other way round. As a result, we get two output sequences fw and bw of size 32×256, which we later concatenate along the feature-axis to form a sequence of size 32×512. Finally, it is mapped to the output sequence (or matrix) of size 32×80 which is fed into the CTC layer.

REFERENCE

[1] Akm Ashiquzzaman and Abdul Kawsar Tushar, ”Handwritten Arabic Numeral Recognition using Deep Learning Neural Networks”, University of Asia Pacifific, Dhaka, Bangladesh.

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

[3] International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-9 Issue-1, November 2019

[4] N. Arica and F. Yarman-Vural, “An Overview of Character Recognition Focused on Off-line Handwriting”, IEEE Transactions on Systems, Man, and Cybernetics, Part C: ,Applications and Reviews, 2016, 31(2)

[5] Ankit Sharma. Character recognition using neural network(IJETT 2013).

[6] Rokus Arnold*, Poth Miklos*.Character Recognition Using Neural Networks.IEEE,2010.