**Topic: Text Scanner  
Mostak Ahmed, Sadia Afrin Khan  
Id: 16155003, 16155008  
Course name: Machine learning Lab  
Department of Information Technology**

**University of Information Technology & Sciences**

**Abstract:**   
Everyday technology is developing and new things are introduced to us to make our life easier. Today’s world we use lots of technique or application to make our works easy. So that we make a text scanner which helps us to convert text from written format to document. So that we need some algorithm like CNN, RNN, CTC etc. Convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing. Recurrent Neural Networks (RNNs) are used for sequence recognition tasks such as Handwritten Text Recognition or trained on the IAM off-line HTR dataset. We will build a Neural Network (NN) which is trained on word-images from the IAM dataset. Connectionist temporal classification (CTC) is a type of neural network output and associated scoring function. As these word-images are smaller than images of complete text-lines, the NN can be kept small and training on the CPU is feasible. 3/4 of the words from the validation-set are correctly recognized and the character error rate is around 10%.We will give some hints how to extend the model in case you need larger input-images or want better recognition accuracy.

## Keywords: **CNN, RNN, CTC,** Data Input, **CNN output, RNN output,** Implementation using Tensor Flow, Training.

## 1. Introduction: Handwritten Text Recognition is the task of transcribing handwritten text into digital text. It is an offline recognition. Offline recognition is performed after the test has been written. The text is captured and the resulting images are processed. Challenges regarding HTR include the cursive nature of handwriting, the variety of each character in size and shape and large vocabularies.

## 

## 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. Fig. 2 shows an overview of our HTR system.

## 

## We can also view the NN 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 (as long as the individual characters get correctly classified).

## 

## 2. Operations

**CNN:**   
Convolutional neural network is a class of [deep neural networks](https://en.wikipedia.org/wiki/Deep_neural_network), most commonly applied to analyzing visual imagery. CNNs are [regularized](https://en.wikipedia.org/wiki/Regularization_(mathematics)) versions of [multilayer perceptron](https://en.wikipedia.org/wiki/Multilayer_perceptron). Multilayer perceptron’s usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. These layers are trained to extract relevant features from the image. Each layer consists of three operation. First, the convolution operation, which applies a filter kernel of size 5×5 in the first two layers and 3×3 in the last three layers to the input. Then, the non-linear Rectified Linear Units 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 32×256.

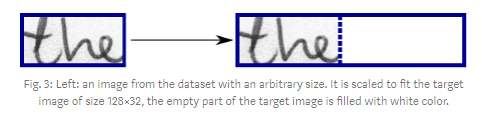
**RNN:**   
Recurrent neural network (RNN) is a class of [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network) where connections between nodes form a [directed graph](https://en.wikipedia.org/wiki/Directed_graph) along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike [feedforward neural networks](https://en.wikipedia.org/wiki/Feedforward_neural_networks), RNNs can use their internal state to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected [handwriting recognition](https://en.wikipedia.org/wiki/Handwriting_recognition) or [speech recognition](https://en.wikipedia.org/wiki/Speech_recognition).

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

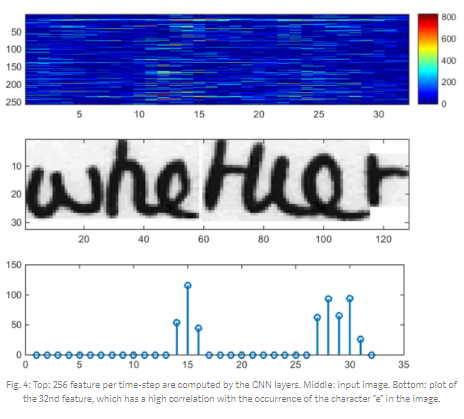
**CTC:**   
Connectionist temporal classification (CTC) is a type of neural network output and associated scoring function, for training [recurrent neural networks](https://en.wikipedia.org/wiki/Recurrent_neural_network) (RNNs) such as [LSTM](https://en.wikipedia.org/wiki/Long_short-term_memory) networks to tackle sequence problems where the timing is variable. It can be used for tasks like handwriting recognition or recognizing phonemes in speech audio. CTC refers to the outputs and scoring, and is independent of the underlying neural network structure.

While training the NN, the CTC is given the 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**. Both the ground truth text and the recognized text can be at most 32 characters long.

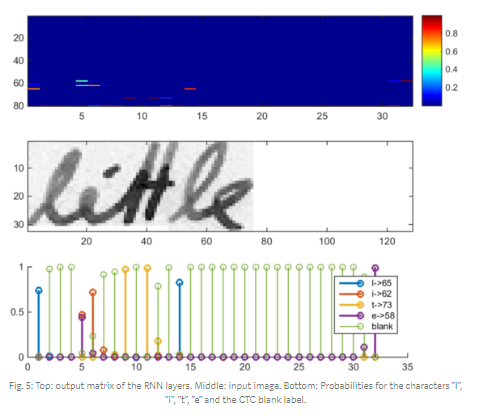
**Data Input:**   
It is a gray-value image of size 128×32. Usually, the images from the dataset do not have exactly this size, therefore we resize it (without distortion) until it either has a width of 128 or a height of 32. Then, we copy the image into a (white) target image of size 128×32. This process 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.



**CNN output**:   
Fig. 4 shows the output of the CNN layers which is a sequence of length 32. Each entry contains 256 features. Of course, these features are further processed by the RNN layers, however, some features already show a high correlation with certain high-level properties of the input image: there are features which have a high correlation with characters (e.g. “e”), or with duplicate characters (e.g. “tt”), or with character-properties such as loops (as contained in handwritten “l”s or “e”s).



**RNN output**:  
Fig. 5 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---ii--t-t--l-…-e” → “l---i--t-t--l-…-e” → “little”.



# **3. Implementation using TF:**

# The implementation consists of 4 modules:

1. SamplePreprocessor.py: prepares the images from the IAM dataset for the NN
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

We only look at Model.py, as the other source files are concerned with basic file IO (DataLoader.py) and image processing (SamplePreprocessor.py).

## CNN: For each CNN layer, create a kernel of size k×k to be used in the convolution operation.



Then, feed the result of the convolution into the RELU operation and then again to the pooling layer with size px×py and step-size sx×sy.



These steps are repeated for all layers in a for-loop.

## RNN: Create and stack two RNN layers with 256 units each.

## 

## 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.

## 

## CTC: For loss calculation, we feed both the ground truth text and the matrix to the operation. The ground truth text is encoded as a sparse tensor. The length of the input sequences must be passed to both CTC operations.

## 

## We now have all the input data to create the loss operation and the decoding operation.

## 

## Training: The mean of the loss values of the batch elements is used to train the NN: it is fed into an optimizer such as RMSProp.

## 

## 4. Improving the model:

In case you want to feed complete text-lines as shown in Fig. 6 instead of word-images, you have to increase the input size of the NN.

## 

If you want to improve the recognition accuracy, you can follow one of these hints:

* Data augmentation: increase dataset-size by applying further transformations to the input images
* Remove cursive writing style in the input images.
* Increase input size (if input of NN is large enough, complete text-lines can be used)
* Add more CNN layers
* Replace LSTM by 2D-LSTM
* Decoder: use token passing or word beam search decoding to constrain the output to dictionary words
* Text correction: if the recognized word is not contained in a dictionary, search for the most similar one.

# **5. Conclusion:** We discussed a NN which is able to recognize text in images. The NN consists of 5 CNN and 2 RNN layers and outputs a character-probability matrix. This matrix is either used for CTC loss calculation or for CTC decoding. An implementation using TF is provided and some important parts of the code were presented. Finally, hints to improve the recognition accuracy were given.

# **References:**

1. <https://en.wikipedia.org/wiki/Convolutional_neural_network>
2. <https://en.wikipedia.org/wiki/Connectionist_temporal_classification>
3. <https://en.wikipedia.org/wiki/Recurrent_neural_network>
4. <https://repositum.tuwien.ac.at/obvutwhs/download/pdf/2874742>
5. <https://arxiv.org/pdf/1507.05717.pdf>
6. <https://repositum.tuwien.ac.at/obvutwoa/download/pdf/2774578>
7. <http://www.fki.inf.unibe.ch/databases/iam-handwriting-database>