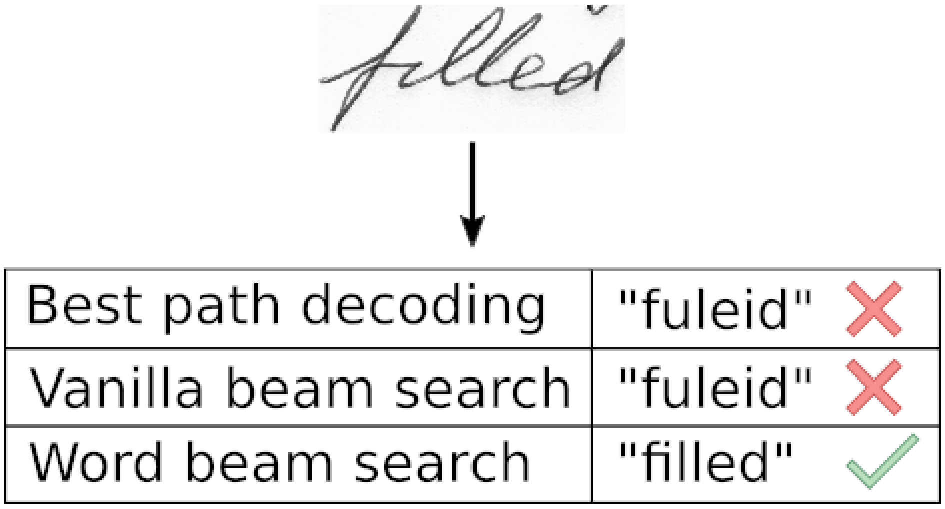
Introduction

Offline Handwritten Text Recognition (HTR) systems transcribe text contained in scanned images into digital text. We will build a Neural Network (NN) which 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. As the input layer can be kept small for word-images, NN-training is feasible on the CPU (of course, a GPU would be better).

The word beam search decoder can be used instead of the two decoders shipped with TF. Words are constrained to those contained in a dictionary, but arbitrary non-word character strings (numbers, punctuation marks) can still be recognized. The following illustration shows a sample for which word beam search is able to recognize the correct text, while the other decoders fail.



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

Literature Overview

1. Understanding of a convolutional neural network | IEEE Conference Publication | IEEE Xplore

One of the most popular deep neural networks is the Convolutional Neural Network (CNN). CNN has an excellent performance in machine learning problems. Specially the applications that deal with image data, such as the largest image classification data set (Image Net), computer vision, and natural language processing (NLP) and the results achieved were very amazing.

2. IET Digital Library: Handwriting recognition using CNN and its optimization approach (theiet.org)

Publication Date: October 2021

3. Exploration of CNN Features for Online Handwriting Recognition | IEEE Conference Publication | IEEE Xplore
   Published in: 2019 International Conference on Document Analysis and Recognition (ICDAR)
4. Offline Handwritten Numeral Recognition using Combination of Different Feature Extraction Techniques

A handwritten numeral recognition system using a combination of different feature extraction techniques has been presented in this paper. SVM classifier has been considered for classification purposes. For experimental results, 6000 samples of isolated handwritten numerals have been considered. The proposed system achieves maximum recognition accuracy of 96.3% using a five-fold cross-validation technique.

5. Handwritten Character Recognition in English: A Survey

This paper presents a comprehensive review of Handwritten Character Recognition (HCR) in the English language. Handwritten character recognition has been applied in a variety of applications like Banking sectors, Health care industries, and many such organizations where handwritten documents are dealt with. Handwritten Character Recognition is the process of conversion of handwritten text into machine-readable form. For handwritten characters, there are difficulties like it differs from one writer to another, even when the same person writes the same character there is a difference in shape, size, and position of the character. The latest research in this area has used different types of methods, classifiers, and features to reduce the complexity of recognizing handwritten text.

Problem Definition

People make notes on pen and paper. Everyone does not have the access to products through which they can create handwritten notes in a digital file. They have to manually type the written notes into a digital file. It is a very cumbersome process as it wastes manpower and time.

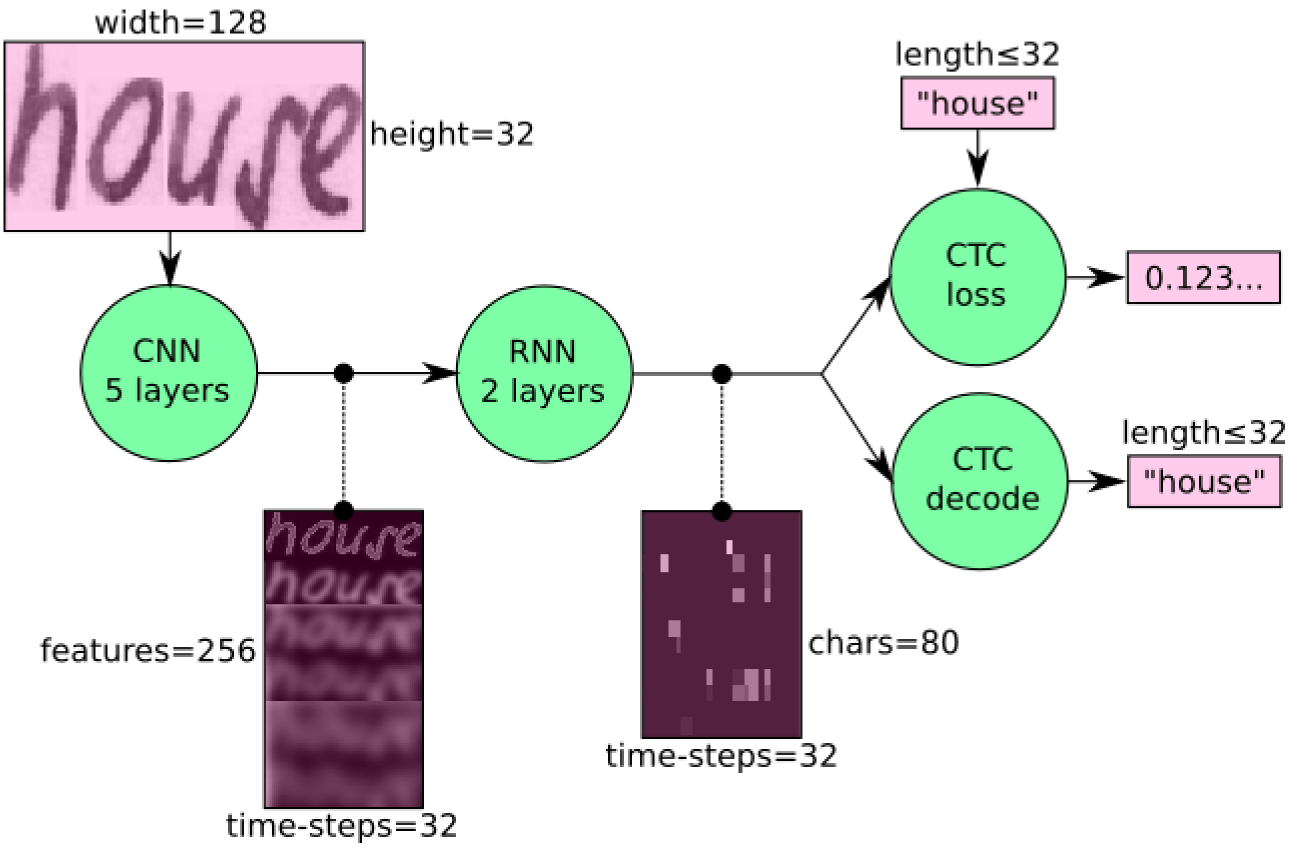
Objective

To provide an easy-to-use application which would convert handwritten images into digital and editable form. To save time and resources by providing a platform which will convert a tedious manual task into an easy to click process.

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.

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. Figure 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).

$$\mathrm{NN}: \underset{W \times H}{M} \rightarrow \underset{0 \le n \le L}{(c_1, c_2, \ldots, c_n)}$$

Operations

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 operation.

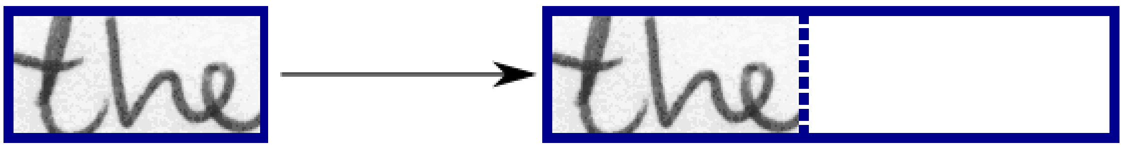
First, the convolution operation, which applies a filter kernel of size 5×5 in the first two layers and 3×3 in the last three layers to the input. Then, the non-linear RELU function is applied. Finally, a pooling layer summarizes image regions and outputs a downsized version of the input. While the image 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 32×256.

RNN: 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), therefore there are 80 entries for each of the 32 time-steps.

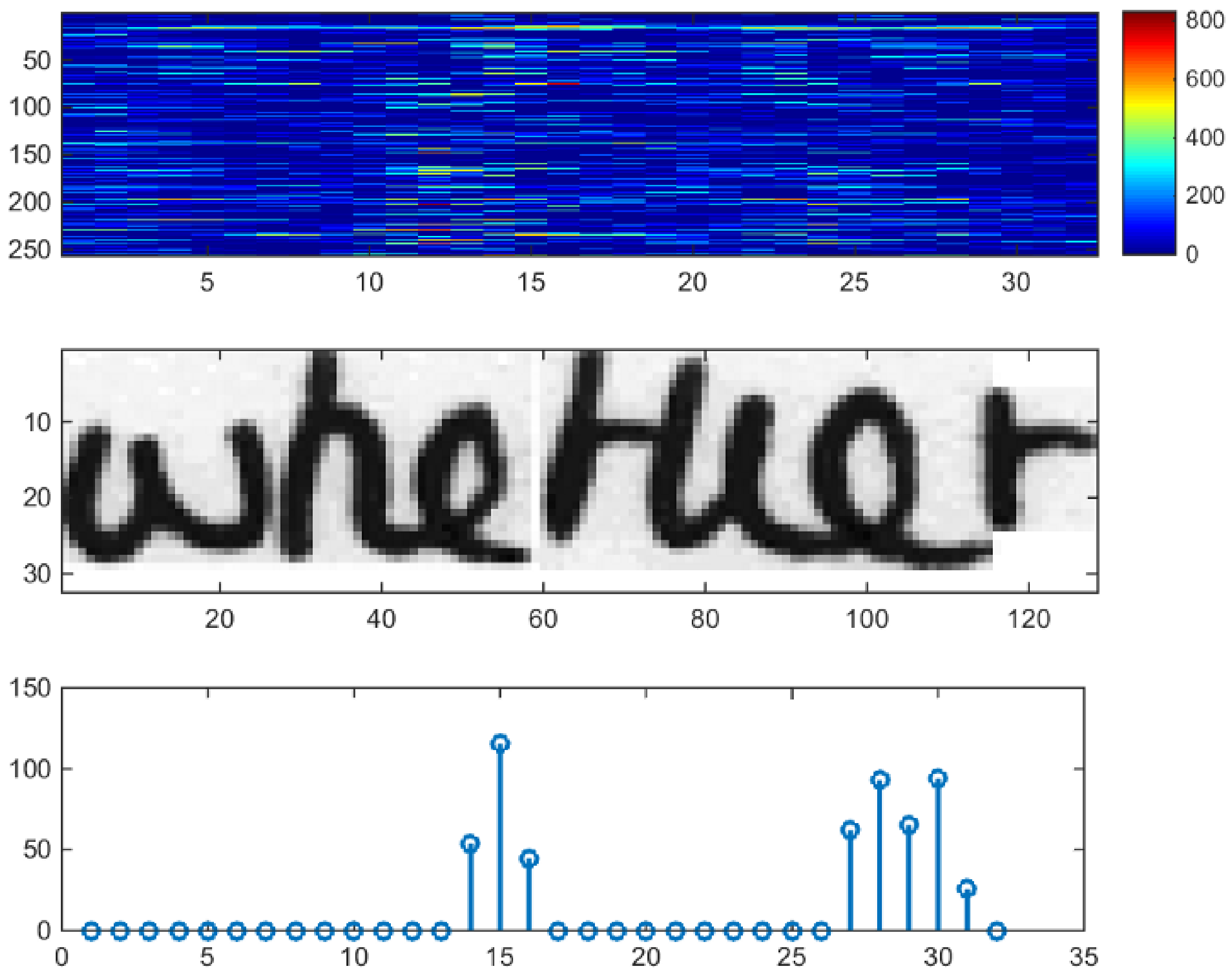
CTC: while training the NN, the CTC is given the RNN output matrix and the ground truth text and it computes the **loss value**. While inferring, the CTC is only given the matrix and it decodes it into the **final text**. Both the ground truth text and the recognized text can be at most 32 characters long.

Data

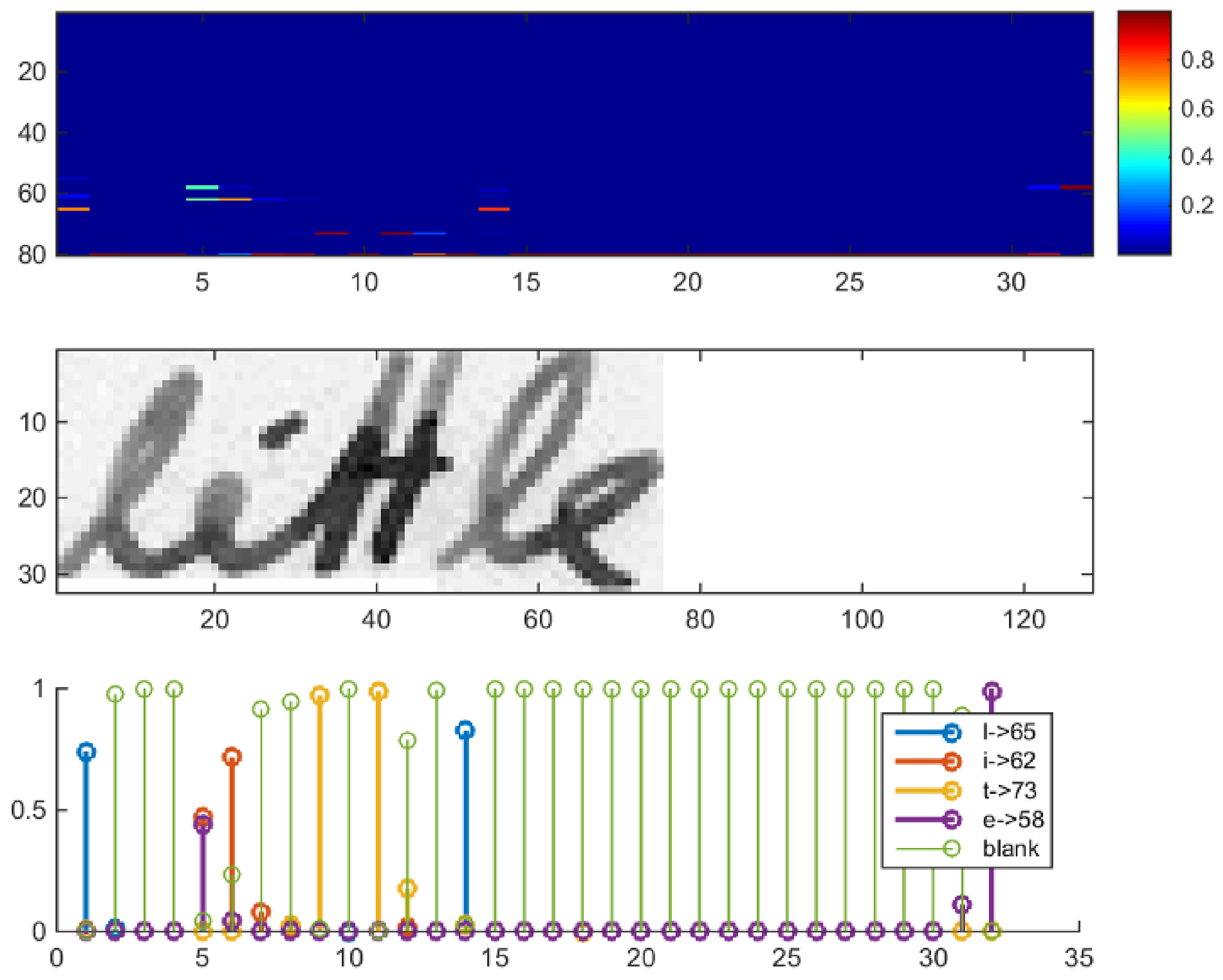
Input: it is a gray-value image of size 128×32. Usually, the images from the dataset do not have exactly this size, therefore we resize it (without distortion) until it either has a width of 128 or a height of 32. Then, we copy the image into a (white) target image of size 128×32. This process is shown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



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.

```
kernel = tf.Variable(tf.truncated_normal([k, k, chIn, chOut],
stddev=0.1))
conv = tf.nn.conv2d(inputTensor, kernel, padding='SAME', strides=(1, 1,
1, 1))
```

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.

```
relu = tf.nn.relu(conv)
pool = tf.nn.max_pool(relu, (1, px, py, 1), (1, sx, sy, 1), 'VALID')
```

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

RNN

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.

```
((fw, bw),_) = tf.nn.bidirectional_dynamic_rnn(cell_fw=stacked,
cell_bw=stacked, inputs=inputTensor, dtype=inputTensor.dtype)
```

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.

```
gtTexts = tf.SparseTensor(tf.placeholder(tf.int64, shape=[None, 2]),
tf.placeholder(tf.int32, [None]), tf.placeholder(tf.int64, [2]))
```

```
seqLen = tf.placeholder(tf.int32, [None])
```

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

```
loss = tf.nn.ctc_loss(labels=gtTexts, inputs=inputTensor,
sequence_length=seqLen, ctc_merge_repeated=True)

decoder = tf.nn.ctc_greedy_decoder(inputs=inputTensor,
sequence_length=seqLen)
```

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.

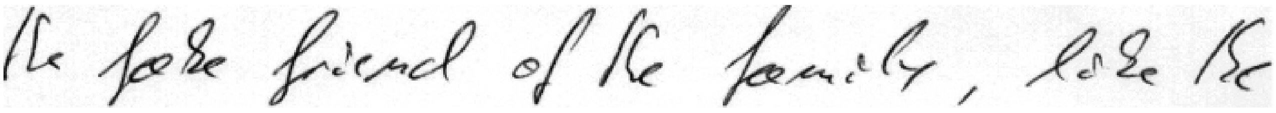
```
optimizer = tf.train.RMSPropOptimizer(0.001).minimize(loss)
```

Improving the model

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

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 (random) transformations to the input images
- Remove cursive writing style in the input images (see DeslantImg)
- 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 (see CTCWordBeamSearch) 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

Methodology

1. Deep Learning: Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign or to distinguish a pedestrian from a lamppost. Deep learning is a branch of machine learning that uses neural networks with many layers. A deep neural network analyzes data with learned representations similar to the way a person would look at a problem.

2. Convolutional Neural Network (CNN): It is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. It is specifically used for image recognition and tasks that involve the processing of pixel data.

3. Recurrent Neural Network(RNN): Recurrent Neural Network is a type of neural network used to deal specifically with sequential data. RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

4. Git: Git is a version control system used for tracking changes in computer files. It is generally used for source code management in software development. Git is used to track changes in the source code. Usually used for coordinating work among programmers collaboratively developing source code during software development.

5. Streamlit*: Streamlit is an open-source app framework in Python language. It helps us create web apps for data science and machine learning in a short time. It is compatible with major Python libraries such as `Scikit-learn`,`Keras`, `PyTorch`, `SymPy(latex)`, `NumPy`, `pandas`, `Matplotlib`, etc.

6. Docker*: It is an open platform for developing, shipping, and running applications. Docker enables us to separate our applications from our infrastructure so, we can deliver software quickly. With Docker, we can manage our infrastructure in the same ways we manage our applications.

> Docker: *Experimental Feature* | *Streamlit*: Experimental Feature

Reference

- Build a Handwritten Text Recognition System using TensorFlow
- Scheidl - Handwritten Text Recognition in Historical Documents
- Scheidl - Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm
- GeeksforGeeks | A computer science portal for geeks
- 3.11.0 Documentation (python.org)
- API Documentation | TensorFlow v2.11.0
- Streamlit documentation
- FAQ

- What a text recognition system actually sees
- Introduction to CTC
- Vanilla beam search decoding
- Word beam search decoding
- Thesis on handwritten text recognition in historical documents
- Word beam search decoding
- Convolutional Recurrent Neural Network (CRNN)
- Recognize text on page-level

- Obsidian Help
- Docker Documentation | Docker Documentation
- Git - Documentation (git-scm.com)
- Medium – Where good ideas find you.