Handwritten Text Recognition (word-level) with TensorFlow

Technische Universität Berlin

QU-Projects-ML-NLP-WS-2019-20
Advanced Projects at the Quality and Usability Lab, QU, PJ, WS 2019/20
TOPIC: Machine Learning in Natural Language Processing and Dialog Systems

Prepared by:
Mahmoud Alsharif 387818

ABSTRACT

The problem of handwritten Word recognition has been still for a long time an open problem in the field of pattern classification. a large amount of studies has shown that Neural networks, machine learning have great and efficient performance. In data classification, Deep learning and Neural Network algorithms are a branch of Machine learning that may automatically identify patterns in the data, then use the uncovered patterns to predict future data, or to perform other alternative types of deciding under unreliability Deep Learning algorithms are used to model high-level abstractions in data. word Recognition is a combination of Deep Learning and Neural Network algorithms, which uses the TensorFlow tool as an interface to develop a model. This paper describes the recognition of handwritten scanned word where the input is given by the user and displays the output as digital words referring as the input provided accordingly by using Machine Learning methods with the assistance of TensorFlow, IAM Database, python thus the image could also be sensed by the system as the user provides barehanded input to the system and then the system shows the respective recognition accordingly.

Keywords: *Handwritten Text Recognition, TensorFlow, IAM Database, Python, Machine Learning, images.*

1. INTRODUCTION

Machine learning and deep learning play an important role in computer sciences, its equipment, and AI. The use of machine learning, deep learning, and related principles have lowered human efforts in the industry. Handwritten word recognition has gained a good amount of popularity from the very beginning of machine learning and deep learning to an expert who has been practicing for years. Developing such a machine needs a proper understanding of the classification of words and the difference between the minor and major points to properly differentiate between different words which can be only possible with proper training and testing.
Handwritten recognition (HWR) is the ability of a computer to receive and understand intelligible handwritten input from sources such as paper documents, user input touch-screens, and other devices. The image of the written text may be sensed from a piece of paper by optical scanning by user input.

This paper presents recognizing the handwritten word from the famous IAM dataset using the TensorFlow framework (library) and python as a language and its libraries as a user enters the respective word, the machine would recognize and show the results with an accuracy percentage.

2. METHODOLOGY

2.1 Tensorflow:

TensorFlow is a software library or framework, designed by the Google team to implement machine learning and deep learning concepts in the easiest manner. It combines the computational algebra of optimization techniques for easy calculation of many mathematical expressions Tensorflow is an open-source library created by the Google Brain Trust for heavy computational work, geared towards machine learning and deep learning tasks. Tensor Flow is built on c, c++ making it very fast while it is available for use via Python, C++, Haskell, Java and Go API depending upon the type of work. It created data graph flows for each model, where a graph consists of two units – a tensor and a node.

- Tensor: A tensor is any dimensional array that is not single-dimensional.
- Node: A node is a mathematical computation that is being worked at the moment to give the desired result.

A data graph flow essentially maps the flow of information via the interchange between these two components. As the graph is completed, the model is executed for the output.[1]

2.2 Numpy:

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays.
It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities.

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.
Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.[2]

2.3 OpenCV-python:

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.
Python is a general-purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability. Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in the background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.
OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays.
This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.[3]

2.4 Python3:

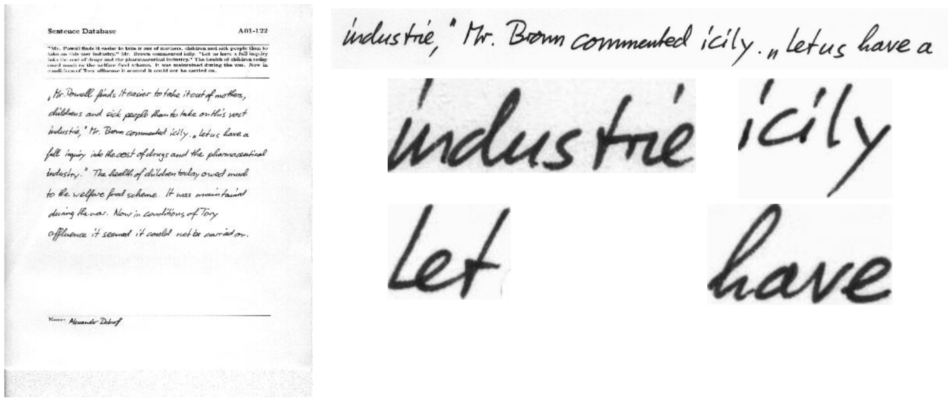
Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language.

Also, Python is a powerful programming language ideal for scripting and rapid application development. It is used in web development (like Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D).

Python 3.0 was released in 2008. Although this version is supposed to be backward-incompatible, later on, many of its important features have been backported to be compatible with version 2.7. This tutorial gives enough understanding of Python 3 version programming language.[4]

2.2 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 was first published at the ICDAR 1999. Using this database an HMM-based recognition system for handwritten sentences was developed and published at the ICPR 2000. The segmentation scheme used in the second version of the database is documented and has been published in the ICPR 2002. The IAM-database as of October 2002 is described. We use the database extensively in our own research, see publications for further details. 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 figure (Fig.1) below provides samples of a complete form, a text line, and some extracted words.



Sentence Database A01-122

"Mr. Powell finds it easier to take it out of mothers, children and sick people than to take on this vast industry." Mr. Brown commented icily. "Let us have a full inquiry into the cost of drugs and the pharmaceutical industry." The health of children today owed much to the welfare food scheme. It was maintained during the war. Now in conditions of Tory affluence it seemed it could not be carried on.

„Mr. Powell finds it easier to take it out of mothers, childrens and sick people than to take on this vast industrie," Mr. Brown commented icily. „let us have a full inquiry into the cost of drugs and the pharmaceutical industry." The health of children today owed much to the welfare food scheme. It was maintained during the war. Now in conditions of Tory affluence it seemed it could not be carried on.

Name: [illegible]

industrie," Mr. Brown commented icily. „let us have a

industrie icily

let have

Fig.1. Some examples of IAM dataset

All forms and also all extracted text lines, words and sentences are available for download as PNG files, with corresponding XML meta-information included in the image files. All texts in the IAM database are built using sentences provided by the LOB Corpus.

3. IMPLEMENTING THE HANDWRITTEN WORD RECOGNITION MODEL

We should be importing the following libraries to prepare the environment and python libraries :

❖ TensorFlow:

```
import tensorflow as tf
```

❖ Numpy:

```
import numpy as np
```

❖ OpenCV:

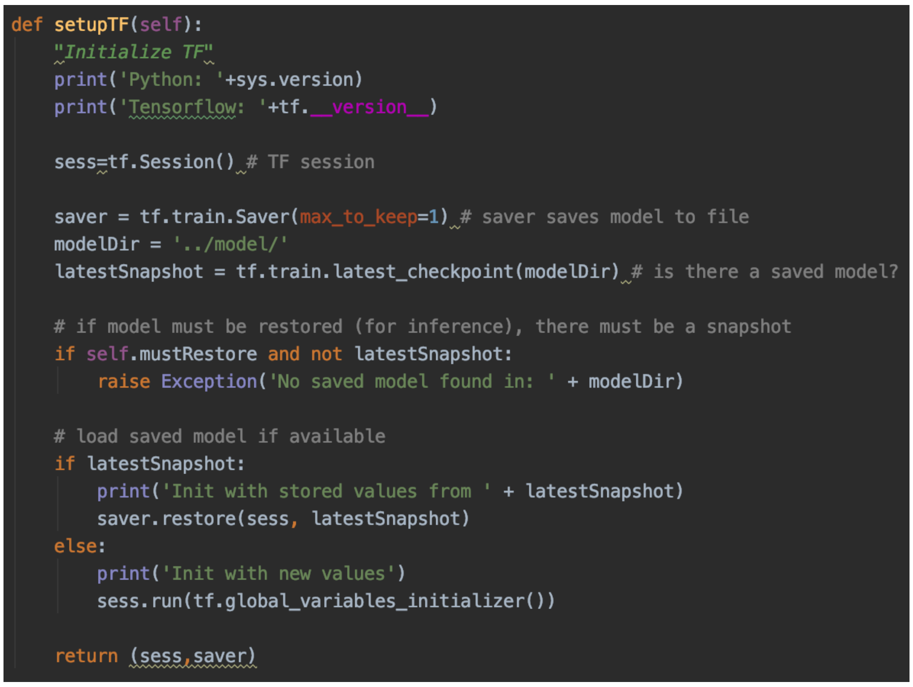
```
import cv2
```

❖ Other libraries:

```
import editdistance
import random
import os
import sys
import argparse
```

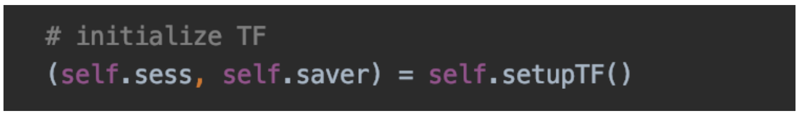
Implementation using TF

- ***Initializing and creating the TF session:***



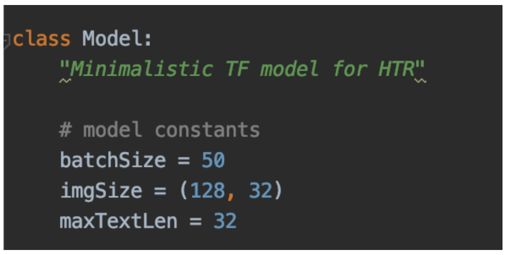
```
def setupTF(self):
    "Initialize TF"
    print('Python: '+sys.version)
    print('Tensorflow: '+tf.__version__)

    sess=tf.Session() # TF session

    saver = tf.train.Saver(max_to_keep=1) # saver saves model to file
    modelDir = '../model/'
    latestSnapshot = tf.train.latest_checkpoint(modelDir) # is there a saved model?

    # if model must be restored (for inference), there must be a snapshot
    if self.mustRestore and not latestSnapshot:
        raise Exception('No saved model found in: ' + modelDir)

    # load saved model if available
    if latestSnapshot:
        print('Init with stored values from ' + latestSnapshot)
        saver.restore(sess, latestSnapshot)
    else:
        print('Init with new values')
        sess.run(tf.global_variables_initializer())

    return (sess,saver)
```

- ***Initializing TF session***



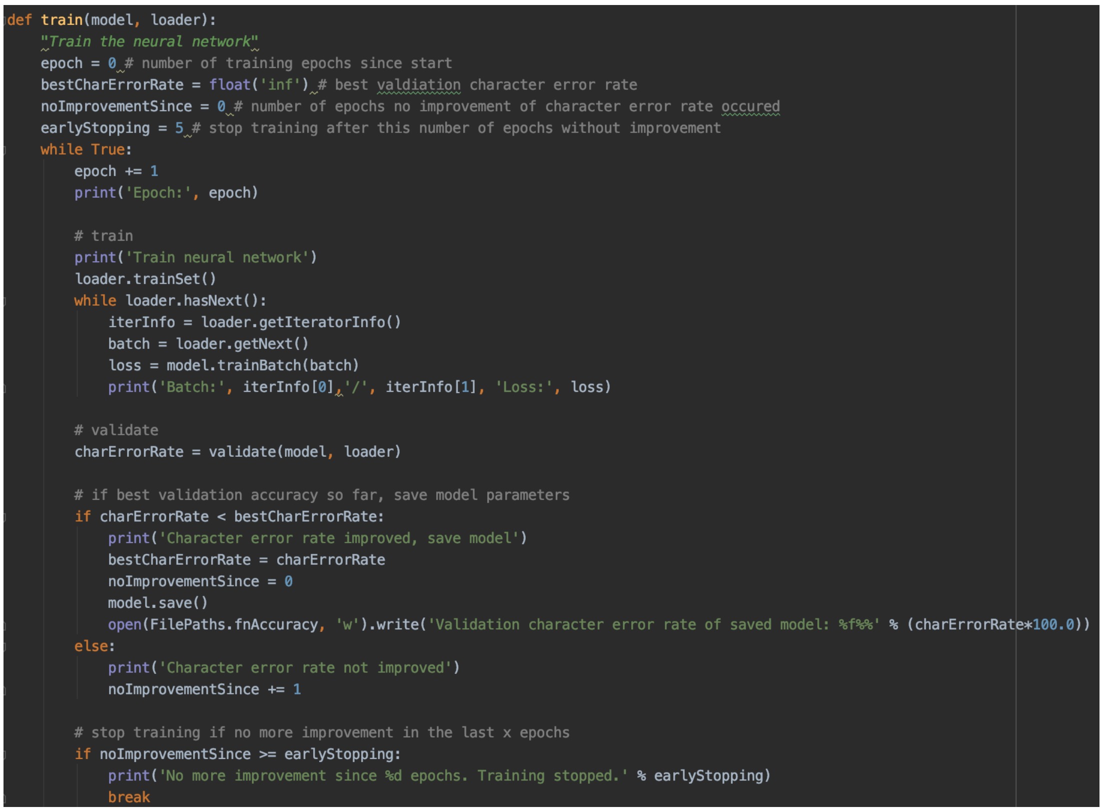
```
# initialize TF
(self.sess, self.saver) = self.setupTF()
```

- ***Minimalistic TF model for HTR:***



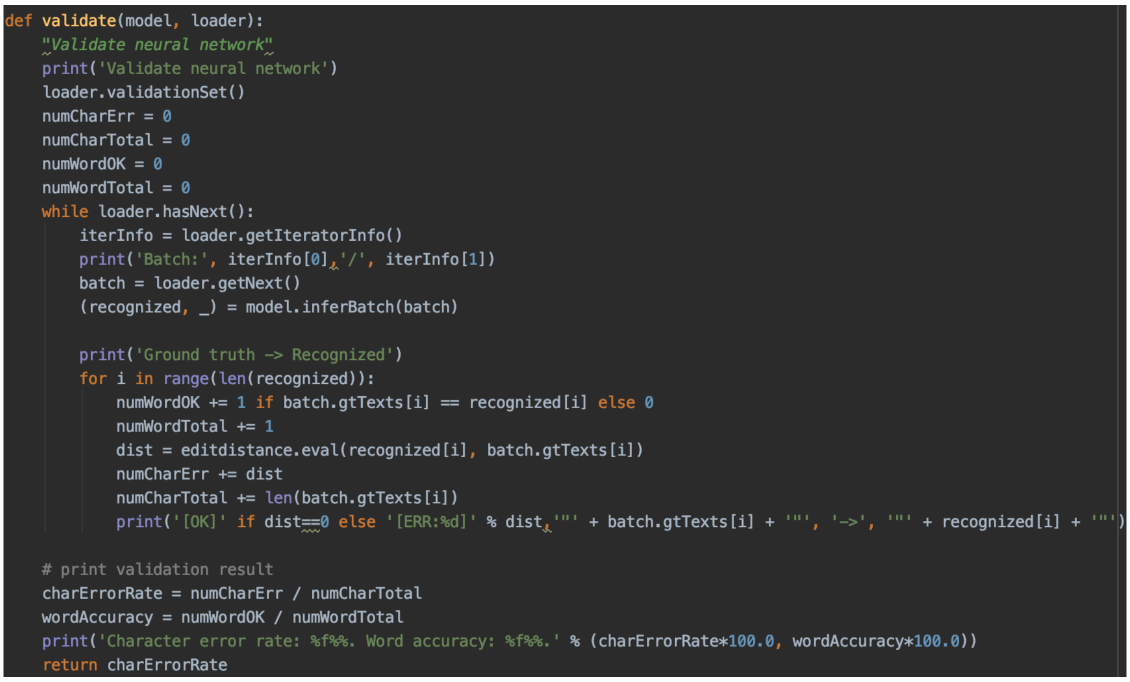
```
class Model:
    "Minimalistic TF model for HTR"

    # model constants
    batchSize = 50
    imgSize = (128, 32)
    maxTextLen = 32
```

- ***Training the Model in main.py:***



```
def train(model, loader):
    "Train the neural network"
    epoch = 0 # number of training epochs since start
    bestCharErrorRate = float('inf') # best valdiation character error rate
    noImprovementSince = 0 # number of epochs no improvement of character error rate occured
    earlyStopping = 5 # stop training after this number of epochs without improvement
    while True:
        epoch += 1
        print('Epoch:', epoch)

        # train
        print('Train neural network')
        loader.trainSet()
        while loader.hasNext():
            iterInfo = loader.getIteratorInfo()
            batch = loader.getNext()
            loss = model.trainBatch(batch)
            print('Batch:', iterInfo[0],'/', iterInfo[1], 'Loss:', loss)

        # validate
        charErrorRate = validate(model, loader)

        # if best validation accuracy so far, save model parameters
        if charErrorRate < bestCharErrorRate:
            print('Character error rate improved, save model')
            bestCharErrorRate = charErrorRate
            noImprovementSince = 0
            model.save()
            open(FilePaths.fnAccuracy, 'w').write('Validation character error rate of saved model: %f%%' % (charErrorRate*100.0))
        else:
            print('Character error rate not improved')
            noImprovementSince += 1

        # stop training if no more improvement in the last x epochs
        if noImprovementSince >= earlyStopping:
            print('No more improvement since %d epochs. Training stopped.' % earlyStopping)
            break
```

- ***Validating the Model in main.py:***



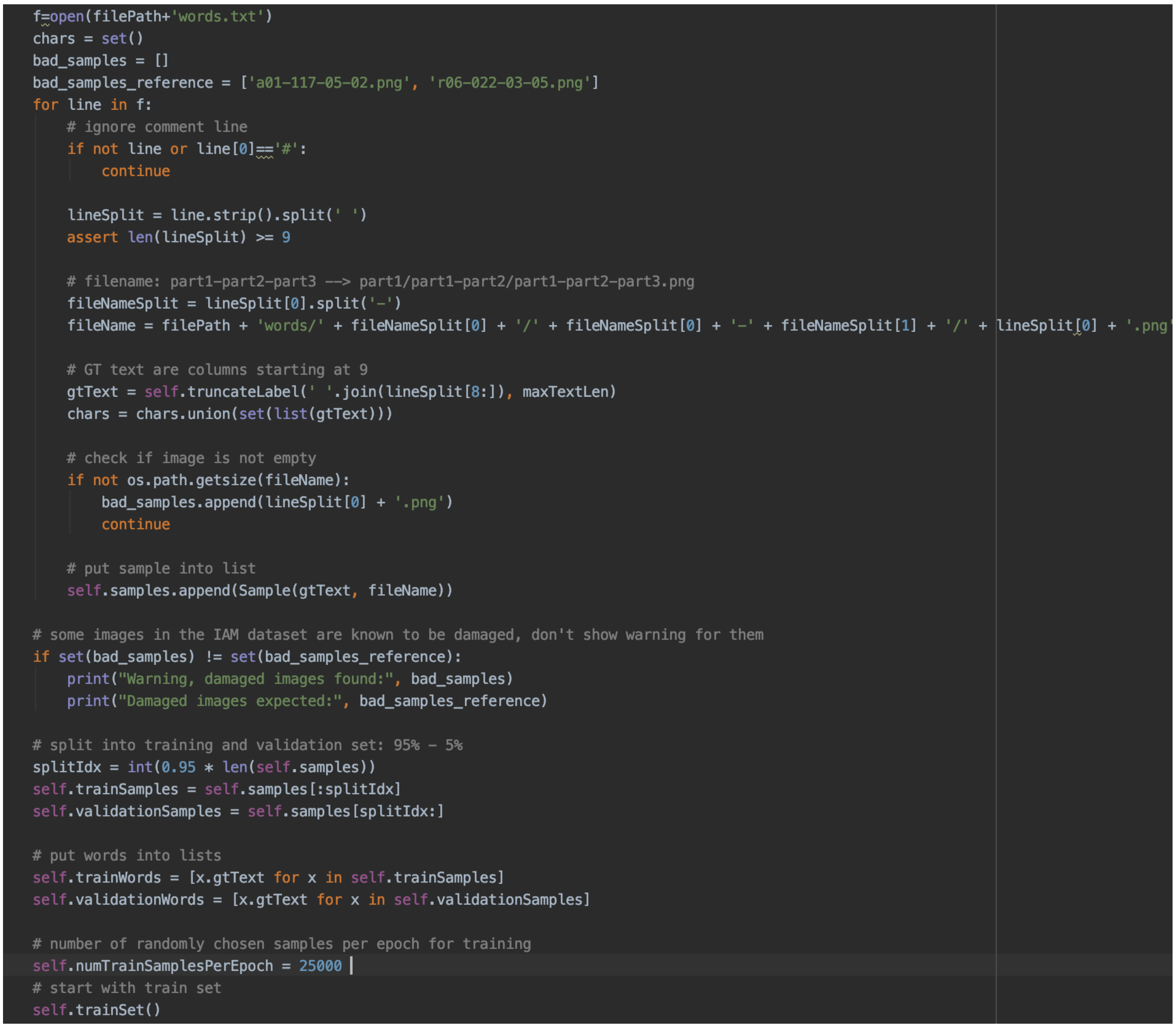
```
def validate(model, loader):
    "Validate neural network"
    print('Validate neural network')
    loader.validationSet()
    numCharErr = 0
    numCharTotal = 0
    numWordOK = 0
    numWordTotal = 0
    while loader.hasNext():
        iterInfo = loader.getIteratorInfo()
        print('Batch:', iterInfo[0],'/', iterInfo[1])
        batch = loader.getNext()
        (recognized, _) = model.inferBatch(batch)

        print('Ground truth -> Recognized')
        for i in range(len(recognized)):
            numWordOK += 1 if batch.gtTexts[i] == recognized[i] else 0
            numWordTotal += 1
            dist = editdistance.eval(recognized[i], batch.gtTexts[i])
            numCharErr += dist
            numCharTotal += len(batch.gtTexts[i])
            print('[OK]' if dist==0 else '[ERR:%d]' % dist,'"' + batch.gtTexts[i] + '"', '->', '"' + recognized[i] + '"')

    # print validation result
    charErrorRate = numCharErr / numCharTotal
    wordAccuracy = numWordOK / numWordTotal
    print('Character error rate: %f%%. Word accuracy: %f%%.' % (charErrorRate*100.0, wordAccuracy*100.0))
    return charErrorRate
```

The implementation consists of 4 modules:

- **helper.py**: prepares the images from the IAM dataset for the NN
- **DataLoader.py**: reads samples, put them into batches and provides an iterator-interface to go through the data.
- **Model.py**: creates the model as described above, loads and saves models, manages the TF sessions and provides an interface for training and inference.
- **Correction.py**: this function to try to correct the word from the English dictionary if no result founded
- **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 (helper.py)

Load the data from the Dataset DataLoder.py:



```
f=open(filePath+'words.txt')
chars = set()
bad_samples = []
bad_samples_reference = ['a01-117-05-02.png', 'r06-022-03-05.png']
for line in f:
    # ignore comment line
    if not line or line[0]=='#':
        continue

    lineSplit = line.strip().split(' ')
    assert len(lineSplit) >= 9

    # filename: part1-part2-part3 --> part1/part1-part2/part1-part2-part3.png
    fileNameSplit = lineSplit[0].split('-')
    fileName = filePath + 'words/' + fileNameSplit[0] + '/' + fileNameSplit[0] + '-' + fileNameSplit[1] + '/' + lineSplit[0] + '.png'

    # GT text are columns starting at 9
    gtText = self.truncateLabel(' '.join(lineSplit[8:]), maxTextLen)
    chars = chars.union(set(list(gtText)))

    # check if image is not empty
    if not os.path.getsize(fileName):
        bad_samples.append(lineSplit[0] + '.png')
        continue

    # put sample into list
    self.samples.append(Sample(gtText, fileName))

# some images in the IAM dataset are known to be damaged, don't show warning for them
if set(bad_samples) != set(bad_samples_reference):
    print("Warning, damaged images found:", bad_samples)
    print("Damaged images expected:", bad_samples_reference)

# split into training and validation set: 95% - 5%
splitIdx = int(0.95 * len(self.samples))
self.trainSamples = self.samples[:splitIdx]
self.validationSamples = self.samples[splitIdx:]

# put words into lists
self.trainWords = [x.gtText for x in self.trainSamples]
self.validationWords = [x.gtText for x in self.validationSamples]

# number of randomly chosen samples per epoch for training
self.numTrainSamplesPerEpoch = 25000
# start with train set
self.trainSet()
```

Model Explanation:

We use a NN (Neural Networks) 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. [5]

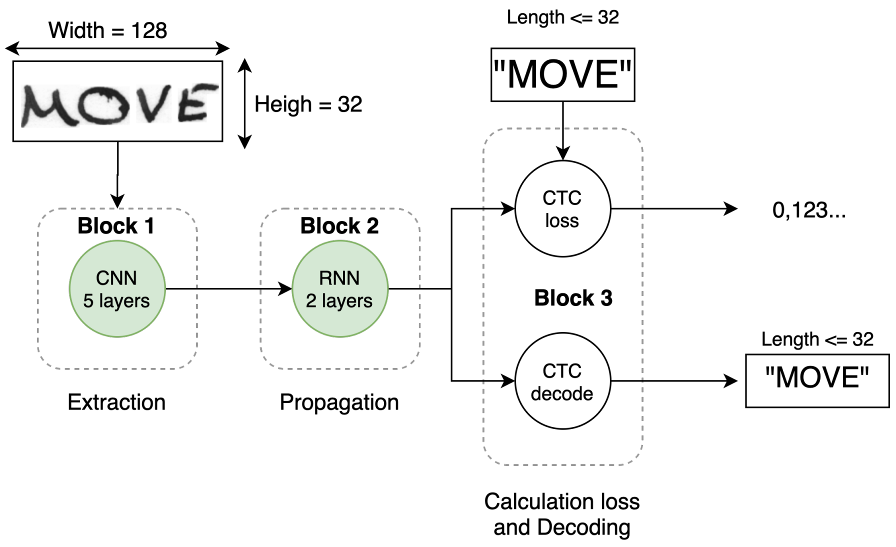
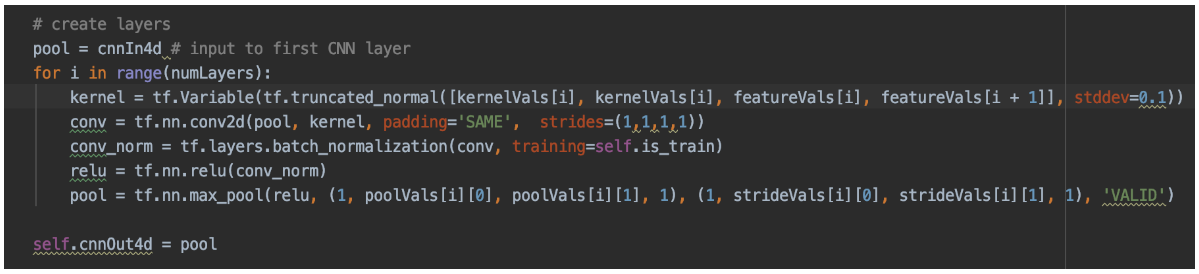


Fig.2. Model Overview

Operations in Model.py

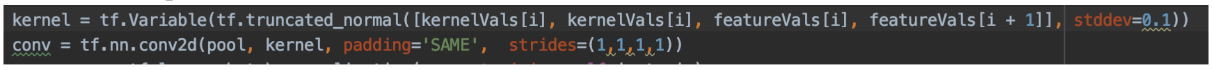
CNN: the input image is fed into the CNN layers. These layers are trained to extract relevant features from the image. Each layer consists of three operations. First, the convolution operation, which applies a filter kernel of size 5×5 in the first two layers and 3×3 in the last three layers to the input. 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.

To create the CNN layers:



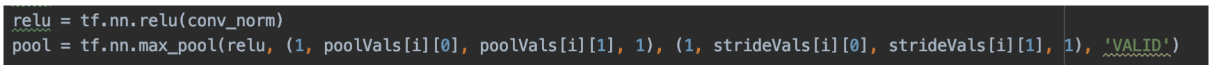
```
# create layers
pool = cnnIn4d # input to first CNN layer
for i in range(numLayers):
    kernel = tf.Variable(tf.truncated_normal([kernelVals[i], kernelVals[i], featureVals[i], featureVals[i + 1]], stddev=0.1))
    conv = tf.nn.conv2d(pool, kernel, padding='SAME',  strides=(1,1,1,1))
    conv_norm = tf.layers.batch_normalization(conv, training=self.is_train)
    relu = tf.nn.relu(conv_norm)
    pool = tf.nn.max_pool(relu, (1, poolVals[i][0], poolVals[i][1], 1), (1, strideVals[i][0], strideVals[i][1], 1), 'VALID')

self.cnnOut4d = pool
```

For each CNN layer, create a kernel of size k×k (kernelVals[i], kernelVals[i]) to be used in the convolution operation.



```
kernel = tf.Variable(tf.truncated_normal([kernelVals[i], kernelVals[i], featureVals[i], featureVals[i + 1]], stddev=0.1))
conv = tf.nn.conv2d(pool, 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 (poolVals[i][0], poolVals[i][1]) and step-size sx×sy (strideVals[i][0], strideVals[i][1]).

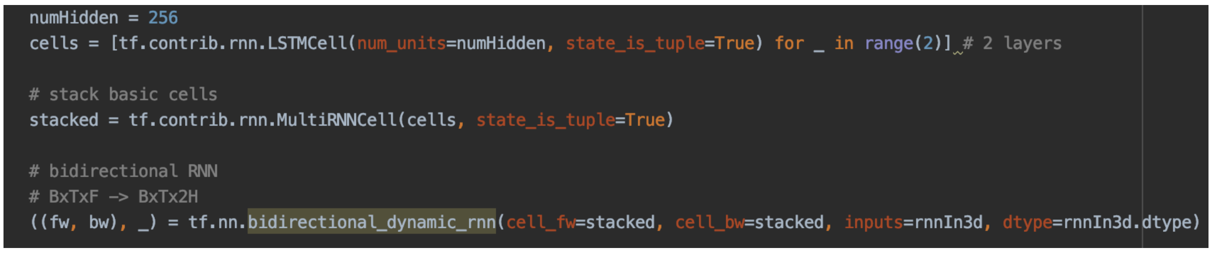


```
relu = tf.nn.relu(conv_norm)
pool = tf.nn.max_pool(relu, (1, poolVals[i][0], poolVals[i][1], 1), (1, strideVals[i][0], strideVals[i][1], 1), 'VALID')
```

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

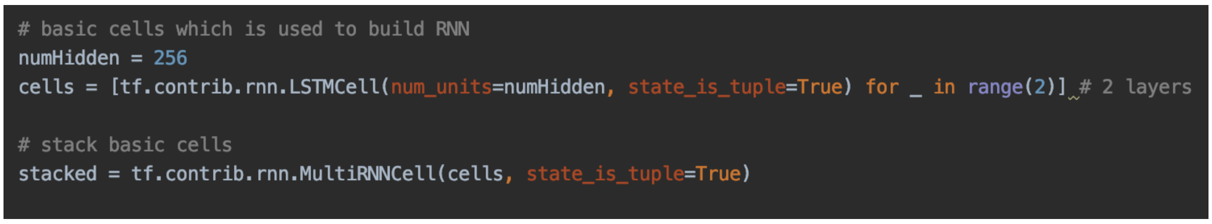
RNN: the feature sequence contains 256 features per time-step, the RNN propagates relevant information through this sequence. The popular Long Short-Term Memory (LSTM) implementation of RNNs is used, as it is able to propagate information through longer distances and provides more robust training-characteristics 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.

To create the RNN layers:



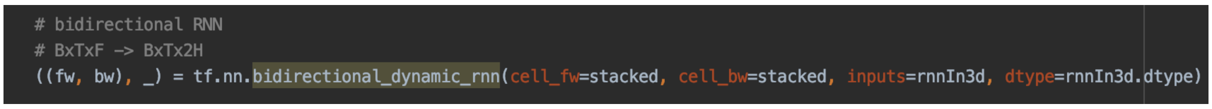
```
numHidden = 256
cells = [tf.contrib.rnn.LSTMCell(num_units=numHidden, state_is_tuple=True) for _ in range(2)] # 2 layers

# stack basic cells
stacked = tf.contrib.rnn.MultiRNNCell(cells, state_is_tuple=True)

# bidirectional RNN
# BxTxF -> BxTx2H
((fw, bw), _) = tf.nn.bidirectional_dynamic_rnn(cell_fw=stacked, cell_bw=stacked, inputs=rnnIn3d, dtype=rnnIn3d.dtype)
```

Create and stack two RNN layers with 256 units each.



```
# basic cells which is used to build RNN
numHidden = 256
cells = [tf.contrib.rnn.LSTMCell(num_units=numHidden, state_is_tuple=True) for _ in range(2)] # 2 layers

# stack basic cells
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.

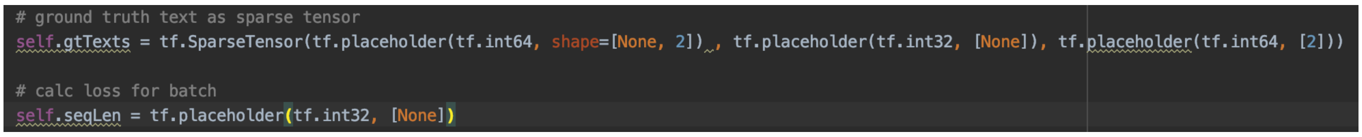


```
# bidirectional RNN
# BxTxF -> BxTx2H
((fw, bw), _) = tf.nn.bidirectional_dynamic_rnn(cell_fw=stacked, cell_bw=stacked, inputs=rnnIn3d, dtype=rnnIn3d.dtype)
```

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. [6]

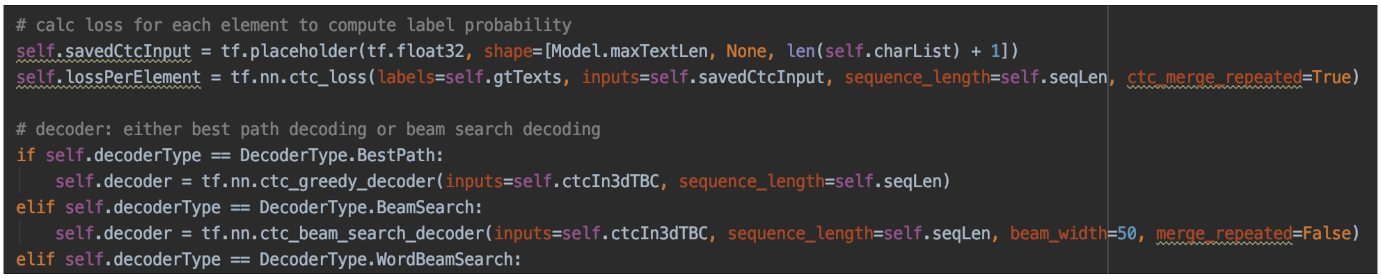
To create the CTC :

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.



```
# ground truth text as sparse tensor
self.gtTexts = tf.SparseTensor(tf.placeholder(tf.int64, shape=[None, 2]) , tf.placeholder(tf.int32, [None]), tf.placeholder(tf.int64, [2]))

# calc loss for batch
self.seqLen = tf.placeholder(tf.int32, [None])
```

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



```
# calc loss for each element to compute label probability
self.savedCtcInput = tf.placeholder(tf.float32, shape=[Model.maxTextLen, None, len(self.charList) + 1])
self.lossPerElement = tf.nn.ctc_loss(labels=self.gtTexts, inputs=self.savedCtcInput, sequence_length=self.seqLen, ctc_merge_repeated=True)

# decoder: either best path decoding or beam search decoding
if self.decoderType == DecoderType.BestPath:
    self.decoder = tf.nn.ctc_greedy_decoder(inputs=self.ctcIn3dTBC, sequence_length=self.seqLen)
elif self.decoderType == DecoderType.BeamSearch:
    self.decoder = tf.nn.ctc_beam_search_decoder(inputs=self.ctcIn3dTBC, sequence_length=self.seqLen, beam_width=50, merge_repeated=False)
elif self.decoderType == DecoderType.WordBeamSearch:
```

Prepare data (Images) to recognize in helper.py :

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.

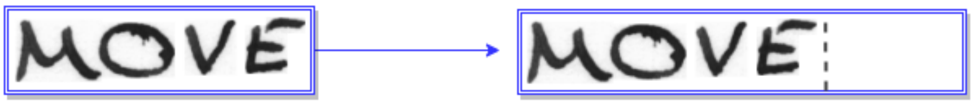
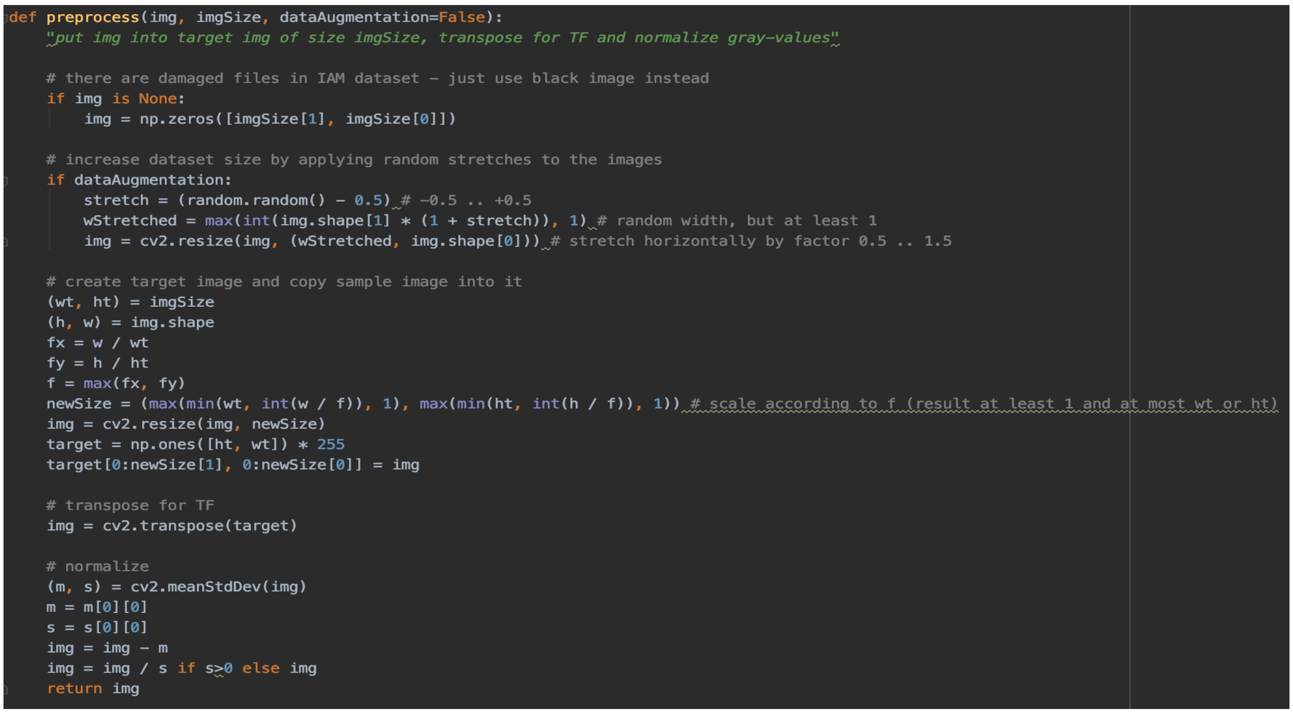


Fig. 3: Left: an image from the dataset with an arbitrary size. It is scaled to fit the target image of size 128×32, the empty part of the target image is filled with white color.



```
def preprocess(img, imgSize, dataAugmentation=False):
    "put img into target img of size imgSize, transpose for TF and normalize gray-values"

    # there are damaged files in IAM dataset - just use black image instead
    if img is None:
        img = np.zeros([imgSize[1], imgSize[0]])

    # increase dataset size by applying random stretches to the images
    if dataAugmentation:
        stretch = (random.random() - 0.5) # -0.5 .. +0.5
        wStretched = max(int(img.shape[1] * (1 + stretch)), 1) # random width, but at least 1
        img = cv2.resize(img, (wStretched, img.shape[0])) # stretch horizontally by factor 0.5 .. 1.5

    # create target image and copy sample image into it
    (wt, ht) = imgSize
    (h, w) = img.shape
    fx = w / wt
    fy = h / ht
    f = max(fx, fy)
    newSize = (max(min(wt, int(w / f)), 1), max(min(ht, int(h / f)), 1)) # scale according to f (result at least 1 and at most wt or ht)
    img = cv2.resize(img, newSize)
    target = np.ones([ht, wt]) * 255
    target[0:newSize[1], 0:newSize[0]] = img

    # transpose for TF
    img = cv2.transpose(target)

    # normalize
    (m, s) = cv2.meanStdDev(img)
    m = m[0][0]
    s = s[0][0]
    img = img - m
    img = img / s if s>0 else img
    return img
```

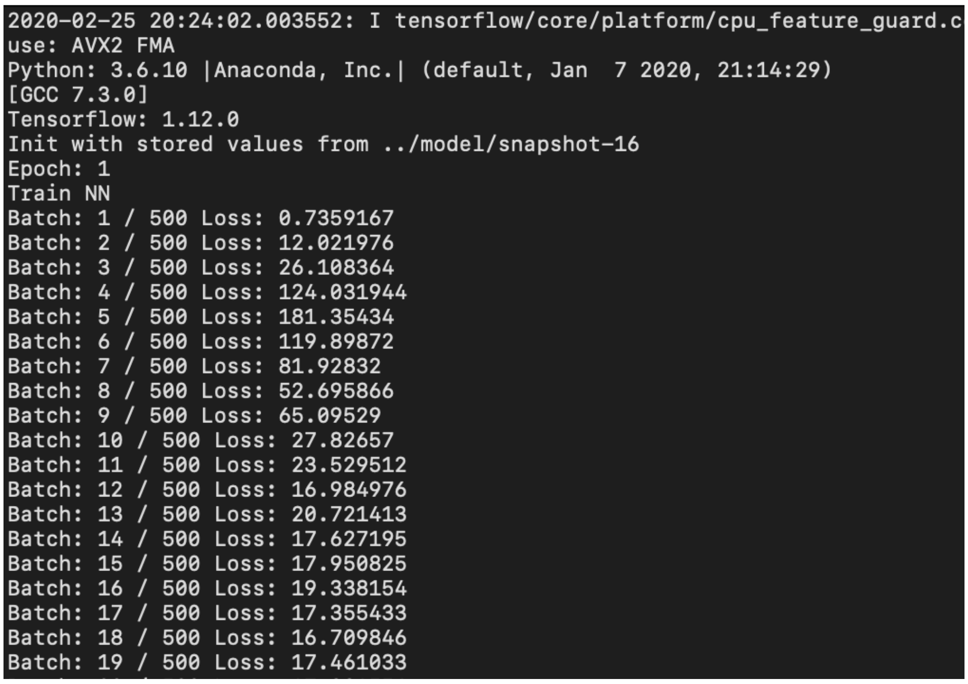
4 HOW MODEL TRAINING AND VALIDATING

Taring and Validating model processes: - I used the TU Berlin HPC Cluster

https://hpc.tu-berlin.de/doku.php?id=hpc:hardware

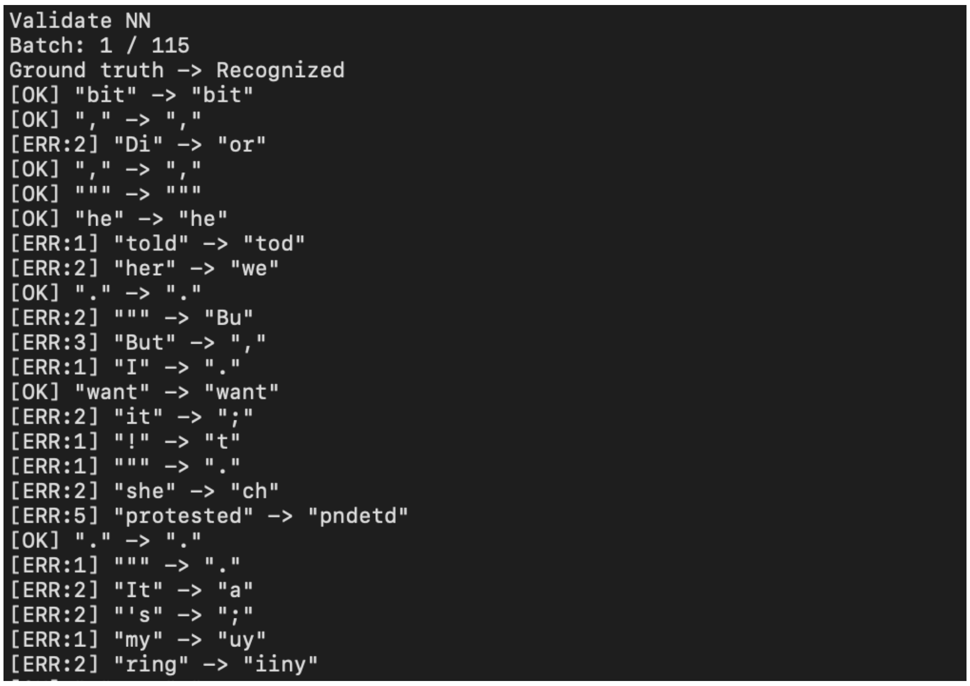
You can see the full output processes by Gitlab inside ***hpc_output.log*** ***file during this link :***

https://gitlab.tubit.tu-berlin.de/QU-Projects-ML-NLP-WS-2019-20/HandwrittenTextRecognition/tree/master



```
2020-02-25 20:24:02.003552: I tensorflow/core/platform/cpu_feature_guard.c
use: AVX2 FMA
Python: 3.6.10 |Anaconda, Inc.| (default, Jan  7 2020, 21:14:29)
[GCC 7.3.0]
Tensorflow: 1.12.0
Init with stored values from ../model/snapshot-16
Epoch: 1
Train NN
Batch: 1 / 500 Loss: 0.7359167
Batch: 2 / 500 Loss: 12.021976
Batch: 3 / 500 Loss: 26.108364
Batch: 4 / 500 Loss: 124.031944
Batch: 5 / 500 Loss: 181.35434
Batch: 6 / 500 Loss: 119.89872
Batch: 7 / 500 Loss: 81.92832
Batch: 8 / 500 Loss: 52.695866
Batch: 9 / 500 Loss: 65.09529
Batch: 10 / 500 Loss: 27.82657
Batch: 11 / 500 Loss: 23.529512
Batch: 12 / 500 Loss: 16.984976
Batch: 13 / 500 Loss: 20.721413
Batch: 14 / 500 Loss: 17.627195
Batch: 15 / 500 Loss: 17.950825
Batch: 16 / 500 Loss: 19.338154
Batch: 17 / 500 Loss: 17.355433
Batch: 18 / 500 Loss: 16.709846
Batch: 19 / 500 Loss: 17.461033
```

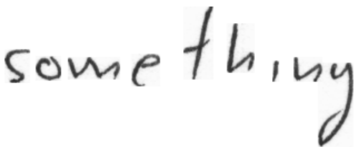
……..

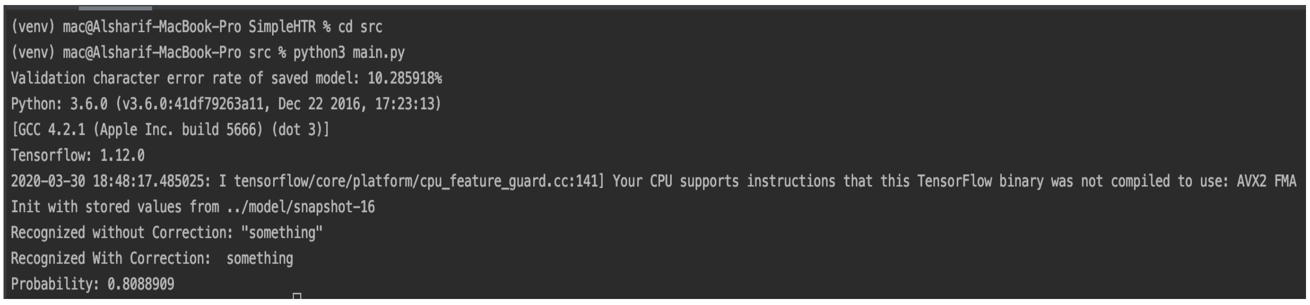


```
Validate NN
Batch: 1 / 115
Ground truth -> Recognized
[OK] "bit" -> "bit"
[OK] "," -> ","
[ERR:2] "Di" -> "or"
[OK] "," -> ","
[OK] """ -> """
[OK] "he" -> "he"
[ERR:1] "told" -> "tod"
[ERR:2] "her" -> "we"
[OK] "." -> "."
[ERR:2] """ -> "Bu"
[ERR:3] "But" -> ","
[ERR:1] "I" -> "."
[OK] "want" -> "want"
[ERR:2] "it" -> ";"
[ERR:1] "!" -> "t"
[ERR:1] """ -> "."
[ERR:2] "she" -> "ch"
[ERR:5] "protested" -> "pndetd"
[OK] "." -> "."
[ERR:1] """ -> "."
[ERR:2] "It" -> "a"
[ERR:2] "'s" -> ";"
[ERR:1] "my" -> "uy"
[ERR:2] "ring" -> "iiny"
```

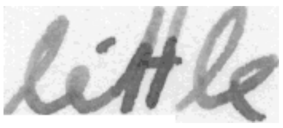
5 EXAMPLES OF SYSTEM OUTPUT

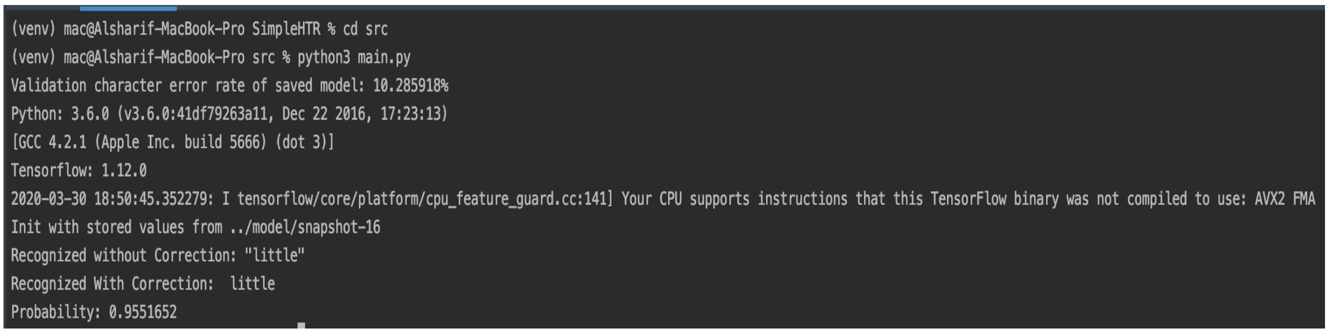
Some examples of IAM dataset with different font style and the output for each one of them:



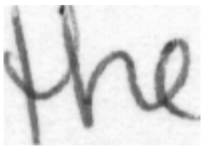


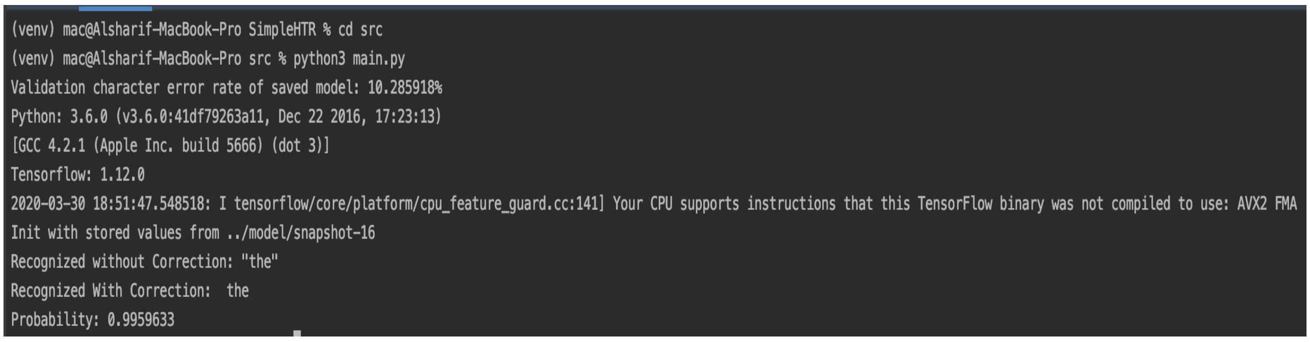
```
(venv) mac@Alsharif-MacBook-Pro SimpleHTR % cd src
(venv) mac@Alsharif-MacBook-Pro src % python3 main.py
Validation character error rate of saved model: 10.285918%
Python: 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
Tensorflow: 1.12.0
2020-03-30 18:48:17.485025: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
Init with stored values from ../model/snapshot-16
Recognized without Correction: "something"
Recognized With Correction:  something
Probability: 0.8088909
```





```
(venv) mac@Alsharif-MacBook-Pro SimpleHTR % cd src
(venv) mac@Alsharif-MacBook-Pro src % python3 main.py
Validation character error rate of saved model: 10.285918%
Python: 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
Tensorflow: 1.12.0
2020-03-30 18:50:45.352279: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
Init with stored values from ../model/snapshot-16
Recognized without Correction: "little"
Recognized With Correction:  little
Probability: 0.9551652
```





```
(venv) mac@Alsharif-MacBook-Pro SimpleHTR % cd src
(venv) mac@Alsharif-MacBook-Pro src % python3 main.py
Validation character error rate of saved model: 10.285918%
Python: 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)]
Tensorflow: 1.12.0
2020-03-30 18:51:47.548518: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
Init with stored values from ../model/snapshot-16
Recognized without Correction: "the"
Recognized With Correction:  the
Probability: 0.9959633
```

6 CONCLUSIONS

This paper has practiced machine learning techniques including the use of TensorFlow to obtain the appropriate word recognition This study built handwritten recognizers that evaluated their performances on the IAM dataset and then improved the training speed and the recognition performance.

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.

7 REFERENCES

[1] ***https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html***

[2] ***https://www.geeksforgeeks.org/numpy-in-python-set-1-introduction/***

[3] ***https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html***

[4] ***https://www.tutorialspoint.com/python3/index.htm***

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

[6] ***https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c***

[7] ***https://github.com/arthurflor23/handwritten-text-recognition***

[8] ***https://github.com/sushant097/Handwritten-Line-Text-Recognition-using-Deep-Learning-with-Tensorflow***