HACETTEPE UNIVERSITY

Department of Computer Engineering

BBM 416 – Computer Vision Project:

"Handwritten Text Recognition"

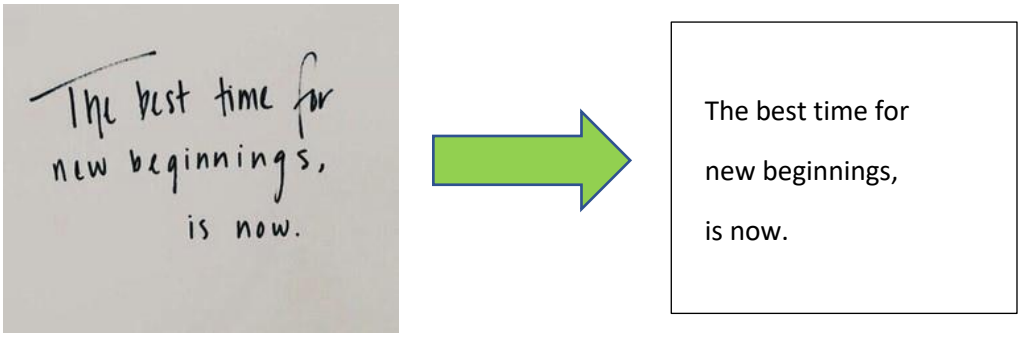
-Final Report-

By

Mehmet Ali GAVCAR - 21803366

Ozan POSTACI – 21627576

Introduction



The goal of this project is to implement a web-based program that converts handwritten texts in inputted image files into computer typed texts with digital characters. This project consists of 2 phases:

1) **Handwritten Text Recognition Phase:** The text in the input image is converted to computer typed text. This is basically a classification problem, and we need to train a model on an appropriate dataset for this phase.

2) **Web Phase:** The relevant section of the image is cropped by the user and given to as input for the converting. We need to build a pipeline between website and HTR module in this phase.

Project Setting

In **HTR Phase** of the project, we will be using **Python** for computer vision tasks like data augmentation, image preprocessing or training a model. In this phase we will use **Pillow** library for ***image processing*** operations and **PyTorch** library in the ***deep learning*** section where we train our model on **IAM handwriting dataset**.

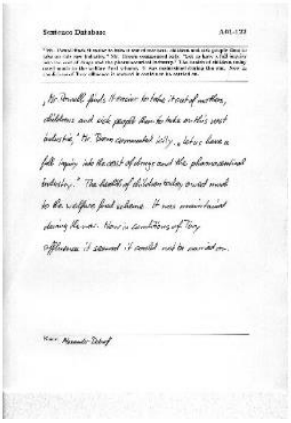
In **Web Phase** we will be using **Java** & **JavaScript** for establishing a ***website*** for taking input from client and passing it into our HTR module. At last, the output text from HTR module is returned to client.

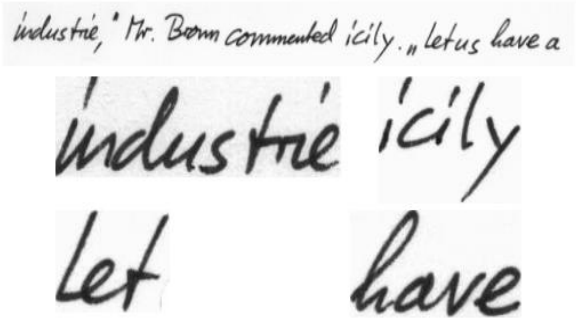
IAM Handwriting Dataset

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 IAM Handwriting Database 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





	file	ok	word
0	a01-000u-00-00.png	ok	A
1	a01-000u-00-01.png	ok	MOVE
2	a01-000u-00-02.png	ok	to
3	a01-000u-00-03.png	ok	stop
4	a01-000u-00-04.png	ok	Mr.

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.

1) Handwritten Text Recognition Phase

Base Model

Our HTR system is a composite NN of 5 CNN layers, 2 RNN layers and an CTC (Connectionist Temporal Classification) layer.

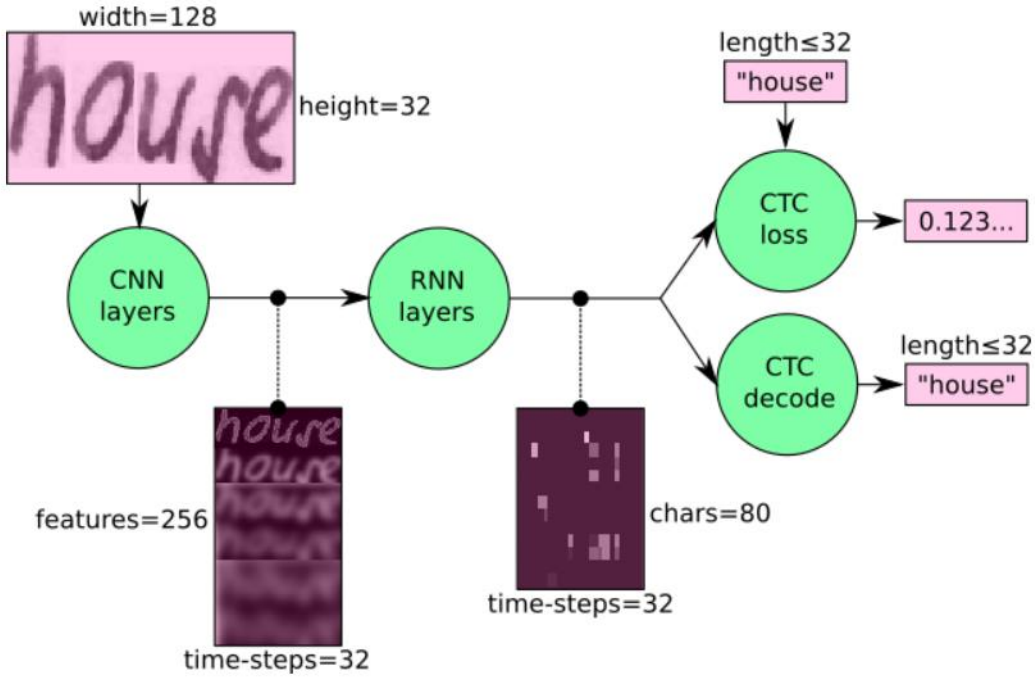


Fig. 2: Overview of the NN operations (green) and the data flow through the NN (pink).

In this network, an input image is mapped to a character sequence so that the text is recognized on **character-level**. Thanks to this, never seen test data can be recognized as well.

The IAM dataset consists of 79 different characters. With the addition of "**blank**" class needed for CTC, we have **80 character classes** in total.

NN Representation: $\underset{W \times H}{M} \rightarrow (C_1, C_2, \ldots, C_n)_{0 \leq n \leq L}$

Input: We preprocess input image and obtain a gray-value image of size 128×32 in the end. 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.

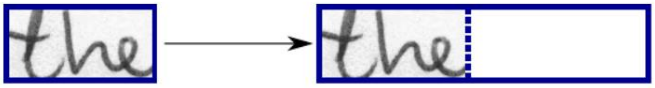


Fig. 3: Left: an image from the dataset with an arbitrary size. It is scaled to fit the target image of size 128×32, the empty part of the target image is filled with white color.

CNN: These layers are trained to extract relevant features from the image. Each layer consists of three operations.

- Convolution with 5×5 and 3×3 filters.
- RELU function is applied. (non-linear).
- Pooling layer down-samples input and summarizes image regions.

We obtain a 32x256 feature map for image as an output. This output is fed to RNN layers afterwards.

We can already see a correlation between some features and image properties at this level.

Ex: Some features have high correlation with

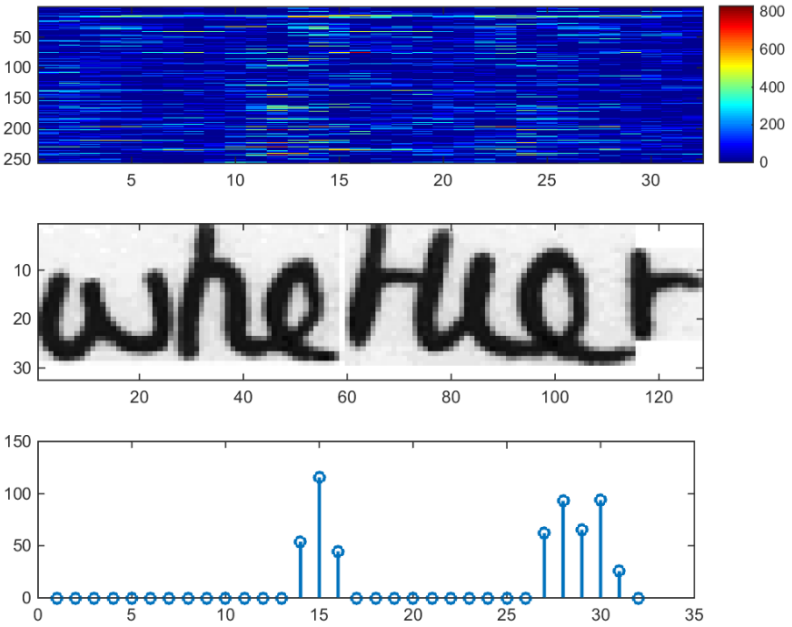


Fig. 4: Top: 256 feature per time-step are computed by the CNN layers. Middle: input image. Bottom: plot of the 32nd feature, which has a high correlation with the occurrence of the character "e" in the image.

RNN: Relevant information propagated through the feature map comes from CNN.

Long Short-Term Memory (LSTM) implementation of is used since its ability to propagate information robustly.

We obtain a 32×80 (80 Character classes) output at the end of the RNN.

This matrix contains the probabilities for each character class.

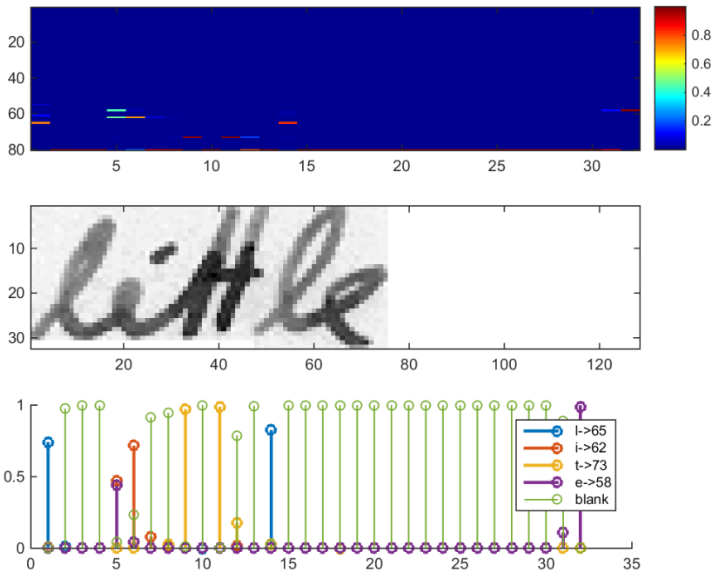


Fig. 5: Top: output matrix of the RNN layers. Middle: input image. Bottom: Probabilities for the characters "l", "i", "t", "e" and the CTC blank label.

CTC: In training, output matrix from RNN and the ground truth text is given to CTC, and it computes the **loss value**.

For classification, CTC is only given the matrix and it decodes it into the **final text**.

little

Improvements Over Base Model

- We apply the **Deslant Algorithm** on training images for data augmentation.

Algorithm: Deslanting Algorithm

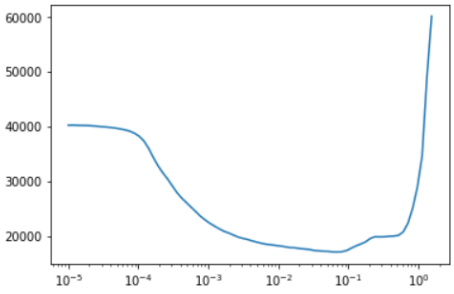
Data: image I, list of shear angles A
Result: deslanted image

$I_{bw} = threshold(I)$;
$S = \{\}$;
for $\alpha \in A$ **do**
  $I_\alpha = shear(I_{bw}, \alpha)$;
  for $i \in colums(I_\alpha)$ **do**
    h_α = number of foreground pixels in column i of I_α;
    Δy_α = distance between first and last foreground pixel in column i of I_α;
    if $h_\alpha == \Delta y_\alpha$ **then**
      $S(\alpha) \mathrel{+}= h_\alpha^2$;
    end
  end
end
$\hat{\alpha} = argmax_\alpha(S(\alpha))$;
return $shear(I, \hat{\alpha})$;

- In CNN part, initially we use a **pretrained Resnet34**, by modifying downsampling steps. Feature map size increased to 512 from 256.
- Correspondingly, RNN layers have been increased to 4 from 2 and the number of the hidden units has been increased to 512 from 256.
- **Levenshtein Distance** is used for evaluation for character error rate. The Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

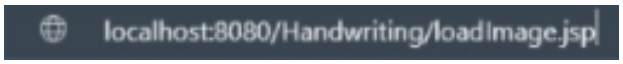
> **Ex:** The Levenshtein distance between "kitten" and "sitting" is 3, since the following 3 edits change one into the other, and there is no way to do it with fewer than 3 edits:
>
> 1. **k**itten → **s**itten (substitution of "s" for "k"),
> 2. sitt**e**n → sitt**i**n (substitution of "i" for "e"),
> 3. sittin → sittin**g** (insertion of "g" at the end).

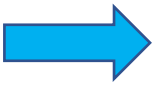
- We implement one cycle policy in training. We start with a small learning rate and increase it linearly over a few iterations, before decreasing back to the base rate again. In this way, we regularize the model and prevent it from getting stuck at local minimums when learning rate is large, while obtaining a good gradient descent loss when it is small.

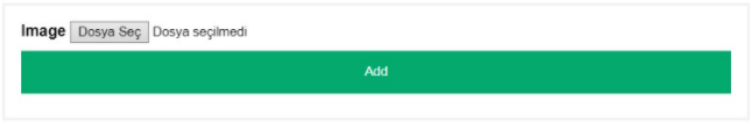


2) WEB Phase

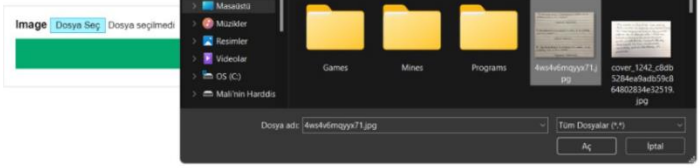
1) We run our project on local host



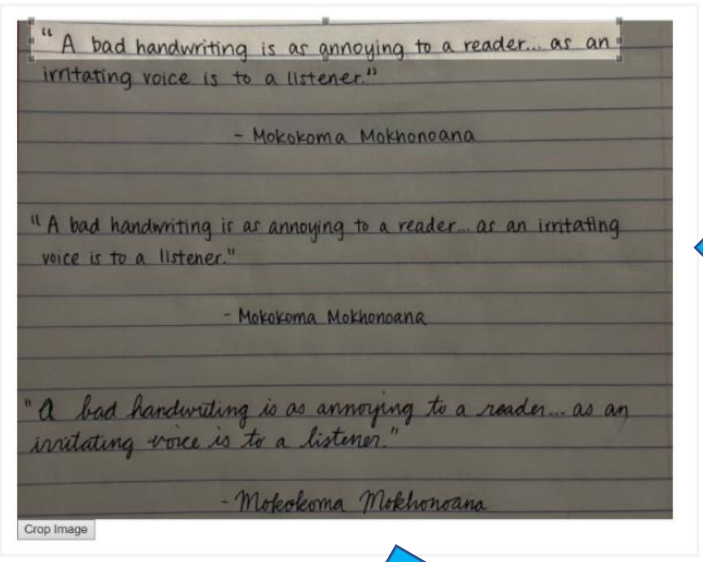


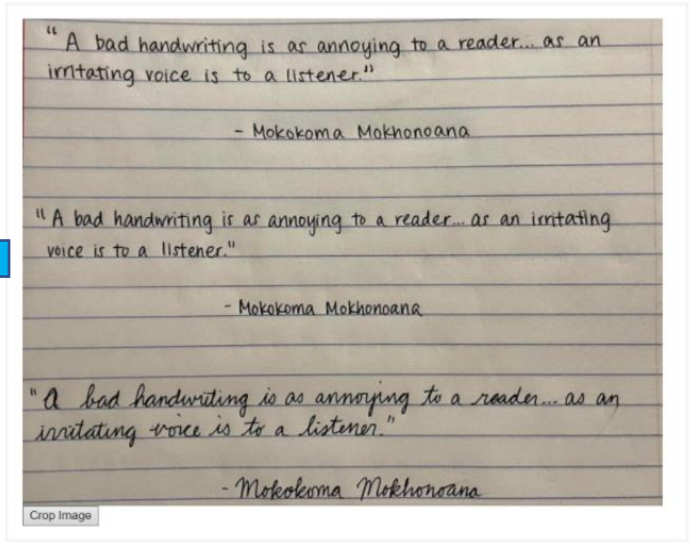


2) The client provides the input image

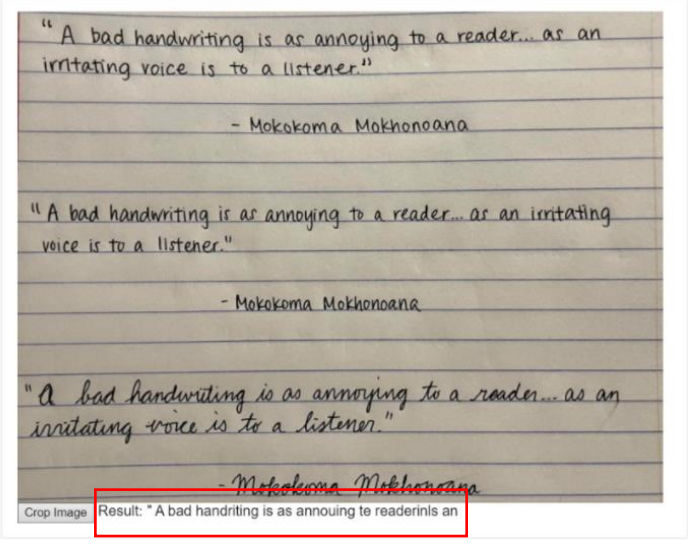


3) Client selects the part of the image to be converted to text





4) HTR Module outputs the resulting text



Results

We can make line level or word level conversions from image to text. We have different accuracy values in these different levels. Our evaluation metric was the **Levenshtein Distance** between predictions and ground truths.

On the line level:

We have a **%7.1 edit rate** between prediction and ground truths.

The overall item level accuracy is **%41** using a best path decoding of the CTC matrix.

(Complete lines with no errors)

On the word level:

We have a similar **edit rate of %7.1** between predictions and ground truths.

Character error rates (edit rates) may be similar but since there is less opportunity for alignment error, we obtain higher accuracy in overall.

The overall item level accuracy is **%86.**

(Complete words with no errors)

Average Runtime

8.9286872 seconds

A feasible duration if we think about the online converters an this can be improved with optimization on website code for speeding up the pipeline between website and HTR module.

Conclusion

This implementation of CNN (pretrained resnet34 based) + RNN + CTC model is definitely a viable solution that can be improved furthermore.

We can consider this project as a success considering its performance and accuracy. Our model can be trained even on the CPU easily.

Learning is the most important hyper parameter here; we can see the effect in the graph on improvements section.

This model is good at word conversions rather than direct line conversions.

How can result be improved?

- A word beam search decoding can be used (CTCWordBeamSearch) to constrain the output to dictionary words.
- Text correction may be applied: if the recognized word is not contained in a dictionary, search for the most similar one.

Web Site Demo Video Link

https://youtu.be/OinXA8DDj2k

References

1- https://fki.tic.heia-fr.ch/databases/iam-handwriting-database
2- https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5
This paper is the basis of this project. Modifications and additions are made to the base idea.

3- https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8
4- https://en.wikipedia.org/wiki/Levenshtein_distance#:~:text=Informally%2C%20the%20Levenshtein%20distance%20between,considered%20this%20distance%20in%201965.
5- https://github.com/githubharald/DeslantImg
6- https://arxiv.org/abs/1803.09820
7- https://stackoverflow.com/questions/10097491/call-and-receive-output-from-python-script-in-java