University of Arkansas – CSCE Department

Capstone I – Final Proposal – Fall 2021

Deep Learning Handwriting Recognition Model

William Farris, Baron Davis, Micheal Oyenekan, Creighton Young

Abstract

The problem to solve is to improve an open-source handwriting recognition model. The overall objective is to improve the already 75% accuracy to a 90% accuracy with several avenues to continue forward. As time goes on, scanning documents for information will get increasingly more important as copying or translating written information to a machine-readable state takes time and money to do. For this reason, it is important for a program to exist that scans documents for letters and words and converts them to a far more readable and easy-to-store state for computers so that chronicling information is faster for people who need to record information but cannot bring devices with them to do so.

1.0 Problem

As we continue to store more data electronically, there are still various documents that are being handwritten, such as loan applications or medical reports. Handwritten reports may not have machine-readable text that document processors can process.

This problem has led to the development of Handwritten Text Recognition (HTR), which applies Deep Learning to process handwritten documents into a machine-readable form. However, open-source HTR projects have yet to reach a level of performance to be used in enterprise applications. Because handwriting differs between each person, so does accuracy. Having a computer be able to read in handwriting regardless of writing style allows data logging individuals to save time when it comes to copying that information and reduce the likelihood of errors, such as hand-typed in typos, in the machine-readable text.

2.0 Objective

The objective of this project is to build upon and experiment with model architectures and data manipulations of an open-source HTR Deep Learning model to improve the accuracy of performance. In order to improve the accuracy of the HTR Deep Learning Model, we will work with several technologies used in the model, including the CNN layers, RELU function, replacing the optimizer with an Adam optimizer, and switching between an LSTM and a 2D LSTM, and increasing the dataset size with the IAM.

We may also try to improve the accuracy of the model by working with the input. Because each person has unique handwriting, we can work with the word beam search, de-slanting the input images, and token passing the input.

Currently we do not know what exactly will happen when we work with each of these individual aspects of the model. By experimenting with the model, we will gain more experience and understanding of each individual part, and we will then apply our knowledge of machine learning to improve the aspects of the model that will result in improved accuracy.

3.0 Background

3.1 Key Concepts

The first key technology that we will be dealing with is **CNN layers**. CNN is short for convolutional neural network. Using a CNN allows us to use nodes to assign importance, also known as **weight,** and also assigns threshold [7]. CNN is a common approach to dealing with handwritten character recognition, and will be one of our focus points while experimenting [1]. Our input image will first be fed into our CNN layers. These layers are trained to extract the relevant features from the image that we need. Every layer in the CNN will consist of 3 operations. The first operation will apply a filter kernel of 5x5 in the first two layers and a 3x3 in the last 3 layers of the input. A filter kernel is a matrix of numbers that will be multiplied times our input image in order to alter the image values. After that we will apply a nonlinear RELU, Rectified Linear Unit, function. Finally, a pooling layer summarizes image regions and outputs a downsized version of the image.

The CNN layer uses nodes known as **Neurons** that are placed in layers. The connections neurons have between each other are known as **synapses.** The neurons first receive an input signal from a source, perform calculations, then send output signals further in the CNN through the synapse.

With each synapse there is a **weight** that goes along with it. This weight represents the strengths between the neurons. If the weight from node 1 to node 2 has greater magnitude, it means that neuron 1 has greater influence over neuron 2. A weight decides how much influence the input will have on the output.

Along with the weight there is a corresponding **gradient** that goes along with it. The gradient tells how sensitive the cost is to change in its weight. If we have synapse A that has a gradient of 3.2 and a synapse B that has a weight of .1, then a change in weight of synapse A is 32x more sensitive than a change in the weight of synapse B.

After our input is run through the CNN, a nonlinear **RELU function** is applied to it. RELU, Rectified Linear Unit, is a piecewise function that will return our input if our input is greater than zero, and will return 0 if our input is less than 0. The RELU function is linear for values greater than zero which makes it great for **backpropagation**.

Backpropagation is a class of algorithms that compute the gradient of the loss function with respect to its weights. Backpropagation is how we are going to adjust the weights and biases of our neural network in order to get the desired neurons to fire. For example, if we are trying to have the letter A be recognized by our neural network, we would want to adjust our values and biases so that neurons that have a positive impact on our neural network recognizing the letter A become more active.

RELU functions are often used because it is a model that is easier to train and often has satisfactory performance [2]. We will apply this nonlinear RELU function in the hope that it will be close to linear or linearly separable. To understand **linear separability,** imagine there is a set of data points with half being blue and half being red. The data set will be linearly separable if there exists a line that can be drawn in the plane so that all of the blue points and all of the red points are on each side of the line.

The other reason a RELU function is used, as opposed to a sigmoid function, which is a function that will return values ranging from either 0.0 to 1.0., is that the RELU function does not suffer from the **vanishing gradient problem**. The vanishing gradient problem, also known as the exploding gradient problem, is a problem that arises when computing gradients during backpropagation. In a neural network in order to calculate the gradient for a node, we have to use all nodes gradients that our current node has outward synapses to. There is no issue at the beginning of, but as we go further back each node starts to have more and more connecting gradients to it, which means more and more calculations. With how large our neural network is, this problem gets worse as worse as there are tons of nodes. It gets worse and worse due to a gradient at any point, being the multiplication of all gradients at prior layers. If we were to use a sigmoid function, all of our gradients would be between 0 and 1, which would mean as backpropagation occurs, each nodes gradient gets smaller and smaller. As a result of these small numbers, our accuracy is very low, along with backpropagation would take a long time. This is why a RELU function is used, as it is not bounded by 0-1, meaning no vanishing gradient.

RNN, Recurrent Neural Network, are a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. RNNs will allow us to use previous outputs to be used as inputs, so for ours we will be using the outputs from our CNN to pass them into our RNN. One problem with RNNs is that they too can also often lead to a vanishing/exploding gradient. A special type of RNN called **Long Short-Term Memory (LSTM)** can also help to solve this issue.

An **LSTM** is a special type of RNN. LTSMs specialize in remembering information for long periods of time. They can propagate information through longer distances and provide more robust training characteristics than normal RNN. The reason why am LSTM can help with solving the vanishing gradient problem is through its use of a thing called a cell state and a forget gate. The reason for what these do is very math heavy, but basically they stop the LSTMs solve the vanishing gradient problem by preventing the gradient from going to 0 as a function of the number of samples seen thus far[16]. it Our model currently consists of two LSTM layers.

Another key technology that we will be using is a **2D-LSTM**. A 2D-LSTM is a special king of LSTM. The difference between a 2D LSTM and a regular LSTM is that a 2D LSTM takes in 2D input as its input, as opposed to the regular 1D input. A 2D LSTM can potentially will be an improvement in this project due to the fact the input will be passing in is an image, which comes in a 2D form. Although we have yet to know if this will improve the accuracy [3], it is one of the focal points of our experimentation.

After our image goes through the LSTM, that output along with the ground text is sent to a **CTC**. CTC stands for Connectionist temporal classification and is a type of neural network output and scoring function that is used for training LSTMs. The CTC function will compute the loss value of the neural network. The loss value is basically how well or how poorly our neural network is doing. The higher the loss value the worse that the neural network is doing. A perfect neural network will have a loss value of zero.

In order to train our dataset, we are going to use an input data set from **IAM**. 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 database consists of 1,539 pages of scanned text, 5,685 isolated sentences, 13,353 lines of text, and finally 115,620 words.

Error in machine learning is known as **Loss.** Loss can occur to due to a variety of reasons, including inaccurate assigned weights. This is especially the case due to how weights are imperfect and not always accurate. In order to deal with loss, we use an **optimization algorithm** to update network weights and the learning curve of the neural network, allowing for more accurate results.

Another term/concept that will be involved in this project is an **Adam optimizer**. An Adam optimizer is an optimization algorithm that can be used to iteratively update network weights based on training data. The Adam optimizer is used instead of a classical stochastic gradient descent procedure. A stochastic gradient descent will maintain the single learning rate, alpha, for all weight updates and the learning rate does not change during training. Adam instead calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages. Beta1 is the exponential decay rate for the first moment estimates, while beta2 is the exponential decay rate for the second-moment estimates. The downside of using Adam is our number of epochs will increase

For our project we may often get inputs of letters that look very similar to each other. Take for example mixing up an a for a o and an i. To fix this we are going to use a decoder with one of two algorithms. One of these two algorithms is word beam search. **Word beam search** works by having each of our inputs be in either two states, a word state or a non-word state. When in the word state we are only going to allow characters that will form words, compared to when we are in a non-word state, we are going to allow characters like " ". We can only move from a word state to a non-word state when we are finished with a word, and can move from the non-word state to the word state when we receive another character.

The other algorithm that we can use is called **token passing**. For token passing we are going to use a dictionary and a word language model. The algorithm will search for the most probable sequence of words in the dictionary in the neural network output. One problem with token passing is that it can struggle with punctuation in words and numbers.

3.2 Related Work

CAPTCHA [11] is a service that checks if the visitor is a robot by sending them a garbled message that only humans can solve while computers struggle with immensely. ReCAPTCHA on the other hand decides to take this pattern recognition ability that human brains have and put it towards digitizing words from printed media and help train AI to recognize images in a photo. Our project is meant to work on its own after a good amount of training and it's also open source in comparison to ReCAPTCHA.

Gboard [12] is a service provided by google that incorporates both predictive text and translating handwriting to text. However, it's only available on android devices and backed by Google while our handwritten recognition model will be open source and thus free to use elsewhere.

Microsoft OneNote [13] is a service that is offered by Microsoft that it not only accepts PDFs, it can also translate any written text on it into machine-readable text. This may seem like a superior program to our project, but, however, it's run by Microsoft, while ours is open source, making it more available for others to use as a basis in their own programs.

4.0 Design

4.1 Requirements and/or Use Cases and/or Design Goals

Our primary design goal is to achieve 90% accuracy from the initial 75% accuracy from the given open-source HTR Deep Learning model. We will experiment with the code from a variety of approaches to attempt to achieve this 90% accuracy.

4.2 Risks

Risk	**Risk Reduction**
Incorrect output poses a risk to the data and thus can cause issues in recording	By raising the accuracy of the output, we can reduce the risk of errors.
Unique handwriting styles or sloppy handwriting may influence the output	Working with a variety of input and working with the de-slant image code may reduce this risk

4.4

We have been provided a repo that already contains prebuilt methods and command line arguments for training and validating the model. Thus, most of our time will be spent on experimenting with different model architectures and image transformation methods rather than building the train/test pipeline from scratch. The current model performs with an accuracy of around 75% on the IAM Offline Handwritten Text Recognition (HTR) dataset. We will be improving the performance of the Open-source Handwritten Text Recognition (HTR) Deep learning mode to meet the enterprise applications performance threshold of ~90%.

For implementation and design, the model that was offered to us is an open-source code in a Github repository. This Handwritten text recognition model uses Python as the programming language, implemented with TensorFlow (TF) and trained on the IAM offline HTR dataset. The model takes images of single words or text lines (multiple words) as input and outputs the recognized text. Three quarters of the words from the validation-set are correctly recognized, and the character error rate is around 10%.

We are using an agile methodology to manage our project and the framework we are choosing is Bi-weekly Scrum. The timeline is between 16 – 20 weeks (about 4 and a half months). CGI acceptance rate for projects is 90%. So, in order to get our project to 90% we will be meeting every two weeks and assigning tasks by sprint.

SPRINT 1

•Data augmentation: As a result of limited data, we will increase dataset-size by applying further (random) transformations to the IAM input images. These changes can be flipping, translations, or rotations to our datasets. By creating these changes our neutral network can this is a different dataset. Now, only random distortions are performed.

SPRINT 2

•Remove cursive writing style in the input images (see DeslantImg). We will be using the **Deslanting Algorithm**, which can be found in an open-source repository. This algorithm sets handwritten text in images upright, i.e., it removes the cursive writing style. One can use it as a preprocessing step for handwritten text recognition.

SPRINT 3

•Increase input size (if input of Neutral Networks is large enough, complete text-lines can be used, see lamhoangtung/LineHTR).

SPRINT 4

•Add more Convolutional Neural Network. This will help with analyzing our image

SPRINT 5

•Replace **LSTM** by **2D-LSTM**.

SPRINT 6

•Replace optimizer: **Adam optimizer** improves the accuracy; however, the number of training epochs increases (see discussion).

SPRINT 7

•Decoder: As explained above in key concepts, we will be using token passing or word beam search decoding to constrain the output to dictionary words. The algorithm will search for the most probable sequence of words in the dictionary in the neural network output.

SPRINT 8

•Text correction: if the recognized word is not contained in a dictionary, search for the most similar one.

SPRINT 9

Final testing

4.5 Schedule –

Tasks	**Dates**
1. Meeting with Industry champion on what needs to be done…	10/19
2. Get familiar with repo, AI, and python	10/26-11/9
3. Data augmentation	11/16 - 11/30
4. Remove cursive writing style in the input image	11/30 - 12/14
5. Increase input size	1/11 - 1/25
6. Add more CNN layers	1/25 - 2/1
7. Replace LSTM by 2D-LSTM	2/1 - 2/15
8. Replace optimize	2/15 - 3/1
9. Decoder	3/1 - 3/15
10. Text correction	3/15 - 3/29
11. Testing	3/29 - 4/19

12. Documentation	4/19- 5/3

4.6 Deliverables

- Design Document: Contains a listing of each major software component and the resulting changes to each component
- Initial data: The starter code: SimpleHTR, DeslantImg, LineHTR, CTCWordBeamSearch
- C++ code for the resulting HTR program
- Final Report

5.0 Key Personnel

William Farris – Farris is a Senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed relevant courses.

Baron Davis is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed relevant courses.

Creighton Young is a senior Computer Science major in the Computer Science and Computer Engineering department at the University of Arkansas. He has completed relevant courses.

Micheal Oyenekan is a senior Computer Engineering major in the Computer Science and Computer Engineering department at the University of Arkansas. He has completed relevant courses

Nathaniel Zinda is a machine learning engineer at CGI and is the main contact for this project.

6.0 Facilities and Equipment

Due to the nature of the project at hand, the equipment and equipment. On the equipment end, personal computers will be necessary to use. For facilities, computer labs will occasionally be useful in the case of meetings and working together.

7.0 References

[1] D. S. Maitra, U. Bhattacharya and S. K. Parui, "CNN based common approach to handwritten character recognition of multiple scripts," 2015 13th International Conference on Document Analysis and Recognition (ICDAR), 2015, pp. 1021-1025, doi: 10.1109/ICDAR.2015.7333916.

[2] A Gentle Introduction to the Rectified Linear Unit (ReLU), https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[3] Mohammad Reza Yousefi, Mohammad Reza Soheili, Thomas M. Breuel, Didier Stricker, "A comparison of 1D and 2D LSTM architectures for the recognition of handwritten Arabic," Proc. SPIE 9402, Document Recognition and Retrieval XXII, 94020H (8 February 2015)

[4] Research Group on Computer Vision and Artificial Intelligence — Computer Vision and Artificial Intelligence. (2021). Retrieved 5 November 2021, from https://fki.tic.heia-fr.ch/databases/iam-handwriting-database

[5] Brownlee, J. (2017). Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. Retrieved 5 November 2021, from https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[6] Word Beam Search: A CTC Decoding Algorithm. (2020). Retrieved 5 November 2021, from https://towardsdatascience.com/word-beam-search-a-ctc-decoding-algorithm-b051d28f3d2e

[7] Convolutional Neural Networks, https://www.ibm.com/cloud/learn/convolutional-neural-networks

[8] Various Optimization Algorithms For Training Neural Network, https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6#:~:text=Optimizers%20are%20algorithms%20or%20methods,help%20to%20get%20results%20faster

[9] Deep Learning Neural Networks Explained in Plain English, https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/

[10] Brownlee, J. (2019). How to Fix the Vanishing Gradients Problem Using the ReLU. Retrieved 29 November 2021, from https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/

[11] Web Security Words Help Digitize Old Books, https://www.npr.org/2008/08/14/93605988/web-security-words-help-digitize-old-books

[12] Gboard, https://en.wikipedia.org/wiki/Gboard

[13] Microsoft OneNote, https://en.wikipedia.org/wiki/Microsoft_OneNote

[14] Data Augmentation, https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

[15] Build a Handwritten Text Recognition System using TensorFlow. (2021). Retrieved 30 November 2021, from https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5

[16] Recurrent Neural Networks, the Vanishing Gradient Problem, and LSTMs

Recurrent Neural Networks, the Vanishing Gradient Problem, and LSTMs. (2019). Retrieved 30 November 2021, from https://medium.com/@pranavp802/recurrent-neural-networks-the-vanishing-gradient-problem-and-lstms-3ac0ad8aff10