Digitizing Documents with a Neural Network Trained for Optical Character Recognition

Jason Albanus
ECE
jasona@vt.edu

Will Fete
ECE
willf00@vt.edu

Cameron Smith
CS
camerons03@vt.edu

1 Introduction

Handwriting recognition, the task of converting handwritten text images into machine-readable characters, has practical applications given the large number of scanned handwritten documents needing to be digitized. Although traditional optical character recognition (OCR) techniques have existed since the 1970s, their effectiveness is limited to printed text due to uniform spacing and font. On the other hand, handwritten text has a lot of challenges due to the considerable variety of handwriting, like stroke shape and size, spacing, and pressure of writing instruments [1]. The variability greatly complicates character segmentation and recognition. As a result, classic OCR is not reliable or effective for handwritten documents and requires a different approach.

Recent advancements in deep learning, specifically in convolutional neural networks (CNN), have shown better results in overcoming the limitations of OCR. No single universal CNN architecture exists as a perfect handwriting classifier. Moreover, no single CNN currently achieves high accuracy across different handwriting styles, which leaves a lot of room for further improvement.

This project aims to address the challenges of OCR by developing and comparing CNN models specifically designed for handwritten word recognition. To facilitate supervised learning, the IAM handwriting dataset from the University of Bern was chosen due to its structured pre-segmented format. The pre-segmentation of words allows for a decreased amount of pre-processing. However, there was still some pre-processing including removing corrupted or unclear images and punctuation marks to improve training efficiency and reliability.

Initially, we developed a baseline CNN model, which consisted of four convolution blocks designed to capture features from input images. However, early experiments revealed lots of limitations, including varying validation performance and poor accuracy. Motivated by these initial findings, we shifted focus and developed an improved model leveraging the ResNet18 architecture. Some of the main modifications, such as reducing convolutional steps and employing other strategies.

Comparative results clearly show the effectiveness of the second approach. The initial baseline CNN plateaued at a low accuracy of approximately 45%, indicating insufficient model complexity. Unlike the baseline model, the optimized CNN achieved training accuracy exceeding 95% and validation accuracy around 80%. Although this improved upon the first, the second model shows overfitting, showing that there needs to be further refinement.

Lastly, a simple graphical user interface (GUI) was created using Tkinter, allowing users to upload and classify handwritten images. Despite its simplicity, this front-end effectively demonstrates the potential practical application of the trained models.

2 Problem Description

The problem this project aims to solve is handwriting recognition. While millions of handwritten documents have been scanned and stored as images, converting their contents into machine-readable text offers significant advantages, such as easier search, storage, and analysis. However, manually transcribing these documents is impractical at scale.

Optical character recognition (OCR), has existed since the 1970s; it is the process of turning text from images to the respective characters. This conversion is fairly easy due to the predictable spacing and similarity of digital fonts. The problem is that traditional OCR systems struggle with handwritten text, instead handwriting recognition is required. Handwriting recognition is the process of turning handwritten text from images to the correct characters. This is significantly more difficult than OCR due to the high variance in human handwriting. The variability in shapes, styles, and writing pressure complicates the task. Unlike text images, the unpredictable spacing and styles with handwritten characters makes it more difficult to segment characters. While deep learning has produced promising results, there is no universal model that performs reliably across all handwriting styles. That universal solution is what this project aims to find, using convolutional neural networks designed and tuned to best accomplish this task.

To train a model capable of recognizing handwriting, a high-quality dataset is needed. The IAM dataset from the University of Bern was chosen for training and testing purposes as it was pre-tokenized in terms of lines, sentences and words. To prevent biased results, punctuation was removed as the images of them were blurred to the point that they were unrecognizable, even to humans.

This problem has many challenges beyond what was mentioned above. To start off, we focused on the recognition of the pre-segmented words from the dataset. This meant that the images used needed to be normalized and padded to provide a uniform dimension for the model. Since the words were pre-segmented, supervised learning can be utilized by providing the correct word transcriptions. There are many limitations to this approach though, as sentences need to be converted word by word. Additionally, limited training input meant that data augmentation techniques were needed to ensure the resulting model would be applicable with unseen data. The model will struggle to recognize words it has never seen in training due to it being trained on a limited vocabulary. Having described the challenges of handwriting recognition, we now turn to the approach taken to solve them.

3 Methodology

3.1 Loading the Dataset

When preparing the data from the IAM dataset for model training and testing, care needed to be taken with regards to the formatting decisions from the original provided file map. Some of the images in the dataset had been corrupted, possibly from the original file transfer itself, thus each image had to pass through a verification process. In addition, we found major sources of statistical noise and error from the considerable amount of punctuation marks included in the dataset. To account for this, those images were removed, and a final mapping was formed to pair each image's relative path with an integer class label. Upon dataset initialization, no transformations were specified, as that was done within the respective CNN model files as necessary. At load time, each image is converted to RGB, and the transform is applied before pairing with the respective class label. Additionally, each image is padded on the right with white pixels to ensure all items in the batch are of the same width. Once initialized, the dataset is split into training and testing subsets with data loaders configured to provide the necessary tensors needed by the networks.

3.2 Our First Model

The first convolutional neural network developed is a baseline CNN model for word classification. It takes a variable-width 3-channel RGB word image and pushes it through four convolutional blocks. The structure of each block consists of a convolutional layer, batch normalization, a rectified linear unit (ReLU), and max-pooling. Specifically, the convolution layers utilize a 3 by 3 kernel with padding set to maintain spatial dimensions. In addition, bias terms are disabled due to the batch normalization after. The batch normalization standardizes the activations, stabilizes training, and speeds up the convergence. After the ReLU enables the model to capture complex patterns, followed

by max-pooling to progressively down-sample feature maps in a 2 by 2 block. This decreases the computational need and highlights dominant features.

After each block, channel depth increases, starting from 3 channels and going to 64 in the first block. The network ends with a 512-channel feature map after the completion of the 4 blocks. The feature depth allows the model to capture many important characteristics, including basic edges and texture. The choice of four blocks was based on GPU hardware constraints. Further, after the completion of the convolution stage, we convert the spatial feature maps into the format of a single column of fully connected layers, regardless of image width and height. This is completed using adaptive average pooling. It compresses the spatial dimensions to a single value per feature channel, which creates a fixed-length representation of the image. This provides the flexibility needed for a wide variety of image sizes. After the adaptive pooling is applied and the feature vector is a one-dimensional tensor, a dropout is applied. The dropout is applied with a probability of 0.3 and serves as a way to mitigate overfitting by randomly deactivating a subset of neuron connections during training. Finally, the flattened vector is put through a single fully connected linear layer, which outputs the logits corresponding to the word classes learned.

After the model was created, we created the training function. Training involved iterative updates to the network parameters, where each epoch comprised a complete pass through the entire training dataset. Images were grouped into small batches for computational efficiency and gradient estimation. For the training, we chose the stochastic gradient descent (SGD) optimization with momentum set at 0.9. This enhanced the convergence stability and speed by smoothing the gradient directions. An initial learning rate of 0.03 was chosen after some initial testing alongside a weight decay of 0.0001. This further helps to prevent overfitting. To dynamically refine the learning rate, we implemented a ReduceLROnPlateau, which allows for convergence on an optimal solution. During training, model accuracy and loss were continuously monitored across both training and validation datasets. Each epoch ended with evaluation across both training and validation datasets, which provided a direct assessment of generalization performance.

3.2.1 Our Second Model

For the improved CNN design, we started by importing a pretrained ResNet18 model to use as a feature extractor. It leverages the ImageNet weights, but includes several modifications from the default configuration to achieve better performance for the handwriting recognition task. Whereas the first convolution's stride is typically 2 by 2 in the ResNet architecture, by reducing it to 1 by 1 we increase the model's ability to preserve fine-grained details, which is valuable given the task at hand. In addition, the first block of the fourth, and final, residual stage has its main convolution and down-sample strides reduced from 2 by 2 to 1 by 1. Our aim was to prevent both premature loss of the fine-grained spatial detail and excessive down-sampling in the feature extraction. When character distinctions can be within a few pixels, these adjustments were necessary for the task to be performed well. Following the first stage's convolution, a ReLU activation and 3 by 3 max pool are used before entering the four residual stages.

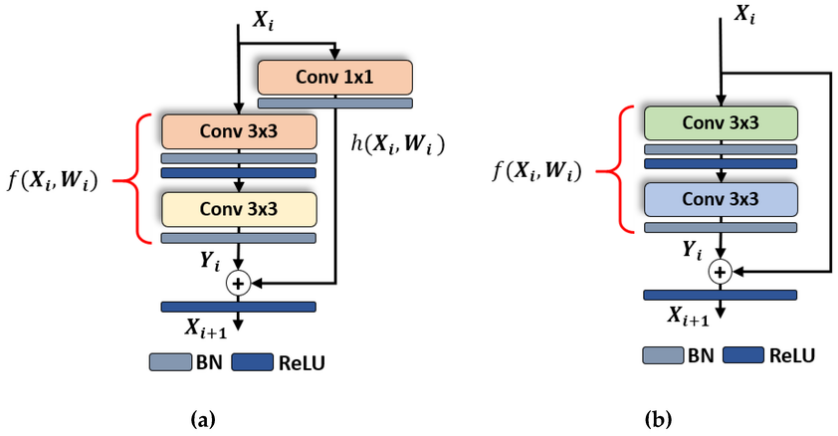

Figure 1: Residual Network Architecture (*Obtained from ResearchGate* [3]).

Within each residual stage, a 3 by 3 convolution, batch normalization, and ReLU activation sequence is applied twice. For the first residual stage, an identity skip is used (as seen in diagram (b) from Figure 1), however for the remaining three, to factor the changes for channel and spatial sizes, we used a 1 by 1 projection convolution (as seen in diagram (a) from Figure 1). The first stage maintains the input resolution and 64 input channels for it's output. The second stage doubles the channel size to 128 with stride 2 down-sampling, and the third stage repeats that doubling to 256 with the same stride 2 down-sampling. Finally, the fourth stage outputs the 512 channels without further changes to the feature map dimensions. Every convolution is followed by a batch normalization and ReLU activation, so as to distribute the nonlinearities uniformly. ReLU Activation is used after the global pooling layer to ensure the classifier receives the raw, unbounded feature map. Using an average pool after the four residual layers, the spatial dimensions are reduced from 16 by 40 (output from the fourth layer) to 1 by 1.

Building upon this, we leverage an optimized DropBlock, as opposed to the conventional Dropout layer, for our regularization prior to classification. The DropBlock converts a scalar drop probability (0.1 for our model) into a spatial gamma, sampling a binary mask, and with max pooling, expands each dropped pixel into a 5 by 5 block. Using the masked feature map, we were able to force the network to distribute information across the map to mitigate over-reliance on any of the pen stroke fragments from the handwriting samples. For every call to the forward method, the DropBlock will flatten the 1 by 1 spatial dimensions to be fed into a single linear layer for classification. The linear layer will output one logit per class, each of which is passed to a Cross Entropy loss function (with applied softmax). As a result, our optimized model is well designed for the handwriting task, keeping the fine spatial details and regularizing against overfitting, with fine-tuned adjustments for memory efficiency.

3.3 Rationale for the Changes in Architecture

The first model we developed relied on the small four block convolutional network optimized with SGD without transfer learning. Early experiments indicated to us that this model's validation accuracy climbed slowly, reaching around 40-50% after many hours of training. To stay within the computational limits imposed by the hardware at our disposal (NVIDIA 3070 TI desktop GPU), development pivoted midway through the semester to start from scratch with the second architecture. By using what we had learned from our original results, the revised model implemented several optimization techniques with the goal of achieving an earlier plateau. This had the effect of reducing the overall training time by several orders of magnitude on the same underlying hardware, allowing a higher performing model to be created that would successfully achieve the results we had sought after. While our original goal was to build a single model, after starting from scratch with the new architecture, we did not want our previous work to go unrecognized. Thus, we made the decision to re-frame our intentions towards a direct comparison between the residual neural network, and a more traditional convolutional network architecture. Even with the under-trained model, it was clear the plateau was approaching at a noticeably lower accuracy, hinting that our expectation was correct, and the major improvements from the ResNet architecture massively contributed to its success.

3.4 User Interface

For our project's front-end, we used a simple Tkinter interface design which uses a file selection dialog for the user to upload handwritten images. After an image is uploaded, we convert it to proper grayscale and pass it through the same prepossessing pipeline used by the second model. In doing so, the image is resized to the fixed height and width, with the same normalizations performed using the ImageNet constants. The pretrained model file is loaded in with the respective label mapping, specific to that model's training results. The inference is executed on said model to compute the class probabilities from the image passed in by the user. The top ten probabilities are written to console, with the most significant being displayed directly on the GUI. While this front-end was slightly more rudimentary in design than what we had originally planned, it serves its purpose as a way to adequately compare results.

4 Results

In this section the results of the models will be shown and discussed. There will be three main implementations that will be evaluated in this section, the first model, the second model built upon the ResNet architecture, and the simple front end. Starting off with the initial model, the training function captured both the accuracy and loss for the validation and training datasets. This will be the main results that will be analyzed for the first model. With the captured accuracies and losses, they can be plotted against the epochs to show the model's progression over time. These results can be seen below.

Figure 2: First Model - Plotted Accuracy.

As clearly seen in Figures 2 and 3, the performance of the first model was not great. The training accuracy was initially very low, around 20%, before gradually increasing before plateauing at epoch

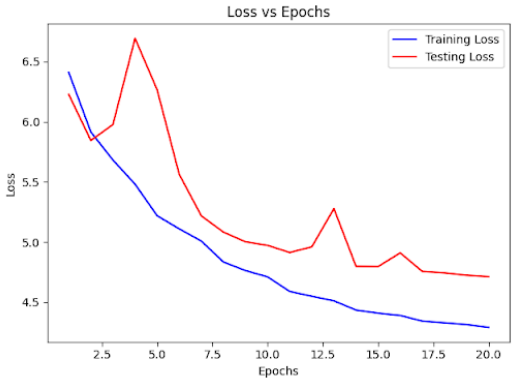

Figure 3: First Model - Plotted Loss.

15 while at 45%. The training loss was very similar to the training accuracy except mirrored. The loss started pretty high before gradually decreasing to about 4.5 and flattening out around the 15th epoch. While the training loss and accuracy left a lot of room for improvement, the testing results were a main reason for the creation of the second model. As seen in the below figures, the testing loss and accuracy was very erratic and inconsistent. This pointed to the model being undertrained and a potential error in the implementation. Besides the dips and peaks, the model achieves similar results to the training sets in the validation phase. As discussed above, this combined with other factors such as computational time and power, led to the decision to move toward the second model.

Now the results of that second model can be shown, and compared with the first model to see whether the decision to create a new model was justified. The results will be shown the same as the first model, in terms of accuracy and loss versus epochs to show the second model's progression over time. These results can be seen below.

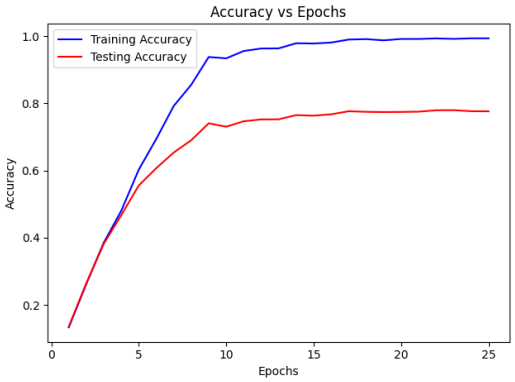

Figure 4: Second Model - Plotted Accuracy.

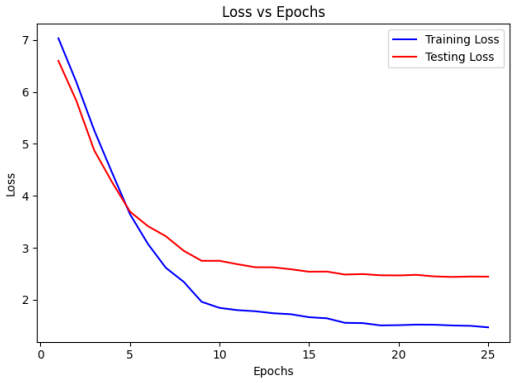

Figure 5: Second Model - Plotted Loss.

As seen in Figures 4 and 5, the results obtained are an improvement. The training accuracy starts low then rapidly increases to above 95% before flattening out. The loss is similar as it starts out around 7 before diving done to around 1. The model had the same behavior for the validation dataset, except the test accuracy was lower and the loss was higher at 80% and 3, respectively. Looking at the graphs, it is clear that this second model is overfitting, which can be due to high complexity, not enough data, and lack of enough regularization. While this second model led to better performance in terms of accuracy, loss, and consistency, the evidence of overfitting is important to recognize. With the results of both models, it is necessary to compare them to decide whether the decision to switch model architectures halfway through the project was justified.

To further test not only the second model, but the front end also, we can feed unseen data to the model through the front end. To do this, we created a small set of input data using our own handwriting of known words in the model's vocabulary. After selecting these images in the GUI, the results were promising. As seen in Figure 6 below, the model made several correct predictions, while the front end displayed the predicted word and its confidence. As you can see, it still is not perfect and limited by the training data size and architecture. The results of the front end testing were satisfactory and as expected. Overall, the results as the implementations discussed in the methodology were acceptable, with room for improvement in the future.

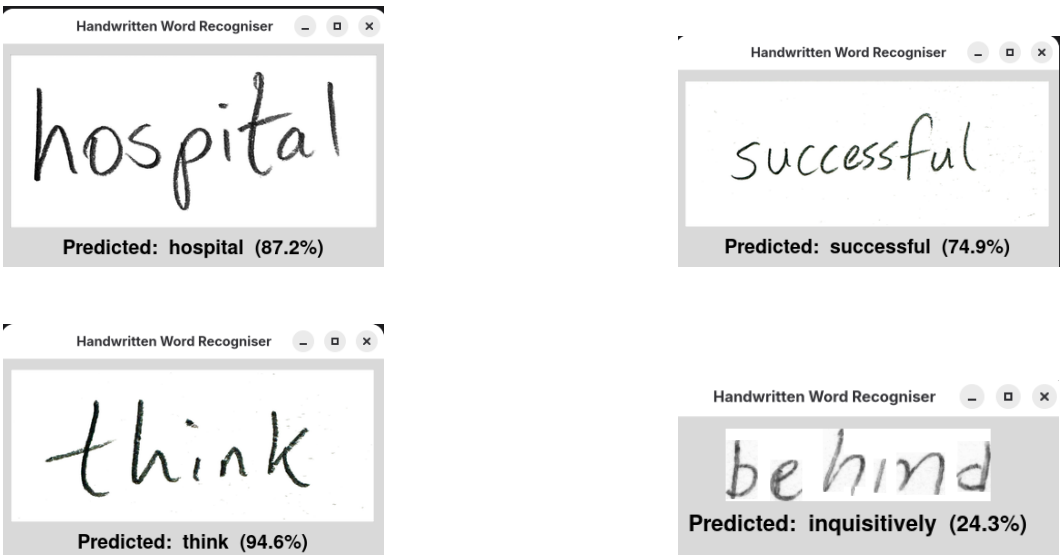

Figure 6: Four Examples from the GUI.

5 Conclusion

This work has shown us the capabilities that convolutional neural networks have at effectively addressing the many challenges imposed by handwriting recognition. There were many instances of success in addition to the setbacks, both of which provided practical exposure and insight. We were able to learn the importance of transfer learning in its ability to enhance accuracy and expedite convergence. In addition, understanding the delicate balancing act between model complexity against available computational resources, data filtering and augmentation to mitigate overfitting on a limited dataset, the effects that preserving fine-grained detail via early down-sampling had, and the improvements to generalization that were made with more advanced regularization techniques like DropBlock. Potential avenues to explore for future work include training the first model to completion on the ARC clusters provided by Virginia Tech, experimenting with attention-based architectures to handle full line and sentence recognition, expanding the dataset to broaden the model's applicability, and refining the front-end user interface for a more seamless interactive experience. Overall, this work helps to demonstrate the practicality of the methods of machine learning used for handwriting recognition tasks.

References

[1] P. C. Vashist, A. Pandey and A. Tripathi, *A Comparative Study of Handwriting Recognition Techniques,* 2020 International Conference on Computation, Automation and Knowledge Management (ICCAKM), Dubai, United Arab Emirates, 2020, pp. 456-461, doi: 10.1109/ICCAKM46823.2020.9051464

[2] Ahlawat, Savita et al. *Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN).* Sensors (Basel, Switzerland) vol. 20,12 3344. 12 Jun. 2020, doi:10.3390/s20123344

[3] *Artificial Intelligence-Based Classification of Multiple Gastrointestinal Diseases Using Endoscopy Videos for Clinical Diagnosis* - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Residual-block- of-ResNet18-with-a-1-1-convolutional-mapping-based-residual-unit-and_fig3_334301817 [accessed 23 April 2025]

[4] Retsinas, George, et al. *Best practices for a handwritten text recognition system.* International Workshop on Document Analysis Systems. Cham: Springer International Publishing, 2022.

[5] Chowdhury, A., Vig, L.: *"An efficient end-to-end neural model for handwritten text recognition"* (2018)

[6] He, Kaiming, et al. *"Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[7] Ogochukwu Patience Okechukwu, Godson Nnaeto Okechukwu, Okwuchukwu Ejike Chukwuogo, & Amanda Uloma Anyigor-Ogah. (2024). *Implementing handwritten text recognition using deep learning with TensorFlow: An MNIST dataset approach*. In World Journal of Advanced Engineering Technology and Sciences (Vol. 12, Issue 2, pp. 544–552). GSC Online Press. https://doi.org/10.30574/wjaets.2024.12.2.0320