Insert Fancy Project Title

Tammer Haddad, Joseph George

Northeastern University, Boston, 02115

December 10, 2025

Abstract

This project explores machine learning approaches to handwritten text recognition and prediction. We developed a system that takes in digital text using a pipeline that includes Tesseract OCR, Convolutional Neural Network, and Markov Model. We source training data from EMNIST and MINST datasets. Our CNN model achieves 85.90% accuracy across 62 character classes.

Introduction

Even the rise of computers, handwritten notes are still a popular method to write down information for classes, meetings, etc. However, the organization and functionality aspect of written notes is inconvenient. For example, finding notes on a previous subject can be difficult manually but it would be easier if the notes were digitized and searchable. Functionally, digitized text allows for editing and sharing of notes. Traditionally, OCR (Optical Character Recognition) has been used to detect characters through rule-based algorithms. However, with the rise of AI approaches, deep learning and convolutional neural networks have improved the accuracy of OCR software. We took inspiration from this technological advancement to create an integrated pipeline that not only transcribes handwritten notes but also predicts subsequent text, making digitization more efficient and useful.

1 Related Work

- most OCR systems use template matching, where images are compared against glyph prototypes from known fonts [5]. these worked on clean inputs but werent great with messy handwriting or strange fonts because they required the templates for every font.
- a 1998 paper made LeNet-5, a CNN that learned handwriting without manual feature creation [4]. they created the CNN structure we use today (convolution, pooling, fully connected layers), and got 99%+ on MNIST. this is what was used commercially after it came out.
- the MNIST is the standard benchmark for character recognition. Cohen extended this into EMNIST by adding uppercase and lowercase letters to bring it up to 62 classes [1].
- n gram language models predict the next word/character based on n-1 previous chars, using markov assumption that the only context that matters is the recent one. theyre good for spellcheck and short autofill.
- modern recognition combines CNNs with recurrent networks (networks that have memory), and use CTC loss for training (so that they dont need character level segmentation) [6]. Tesseract 4.0 used this and gets 95%+ accuracy on character recognition now. We are simulating this but instead of having it as one model/program we keep them modular, only linked by the runner script.

2 Related Work

- Traditional OCR systems used template matching, where character images are compared against stored glyph prototypes from known fonts [5]. These rule-based approaches worked well on clean, constrained inputs but struggled with novel fonts and handwriting because they required extensive libraries of character templates for each font they could recognize.
- LeCun et al.'s 1998 paper introduced LeNet-5, a convolutional neural network that learned to recognize handwritten digits end-to-end without manual feature engineering [4]. LeNet established the fundamental CNN structure we still use today (convolution, pooling, fully connected layers) and achieved under 1% error on MNIST. It was deployed commercially for

reading handwritten checks and inspired later architectures like AlexNet and ResNet.

- The MNIST dataset (70,000 handwritten digit images) became the standard benchmark for character recognition. Cohen et al. extended this with EMNIST in 2017, adding uppercase and lowercase letters to create a 62-class task while keeping the same 28x28 grayscale format [**?**]. Their baseline results showed 51.80% accuracy with linear classifiers and 77.57% with neural networks. Ciresan et al. achieved 88.12% using committees of deep CNNs [2], and recent work like SpinalNet has reached over 91% on balanced splits [3].

- N-gram language models predict the next word or character based on the previous n-1 elements, using the Markov assumption that only recent context matters. The math dates back to Markov's 1913 work on letter sequences, with Shannon applying it to English in 1948. These models are simple and efficient for tasks like spelling correction and text prediction, especially when trained on large corpora like Common Crawl.

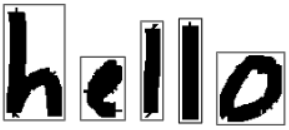
- Modern end-to-end handwriting recognition combines CNNs with recurrent networks (LSTMs) and uses CTC loss for training [6]. CTC allows the network to learn from unaligned data without character-level segmentation. Tesseract 4.0 adopted this LSTM-based approach and saw major accuracy gains over its legacy engine. State-of-the-art systems on benchmarks like IAM now achieve character error rates below 5%.

- Our approach differs by keeping a modular pipeline: Tesseract for segmentation, a custom CNN for EMNIST classification, and a separate Markov model for prediction. This lets us optimize each component independently and see what's happening at each stage.

3 Preliminaries

- OCR stands for Optical Character Recognition, which is a fancy way to say reading. we take an image of text and convert it into readable characters (text letters). we specifically are only using one to chop up images into individual characters, because we are using a CNN to read the text.

- a CNN (Convolutional Neural Network) is a type of deep learning model that is especially good at processing images. it uses layers of filters to detect patterns in images (edges shapes



and textures), then looks at patterns WITHIN the patterns to recognize things like letters and numbers. this can be seen in figure 1, where the first layers detect simple edges, and later layers detect more complex features, finalizing in the last layer with character predictions.

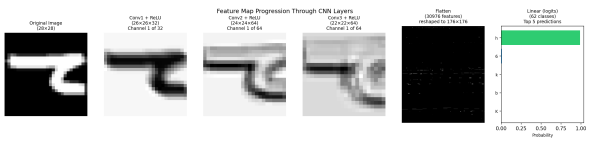


Figure 1: CNN Feature Progression Across Layers

- a Markov Model is a statistical model that predicts the next item in a sequence based only on the current item (or several current items). in our case, we use it to predict the next letter in a sequence of letters based on the previous letters (if you see "good morni__", you can guess the next letters are "ng").

- we have images of handwritten text, which we first segment into individual character images using OCR. then each image is fed into our CNN, which outputs several predicted characters. these predicted characters form a sequence that can be formed into sentences or words, which is then processed by our Markov model to predict the most likely next characters based on learned language patterns taken from the internet.

4 Data

- the MNIST dataset is a dataset of handwritten numbers 0 to 9, with 60k training samples and 10k for testing. the images are 28x28 greyscale images of single digits.

- the EMNIST dataset is the *E*xtended Mnist dataset. this includes uppercase and lowercaqse letters on top of the MNIST digits, totalling to 62 classes. it has 240k training samples and 40k testing samples, also 28x28 greyscale images of single characters (except these are transformed as well, but more on that later).

- in our the Markov model, we used the common crawl dataset, which is a massive dataset of web pages crawled from the internet. we used

this because it is nigh infinite and has a wide variety of text. we can also pull it in batches to train our model so we dont have to load the same amount every time.

- the EMNIST and MNIST datasets we just got off of torchvision datasets, and for the common crawl dataset we used the publicly available dataset on AWS S3
- For training, the EMNISt and MNIST already have training and testing splits, so we used those. for the markov model, we tested on the C4 validation set, which is a held out portion of the common crawl dataset specifrically for testing.
- in terms of preprocessing, we didnt need to do anything for MNIST and EMNIST, since theyre very well formatted. common crawl needed a little: we discared docs shorter than 100 chars, stripped whitespace from beginning and end, and joined with newline.
- we had no class imbalance issues, since we are using established and well balanced datasets.

5 Methodology

- general pipeline (input image -¿ OCR -¿ Preprocess -¿ CNN -¿ letter combination -¿ Markov -¿ output)
- We are using tesseract OCR to take an image of standard handwritten text and turn it into individual character images because our CNN is trained on single characters
- Once we have those split characters, we preprocess them by greyscaling and inverting, ressizing by adding borders (to account for non square chars) and scaling to 28x28 pixels. for MNIST we then run it, but for EMNISt we also need to transform the image because EMNIST chars are rotated 90 degrees and flipped
- Once we have those character predictions from the cnn, we need to DO something with them. for this we have the markov model we trained on the common crawl dataset
- The tesseract is used beforehand (for now) to segment the text images, but then it is brought into a folder. Once there, there is a single python file that runs the pipeline from start to finish. the CNN and Markov models are completely seperate and run individually.
- There is the possibliity of combining them by using the context given by the markov model to increase the accuracy of the CNN, but we believe it is better to implement that in the runner script rather than within either model.

6 Experiments

- Optimizer: Adam, Learning rate: 1e-3, Batch size: 32, Loss: CrossEntropyLoss
- FOR EPOCHS, We trained 50 MNIST epochs and 20 EMNIST epochs, then saved the model that had the highest test accuracy. for EMNIST this was epoch 3, and MNIST it was 40. after epoch 3-4 of EMNISt, the accuracy decreased we believe due to overfitting (since loss kept decreasing, shown in figures 2 and 3).

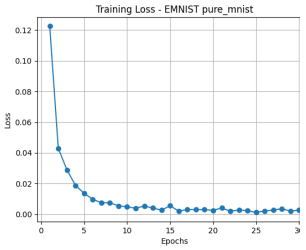


Figure 2: MNIST Model Loss

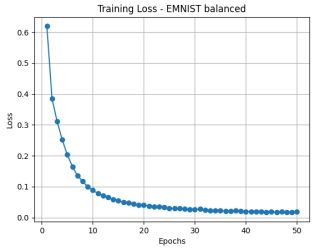


Figure 3: EMNIST Model Loss

- There were several hardware and software environments used, sometimes locally on a local machine with an NVIDIA GPU, and sometimes on a remote cluster using a V100 GPU. The code was written in Python 3.8 using PyTorch and torchvision
- Our evaluation metrics were primarily accuracy. we chose the epoch level based on highest testing accuracy. We also checked the confusion matrix as seen in figure 4 to see which characters were being confused the most.
- Using Linear classifiers and other CNNs as a baseline, we compare them to ours. The original EMNIST paper [1] gets 51.80% from a linear classifier, and 77.57% with an OPIUM Neural Network with 10k hidden neurons. Other work [2] has managed 88.12% using a committee of seven deep CNNs, and we managed 85.90% accuracy on EMNIST byclass with a single CNN.
- We have not yet combined the Markov model with the CNN for the single character prediction with context, we have them working seperately, but hope to combine them in the near future (working versions are already done but not solid).

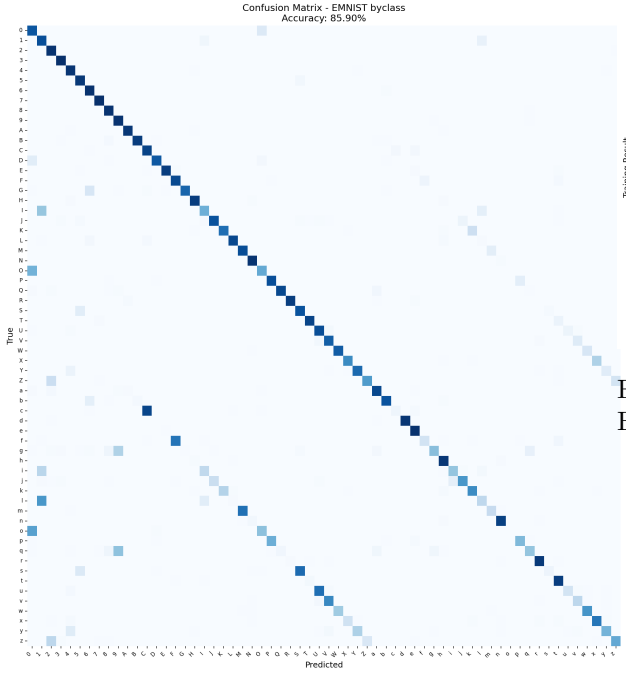


Figure 4: EMNIST Byclass Confusion Matrix

7 Results

- test loader successful: 116323 test samples Loaded model trained on 'byclass' split with 62 classes Accuracy on EMNIST byclass Test Set: 85.90Correct: 99921/116323 test loader successful: 10000 test samples Loaded model trained on 'mnist' split with 10 classes Accuracy on EMNIST mnist Test Set: 99.21Correct: 9921/10000

- Figures 5 and 6 show accuracy improvements across training epochs for both datasets.

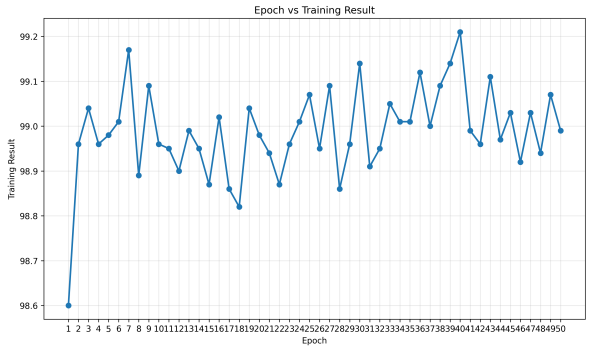


Figure 5: MNIST Model Accuracy Over Epochs

- The most common mistakes we see are lowercase characters being confused for their upper case variants (ex: 'O' and 'o', 'I' and 'l' and '1'), which makes sense, since they they often have nearly identical shapes, with size being the differing factor. in fact the lowercase letter

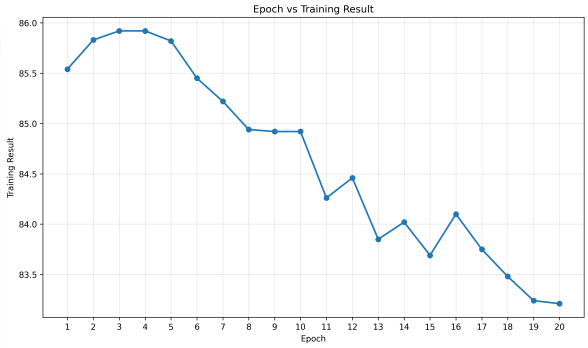
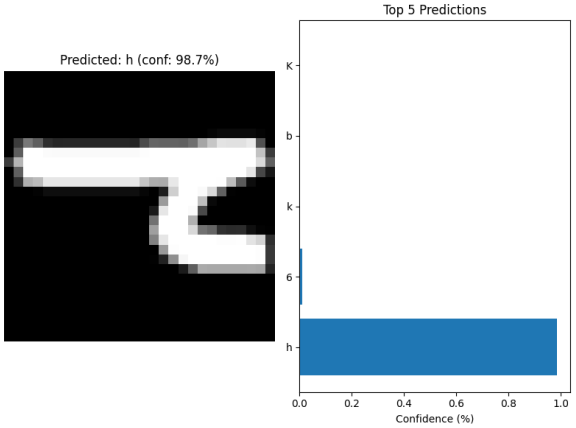


Figure 6: EMNIST Byclass Model Accuracy Over Epochs

'c' is almost always misclassified as uppercase 'C', showing that size is a major factor in classification.

- Markov model prediction accuracy: Top-1 Accuracy: 59.84%, Top-3 Accuracy: 77.03%, Top-5 Accuracy: 82.43%. The more often the charater appears the higher accuracy it predicts it. The model can predict spaces with 88.72% accuracy, 'e' with 71.87%, but 'm' for example only 40.12%.

- A good example output:



8 Discussion

- The current main limitations is that messing up a single character completely throws the markov model out of whack, since it relies on the previous characters to predict the next one. so if the CNN misclassifies a character, the markov model will be predicting based on that wrong character, leading to error hell.

- State of the art methods (transformers, larger datasets) would vastly improve performance,

and even other EMNIST models have been trained with 95.88% accuracy on the letters, and 91.05% on balanced [3], which is much higher than ours. They have obviously had a lot more time to fix theirs, so we hope to improve ours in the future as well, or we could pivot to simply using theirs.

- Future work: we hope to streamline the process by including the tesseract into the main program (within the github), and make it a web app able to take images directly from users. This could be extended to a video feed as well by taking snapshots. Using the markov model to increase the accuracy of the CNN is also well within possibility, and is already being worked on.

- There are quite a few practical applications to the mix of Markov models and CNNs for handwritten text recognition. For one thing there is the easy answer in saying that nearly anything a CNN is used for the markov could be included to improve it's accuracy. for example, license plate recognition could be improved by using a markov model to predict the next character based on state specific license plate formats. We specifically were thinking of handwriting because of note taking and digitization because we are viewing it through the lens of students, but there are swathes of other contexts in which people write things, and this would be useful for all of them.

9 Github Link & Team Contributions

You can find the complete project repository at `github.com/jgeorge123george/CS4100_final_project`.

9.1 Team Contributions

- **Tammer Haddad:** Came up with idea for project. Expanded CNN from just MNIST to full EMNIST. Created Markov model for text prediction. Combined the models to create the pipeline. Wrote majority of report.

- **Joseph George:** Created original MNIST CNN for handwritten digit recognition. Ran Tesseract OCR for character segmentation. Wrote Abstract and Introduction of report, and edited report from simplified bullet points to full sentences and formal paragraphs.

References

[1] Cohen, Gregory, et al. "EMNIST: An Extension of MNIST to Handwritten Letters." *arXiv preprint arXiv:1702.05373*, Mar. 2017. https://doi.org/10.48550/arXiv.1702.05373.

[2] Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. "Convolutional Neural Network Committees for Handwritten Character Classification." *2011 International Conference on Document Analysis and Recognition*, Beijing, China, 2011, pp. 1135–1139. https://doi.org/10.1109/ICDAR.2011.229.

[3] Kabir, H M Dipu, et al. "SpinalNet: Deep Neural Network With Gradual Input." *IEEE Transactions on Artificial Intelligence*, vol. 4, no. 5, Oct. 2023, pp. 1165–77. https://doi.org/10.1109/TAI.2022.3185179.

[4] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] Eikvil, L. "Optical Character Recognition." Norsk Regnesentral Technical Report, 1993.

[6] Shi, B., Bai, X., and Yao, C. "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 11, pp. 2298–2304, 2016.