Project Topic

Miles Fike

December 4, 2025

1 Introduction

Currently many different computer vision systems are used to interpret handwriting as digital text, but these systems are constrained usually to one style or lack the complexity to comprehend more challenging handwriting. The goal of this project is to create a series of computer vision systems that maintain accuracy while circumventing some of these constraints by having an AI redirect the given image of text to the appropriate computer vision system to accurately interpret it. By providing a single interface able to interpret and return text to users from diverse styles, a functional multipurpose AI system implementing computer vision systems has been created that can be implemented for many different purposes from a single source. This approach allows for greater adaptability and flexibility, as it enables the AI to recognize a wide variety of handwriting styles without being limited to a single computer vision system. The system is designed to evolve over time, allowing for the continual input of new computer vision systems to expand the AI models capabilities in reading and transcribing different kinds of text. In addition, by integrating multiple specialized vision modules, the AI can effectively balance precision while also ensuring that even uncommon or low quality handwriting can be interpreted with a reasonable degree of accuracy.

For the implementation of this project, a simple application was created allowing image and optional text input. The image input will take the image of the text while the optional text box would allow for a description of the writing style, skipping the optional analysis from a large AI model to detect a specific writing style. From here, the file will be run through the required computer vision system or systems to convert it to plain text by an MCP Client sending the image to the MCP server for the determined writing style. This will be returned to the user in the form of a text file that they can download and if the size permits, it will also be displayed onto the page itself.

The images chosen are diverse Latin alphabet handwriting styles including the generic modern handwriting found in the EMNIST data sets. The images of this text are converted using the PIL library to a similar size to that of the individual image files in the EMNIST database. From here, if they are a part of a larger or more popular writing style, the images will be fed into a convolutional neural network module which will perform the training and measure accuracy as it goes. If the accuracy is satisfactory, the resulting computer vision system will be made accessible to the AI implemented by MCP. Small fluctuations in accuracy do occur in each run of the program. If the text is from a smaller dataset

then the necessary labelling is performed manually and text processing is attempted with the unmonitored clustering method. This provides far less accuracy, but the system still provides an output. This involves the same image processing steps as convolutional neural networks and also may be used by the AI. The text is output on the simple user interface in the same manner regardless of whichever computer vision system the AI chose to use.

By connecting these systems through the Model Context Protocol, this project demonstrates how new AI architectures can expand beyond the limited capabilities of individual datasets and models to form expandable systems for complex computer vision tasks. Future developments may include the implementation of more complex output with potential areas of inaccuracy, such as typos analyzed and identified by a program linked to a dictionary dataset, and the continuous integration of more and more computer vision systems.

2 Related Works

Computer Vision Methods

Within the AI field of computer vision, one popular goal is the transcription of handwritten text. By different organizations, this is called handwriting recognition technology or Handwritten Text Recognition (HTR). Researchers implemented a convolutional neural network system was utilized to build a computer vision system able to recognize digits from the MNIST and achieve accuracy of up to 100% on the number 1 in the test data[**?**]. Essentially a convolutional neural network uses a combination of a single input layer then convolution layers that create feature maps, pooling layers that decrease data, and fully connected layers to create and process feature maps from images and fully connected layers that link the neurons to each of the potential final output results[**?**]. The created convolutional neural network is able to receive further input images and can assign a label to them based on what it interprets the image to be. These are very effective systems when a large, thoroughly labeled, dataset is available, but according to the research focused on obscure texts, specifically a medieval book with a Runic script known as the *Codex Runicus*, there are issues with methods like convolutional neural networks, and recurrent neural networks because they require deep learning that involves these vast labeled datasets inaccessible to some less common scripts and styles[**?**]. Alternative options that did not involve deep learning were studied and implemented to accurately transcribe segments of the *Codex Runicus*. While doing this, they used learning free methods such as unsupervised clustering which creates clusters which are subdivided, then the labels of each cluster are propagated through the other symbols, and the few-shot classification method which seeks to represent each individual symbol as a node in a graph and compare similarity between each pair of symbols[**?**]. With modern methods, it is very possible to perform highly accurate machine learning for popular fonts and styles but the pursuit of handwritten text recognition for any obscure text may result in lower accuracy systems that avoid machine learning methods. These methods may all be used to accomplish similar tasks but at different scales, when attempting to transcribe a frequently used writing style with a large, labeled database available. a convolutional neural network should always be used.

Datasets

There are several different frequently implemented datasets for handwritten text recognition computer vision. As was seen in the implementation of a high accuracy convolutional neural network, the MNIST is a dataset used in computer vision systems[**?**]. This is a popular data set used for testing computer vision systems provided by the National Institute of Standards and Technology consisting of 70,000 grayscale images of 28 by 28 pixel digits[**?**]. Because the text is labelled accurately and organized, it allows for easy implementation of computer vision systems, but to add further data to a dataset like this or use the output computer vision system to analyze another image, it would have to be processed similarly, a task that is much more complicated. On the NIST website for another dataset, the EMNIST, documentation clarifies that this new data set includes images of characters from a-z in the same grayscale 28 by 28 pixel format used by the MNIST dataset[**?**]. Each character in the dataset is labelled with its ASCII value. This makes many methods described for the other computer vision system very easily applicable to this extended version of the dataset they had used. More broadly, datasets are accessible for many written styles and languages, and complications only seem to arise when pursuing data sets for languages and styles that are rarely used.

The preparation of images for computer vision is a complex process involving simplifying images and often storing them on a gray scale. As was seen with the EMNIST images, they were represented in gray scale with gray scale pixels. Additionally, a gaussian blur smoothed the image, ROI Extraction Centered Frame, and resizing and resampling were applied to ensure that the images were the correct consistent size and had the proper shading[**?**]. The final result was images with white text with pixels blurred into a black background along the edges of the letters. In implementations of interactive image measurement software using Tkinter and PIL libraries from Python, researchers showed how tools may be used to perform the individual tasks implemented for the EMNIST database[**?**]. They explain that the Python PIL libraries may be used to perform image processing such as grayscale conversion, Gaussian blur, image sharpening, binarization and more[**?**]. The PIL package in Python could effectively be used for the preparation of images for EMNIST computer vision systems. The same methods could also be easily implemented to feed images into the output computer vision systems during their implementation.

Model Context Protocol

To connect AI models with the tools that they require to function optimally and perform all tasks that may be required of them, there are several different popular methods. One recently released was the Model Context Protocol (MCP) and according to its own website, this protocol is a standardized way to connect AI applications to other systems[**?**]. This is primarily described in the context of agentic AI applications. The Model Context Protocol simplifies the integration of algorithm models, platform tools, and data sources to an AI agents[**?**]. Using the model context protocol, AI systems would be able to access and use the computer vision systems that are created using different datasets. It would become the AI's task to select the appropriate computer vision system to analyze an image that is input into it.

3 Methodology

While seeking to implement the necessary computer vision systems to complete this project, the computer vision systems frequently used by other researchers in the field were first explored. It was discovered that among them one of the most popular was the MNIST[**?**]. From there, further research soon revealed the existence of the EMNIST which included the full set of characters required for this project. Thus, the dataset's parameters were used to create a convolutional neural network (CNN), a multi-layer architecture designed for image classification tasks. To perform the machine learning tasks involved, the Python PyTorch library was utilized. The model begins with an input of a single-image, 28 by 28 pixels. From here, kerneling with 10 filters with a 5 by 5 size and stride of 1 are applied in the first convolutional layer. This operation extracts 10 feature maps with emphasized details of the image. From here, a pooling layer halves the dimensions of each feature map reducing them to 12 by 12 squares. Next, another convolutional layer expands the feature map depth to 20 using the same kernel size and stride. From here, a second pooling layer simplifies the feature map to dimensions of 4 by 4 for each map. Finally, three linear fully connected layers map the 320 inputs to 104 neurons, then reduce that to 52, and finally bring that down to 26, fully aligned with the number of potential characters in the dataset. This method was effective. Through three separate attempts at processing the image, accuracies between 84% and 86% were attained through several attempts at running this code. This shows that the system was successfully recognizing many images, but an accuracy of 86 percent means that many words would still be very misspelled. For example, by inputting the image of the letters in the author's name, Figure:1, the following characters were received as the output: "ailksfike" This is very similar to the intended result but still imperfect. Because the intention of this project is to accurately transcribe text, the program should be expected to correctly identify all characters. The system was rewritten a few times.

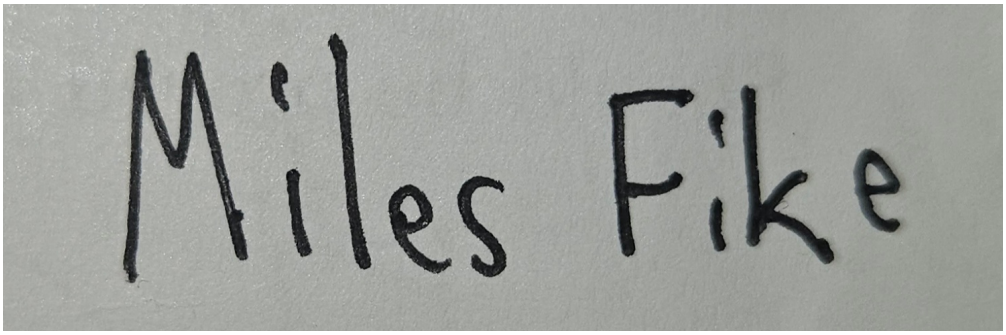

Figure 1: Miles Fike Name

Next, the program was rewritten to utilize more convolutional frames. In the second attempt, a convolutional layer with 32 output channels was used to iterate over the input layer and return 32 feature maps. From here the same pooling layer was used to halve the length and width of each individual feature map produced by the convolutional layer. The second convolutional layer output 64 different channels, and the following pooling layer compressed them. From here, 3 fully connected layers took the 1024 outputs and connected them to 256 neurons, then 256 neurons are connected to 104, and finally the 104 neurons are

connected to the 26 aligned to each image. Due to the more extensive analysis of each input image by the increased number of feature maps detecting more details, this system, which will henceforth be referred to as Model2, had consistently higher accuracy. Model2 achieved accuracy of up to 89% and no accuracy was recorded below 88

From here, another system was created starting by filtering over the input layer for 100 feature maps. This used the same pooling layers and eventually attained 200 4 by 4 feature maps. The 3200 different outputs were connected to 208 neurons, then they were connected to 104, and finally, they were connected to just 26 by the fully connected layers. This system, henceforth referred to as Model3, usually resulted in a final accuracy of 90% through every attempt at running it. Example outputs from the system using figure1 include, "rilesfike." With only one letter off, it is similar to the outputs of the Model2. The lack of significant difference and significantly longer run time for this application indicates that the number of convolutional layers in Model 2 may have been enough.

A fourth and final model was created using the same convolutional and pooling layers as Model3, but with different fully connected layers. This model used 6 separate fully connected layers to link the neurons to the feature map. The first linked the 3200 outputs to 1600 neurons, then 1600 to 800, then 800 to 410, then 410 to 208, then 208 to 104, and finally 104 to 26. This resulted in an accuracy of 89% or 90%. This shows that the fundamental issue is likely the amount of training data input into the system. Because Model3 was the optimal system, it was the system that was exported to the MCP server.

To account for a wider range of potential characters including uppercase letters, lowercase letters, and digits, a separate computer vision system was developed using the EMNIST ByClass dataset. This dataset contains 62 distinct character classes rather than the 26 used in the previous models. The architecture of this system follows a similar structure to Model2 but with adjusted output dimensions. The first convolutional layer applies 32 filters with a 5 by 5 kernel and stride of 1 to the 28 by 28 pixel input, producing feature maps of dimension $(28 - 5) + 1 = 24$, which are then reduced to 12 by 12 through max pooling with a 2 by 2 kernel. From here, a second convolutional layer expands to 64 output channels using the same kernel size, resulting in feature maps of dimension $(12 - 5) + 1 = 8$, which are further compressed to 4 by 4 through another pooling layer. The 1024 outputs ($64 \times 4 \times 4$) are then passed through three fully connected layers: the first connects to 256 neurons, the second reduces to 104 neurons, and the final layer outputs to 62 classes corresponding to each possible character. The ReLU activation function is applied after each convolutional layer to introduce non-linearity by converting negative values to zero, while the hyperbolic tangent (tanh) function is used between fully connected layers to normalize outputs between -1 and 1. All of the EMNIST-based models described above were trained for 2 epochs, as the large size of the dataset allowed the system to achieve adequate accuracy within this limited training period. Due to the significantly larger number of character classes being trained, this system had a much longer setup and training time compared to the letter-only models. However, this was the computer vision system that was ultimately implemented for the EMNIST recognition in the MCP Server, as it more accurately accounts for capitalization and can distinguish between digits and letters.

The full EMNIST was broken into training and testing sets using its own built in attributes. This will be explained further in future drafts.

Loss is the indicator of a systems assumption of the correct character. It is derived from

the system's perceived percentage likelihood that a given image may be a given character. As the system is trained, the loss changes with each image it processes. Ideally, loss should be 0 indicating that the system is operating with 100% certainty that the image input is a specific character. Model3's loss is seen below

There will be a graph for the loss of Model3 here, but this proves difficult to construct as there are so many data points. The program will be altered to get averages over a wider area before making this graph.

As the graph indicates, in the first several hundred images processed, loss decreased drastically indicating rapid development by the system. However, in the final hundreds of images loss fluctuated around 0.3 and does not as consistently decrease. This indicates that the system may require substantially more training data to improve its accuracy further.

For the preparation of images, processing first needs to be applied to each individual letter that needs to be read by the system to separate each of them from the larger image of text input. To do this, the first program sought to separate each individual image of a letter by identifying isolated blobs of white pixels within the image of text. The program would identify a pixel and try to iterate through the surrounding pixels to identify if they were also part of the letter. This, however, proved ineffective for a few reasons. First of all, the system picked up noise in the background, such as smudges on paper, and tried to label it as letters. However, that could be prevented by applying a Gaussian blur that smoothed the image before black and white thresholding converted the image to two colors. The real issue appeared with the letters "i" and "j." The dots on these letters were identified as separate letters by this method making it impossible to properly isolate the full "i" and it created an extra letter in the output. Simply looking for nearby images was not possible because often the dots of "i's" were significantly separated from the body of the letter.

Because of the flaws with this method, a separate program was prepared to read the complex. The program starts by converting the initial image into gray scale then using Opencv's threshold function to convert the image to black and white. From here a black background was created for the image and any pixel that is not absolutely black was made white by iterating through the image. From here the image is iterated through by row to separate each horizontal line of text, which are stored as separate PNG files. Next, each PNG of a line of text is iterated through by each column of pixels. If there is a pixel in that column it is marked and when there are no longer any pixels that is marked as well. These sets of two columns are used to crop each separate each image from the original picture, not the grayed version. The resulting images tend to have unusual proportions and a tall height.

A separate Python module was made to process individual images of letters and to prepare them to be read by the computer vision system. This was applied iteratively to the results of the line reading function. From here, the characters were blurred using a Gaussian blur which smoothed the colors into transitions between black and white rather than just the two colors while also blurring out some background noise to produce a smoother overall picture. From here, a threshold was set that converted all pixels with a color greater than 100 on a 100 to 255 scale to black. Next, black pixels are converted to white and white pixels are converted to black to make the image white text on a black background. The images were converted into perfects squares by adding extra rows and columns to each side until the width and length were equal. This ensured that when the size of the images was decreased, the images would not be heavily compressed or altered. The images were then converted

to 128 by 128 resulting in no apparent warping in any of the tests. Next, the images were iterated over to identify the highest, lowest, furthest left, and furthest right pixels. The image was reduced to just these pixels with two pixel padding before being padded back into a square before being further reduced once more into a 28 by 28-pixel square. To finish the image processing, each individual image was mirrored and rotated 90 degrees to match the image orientation provided by the EMNIST in their data set. The motivation behind the orientation of these images remains unclear. The computer vision system from here assigned the appropriate labels to the images. The labels are returned in order from the left to right side of the original image to give the final text output.

In addition to the EMNIST-based computer vision system, a separate system was developed using the CHoiCe (Characters of Historical Choice) dataset to demonstrate the expandability of the MCP architecture for handling alternative handwriting styles. The CHoiCe dataset is a significantly smaller collection of handwritten character images compared to the EMNIST, requiring a different approach to training. The same neural network architecture used for the EMNIST ByClass system was employed, utilizing 32 and 64 channel convolutional layers with identical pooling and fully connected layer structures outputting to 62 character classes. However, due to the limited size of the CHoiCe dataset, the training process required substantially more epochs to achieve reasonable accuracy.

The CHoiCe computer vision system was trained for 45 epochs, a dramatic increase compared to the 2 epochs used for the EMNIST models. This extended training period was necessary because smaller datasets require the model to iterate over the available data many more times to learn meaningful patterns. The dataset was split into 80% training data and 20% testing data using a random split. During training, the system used stochastic gradient descent with a learning rate of 0.01 and momentum of 0.9, with cross-entropy loss as the optimization criterion.

The accuracy results for the CHoiCe system demonstrated significant inconsistency and variability between training runs. When trained for 30 epochs, the system achieved accuracies of 61% and 65% across different runs. When the training was extended to 45 epochs, the accuracy measured at 60%, which was actually lower than some of the 30-epoch results. This inconsistency shows the challenges of training neural networks on small datasets, where random weight initialization and the stochastic nature of the training process can lead to substantially different outcomes. The variability in results suggests that the limited training data makes the model sensitive to initial conditions and may cause overfitting in some runs while underfitting in others. The first 10 output classes corresponding to digits (0-9) were eventually excluded from the CHoiCe MCP implementation due to higher confusion rates between numeric and alphabetic characters, an adjustment made in an attempt to sustain higher accuracy for letter recognition. Despite these limitations, the CHoiCe computer vision system was successfully integrated into the MCP Server, demonstrating that additional handwriting recognition systems can be incorporated into the framework regardless of their individual accuracy levels.

The MCP elements of this project were developed by allowing an MCP client to connect to each computer vision system exported as a MCP server or tool. The client registers each individual computer vision system as a tool accessible for the MCP client allowing AI to fluidly interact with each computer vision system. The image processing is built in as a function call within the running of the system, ensuring that the program runs effectively.

The LLM when prompted with an image file identifies which tool is best to interact with the image if any and sends a tool call in the form of a JSON file containing the name of the tool and its parameters. If it is correct in its assumptions, a text file is returned containing all of the words transcribed by the MCP system.

EMNIST MCP Server

The EMNIST MCP Server was developed to handle general handwriting recognition for modern handwriting styles. This server loads the trained EMNIST.pth model weights and exposes two tools to the MCP client: recognize_character and recognize_line. The recognize_character tool accepts an image of a single handwritten character and returns the predicted character along with a confidence score. The recognize_line tool accepts an image containing a full line of handwritten text, automatically segments it into individual characters, and returns the complete recognized text string along with confidence scores for each character.

The server includes the full image preprocessing pipeline built directly into its functions. When an image is input, it is first converted to grayscale and a Gaussian blur is applied to reduce background noise. From here, a threshold is applied to convert pixels to black and white, and non-black pixels are converted to white to create white text on a black background. The image is then padded into a square, resized to 128 by 128 pixels, cropped to the bounding box of the character with 2 pixel padding, and finally resized to 28 by 28 pixels to match the EMNIST format. The image is mirrored and rotated 90 degrees to match the orientation of images in the EMNIST dataset. The server can accept both local file paths and HTTP/HTTPS URLs for input images.

The neural network architecture embedded in the server uses the same structure as the EMNIST ByClass computer vision system with 32 and 64 channel convolutional layers, max pooling layers, and three fully connected layers outputting to 62 character classes. The ReLU activation function is applied after convolutional layers and the hyperbolic tangent function is used between fully connected layers. The server runs the model in evaluation mode and uses CUDA if available, otherwise defaulting to CPU processing.

CHoiCe MCP Server

The CHoiCe MCP Server was developed to handle cursive and historical handwriting styles using the CHoiCe dataset trained model. This server loads the trained CHOICE.pth model weights and exposes two tools: recognize_cursive_character and recognize_cursive_line. These tools function similarly to the EMNIST server tools but are optimized for cursive handwriting. The server explicitly informs users that it cannot recognize digits (0-9) and only recognizes letters (A-Z, a-z) due to the exclusion of digit classes from the training process.

The image preprocessing for the CHoiCe server differs slightly from the EMNIST server to better handle the characteristics of cursive handwriting. The server uses Otsu's method for automatic threshold detection which handles images with varying brightness levels more effectively than a fixed threshold. The image is converted to grayscale, binarized using Otsu's threshold, and inverted to create white text on a black background. From here, the bounding box of the character is identified and the image is cropped tightly to the character

boundaries. The cropped image is padded to a square, resized to 28 by 28 pixels, and converted to grayscale format matching the CHoiCe dataset training format.

The CHoiCe server uses the same neural network architecture as the EMNIST server with 32 and 64 channel convolutional layers and three fully connected layers outputting to 62 classes. However, when a digit class (0-9) is predicted, the server may produce unreliable results since these classes were excluded from training. The server provides confidence scores for each prediction allowing users to assess the reliability of the recognition results.

The recognize_cursive_line tool has a significant limitation due to the nature of cursive handwriting. The line segmentation algorithm relies on identifying gaps between characters by scanning for columns of pixels that contain no white pixels. This works for print handwriting where characters are naturally separated, but cursive handwriting connects letters together in a continuous flow. Because of this, the segmentation algorithm cannot properly separate conjoined cursive characters. It will often interpret an entire word as a single character or incorrectly split characters at arbitrary points where strokes happen to thin. This makes the recognize_cursive_line tool only functional for cursive text where characters have already been separated, which is not particularly useful for most real-world cursive handwriting recognition scenarios. The tool does still allow multiple pre-separated cursive characters to be processed at once if circumstances requiring such arise, but this is a narrow use case. Developing a proper cursive segmentation algorithm proved extremely difficult. Cursive connections vary significantly between writers and even between different letter combinations within the same writer's handwriting. Methods such as analyzing stroke direction changes, identifying local minima in character width, and using machine learning for segmentation were considered but not implemented due to the complexity involved.

To interact with this system, Anthropic's Claude Opus 4.5 model serves as the MCP client. Claude Opus 4.5 is a large language model capable of making tool calls through the Model Context Protocol. When given an image path and information about the handwriting style, Claude determines which MCP server tool is most appropriate to use based on the tool descriptions provided by each server. Claude then sends a tool call in the form of a JSON request containing the tool name and image path parameter. The MCP server processes the image through the appropriate computer vision system and returns the recognized text to Claude, which then presents the results to the user. This approach allows Claude to intelligently select between the EMNIST server for modern print handwriting and the CHoiCe server for cursive or historical handwriting styles based on context provided by the user.

4 Results

For the results section, a graph of the loss of each computer vision system will be shown as the project progressed. This will be shown for earlier renditions of the system as well as the final chosen system. It will be used to demonstrate the accuracy of the system. This has not been done yet because it is believed that a lower loss from the system will still be attained.

Examples of lines of text and the output from Claude's tool calls will also be shown. This unfortunately does not exist yet.

When both MCP servers were given an image of a cursive lowercase "l" shown in Figure **??**, the results demonstrated the challenges of handwriting recognition. The EMNIST server

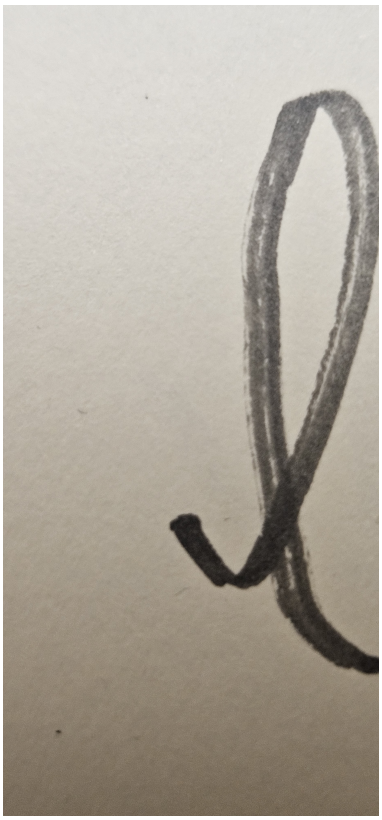

Figure 2: A cursive lowercase "l" used to test both computer vision systems.

predicted an uppercase "I" with 29.4% confidence. The CHoiCe server predicted a "D" with 96.77% confidence. The intended character was a lowercase cursive "l" written in a round cursive style. This particular "l" was fundamentally flawed as input because the round cursive shape became distorted during image processing, making it appear more circular than the typical pipe-shaped lowercase "l" that OCR systems expect. The EMNIST server's low confidence guess of uppercase "I" shows the model recognized that the input did not match any character well. The CHoiCe server's high confidence prediction of "D" represents a false positive, where the model is very certain about an incorrect answer. This is a known issue with neural networks trained on small datasets where the model can become overconfident in its predictions.

As was previously shown in the example with figure1, the handwritten name input into the system, errors in transcription are frequent and often very obvious. Additionally, spaces may not be detected by the system. Among x attempts in running our convolutional neural network Model3 on Figure1, an average of y characters were seen incorrectly transcribed. This reflects the system's overall accuracy but also shows that the system is far from perfect. Generally

The next section of the results will show how different text styles were successfully identified by the Model Context Protocol. This also does not exist yet but the MCP will be configured to return the name of the system that it used.

5 Conclusion

In conclusion, this project demonstrates the potential of integrating multiple specialized computer vision systems into a single, adaptive AI framework for handwriting recognition. By redirecting input images to the most suitable model, the system effectively overcomes the limitations of single computer vision system strategies. This design not only enhances accuracy across diverse handwriting styles but also provides a foundation for expansion as new models are developed. The result is a flexible, scalable, and intelligent handwriting recognition system capable of evolving with future as more computer vision systems are added to it.

A comparison of how each individual system compares to other computer vision systems that have already been released will also be included, along with a demonstration of how well different styles fare when input into unprepared systems versus a system built to handle diverse styles.