Convolutional Neural Network

Hyperparameter Tuning

Project Goal and Significance

The focus of this project was to explore a convolutional neural network (CNN) for computer vision, specifically looking at handwritten letters for character recognition. By gaining a deep grasp of how to use CNNs effectively and efficiently, the project in the long-term can be applied to other CNN image frameworks. Recognizing the hyperparameter relationships and tuning them to yield the best model accuracy with a reasonable processing time, can ultimately work for a conversion application that ingests handwriting images and returns digital characters. This is useful in many industrial domains including medical records, historical documents and surveillance.

Research Question

This project focuses on 8 hyperparameters of a keras CNN. We answer the question: Which of the 8 hyperparameters, in what combination, can provide the best accuracy for the model while minimizing time computer resources?

Methodology

The 8 hyperparameters divide into 2 categories, Additive and Subtractive. (*Table 1*)

Each hyperparameter tested is done so while holding the others constant, until specifically exploring their combinations. Our

Additive	Subtractive
Training Dataset Size	Pooling Layers
Number of Layers	Padding
Number of Nodes	Stride Length
Kernel Size	Dropout Layers

Table 1

process begins with the additive hyperparameters, then subtractive hyperparameters were tested on the updated model. For each test, the training data was divided into 4 equally sized sets. Each version of the model was then trained on all four training sets and evaluated on the same validation set. The accuracy and processing time were then averaged across the four models to mitigate the impact of one set having more upper or lower case letters than another.

Data Description

Data Source: https://www.kaggle.com/datasets/vaibhao/handwritten-characters

The data used for this project is a series of 32x32 greyscale images of handwritten characters. The original dataset is divided into training and validation sets and includes upper and lower case letters A—Z, as well as numerical characters and special characters such as “@” and “#”. Because this project is focused on letter characters the numerical and special character images are excluded. Each character in the training dataset has between 4,261 and 65,500 images. Each character in the validation dataset had between 119 and 392 images.

Data Preparation and Exploration

The provided dataset is labeled and clean, with all the grayscaled images centered and scaled to 32 x 32 (*Figure 1*).

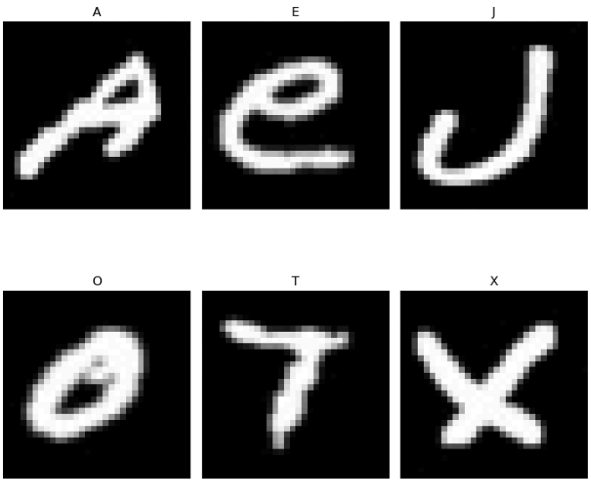

Figure 1

Exploring the data indicated changes were needed for the model to work best. While the images are centered, the quirky nature of handwriting is that the same letters are formed different; one “A” has narrower strokes, a non-uniform slant or shorter than another “A”. (Figure 2) To

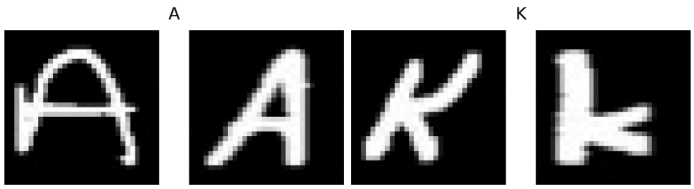

Figure 2

minimize these factors influencing the model beyond the scope of the hyperparameters, the classes were balanced by loading an equal number of images for each letter. We also changed a label from the original project—since we were not including numbers in this model, we relabeled the letter “O” to the letter, instead of “0” (zero). the largest was limited to 4,200 per character.

Solution Development

Additive Hyperparameters

Additive hyperparameters are those that when increased, increase the amount of information that a model uses. The expectation was with an increase in information would yield a higher accuracy, but also a higher processing time. For these tests, we used a simple model as a baseline: a single CNN layer architecture with 32 nodes, a kernel size of 3x3, and padding added. This layer was then flattened and fed into a softmax prediction layer. All models ran for 30 epochs with a batch size of 128.

Training Dataset Size

The first test aimed to see how adding more images would change the model’s accuracy and run time. The model was trained on a designated training dataset with the range of sizes from 25 images per character to 1025 images per character and a step size of 100.

As expected, the larger the number of images used, the higher our accuracy became and the longer it took to get results. However, there was a point where the larger numbers of images effectively stopped producing a better accuracy score. (*Figure 3*)

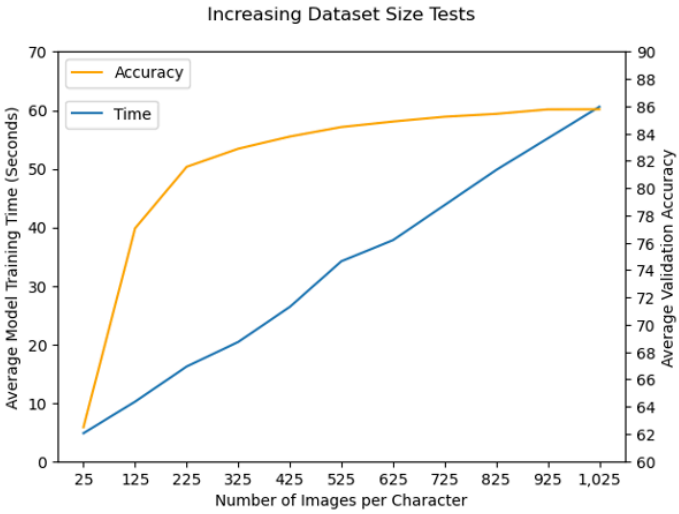

Figure 3

Increasing the data size is a good way to improve our model, but this method is only valuable if the data is introducing new features and not using images that are too similar. Our models ranged form 62.5% accuracy with a time of ~5 seconds, to 85.79% which took ~1 minute. Moving forward, we selected a data size which was large enough to a get strong accuracy with a practical return time; 500 images per letter.

Number of Layers

A common method of improving a model's predictive power is to add more layers. Each layer provides more information by performing a linear combination and non-linear activation on the observations. This helps the model differentiate between images that have similar features. In this test, layers were added to measure adding the effects of 1 2, 4 and 16 layers.

Adding more layers also increased accuracy, although the proportional increase in computation time is much higher than only adding data. The most accurate was the model that used 16 layers, with 81.62%, this is only a 0.9% improvement over the 4-layer model with 80.72%. The 16-layer test took 5 times longer to process. (*Figure 4*) We noted that the largest dataset size took the same amount of time as the 4-layer model while giving an accuracy 5% higher.

During the data size tests we found that the models were misclassifying the letter “D” and “G” in most cases. Adding more layers had different letters labeled incorrectly; they were more distributed among the alphabet and different for every layer test. This indicates that adding layers increases the model performance by learning new information, compared to the data size tests which were reinforcing what was learned at earlier tests. Still, while both showed increased accuracy, a combination was likely to produce better results than these hyperparameters alone.

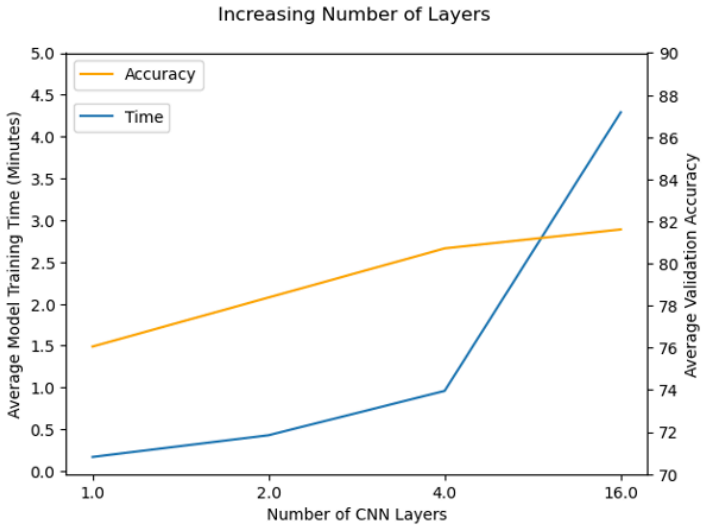

Figure 4

Number of Nodes

The number of nodes, also known as filters, represent the number of linear combinations that are performed on each observation in a given layer. The baseline model uses a standard size of 32 nodes for all layers, our next hyperparameter to test is increasing the number of nodes. Again, by increasing the amount of information the model uses the accuracy is expected to also increase—more nodes might create the opportunity to prioritize features from any layer. We tested the range of nodes from 16 to 256 (doubling the number with each step). The models increased

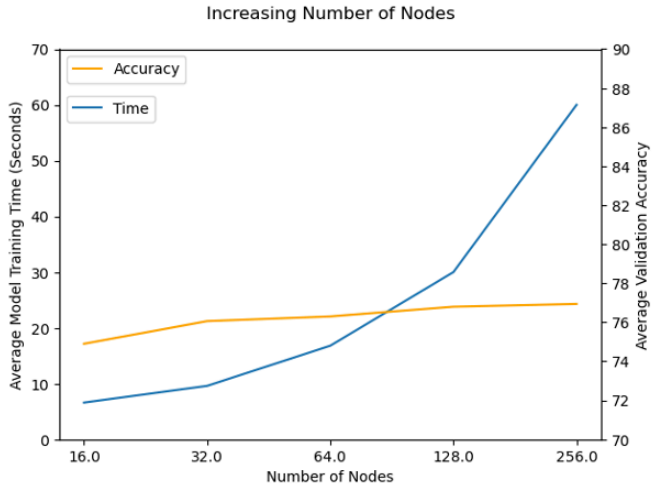

Figure 5

with run time, but showed little impact on accuracy. (*Figure 5*) This demonstrates that increasing the node size did not add value to the experiment.

Kernel Size

In a CNN, the kernel size is the coefficient matrix applied across the input matrix to weight each pixel value. By increasing the size of the kernel, each calculation uses a wider number of adjacent pixels for its output, introducing more information for the layer to use. The kernel sizes tested were 2, 3, 5 and 7. Each increase in kernel size moved the accuracy to higher scores. The 7x7 kernel gave us the best results of 80.19%. Surprisingly, the kernel size had little impact on the training time. (*figure 6*)

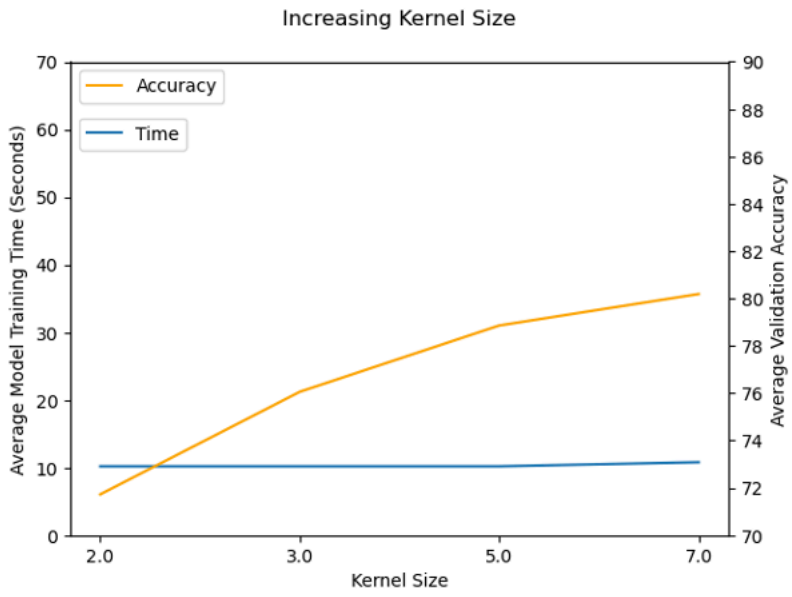

Figure 6

Node and Layer Combinations

After testing nodes alone, we tested the combination of a multilayer model and nodes to determine what is gained. The nodes were set from 32 to 64 and set in tandem with layers

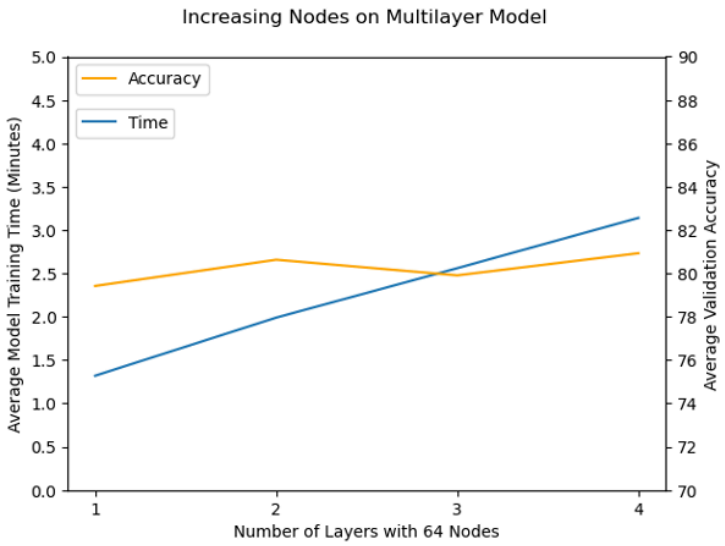

Figure 7

ranging from 1 to 4. There was no noticeable gain in accuracy, yet the run times continued to get longer. (*Figure 7*)

Given that the accuracy did not increase with a multi-layer model, we concluded that more nodes would not benefit the model. Because of the simplicity of the images adding more filters would not filter more pixel types, they would find the same features multiple times and not add new information.

Combining the Best Hyperparameters

After comparing the additive tests results, the best tuning hyperparameters settings were dataset size, the number of layers and kernel size. For thoroughness and verification, we used the best performing parameters at different settings to further the possibilities of combinations. The dataset size was increased to 225, the number of layers holding at 4, nodes at 32 and again looking at kernel sizes of 3, 5, and 7. We then used the number of layers and kernels as a 'mixed' run: layer 1—kernel 7, layer 2—kernel 5, layer 3—kernel 3, layer 4—kernel 3.

The kernel size continued to have a significant impact on the accuracy while increasing the time on this multi-layer model, confirming that, to our CNN, larger kernels mean longer run time due to more calculations performed.

Because there are only minor differences in the surrounding pixels and the data is very sparse, using a wider radius from the targeted information was valuable. Expanding the view allowed the comparison between one section to another and include nuances like a faint continuation of a line, which could mean the difference an “E” or an “F”. Still, with the new combinations we found the model with a kernel size of 7 had the best accuracy of 86.24%, but it only gained 0.8% and took twice as long to run as a kernel size of 5. (*Figure 8*)

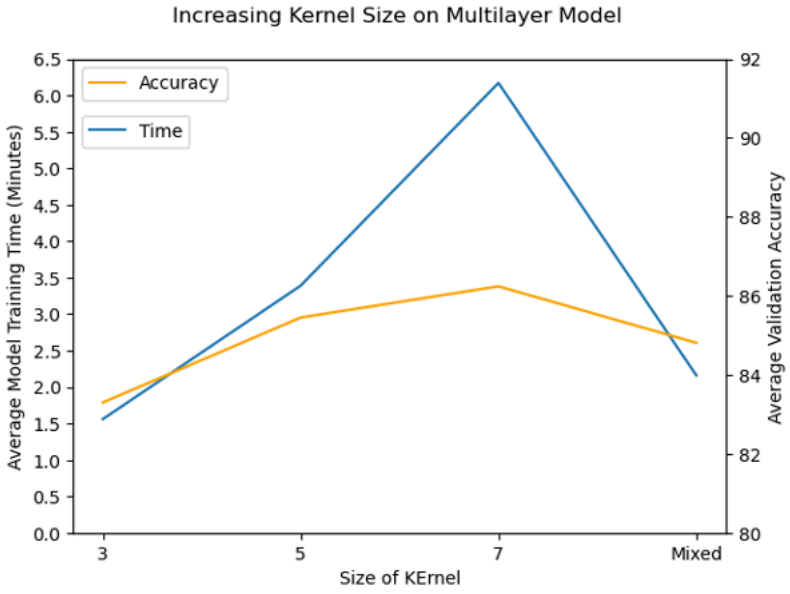

Figure 8

Additive Findings

After exploring the additive hyperparameters, the best performing model had an accuracy of 85.79%, using the simple single layer architecture at 1,025 images per letter, which took only about a minute to run. Adding only the data did not help the model perform better overall, given the trade-off in time. This model did not learn “G” or “D” to our satisfaction and unless new, substantial data to add, exploring this dataset in this way had reached its potential.

With all three of the the most impactful hyperparameters, data size, layers and kernel size, we increased the accuracy to 86.25% and a bit more flexible model except for the 6+ minutes it takes to run. Because of this flexibility, the subtractive hyperparameter tests are allowed the most opportunity to improve. We went forward with a balance of accuracy and time within the combinations, holding the additive parameters at a constant kernel—5, layer—4

and data size—225 images per character, which completes in 4 minutes and has 85.45% accuracy.

Subtractive Hyperparameters

Subtractive hyperparameters are those that when increased, decrease the amount of information that a model uses. Our expectation for these subtractive experiments was that the computational time would decrease, but with what impact on accuracy.

Pooling Layers

Pooling is a process where, after the observation passes through an activation function, the output matrix is reduced one section at a time, transitioning into an aggregated value output matrix. This decreases the spatial dimensions of the data. We experimented by adding a 2x2 MaxPooling layer between various layers in our CNN to assess their impact on the model's performance. This resulted in a better run time with high accuracy. (*Figure 9*)

Pooling benefits image classification in two main ways: First, it helps with overfitting since it removes noise from the image. Second, it helps with translation invariance—which is the models ability to recognize patterns regardless of their location or rotation in an image. By aggregating the information from neighboring regions, pooling facilitates this generalization.

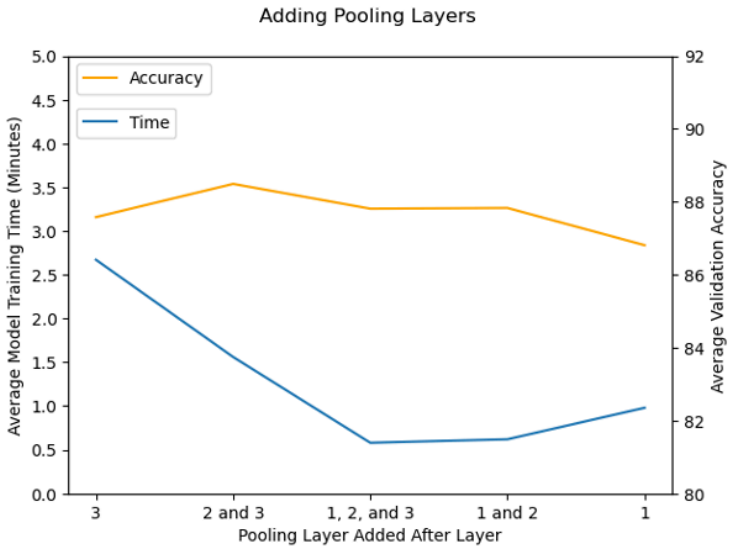

Figure 9

Padding

When a kernel is applied to a matrix, if the data is not augmented, the edges will be lost —the output matrix will be slightly smaller than the original image. Padding refers to adding zeros around the edges and so preserving the image size. This also will avoid introducing bias towards the image centers. To this point padding was included with all of the models. We iteratively removed the padding one layer at a time from both the beginning and the end of the network. The accuracy results were varied and so were there in the computational time. (Figure 10)

The increase in accuracy is likely a result of the model eliminating noise that was causing overfitting, but since the accuracy does not show consistent improvement, this hyperparameter needs to be explored further to find its total influence.

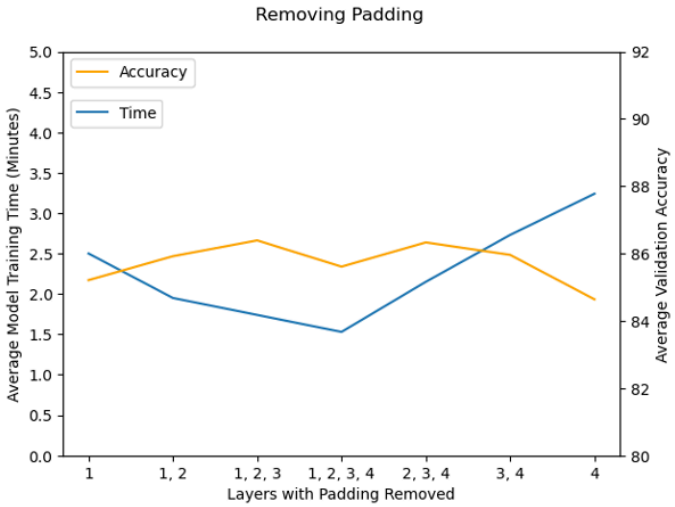

Figure 10

Stride Length

Stride length is the distance that the kernel moves every time it shifts along the input matrix. By the keras default it moves a distance of 1 column horizontally, and 1 row vertically (1,1); this is the setting that we have used to this point. Increasing the stride will skip information based on the parameter

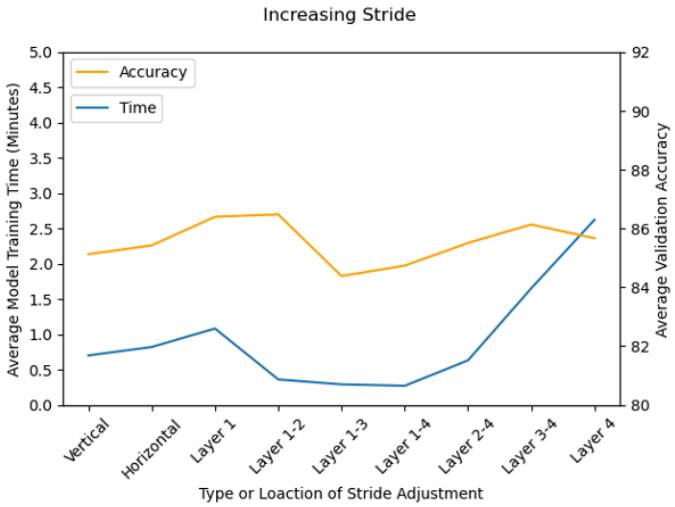

Figure 11

change. We tested (2,1), (1,2) and (2,2). This reduced the computation time in the tests through layers 1-4, but layers 2-4, 3-4 and 4 only, showed a steep rise in time. We concluded that this parameter would also need further testing to find its total influence. (*Figure 11*)

Dropout Layers

Dropout layers randomly select a specified percentage of the outputs from the previous layer and sets them equal to 0. The goal of adding dropout layers is to prevent overfitting by training the model on a more randomized dataset. The dropout rate can be set for any percentage; between 10% and 50% is typical, we used 20%.

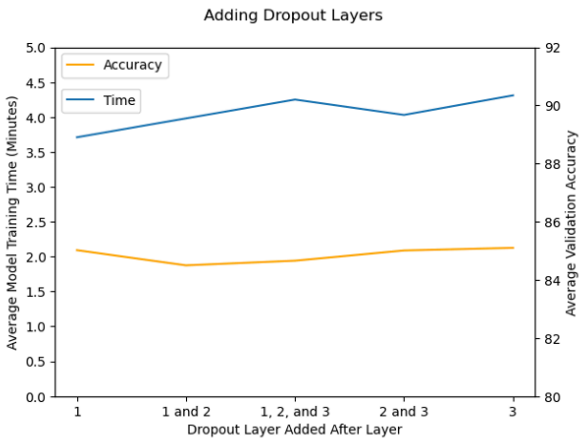

Figure 12

We introduced dropout after the CNN layers in multiple combinations. The dropout tests were not useful to progress our model. The accuracy change was insignificant on every iteration and the additional step of altering the data increased the computation time.

Subtractive Findings

One challenge with subtractive hyperparameters is that combining them can result in an output that is too small to be of use. Because of this, and how well the model performed well by pooling alone, we decided to continue with the CNN tuning with just the pooling as our subtractive hyperparameter set after the 2nd and 3rd layers.

Combined Subtractive and Additive Hyperparameters

Introducing pooling to our model in the subtractive tests, cut our run time by more than half while giving us a significant boost in accuracy. Because we gained so much speed, we decided to do one more test to see if we could increase our two best additive hyperparameters, data size and number of layers, to improve their accuracy scores while keeping the time low, with the pooling method.

	Hyperparameter	Accuracy	Loss	Avg Time	scaled_time	scaled_acc	New Label
0	Data = 4200	91.49	0.741356	6.75	1.000000	1.000000	1,050 Images\n4 Layers
1	Layers = 8	87.98	0.883790	1.95	0.000000	0.000000	225 Images\n8 Layers
2	Layers = 8, Data = 3200	90.72	0.661991	5.99	0.841667	0.780822	800 Images\n8 Layers

Table 2

We tested the full dataset of 1,050 images per character. We increased the layers to from 4 to 8. We tested 8 layers with less data at 3200. (*Table 2*) Of these three tests, increasing the layers alone was the only that did not yield a better accuracy. The addition of the 800/per image data points at 8 layers gave us a significant improvement, increasing the accuracy to 90.72%, and further to 91.49% when only increasing the data. All three tests had increased run times, and the data-only increase took over 6.75 minutes to finish. (*Figure 13*)

Considering the compound effects of computational complexity that we have introduced to this CNN, a run time under 7 minutes shows the effectiveness of pooling on the time. The tests for the bulk amount of data took about an hour before we tuned hyperparameters. (*Table 3*)

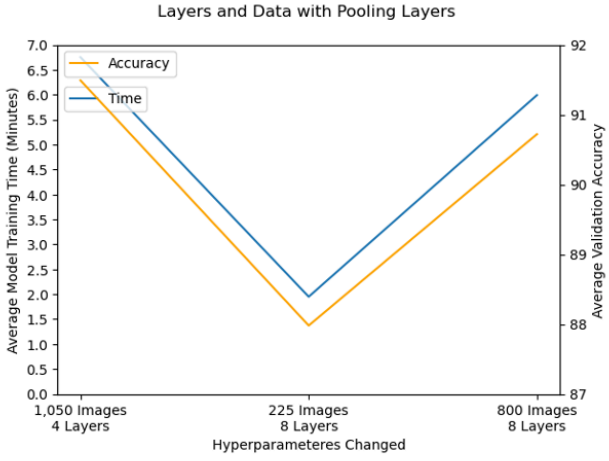

Figure 13

We noted that by adding more data

with 4 layers, the model did better at recognizing the letter “D” although it did still often misclassify “G” and “Q”.

Beginning EDA Phase

	Number Images	Accuracy	Loss	Avg Time	scaled_time	scaled_acc	Num Img / 4
0	100.0	62.50	2.091600	4.8	0.000000	0.000000	25
1	500.0	77.07	1.268644	10.2	0.096774	0.625516	125
2	900.0	81.57	1.173174	16.2	0.204301	0.818745	225
3	1300.0	82.90	1.220078	20.4	0.279570	0.876135	325
4	1700.0	83.79	1.164347	26.4	0.387097	0.914121	425
5	2100.0	84.49	1.221258	34.2	0.526882	0.944261	525
6	2500.0	84.89	1.314335	37.8	0.591398	0.961602	625
7	2900.0	85.24	1.338991	43.8	0.698925	0.976466	725
8	3300.0	85.45	1.385273	49.8	0.806452	0.985549	825
9	3700.0	85.78	1.443730	55.2	0.903226	0.999587	925
10	4100.0	85.79	1.457841	60.6	1.000000	1.000000	1,025

Table 3

Conclusion

While there are many more combinations for these hyperparameters, and there are other CNN functionalities that were not a part of this exploration, we gained valuable insight into how to manage and improve a convolutional neural net for handwriting recognition.

We successfully identified combinations of data that best helps the model learn different classes of images, layers to improve the ability to recognize detailed differences, kernel size to get a broader scope of analysis and pooling to counteract overfitting and introduce translation invariance. While improvements can still be made to this model, a 92% accuracy score from only 4 hyperparameters, shows what a significant impact they have on the model’s capabilities.

These results are strong enough to advance to the next step in handwriting analysis, which is to be able to distinguish letters in words. This is usually achieved by a bounding boxes method for printed handwriting; cursive is yet another hurdle.

Inspiration Sources

https://www.kaggle.com/code/aman10kr/offline-handwritten-text-ocr/notebook

https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fca63a

https://deepai.org/machine-learning-glossary-and-terms/padding

https://www.tensorflow.org/api_docs/python/tf
https://www.tensorflow.org/api_docs/python/tf/keras/

https://www.tensorflow.org/guide/migrate/early_stopping

https://inside-machinelearning.com/en/bounding-boxes-python-function/

http://alvarestech.com/temp/deep/Deep%20Learning%20by%20Ian%20Goodfellow,%20Yoshua%20Bengio,%20Aaron%20Courville%20(z-lib.org).pdf

https://medium.com/geekculture/10-hyperparameters-to-keep-an-eye-on-for-your-lstm-model-and-other-tips-f0ff5b63fcd4