Optical Character Recognition

Converting Handwritten Equations to LaTex via Trained CNN Models

ME 8813 – Machine Learning / AI for Mechanical Engineers

Spring 2023 Final Project – Report 2

Ryan Grajewski

Introduction

The purpose of this project is to develop a robust Optical Character Recognition (OCR) algorithm that is capable of interpreting handwritten mathematical equations and converting them to LaTex format – a popular document preparation language. At its core, OCR is a technology that aims to automatically characterize and convert image data or scanned documents to machine-readable text. Pattern recognition and feature extraction are thus the central machine learning principles that are leveraged to make sense of unclassified image data in any OCR implementation.

The applications for such algorithms are valuable in a range of industries where high volumes of image data need to be processed in order to make intelligent decisions. OCR's earliest applications included industrial uses for identifying engraved symbols on metallic bars and also for historical preservation of documents in digital form[1]. Beyond industrial applications, OCR algorithms can be used to decrease the manual burden that comes with transcribing handwritten notes or equations to digital form. Especially in an academic context, OCR can enable researchers to convert pages of handwritten calculations to LaTex without ever needing to type a single character. This application is precisely the motivation for this project.

Project Objective

The goal for this project can be categorized into two principal functionalities: character classification and LaTex conversion. The first objective in this process involves capturing and classifying individual handwritten symbols in an input image. Once the process is in place for individual symbols, the algorithm will be expanded to handle full equations comprising numerous symbols. This approach will entail training the OCR algorithm so that it is capable of parsing through said equations and identifying each unique symbol accurately. A convenient way to generate sample equations is by concatenating images of singular characters together rather than loading in genuine handwritten samples and requiring additional software to calculate each feature's bounding box. The dataset[2] that will be used to train and test the devised algorithm supplies 82 distinct character classes (symbol names) and a total of 375,974 different handwritten character images distributed among those 82 classes. For illustration, a plot of image distribution among each character class is shown in Appendix A.

Now, because character classification and, more broadly, pattern recognition is a fundamental challenge that faces both machine vision and machine learning disciplines,

numerous researchers have already proposed viable methods that can be used to classify patterns in images. Such methods range from simple, offline schemes to more intelligent and adaptive strategies. Rather than reinventing an already widely implemented pattern recognition scheme, the supervised learning method that was selected for exploration in this project is the convolution neural network (CNN). CNNs are a form of deep learning method which adaptively learns the key attributes of a desired output by optimizing neuron weight functions during the training process. The reason for electing a CNN architecture is that, in comparison to other deep learning methods, neural networks are particularly well-equipped to handle the complex data structures that are inherent in image data. Along with CNNs, a more in-dept exploration of some alternative OCR methods is presented in the next section.

To address the second functional goal of this project – text-to-LaTex conversion – custom python modules were developed which can interpret a plain-text equation that is outputted from the CNN and match each symbol with its corresponding LaTex representation. This process involves receiving a 1D vector string of all the symbol predictions that make up a given equation, converting each symbol string to its corresponding LaTex format (for example, $\sum \rightarrow \$\backslash sum(\quad)\$$), and outputting an updated version of the concatenated vector string. While this is an equally interesting aspect of the project, the majority of the machine learning principles discussed throughout this semester pertain more directly to the character recognition aspect of the project, and thus the CNN will be a larger focus of this paper.

Methodology

A Brief Presentation of Alternative Approaches:

The alternative approaches for character recognition algorithms that have been developed in recent decades include[2]:

Kernel-Based Methods

- Support Vector Machines (SVM)
- Kernel Fisher Discriminant Analysis (KFDA)
- Kernel Principal Component Analysis (KPCA)

Statistical Methods

- Hidden Markov Models (HMM)
- *K* Nearest Neighbor (*K*NN)
- Decision Trees (DT)

Template Matching Methods

- Deformable Template Matching

All of these methods implement a combination of machine learning principles with image processing techniques. In particular, Support Vector Machines were previously the preferred

method of character classification prior to the development of neural network computing. In this method, a kernel function (1) is applied to map image feature vectors into higher dimensional subspaces in order to find the hyperplane that captures the image features most distinctly[1].

$$f(x) = sng(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b) \qquad (1)$$

The constraint with kernel methods is, of course, the selection of kernel; poor kernel selection can limit the ability of the model to adapt to the given dataset in the way one may desire. *K* Nearest Neighbors is a non-parametric statistical method that is a second viable alternative approach. Based on the value of a target function, *K*NN models compute the number of input data points which are closest to the target value, and subsequently infers the output class value[3]. This is a probabilistic approach in that, for each iteration, the probability that an unknown data point (or sample) "x" belongs to a certain class "Q" is computed:

$$p(Q|x) = \frac{\sum_{k \epsilon K} W_k \cdot 1_{(k_x = x)}}{\sum_{k \epsilon K} W_k} \qquad (2)$$

Where

$$W_k = \frac{1}{d(k,x)} \qquad (3)$$

Non-parametric statistical approaches like *K*NN are advantageous for the given problem because they demonstrate more flexibility in terms of learning data attributes. However, with computation time and complexity as a concern, these statistical models often fall short because of their tendency to increase in complexity as the size of the input data grows[1]. With image data, the volume of datapoints is inherently high.

The simplest of all the approaches investigated is template matching. In this approach, the input image is compared with a known template or feature mask in order to conclude whether the image satisfies a certain class. There exist both rigid shape and deformable shape template techniques, of which the deformable template matching method would likely be most suited for this project. Since different authors could potentially have very different handwriting, the number of representations of the same character are limitless, and the likelihood of a rigid template capturing every rendition is miniscule (for example, think about using the symbol "A" as a template to capture the lowercase version "a" – they are the same letter but their form-factors are nothing alike!)

Presentation of Selected Approach:

Prior to any discussion pertaining to the chosen CNN architecture, it is important to note that image pre-processing was required to prep each input image array for the neural network. Each raw image in the selected dataset comes with a uniform size of 45x45 pixels in .jpeg format. To be a valid input for the designed CNN, each image was further reduced to a size of 32x32 pixels. Also, a common practice for pre-processing image datasets used for character recognition is to convert each RGB image to grayscale, color threshold the image to maximize contrast between character strokes and its background, and then normalize each pixel value to be between 0 and 1. The following figures depict sample images for each character class contained in the selected dataset; Figure 1 shows both unprocessed and pre-processing images.

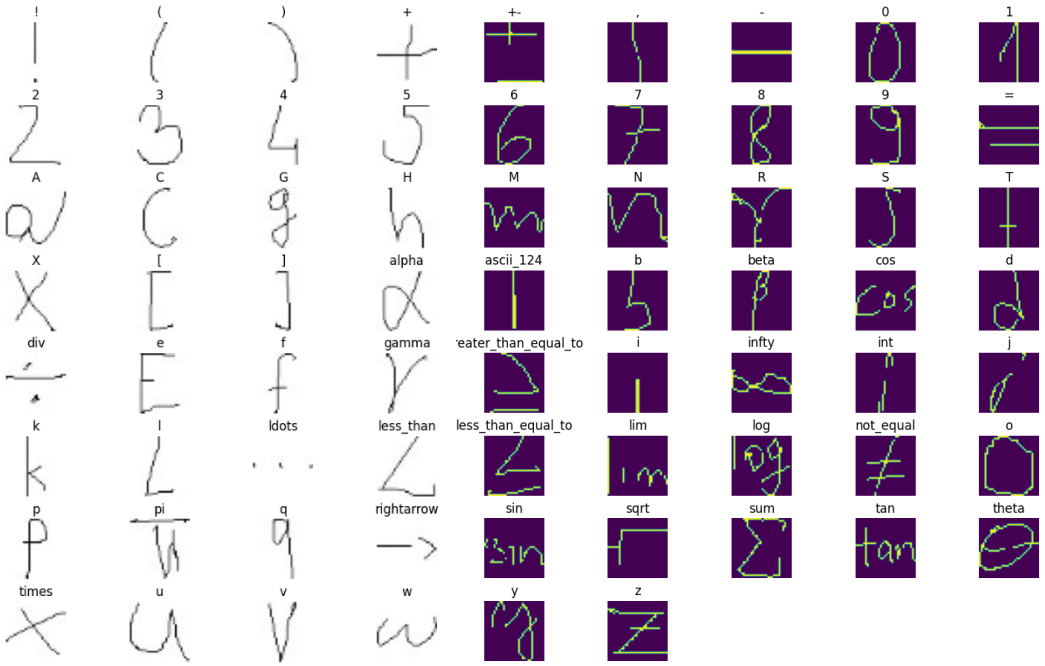

Figure 1: Depiction of sample images that belong to each class of the selected dataset. On the left(white background) are unprocessed images and on the right (purple background) are pre-processed images. Note: together, Figure 1 shows the total set of character classes.

The selected approach for character classification is an implementation of a convolution neural network. The structure of the neural network is the most defining aspect of this method, and the choice of number of activation, hidden, and output layers allows for a large degree of freedom in representing the characterization problem. A general depiction of how a neural network operates is shown below:

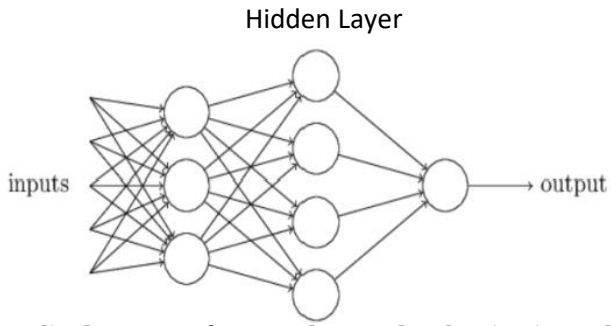

Figure 3: Generalized structure for neural networks, showing input, hidden, and output layers

The number of input layers corresponds to the size of the input data you are passing through the CNN, and after image pre-processing, the value for this implementation is 32x32x1. Hidden layers are used to adaptively process the signals passed from the input layers (or previous hidden layers) by activating, updating, or collecting a signal and passing it to the next layer if a specific parameter criteria is met. The CNN architecture implemented in this project is the classic Lenet-5 structure. This architecture has been demonstrated to be particularly well-posed for processing image datasets and is one of the earliest proposed methods having been published in 1998[5]. An illustration of this CNN architecture is shown below:

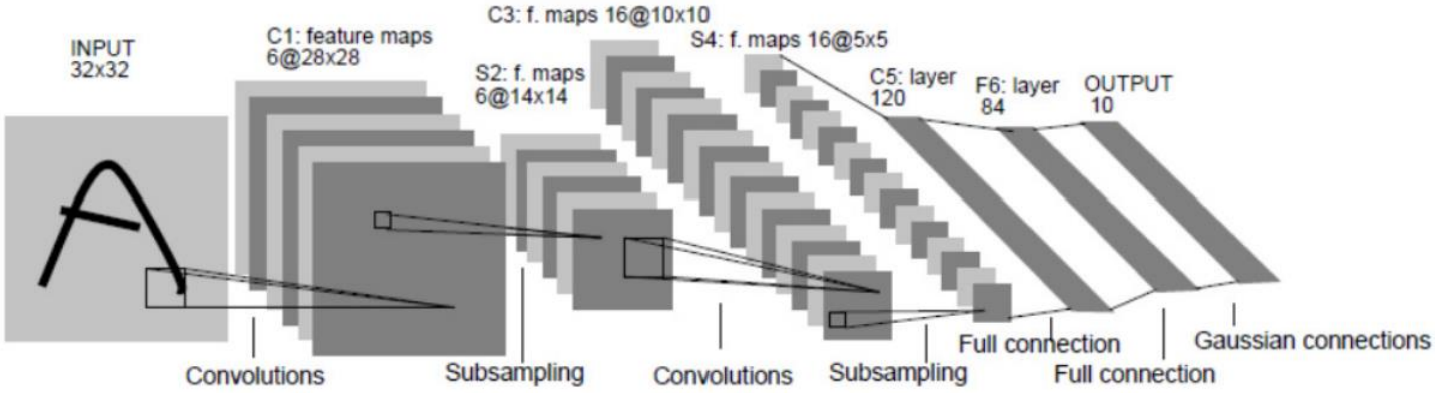

Figure 4: Illustration of LeNet-5 CNN Architecture[5]

The Lenet-5 architecture makes use of a sequence of convolution, sub-sampling, and fully-connected layers. First, a full input image (pre-processed to be size 32x32x1 pixels) is passed to the CNN and undergoes a convolution operation. Each convolution operation applies several filters over the input sample which acts to extract the most important features by minimizing spatial redundancy in the image[6]. The activation function proposed for the Lenet-5 architecture is the tanh function, however for the CNN model used in this project, ReLu was implemented. After each convolution layer, a Max-Pooling layer is implemented to accomplish the subsampling function that helps to prevent overfitting and reduces the size of the feature map. Each Max-Pooling operation reduces the feature map size by half. A flattening layer and

three dense layers are implemented after the series of convolution-subsampling layers to accomplish the "full connection" layers shown in Figure 4. These final layers act to fully reduce the channel size of the feature map back to 1 and output the desired number of feature map attributes – in our case, 82. On the final layer, a softmax activation function is used instead of ReLu in order to compute the probabilities of the current symbol image pertaining to each possible class. The Adam optimization method was elected as the optimizer for the CNN and multiclass cross-entropy was used as the loss metric during model training.

With the CNN architecture established, the next task in our methodology is to develop a scheme for converting sample equations to LaTex. Before a conversion module could be drafted, it was decided that the sample equations used for validation would be built from images in the test dataset. This was a convenient choice to make because all images in both the test and train dataset splits are of a uniform 32x32 pixel size. Thus, to form a sample equation, all one would need to do is concatenate several individual character images together. For example, the first test equation used to validate the algorithm is: Ax+By=C. This equation was generated by concatenating random images from the character classes 'A','X','+','B','Y','=', and 'C'. Three test equations were developed for validation purposes and are shown below:

$$Ax + By = C \tag{4}$$

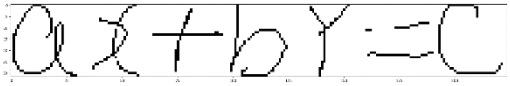

Figure 5: Test Equation 1. A simple linear equation.

$$[F \cdot G] = \int f(x)g(t-x)dx \tag{5}$$

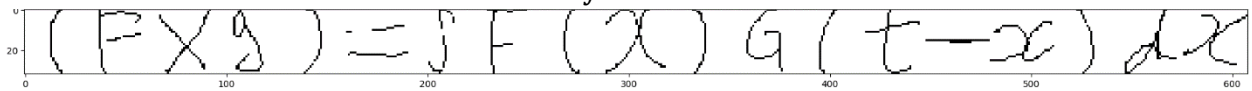

Figure 6: Test Equation 2. The Convolution Integral.

$$\frac{dT}{dm} = \sum \frac{(Xm + c - Y)(X)}{n} \tag{6}$$

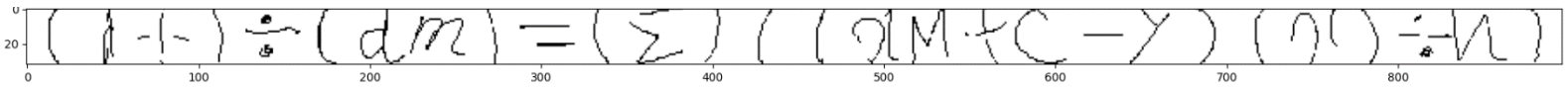

Figure 7: Test Equation 3. A more complex heat transfer equation using the Sum operator.

To accomplish the LaTex conversion functionality, a custom python module was built on top of an existing conversion function py2tex(). A custom conversion helper file was needed to account for certain symbols and operators that the existing function was not built to handle. The method for developing both the custom module and enhancing the existing function was similar to developing a lookup dictionary where character classes are compared to the dictionary entries, and each classes' corresponding LaTex representation could be retrieved.

Results:

The results of the CNN implementation for OCR in this project fall under the same two categories of functionality. The first results to report on pertain to the CNN training process and subsequent prediction outputs.

The CNN discussed in the previous section was trained over a batch of training data which was extracted at an 80/20 train-test split from the original dataset. Training was performed for 100 epochs with a batch size of 400 – each epoch spanned an average 3.21 minutes, which resulted in a total of 5.35hrs required to train the CNN. The loss value present at the final epoch of training amounted to 0.0447, and the final prediction accuracy score was 0.9913. Again, the accuracy metric used for this calculation was multiclass cross-entropy which is an average difference between the actual and predicted probability distributions for all character classes. Shown below in Figure 8 and 9 are the Loss and Accuracy plots over the total 100 epochs:

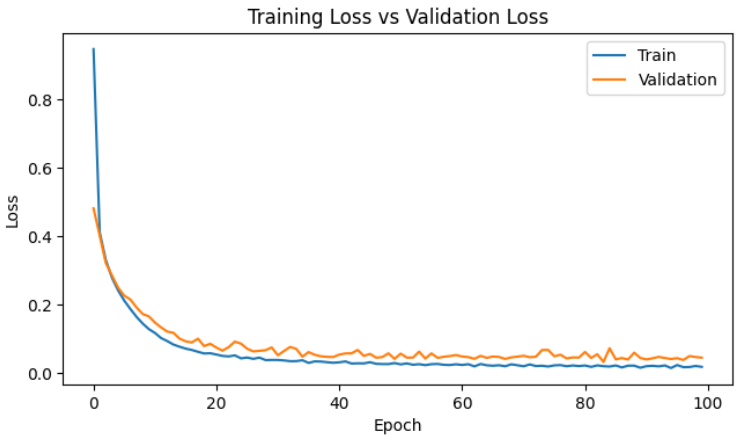

Figure 8: Loss Plot of Model over 100 epochs

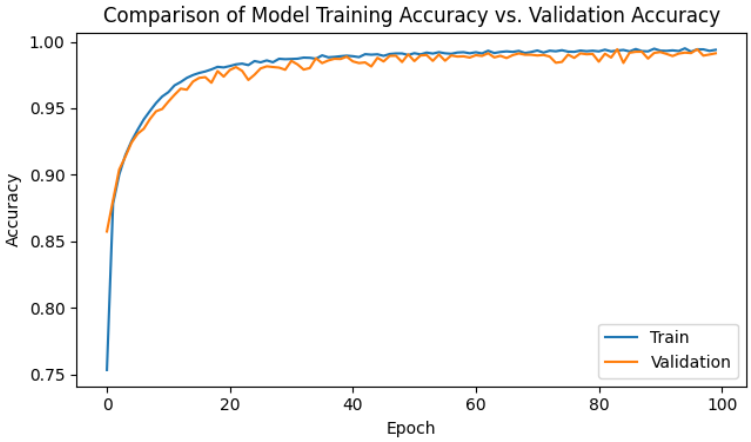

Figure 9: Accuracy of model over 100 epochs

An accuracy score of 0.9913 is very satisfactory for this use case considering prediction perfection would require an ultimate score of 1. This result demonstrates the robustness of the LeNet-5 CNN architecture.

With the CNN model trained, character class predictions could be made. As a simple starting point, individual character images were passed through the model for class prediction. Figure 10 depicts a random sample of input test images, with the titles of each plot representing the model's output class prediction.

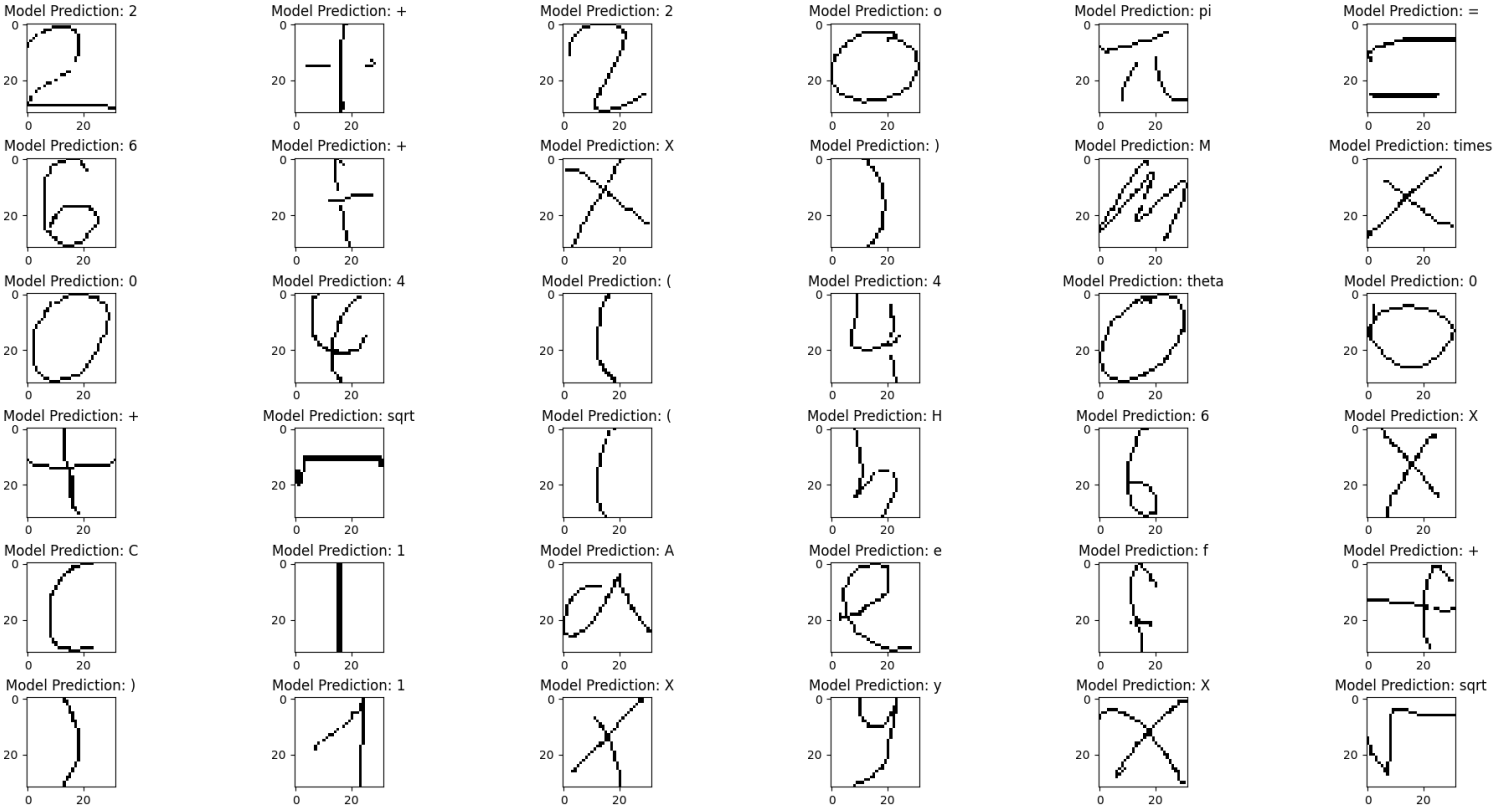

Figure 10: Sample input images and their corresponding class predictions from the CNN model

It is important to note that, even for individual characters, the CNN still demonstrated difficulty classifying each image accurately. The results from this demonstration show that the CNN model has difficulty accurately classifying input images that are relatively ambiguous – for example, in array position (5,2) of the above figure, the model's prediction is the class '1', whereas the symbol may actually represent the class '|' or 'vert'.

The next step was to validate the LaTex conversion functionality by passing test equations through the CNN and retrieving the corresponding class predictions for each symbol in the equation. First, the LaTex conversion module would gather the output class predictions into a vector of strings, then it would concatenate each entry in that vector into one single string

representing the full formula. As a test of the model's prediction capabilities, the concatenated string formulas for the three test equations are shown below:

Table 1: Intermediate result showing concatenated string formulas before LaTex Conversion

Test Equation 1	A(x())+by==C
Test Equation 2	(f*mul()*(G()))==quad()*f((X()))(G())((T())-(X())) (d()) (X())
Test Equation 3	((T())(T()))/((d()) (M()))==((sum()))(((X())(M())+C-y)((X()))/N)

Both equations 1 and 2 were formulated with 100% class prediction accuracy from the CNN model. However, the first symbol in equation 3 should be a 'd' rather than a 'T'. To quantify these accuracy measures, one metric to use would be total formula prediction accuracy. So, for equation 3, with one symbol incorrect, the accuracy of prediction is still 96.43% (27/28 characters correct) which is slightly lower than the prediction accuracy demonstrated at the end of the training phase.

Passing each of these string formulas through the LaTex conversion module yielded LaTex representations for each of the equations:

Table 2: Final result from LaTex Conversion functionality, including LaTex representation of each equation and their corresponding rendition.

Test Equation 1	$$\operatorname{A}\left(X \right)+by=C$$	$$A\left(X\right)+by=C$$
Test Equation 2	$$f \cdot G =\int \operatorname{\operatorname{\operatorname{\operatorname{f{\left(X \right)}}\left(G \right)}\left(T - X \right)}\left(d \right)}\left(X \right)$$	$$f \cdot G = \int f(X)\,(G)\,(T-X)\,(d)\,(X)$$
Test Equation 3	$$\frac{\operatorname{ T }\left(T \right)}{\operatorname{ d }\left(M \right)}=\operatorname{\sum }\left(\frac{\operatorname{\operatorname{ X }\left(M \right)+C-y}\left(X \right)}{N}\right)$$	$$\frac{T\,(T)}{d\,(M)} = \sum\left(\frac{X\,(M)+C-y\,(X)}{N}\right)$$

To the naked eye, the LaTex representation of each equation is rather unhelpful for understanding algorithm performance. To better understand the output of the LaTex conversion module, each of the equation's latex formulas were compiled in a LaTex editor and outputted in column 3 of Table 2. By visual inspection, it seems the functionality of the LaTex conversion task is a success; each equation's LaTex rendition is nearly accurate to the desired input equation. One thing to note is that there exist numerous extraneous parenthesis in both the LaTex representation and rendition of each equation. This is because of the strategy used in developing the custom conversion module to support the py2tex() function – surrounding classes with difficult names (such as 'ascii_124') with redundant parenthesis was a way to trick the py2tex function into interpreting the class name as a single input, as opposed to seeing each character as a new class name to be converted.

Future Work

While these results are promising, there is some major work to be done that will elevate the application of the modeled CNN. In order to fully develop the first functional objective of character classification, some tinkering will need to be done to decrease the program runtime – currently, 100 epochs is taking 5+hrs to compile which is an obvious impedance to any real-time predictions in the future. Further, to truly validate the algorithm's ability to classify handwritten equations, several tests should be done for actual handwritten equations rather than concatenated test images. For this aspect to be functional, a method for detecting symbol bounding boxes and center of masses would need to be developed in order to enable pre-processing and resizing.

After that, and for clarity of model performance, it may be advantageous to develop a confusion matrix to visualize the symbols that are the most difficult for the model to predict. Perhaps with this knowledge, the LaTex conversion functionality and class predictions to formula string process can be improved.

Finally, an exploration into dimensionality reduction methods may be advantageous. Methods such as t-SNE and PCA are used to reduce the order of a given dataset and identify principle components within a sample – application of PCA for this OCR algorithm could take the form of a prediction validation check. Since implementing PCA is a rather inexpensive

process, having predictions from a PCA model to validate the primary CNN's predictions against would be a secondary form of security.

Conclusion:

In summary, the goal of Optical Character Recognition algorithms is to convert text captured in digital images into machine-readable text. The applications of such algorithms impact numerous industries, and as demonstrated with this project, can also increase the work efficiency of day-to-day tasks.

The two functional goals of this project were to develop a character recognition scheme using a convolution neural network and then implement a LaTex conversion scheme for converting the outputs of the machine learning model to LaTex equation representations. Intermediary results demonstrated small successes in developing and training the CNN, where class prediction accuracy for single character images was close to the 99% during the training phase of development. Graduating from single characters to full equations provided some valuable lessons on how CNNs interpret handwritten symbols, and the largest takeaway here was that ambiguity in handwriting can lead to confused class predictions despite having rigorously trained the model. Intermittently retraining the CNN with an emphasis on the characters that produced the prediction errors is one way to enhance the robustness of the OCR algorithm.

The LaTex conversion functionality was also a demonstrated success, having satisfactorily produced a LaTex representation of the input equation image. Success in this aspect of the project was highly dependent on the class prediction accuracy from the CNN, and simple errors in predictions would track through the LaTex conversion process without remedy. Despite demonstrated shortcomings, the ultimate result from this project is a proof of concept for using Optical Character Recognition to convert equations to digital LaTex format.

References

[1] *Semantic Scholar | AI-Powered Research Tool*. https://www.semanticscholar.org/.

[2] Nano, Xai. “Handwritten Math Symbols Dataset.” *Kaggle*, 15 Jan. 2017, https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols.

[3] *OpticalCharacterRecognition,Usingk-Nearest Neighbors Arxiv:1411.1442v1 ...* https://arxiv.org/pdf/1411.1442.pdf.

[4] *DTM: Deformable Template Matching Hyungtae Lee, Heesung Kwon ... - Arxiv*. https://arxiv.org/pdf/1604.03518.pdf.

[5] Lecun, Y, et al. “Gradient-Based Learning Applied to Document Recognition.” *IEEE Xplore Full-Text PDF:* https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7169508.

[6] Saxena, Shipra. “The Architecture of Lenet-5.” *Analytics Vidhya*, 30 Mar. 2021, https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/.

[7] Cope, Greg. “Https://Www.algosome.com/Articles/Optical-Character-Recognition-Java.html.” *Optical Character Recognition Using PCA*, https://www.algosome.com/articles/optical-character-recognition-java.html.

[8] J. Memon, M. Sami, R. A. Khan and M. Uddin, "Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)," in IEEE Access, vol. 8, pp. 142642-142668, 2020, doi: 10.1109/ACCESS.2020.3012542.

[9] Yu Han Liu 2018 J. Phys.: Conf. Ser. 1087 062032

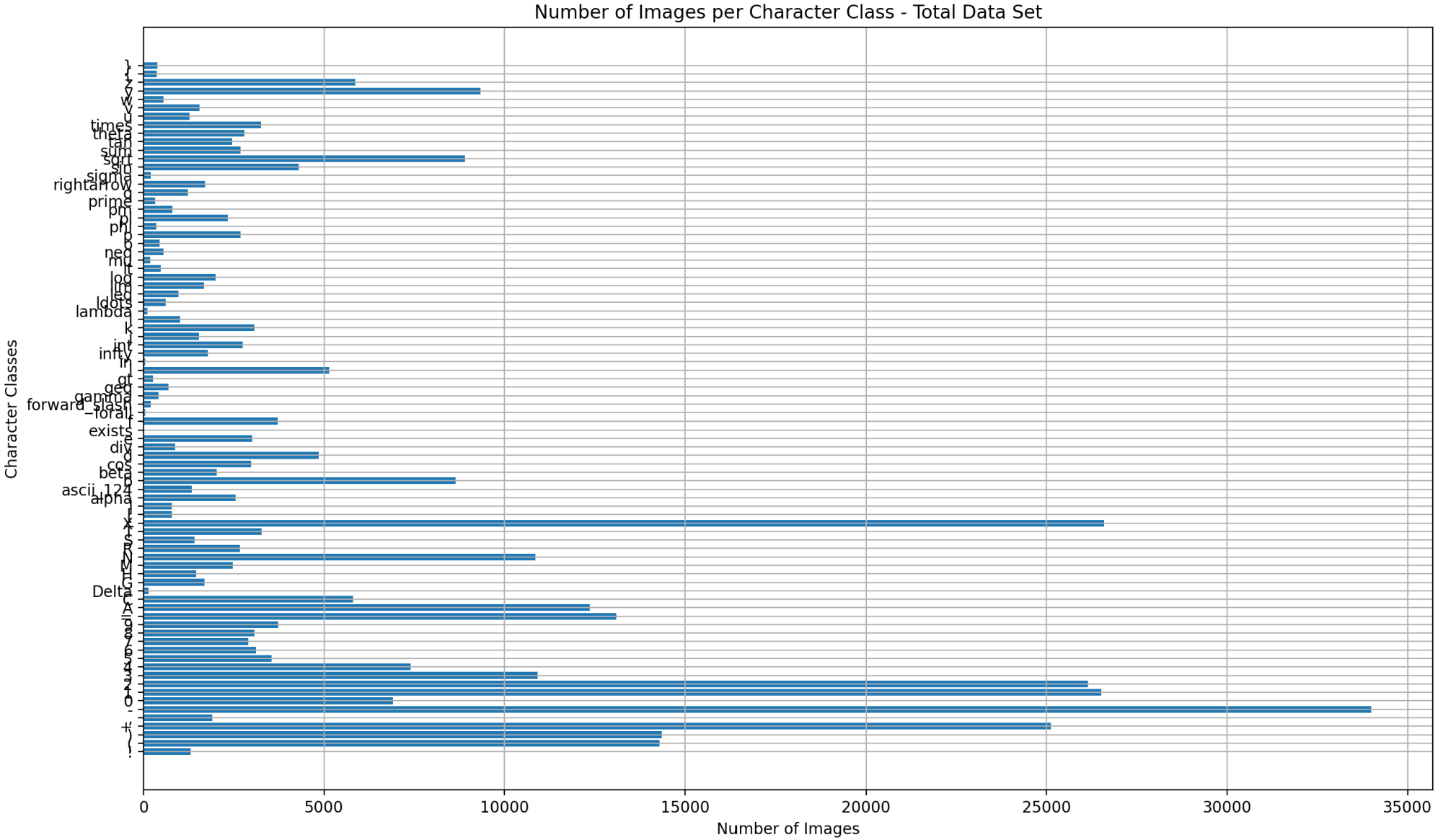

Figure 1A: *Symbol Image distribution among all 82 character classes (this was the most readable version I could come up with)*