Multi-Symbol LaTeX Conversion

Peter Do
peterhdo@stanford.edu
Stanford University
CS 230 and CS 231N

Kevin Baichoo
kbaichoo@stanford.edu
Stanford University
CS 230

Jonas Shomorony
jshom@stanford.edu
Stanford University
CS 230

Abstract

We propose a system of deep neural network models that convert handwritten LaTeX into the corresponding symbol form. In this paper, we train both VGG-16 and ResNet50 PyTorch [4] implementations over various numbers of symbol classes on the LaTeX classification task. We find that top-1 accuracy decreases significantly as the number of classes increases, potentially due to the similarity of symbols, but top-5 accuracy remains strong. We also analyze performance on sequences of symbols that a user may typically enter into a system and find that sequences with unique symbols are predicted accurately.

1. Introduction

We propose a solution to the problem of converting handwritten mathematical symbols to their corresponding LaTeX form. Oftentimes, it is much easier to hand-write equations, but converting the symbols to LaTeX can often lead to a clearer write-up, whether it be for a homework assignment or research paper. The input to the model is handwritten LaTeX symbols and the output is the corresponding LaTeX code for the symbols. We pass the images through a deep neural network model (either VGG-16 or ResNet50), which outputs a set of scores for each of the image classes in our dataset.

2. Related Work

There have been multiple approaches with the goal of developing a model to recognize LaTeX symbols. In "TexNet" [5] YOLO was used for object detection on 82 classes of LaTeX symbols. In our model, we aim to expand this number of classes to 200 without suffering a significant loss in accuracy. The original project that motivated our exploration is DeTexify [3]. This approach did not make use of neural networks – instead it used a variation of K-nearest-neighbors. One of the driving forces for our project was the belief that using a deep neural network would help us achieve much better results. In another paper, titled "Image-to-Markup Generation with Coarse-to-Fine Attention" [1], a neural encoder-decoder model based on a scalable coarse-to-fine attention mechanism was presented, which was shown to outperform classical OCR systems by a significant margin on rendered data.

Figure 1. Examples of images in our dataset: at, intersection, inf

3. Dataset and Features

Our dataset originates from Detexify's dataset [3]. The data is provided as "strokes" to represent handwriting, tuples of (X,Y, timestamp) in a Postgres database. We converted this stroke data into a binary image, connecting different points of the strokes with lines. We then distributed all the data into train, dev, and test sets, using a random 98/1/1 split, while making sure that there was at least one image for each type of symbol in each set.

In total, we have 208,171 400x400 black-and-white images distributed across 959 classes. In order to run our experiments, we focused on N classes, with the largest number of examples, such as N = 50, 200, and 500. Below are some statistics on the 50 largest classes:

Number of images per class per dataset, N=50					
	max	**min**	**median**	**mean**	**total**
train	3906	1011	1536.5	1708.3	85415
dev	29	7	12	12.92	649
test	32	5	11	12.82	641

4. Methods

4.1. ResNet50

We decided to use ResNet as it has been shown to perform incredibly well on object detection tasks [2]. Our reasoning was that it would continue to perform well on simpler images (as LaTeX image recognition should be simpler than more complex image classification datasets, such as ImageNet). However, we wanted to strike a fine balance between a network that was large enough to be expressive for a large number of classes but small enough to train in a reasonable amount of time. With this in mind, we settled on using an implementation of ResNet50 from PyTorch without pre-training.

ResNet is short for residual network, which was created to allow for easier training of deeper networks. Prior to ResNet, training deep networks was a significant challenge. Oftentimes, gradients had difficulty flowing backwards across a large number of layers. However, if stacks of layers of a neural network were treated as an identity mapping, the error of a deep network should be no worse than a shallower one. The main advancement with ResNet is the addition of skip connections which allow the model to pass values forward multiple layers. ResNet50 was able to achieve a top-1 error of 20.74 and top-5 error of 5.60 on ImageNet [2], which was among the top scores at the time.

To train, we utilized the Adam optimizer with a learning rate of 1e-4, a batch size of 32, and measured loss with cross entropy loss. We trained over 10 epochs to be consistent across models and experiments. We found that ResNet152 was more difficult to train as expected and using pre-training did not have as much of an effect as it does with VGG-16.

4.2. VGG-16

VGG-16[6] is well-known for its performance on ImageNet. One of the major improvements over AlexNet (the first model that performed reasonably well on ImageNet) was its use of multiple 3x3 filters in lieu of larger kernel filters such 11x11 or 5x5, which allowed better weight sharing with deeper layers and more localization to small features. Since our problem space was also bounded by 1000 classes, we figured VGG-16 would be applicable to our problem. We thought VGG would be a simpler model to train compared to ResNet (with far fewer parameters) while being complex enough to capture the different dynamics of the symbols.

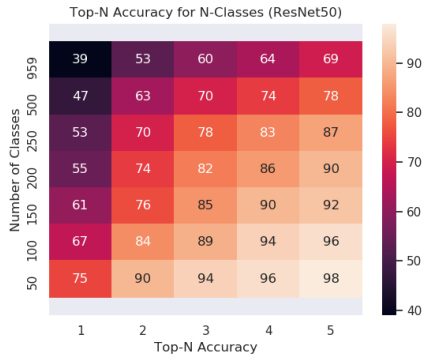

Figure 3. Heatmap for top-1 to top-5 accuracy with ResNet50 over various numbers of classes.

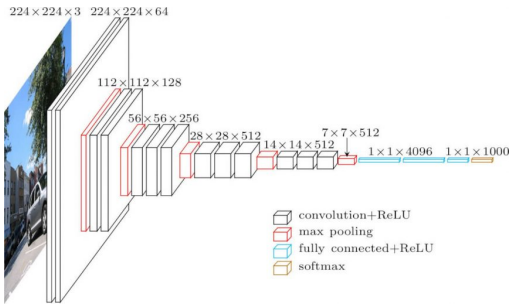

Figure 2. VGG-16 Architecture

To train, we utilized the Adam optimizer with a learning rate of 1e-4, a batch size of 64, and measured loss with cross entropy loss. We used a batch size of 64 as it was the largest we could accommodate on our GPU. As before, we trained over 10 epochs. We used pre-trained weights as we felt this would allow us to apply transfer learning to our specific problem. To improve the training of our model, we used batch normalization which tends to improve the speed, performance and stability of Neural Networks.

5. Results

5.1. N-class Experiments

We ran a set of experiments to determine how the models performed with varying numbers of classes, from 50 classes to the maximum number of classes, 959. We selected the classes by sorting by the classes with the most examples. For example, the 50 output class experiment is on the top 50 classes by the amount of examples each class has. We also vary the accuracy metric between top-1 accuracy and top-5 accuracy (which is what Detexify uses).

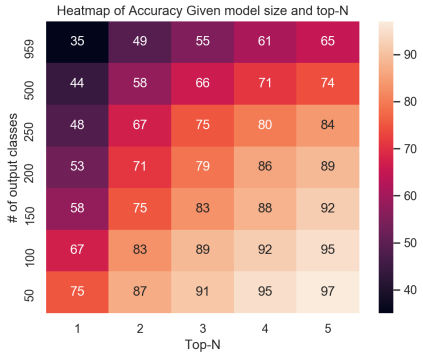

Figure 4. Heatmap for top-1 to top-5 accuracy on the test set with VGG-16 over various numbers of classes.

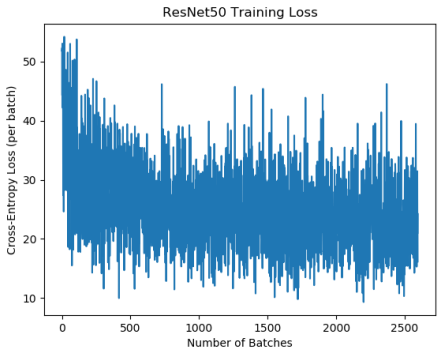

Figure 5. Loss curve for ResNet50 when training for 10 epochs with our 200-class dataset.

5.1.1 ResNet50

5.1.2 VGG-16

At best, the models achieve 98% top-5 accuracy over 50 classes. Performance degrades as we increase the number of classes. We also note a large gap between top-1 and top-2 accuracy, which can be attributed to similar-looking symbols.

5.2. 200-class results

We ultimately settled upon using 200 classes as this struck the balance between having a large number of symbols available and strong accuracy numbers.

5.2.1 ResNet50

ResNet50 is able to achieve a top-1 accuracy of 55% and a top-5 accuracy of 90% on the test set. ResNet has a steep downward trend in its loss curve, but seems to plateau. Training accuracy and validation accuracy are reasonably close together, with both around 70% after the full training loop.

Figure 6. Training accuracy for ResNet50 when training for 10 epochs with our 200-class dataset. It is likely that this could continue to improve if trained for longer.

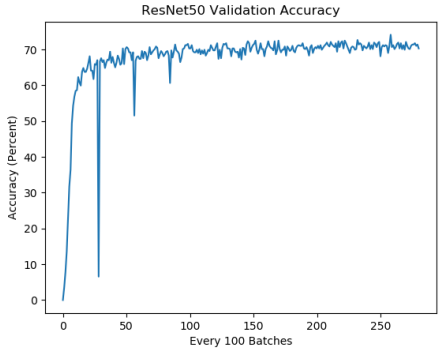

Figure 7. Validation accuracy for ResNet50 when training for 10 epochs with our 200-class dataset.

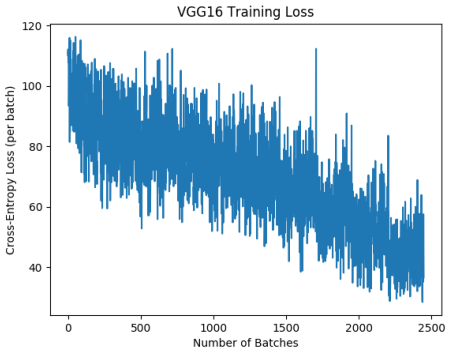

Figure 8. Loss curve for VGG-16 when training for 10 epochs with our 200-class dataset.

5.2.2 VGG-16

VGG-16 is able to achieve a top-1 accuracy of 53% and a top-5 accuracy of 89% on the test set. We also note that VGG has a more obvious downward trend in its training

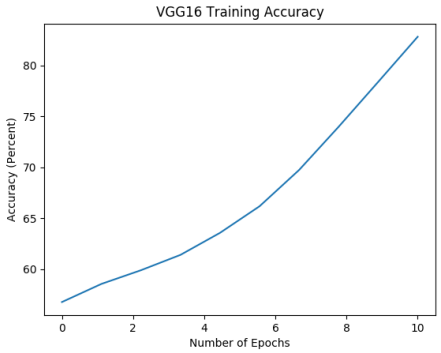

Figure 9. Training accuracy for VGG-16 when training for 10 epochs with our 200-class dataset.

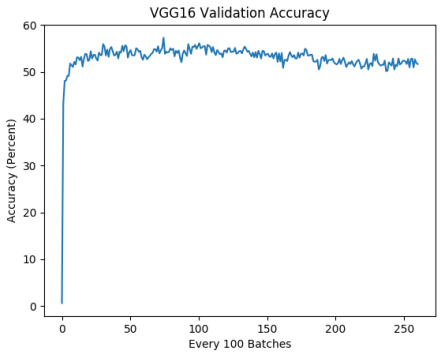

Figure 10. Validation Top-1 accuracy for VGG-16 when training for 10 epochs with our 200-class dataset.

loss curve than ResNet. We see that, like ResNet, VGG's training accuracy continues to increase. In addition, there is almost a 30% gap between training accuracy and validation accuracy for VGG-16, which indicates overfitting. Regularization would likely improve performance, and we note that our avoidable bias problem, is relatively small, around 16% assuming that Bayes error is 1%.

6. Analysis

6.1. Comparison with Initial Baseline

In our initial attempt, we used ResNet152, which was able to achieve up to 80 percent accuracy (over 959 classes) on the training set. However, this did not translate well to the validation and test sets, both of which had accuracy values around 2 to 3 percent. With some tuning of the model and ensuring the data was properly distributed among all datasets, we were able to improve the generalization of accuracy to the values shown in the results section.

6.2. Analysis of Results

With both ResNet and VGG, training accuracy continued to increase while validation accuracy plateaued. We may have seen further improvements in training accuracy if we trained for longer than 10 epochs. In addition, with regularization techniques, we may have been able to close the performance gap between training and test. Overall, our models performed better on a smaller number of classes than a larger number of classes, as expected. The model does have to contend with similar looking symbols such as "Sigma" and "Sum", which could explain the large gap between top-1 accuracy and top-2 accuracy. If presented with the top 5 results over 200 classes, a user would have the correct symbol represented 89 to 90 percent of the time with our models.

With pre-training, we found that ResNet was not able to reach its highest peak accuracy, but VGG benefited greatly from utilizing pre-training. In addition, ResNet took longer to train, although eventually output slightly higher test accuracy scores over the same number of epochs. Future work may need to weigh the benefits and drawbacks between the two models in order to best fit their use case. Experiments with both ResNet50 and ResNet152 indicated that the larger model was more difficult to train and could have benefited from running for more epochs.

6.3. Sequence (Multi-Symbol) Results

We also evaluated the model on sequences of symbols. To do this, we separated a "sequence" into its own evaluation data set and ran the model across the examples within that data set. As expected, the results were similar to test accuracy (55% with ResNet50 on 200 classes), with some variance between examples. Some sequences had lower accuracy scores (less than 50%) while others performed well (between 60% and 100% accuracy).

6.4. Error Analysis

As stated previously, the model generally performs well on a smaller number of classes (50). We constructed a confusion matrix on this smaller subset of classes for ease of visualization in order to gain insight into where the model may be having trouble. As expected, symbols that looked similar (sigma and sum, oint and int) were often classified as the other. These seemed to be the majority of mistakes, with distinct symbols being classified properly.

7. Conclusion and Future Work

We build a multi-symbol Latex conversation system that takes an image and transforms it to the LaTeX representation. An ideal system would include symbol detection, but for our sequence experiments we classify small datasets of desired symbols. We find that both ResNet and VGG per-

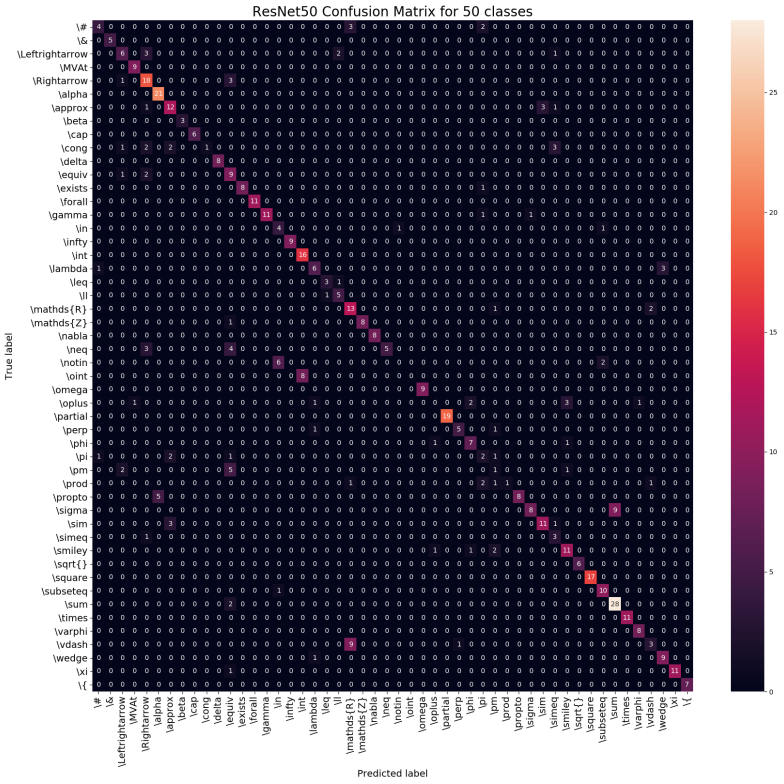

Figure 11. Confusion Matrix for ResNet50 over 50 classes.

Figure 12. Sequence of symbols that our model classified with 60% accuracy (ResNet50 over 200 classes). The original sequence is Sigma, alpha, approx, gamma, and beta. The predicted sequence is sum, alpha, sim, gamma, and beta. This also showcases a potential bad example in our dataset (approx).

Figure 13. Sequence of symbols that our model classified with 100% accuracy (ResNet50 over 200 classes). The original and predicted sequence is int, alpha, sim, partial, omega.

form similarly. Future work could experiment with additional regularization, as it is clear that both of our models overfit the training set, which caused us to suffer from high variance issues.

At best, we were able to achieve 98% top-5 accuracy. If we were able to increase top-1 accuracy of 98%, our system would still have significant flaws from treating each symbol as independent events. For example, in a formula with 10 symbols, treating each symbol independently would lead to an end-to-end correct translation $.98^{10} = 81\%$ of the time, which is good, but quickly degrades as the sequences get longer. A more 'complete' approach should approach the problem completely end-to-end without any segmentation. In this sense, an RNN model could be a viable area for future work.

8. Contributions

Peter worked on building the general PyTorch framework, ResNet tuning and experiments as well as confusion matrix, loss curve generation, and other visualizations. Kevin worked on Pytorch, VGG tuning and experiments, and data generation from Detexify's Postgres stroke database. Jonas worked on automating the generation of datasets for different numbers of classes given the original images, sequence datasets, and configuring model parameters given varying number of classes.

References

[1] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M. Rush. Image-to-markup generation with coarse-to-fine attention. *arXiv arXiv:1609.04938v2*, 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] Daniel Kirsch. detexify-data. `https://github.com/kirel/detexify-data`, 2014.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[5] Zen A. Simone, Cameron Cruz, and Kofi Adu. Texnet: The first stage of a handwriting to latex pipeline. `https://cs230.stanford.edu/projects_spring_2018/reports/8291079.pdf`, 2018.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.