Offline Mathematical Expression Recognition

Yisu Peng
yisupeng@iu.edu

Yang Zhang
zhang505@indiana.edu

1 Introduction

In academic area, LaTeX is widely used. But it takes people a large amount of time to write the code for formulas and mathematical expressions. Besides this, people often don't know how to write some special mathematical symbols and then need to go to do a search. Therefore, it is very useful to have some tool that can translate a math formula image into LaTeX code. With such a tool, when one wants to insert some math formula in LaTeX, they only need to write formulas on paper, take pictures of them then our tool will give out the code. This can save the researchers a large amount of time. In addition, another very promising application is to help people solve equation. At now when people want to get some assist in solving an equation, most will go to wolframalpha.com and type the equation in the system. The typing process is awful. It will be so great if there is a software to handle this process automatically. Some studies have shown that on-line methods can get better result than offline ones[6]. In intuition, it is obvious because on-line method store strokes as data which have more information than offline method. The advantage of offline method is that which has much wider applications. For example, one of our project goal is to take photo of hand written mathematical expression and the program will recognize it. In this case, the goal cannot be achieved by using on-line. That is why we choose offline approach.

2 Related Work

Nicolas et al. [14] introduced a method to recognize a math expression (online) using Pyramid Histogram of Oriented Gradients (PHOG) features. In 1998, Lecun solve the handwritten digit problem by LeNet [12]. We mainly use LeNet architecture to solve characters recognition. Yangqing Jia developed caffe [9], which make Deep Learning very easy to use. Thanks to caffe, we can reuse LeNet architecture and train collected data.

The research of the Convolutional Neural Network can be traced back as early as Hubel and Wiesel's biological work [8]. The visual field is affected by the sub-regions of cells which act like local filters. CNN has one or more convolutional layers, often followed by one or more fully connected layers [17]. Conventionally, a full ConvNet architecture consists of Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks) [16]. LetNet is considered as the first successful applications of Convolutional Networks and the major usage of LetNet architecture is to read zip codes, digits, etc[16] in 1990s'. However, CNN hardly gained its attentions by the vision community only after 2012 ImageNet challenge. Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton created a deep CNN to get incredibly good result [11]. Although CNN is quite prevalent today, Hinton gave a talk[7] about the dark side of CNN indicate that human beings have not really understood what CNN was.

3 Approach

In order to solve such a problem, we decompose it into several subproblems. The first step is to preprocess the raw input image so that it can be handled better in later steps. The second task is to do a segmentation on the image to have all pixels for one symbol in a set. The next one is to figure out the location and size of each single symbol with the information provided by the previous segmentation process. Then, we need a classifier to recognize each symbol. At last, there should be some parser which in cooperate both the location and classification information to get the final result. For the parser, we have not started because we did not get good enough accuracy for classification.

3.1 Data prepare

One of the biggest challenges in machine learning is to prepare an appropriate data set. In our case, first we need to collect LaTeX symbols. Luckily enough, we found Dr. Daniel Kirsch had collected lots of LaTeX symbols in his detexify project [10]. While detexify data set do not symbols that can be typed directly in LaTeX such as digits, English letters and some simple math symbols. For digits, we use the MNIST databases [13]. Since it is relatively large, we only use part of its data to train. The handwritten English letters is from The Chars74K data set [4]. At last, we collect a few extra math symbols data from write-math [15] to our data set.

3.2 Data preprocess

Since our images are from several data sets, we need to unify these data sets by some preprocessing measures. First step is to convert all the images to the "png" format. The image data we use are in several different image formats in their original data set. For the Detexify and Write-Math data set, the authors use SVG stroke data to save the writing trail of symbols. We need to decode them and stored them in png formats. Second step is to take gray scale and negate the color of the image. We need to make sure all images in our data set have the black background. This will benefit the computation. Think about when we do the matrix multiplication by hand, how happy we will be if we saw a matrix has lots of 0. Then we crop the symbol in the image and add the padding to make it square. This step is necessary because it extract the symbol and normalize it, while it causes some problem such as the lower case 'c' and upper case 'C' will become the same. We can solve this problem when we do the semantic parsing. The last step is to move the image data to LMDB database which can be used by Caffe. The procedure is shown in Figure 1.

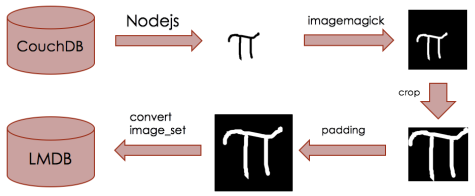

Figure 1: Data Preprocess

3.3 Data Resampling

Some symbols have enough data (like digits), while some symbol has very few data (like English letters). For those symbols lack of image samples, we copy them and make them up to 1,000 images. For those symbols have lots of image samples, we choose randomly up to 1,000 images. The reason we need to balance the data set is that the strategy we chosen for caffe is SDG, which will randomly pick up data sample. If we do not balance the data set, then the neural network will have biased prior for categories, which will cause it to have some unfair preference to those categories that have more data points in the training set. This is due to the neural network is a statistical approach, if we give it an unbalanced data set, it will get an inaccurate statistical conclusion that some category has a much more occurring rate than others. This should be avoided.

3.4 Classification

Since LeNet got a successful application on hand written digit, we think this architecture should also have reasonable performance on our classification task, to recognize digits, English letter, Greek letters, and some math symbols. Mostly we keep the original architecture of LeNet. The only thing we need to modify is the last layer. We changed the output classes from 10 to 101 (we have 101 categories for symbols). To be specific, our neural network has 7 layers, consisting 2 convolutional layers, 2 pooling layers, and 3 full connected layers. The first layer is a convolutional layer one, with 28x28 inputs with 1 channel and 20 filters with kernel size 5x5. The output is in dimension 24x24x20. The pooling layer one following this has 2x2 kernel size and moving in stride 2 with a maximum kernel. Then the next one is convolution layer two which takes input of 12x12x20 and has 50 5x5 filters. The output is 8x8x50. Upon this is another pooling layer with 2x2 and 2 strides, resulting in a 4x4x50 output. The following layer is a fully connected layer with 500 neurons and ReLU activation function. Then following this is another fully connected layer with 101 outputs and identification transfer function. Finally the last layer is a softmax layer.

3.5 Segmentation

We use Efficient Graph-Based Image Segmentation (EGBIS) algorithm[5] to do the segmentation job in method. This algorithm is a spanning tree algorithm for the segmentation task. It treats the image as a graph, and assign edge to every two adjacent pixels. According to their difference in color the algorithm assigns weights to edges. After we construct the graph, the algorithm first finds the minimum spanning tree for this graph, and then cut those edges with weights over a threshold to get a forest.

3.6 Localization

After the segmentation step, the task is to locate symbols in the image based on the segmentation result. Here is what we do. We find all pixels according to the segmentation. For each segment there will be a tree structure to organize all pixels in a segment. This is because the "EGBIS" algorithm is a spanning tree algorithm for the segmentation task. One problem with this method of localization is that the output of the segmentation algorithm does not only containing the segments for symbols, but also those for the background. Here we use a naive method to separate these two cases. We compute the average intensity of all pixels in one segment and set a threshold for this. If the average intensity is below this value we consider the segment to be part of the background, otherwise we adopt it as a symbol. To get the location and size, we just find minimum and maximum of x and y. Some desired output is shown in the following figure 2,

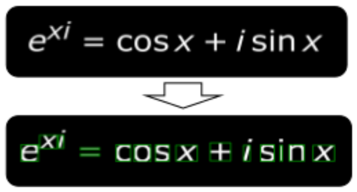

Figure 2: Localization

4 Experiment

To evaluate the neural network we trained, we used the test set that is subsampled from the whole data set before we do the resampling. We are ruling out the test samples firstly in case the training data have some test samples. The accuracy we got for 101 classes we chose on the test data set is 81.5%

4.1 Learning Rate and Batch Size

We trained several versions of our convolutional neural network on our combined data set, for each configuration we train generally 30,000 iterations with batch size 64 batch size or 256 batch size. We use decay on the values of the global learning rate η with a factor 0.00028. This is set as we downloaded caffe, we find this parameter performs well, so we keep it unchanged. We tried a lot different learning rates, range from 0.00001 ˜0.01. At first we use quite a large learning rate, that is the default value in the package for LeNet model in caffe. However, this caused the training of neural network to diverge. The loss of the output of the neural network keeps at a large value. After trying different values, finally we find the value 0.002 is good for our problem.

Later we also found that the batch size has a large relationship to the step size. With large batch size, we found that we can use relative large learning rate.

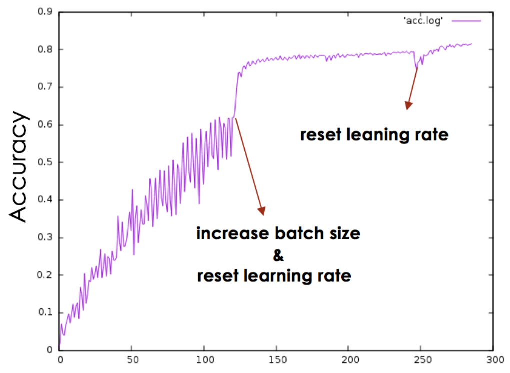

Figure 3: Learning Curve of Neural Network

One explanation for this could be that since we are using mini-batch stochastic gradient descent, the input data are random, and this means that the gradient we use to update the weights is a not a true gradient for the optimization problem, but a noised approximation to it. So, when using small batch size, the gradient might differ significantly, and thus lead the weights of the neural network to be updated in a wrong way. In such circumstances, if we are just using some small learning rate, the noise will not cause much hurt, and as long as the whole dataset is stable enough and do not have a too big variance, the weights will converge to some point. However, if we use some big learning rate, we will move too far along the noised gradient. Then for a next given input data batch, we will probably get bigger loss/error. This increased loss in turn will result in an even larger gradient, according to the partial gradient derived from the loss function. Therefore, following this loop, the loss grows larger and larger, at last stops at some boundary.

On the other hand, if we choose not only a relative large learning rate but also a larger batch size. The situation will be different. Here is one way to consider this. The enlarged batch size will be better in averaging out the noise in the gradient. With the noise smoothed out, we are able to get closer approximation. As long as we are keeping the track of some nearly right direction, we don't need to set the learning rate at a very small value.

4.2 Stroke Size

For the image generated from SVG data, at first we used a thin stroke, 5px, to convert it into an image, and get the image on the left in figure 4. For now it seems good.

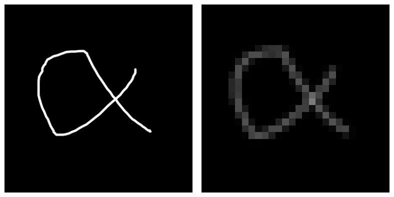

Figure 4: with stroke 5px,
original on the left, and resized on the right

However the problem occurred after we resize the image to a 28x28 size. The symbol became very blur. The reason is that the original SVG image is quite large (400x400). This large ratio resizing averaged the pixels out. After we trained the convolutional neural network using this kind of images, we only got very poor result, about 40% even only for 10 categories. Then we tried different widths for strokes, say 7px, 11px, 14px, 18px. Then, we compared the effects of resized image and picked 14 and 18 to make further comparison in the neural network. After experiments we found that 14 is slightly better than 18, so we choose this as our final stroke size.

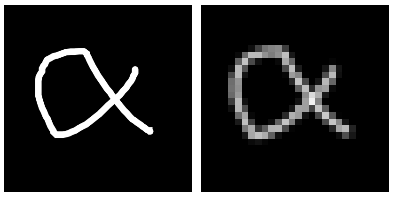

Figure 5: with stroke 14px,
original on the left, and resized on the right

4.3 Fine Tuning

We also tried two fine tuning method. One is to fine tuning the pre-trained LeNet model on MNIST data set. Even this model is only trained on the MNIST data set that only consisting of hand written numbers, but we think that this might be helpful for us to get some better result. Our reasoning is that even the numbers are quite different from English and Greek characters, and also math symbols, they still share a lot of similarities. For example, they are all written by hand, so that they can be seen as some subsets that belonging to a same large category. Also, they share a lot of similar local features, like straight line, curve, circle, angle, corner, and so on. Based on these guess, we think that the feature extracting filters learned by LeNet through the MNIST data set should also work on

The other try we did is to use a pre-trained model of Alexnet[11]. This model is trained on a LSVRC-2010 ImageNet training set. We successfully run this pre-trained CNN, but we failed to do the right fine tuning because our laptop cannot handle this CNN on caffe. Each iteration needs lots of time. We end up with less than 10% accuracy with a few iterations.

5 Conclusion

First, we found it is very useful to have the character to be properly bold before we send it to the convolutional neural network. For svg image this can be done very easily, for images one thing we can use is the dilution. For now we just use the simplest dilution operation that operates a squared region, but we notice that this kind of operation will cause aliases. We can use some structured dilution for future.

Second, we found there should be some relationship between batch size and learning rate. With larger batch size one may have larger learning rate, and this allows us to train the neural network in a faster speed.

Third, we have not figured out the fundamental reason why we only achieve 81.5% accuracy. Lack of handwritten English letters may be a major reason. An evidence is that we can easily recognize $\alpha + \beta$, but our program is always failing to recognize $a + b$. Another reason, maybe overfitting. The duplications of English letters data might cause this. We can solve this problem by increasing handwritten English letters training data.

6 Future work

We notice that the Chars74K data set has lots of images about English letters that are not hand written. We have not tested those data because those English letters may be dirty data, but we only get a few samples of English handwritten letters. Therefore, we would like to have a try on those data, though we are not sure whether these data are useful for our neural network or not. We also think about another way of fine tuning the LeNet. For example, we extract the first layer of filter, and keep it fixed and then start fine tuning the rest layer. Visualizing the filters in LeNet may be helpful because we can compare them with ours in order to do more analysis. We also want to work on parser work after improving the classification accuracy. This work includes a segment correctly split symbols, such as ‘i’, ‘=’. We may consider the grammar based inference or consider the mathematical formula as sort of prior knowledge [2]. Belief propagation will also be a appropriate tool to do the segmentation to deal with cases when the background contains much noise.

7 Acknowledge

We thank Prof. David Crandall for really helpful comments and discussions. We also thank Dr. Daniel Kirsch and Martin Thoma to provide us the permission to the handwritten data. The idea to generate feature images is from this tutorial [1]. We read caffe office[9] tutorial and adapt LeNet prototxt file. Christopher Bourez’s blog helps us to understand how to use caffe in

python[3].

References

[1] Classification: Instant recognition with caffe in, January 2015.

[2] Francisco Álvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. An integrated grammar-based approach for mathematical expression recognition. *Pattern Recognition*, 51:135 – 147, 2016.

[3] Christopher Bourez. Deep learning tutorial on Caffe technology : basic commands, Python and C++ code. `http://christopher5106.github.io/`, 2016.

[4] Teófilo Emídio de Campos, Bodla Rakesh Babu, and Manik Varma. Character recognition in natural images. In *VISAPP (2)*, pages 273–280, 2009.

[5] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, September 2004.

[6] I. Guyon, P. Albrecht, Y. Le Cun, J. Denker, and W. Hubbard. Design of a neural network character recognizer for a touch terminal. *Pattern Recogn.*, 24(2):105–119, January 1991.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Dark knowledge, January 2015.

[8] David H. Hubel and Torsten N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195:215–243, 1968.

[9] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[10] Daniel Kirsch. Detexify. `http://detexify.kirelabs.org/classify.html`, 2016.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[13] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits.

[14] Lan Nguyen Nicolas D. Jimenez. Recognition of Handwritten Mathematical Symbols with PHOG Features.

[15] Martin Thoma. Hwrt database of handwritten symbols, January 2015.

[16] Stanford University. CS231n Convolutional Neural Networks for Visual Recognition. `http://cs231n.github.io/convolutional-networks/#layers`, 2016.

[17] Stanford University. UFLDL Tutorial - Convolutional Neural Network. `http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/`, 2016.