Optical Character Recognition through Convolutional Neural Networks

Jay Szeto
jsa143@sfu.ca

Adrian Lim
aclim@sfu.ca

April 8, 2017

---

1 Introduction

Optical Character Recognition (OCR) is the process of converting an image of text into a computer readable format. [1] It can be used in applications such as data entry, license plate recognition, and defeating CAPTCHA anti-bot systems. [1] One approach for OCR involves the segmentation of an image into characters before the classification of each character. Of the many different ways to classify an image, one of the more popular approaches involves the use of Convolutional Neural Networks (CNN). [2] This type of approach has grown more popular and complex in recent years which can be seen by the appearance of neural networks such as LeNet, ImageNet, and ResNet. [2–4]

Due to the popularity of deep learning in recent years, we have decided to focus on CNNs for this project. We construct multiple different CNNs using the TensorFlow library in order to classify digits, lower-case, and upper-case letters. [5] We have also attempted to implement a basic line and character segmentation algorithm to detect words in examples such as in text documents.

2 Related Work

TensorFlow uses an assortment of procedures to enable training. [5] This section will focus on highlighting existing concepts and mathematical applications that are core components in the tested neural network. Using these concepts, TensorFlow is able to facilitate the construction of different type of layers to construct neural networks. [6]

2.1 Model Generation Function

TensorFlow uses the softmax regression function to calculate loss when assigning weights. [7] This model uses two steps:

1. Collect a sum of evidence values from input belonging to a certain class.

2. Convert that evidence into weights.

Evidence is gathered by performing a weighted sum of the pixel intensities. A negative weight for a pixel represents evidence that is against the image being in that class. Positive weights are treated as evidence in favor of labeling an image to said class. Extra evidence, called bias, were also added. These extra data allows the algorithm to affect the evidence with weights "more likely independent of the input". [7] Evidence for a class i given an input x is:

$$evidence_i = \sum W_{i,j} x_j + b_i \tag{1}$$

where W_i is the weight and b_i is the bias for class i, j being an index allowing the summation of the pixels of the input image x. Once these evidences are available, a normalization is required to create a valid probability distribution:

$$softmax(x)_i = \frac{exp(x_i)}{\sum exp(x_j)} \tag{2}$$

Exponentiation allows increases in evidence units to amplify our hypothesis of prediction probabilities multiplicatively. On the other hand, reduction in evidence amounts results in the hypothesis to reduce to a fraction of its former amount. [7]

2.2 Error Minimization Algorithms

Although a method for calculating probability via evidence is available, such a model requires a training process in order to output the desired outcome. A model's effectiveness is measured by how minimal its error margin is. A model's loss is measured through the cross-entropy function:

$$H_{y'} = -\sum y'_i log(y_i) \tag{3}$$

Where y is the predicted probability distribution, and y' is the true distribution of probabilities. [7]

The minimization of the loss output function is the responsibility of the backpropagation algorithm. By minimizing the loss function output,

users can maximize the amount of confidence behind a model's prediction. After evaluating the derivative of each weight, this algorithm can determine whether or not a weight must increase or decrease to minimize loss. [8]

Finally, a gradient descent algorithm will utilize the results of the backprogation algorithm to alter weights for determining a minimum cross entropy value. Although TensorFlow provides a multitude of gradient descent variations, the tested neural network utilized TensorFlow's stochastic gradient descent algorithm. [6]

2.3 Layers

Neural networks consists of a different range of layer types. Below are three different types that were used: convolutional, pooling, and fully connected layers. [6]

1. **Convolutional Layer:** A layer that applies convolution, filters the input image with a ReLU activation function. Each portion of the image will produce an individual value to the output map. [6]

2. **Max Pooling Layer:** extracts the data from convolutional layers to reduce the dimensions of the received feature map. Due to the max pooling algorithm, only the maximum values will persist after such downsampling. [6]

3. **Fully Connected Layer:** Unlike the convolutional and pooling layers, a fully connected layer receives and outputs a map of a constant size. [6]

3 Approach

The Char74K's computer "EnglishFnt" data set was used for training and testing of the CNNs. Such font data seemed to greatly reflect the type of characters we were trying to detect. This data set contained 62992 computer generated images of 62 classes of digits, lower-case, and upper-case letters. [9] They consisted of 254 different fonts with 4 different styles; normal, bold, italic, and bold and italic. [9] The images were additionally centered, padded, and were of a constant size of 128 x 128. [9] Images from this data set can be seen in Figure 3.

To observe the effect of different parameters on the CNN, we constructed multiple networks using the TensorFlow library. In both CNNs, the Char74K data set was randomly split into a training set containing 90% of the data,

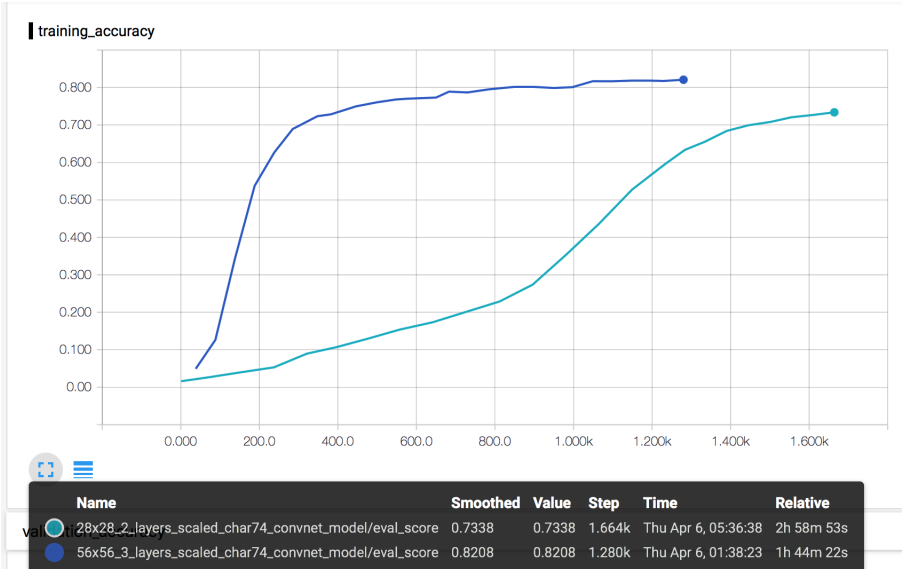

Figure 1: Training Accuracy

a validation set containing 5% of the data, and a test set containing the remainder of the data. Validation and training accuracy was plotted to monitor for overfitting. these graphs can be seen in Figures 1 and 2.

When training the network we used a batch size of 100. We also used 3 epochs of the training data. In the case of the 3 convolutional layer model, we stopped early as it appeared the results were not improving after multiple steps.

The first tested CNN was a modified version of the neural network in the MNIST TensorFlow tutorial. [5] The only modification that was made was that the final fully connected layer in the network was modified to output to 62 nodes rather than 10. This is to reflect the 62 different character types in our training data. The image is first resized to 28 x 28. After, it is convolved with 32 filters with a kernel size of 5 x 5 in the first convolutional layer. In the first pooling layer, the image size is reduced by 2 by applying a pooling size of 2 x 2 with a stride of 2. In the second convolutional layer we apply 64 filters but keep the same kernel size as the first layer. This is followed by another pooling layer and 2 fully connected layers. To prevent overfitting we apply dropout to the first fully connected layer. The layer structure of the first CNN can be seen in Figure 4.

Another CNN that was tested was an extended version of the previous

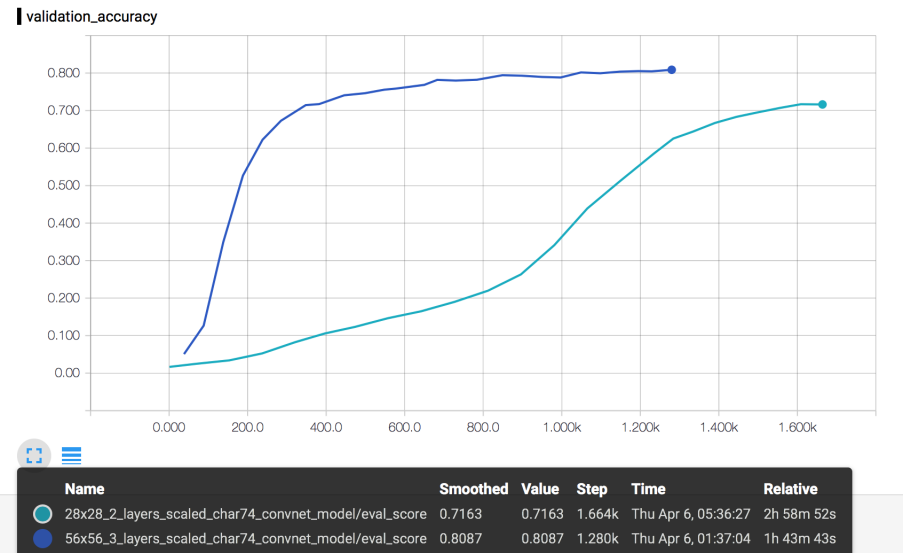

Figure 2: Validation Accuracy

Figure 3: 4 Images from the Char74k "EnglishFnt" Data Set

CONV -> $RELU->POOL->CONV->RELU->POOL->$
$FC->RELU->DROPOUT->FC$

Figure 4: Model from TensorFlow Tutorial

$$\text{CONV -} > RELU- > POOL- > CONV- > RELU- > POOL- > CONV- > RELU- > POOL- > FC- > RELU- > DROPOUT- > FC$$

Figure 5: Model with Additional Convolutional Layer and Pool Layer

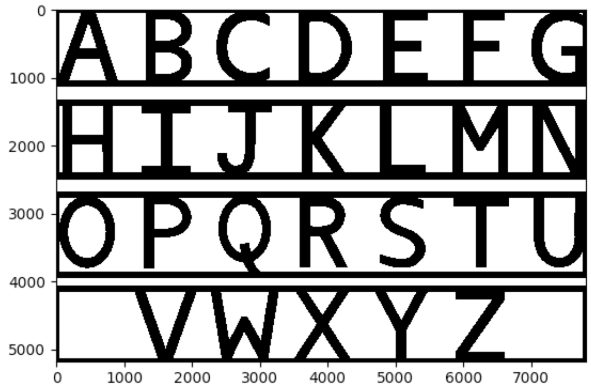

Figure 6: Line Detection and Segmentation

network. In the newest neural network we doubled the size of the input image and added an additional convolutional layer and pooling layer. The added convolutional layer and pooling layer have the same parameters as the previous layer in the CNN.

This CNN an be seen in Figure 5.

In order to process images containing a set of characters, a way to normalize and segment characters in images was developed. This was accomplished using a basic algorithm that first segmented lines in the text. This was done by searching each row for white space. If a row consisted entirely of white space it was assumed that the end of the line was found. An example output from this line segmentation algorithm can be found in Figure 6. The characters were then isolated from each line by running the previous algorithm on each column. An example output from this character segmentation algorithm can be found in Figure 7. These characters were then scaled and padded appropriately and classified by the CNN.

Figure 7: Character Detection and Segmentation

4 Results

The test set consists of 12600 images from the Chars74K data set. A number of images were programatically set aside for the test set but an error caused this set to become irretrievable. Due to the large amount of time required to retrain a new network with a new set, we decided to continue using the aforementioned 12600 random images from the complete set.

Using the TensorFlow MNIST tutorial modified example, we get a 74% accuracy against our test set. A bar chart of the incorrectly predicted images can be seen in Figure 8.

Our model with the added convolutional layer and pooling layer achieved an 81.8% accuracy against our test set. A bar chart of the incorrectly predicted images can be seen in 9.

The combined character segmentation algorithm and improved CNN was applied to the image in Figure 6. Every letter was predicted correctly except for the letter O which was misclassified as the number 0.

This was then applied to a more ambitious example, a sentence taken from Wikipedia. [1] Though the neural network was not trained with symbols other than the alphabet and numbers, and the segmentation algorithm was quite basic, the output from the program was still somewhat ledge-able. This is shown in Figure 10.

5 Discussion

The results show that adding a convolutional layer and a pooling layer in the neural network increased the accuracy of the neural network by 7.8%. However, this may be due to overfitting as we were unable to test our network against the actual test set. That being said, we monitored the networks for overfitting by plotting the accuracy in our training set and validation set which can be seen in Figures 1 and 2. From observing these two graphs, the two networks do not appear to be overfitting, since the validation data is not decreasing.

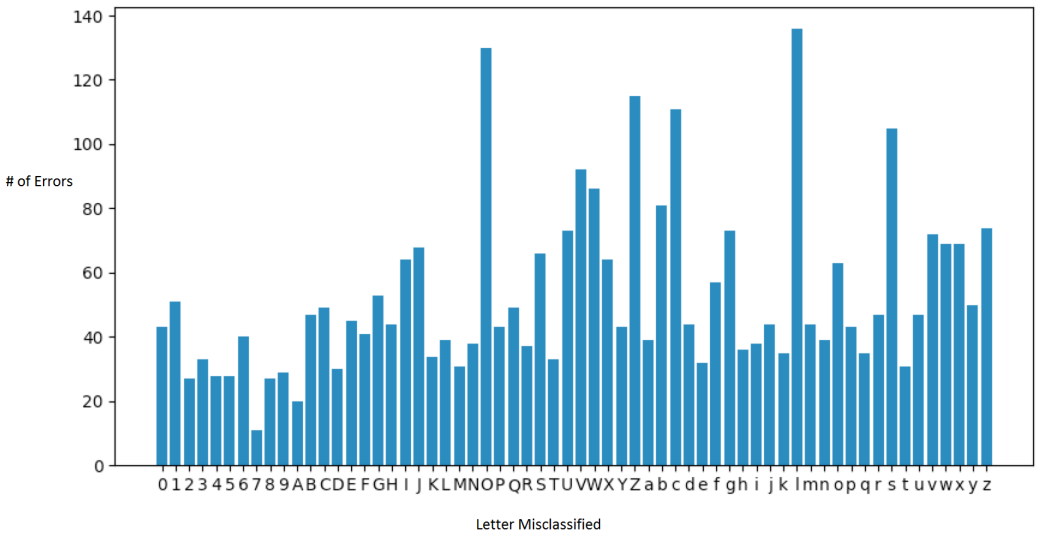

Figure 8: Bar graph showing which sample images were misclassified in the TensorFlow example prediction

Analysis of the misclassified images of the trained neural networks seems to show that the majority of errors occur in images that are extremely similar to other images, 0 and O or 1 and l, for example. Due to how the images were processed in the Char74k training set it makes sense that these images become almost indistinguishable from one another and would cause at least 1 of these characters to be misclassified more often. 11 This suggests that it may be impossible to correctly distinguish certain characters from each other without additional information such as the letters surrounding the character of interest.

When attempting to classify segmented characters outside of our data set without normalizing the characters, the predictions would be poor. Once we normalized both the training data and the characters taken outside of the data set, these predictions became much better.

In order to apply this OCR solution to text documents, a better segmentation algorithm will need to be implemented.

The results of the character segmentation shows that there is much room for improvement. However, a very basic character segmentation algorithm was written solely to apply our CNN on a more practical and interesting

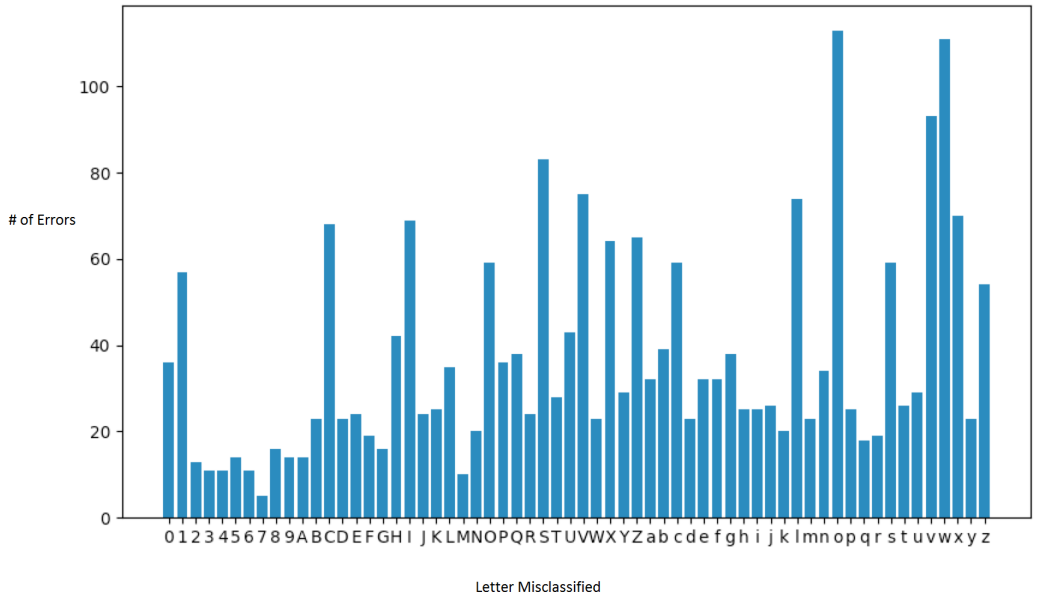

Figure 9: Bar graph showing which sample images were misclassified in the 3 convolutional and pool layer model

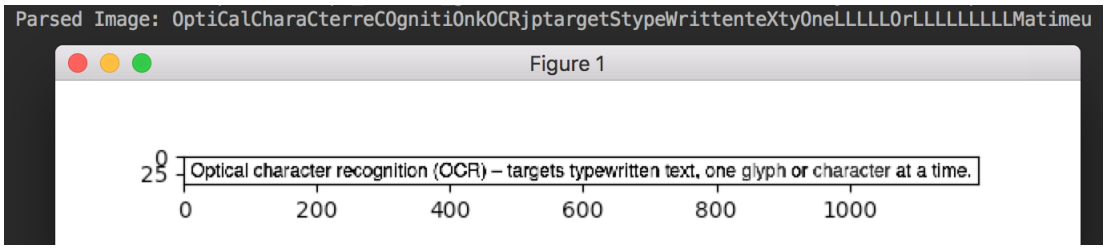

Figure 10: OCR on a Sentence Image Taken from Wikipedia

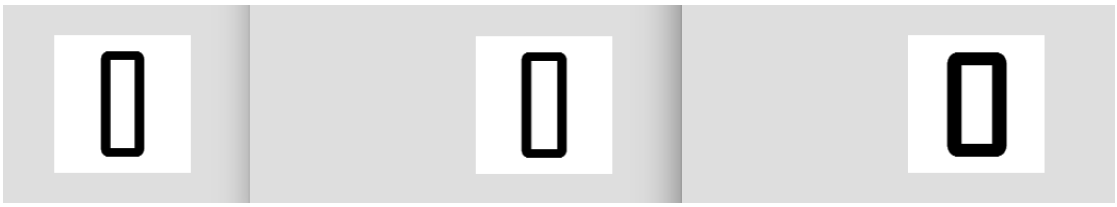

Figure 11: From left to right: 0, O, o from the Char74K training set

example.

6 Future Work

There are multiple areas that could be conducted to improve accuracy in our program. First, the data set that was used in the experiment did not contain punctuation which is critical for processing a text document. This could be addressed by adding these images to our data set and increasing the number of classes in our CNN. Another way we could improve our program for OCR would be to implement a better character segmentation program which could account and report spaces between words. Finally the character segmentation algorithm currently does not account for images in text, a pattern matching algorithm could be used to obtain solely the part of the image containing text.

7 Conclusion

Our results show that the CNN reported decent results if the input data was normalized correctly. In order to create an end-to-end OCR solution a better segmentation algorithm would need to be implemented. The results also show that it is difficult to classify the differences between similar characters such as O and 0 especially if the data is normalized in a similar way as in our data set. Although adding additional layers may increase over fitting as described in the ResNet paper, in our case it did not negatively impact our results and instead improved them. [4]

References

[1] "Optical character recognition." Wikipedia, 2017. Web. 6 Apr. 2017.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, 2016.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[6] "A Guide to TF Layers: Building a Convolutional Neural Network." TensorFlow, 2017. Web. 6 Apr. 2017.

[7] "MNIST For ML Beginners." TensorFlow, 2017. Web. 6 Apr. 2017.

[8] Y. LeCun, L. Bottou, G. Orr, and K.-R. Muller, "Efficient backprop," tech. rep., Willamette University, Oregon, USA, 1998.

[9] T. E. de Campos, B. R. Babu, and M. Varma, "Character recognition in natural images," in *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009.