COMP551-MiniProject3

John Flores, Luis Pinto, Rebecca Salganik

March 2019

Abstract

In this project we used multi-layer convoluted neural networks for handwritten numerical image classification. Our work utilized a modified MNIST dataset which contained a collection of handwritten numbers between 0 and 9 in order to classify the images in our dataset. The final model achieved a final accuracy of 93.4% on the Kaggle test set.

1 Introduction

In this project, we are tasked with creating a model which can identify the largest digit by area in a grey-scale image containing multiple digits. The dataset, is comprised of 40,000 examples of grey-scale pictures containing multiple handwritten digits. Each example is a 64 x 64 matrix of pixel intensity values, which can be interpreted as grey-scale pictures, as seen in Figure 5 in Appendix D. Our work revolves around convolutional neural networks (CNN); our results demonstrate that the user-defined hyper-parameters and the structure of the CNN contribute to a high accuracy model. The learning rate, batch sizes, and number of convolutional layers play an integral role in the performance of the final model. A final test accuracy of 93.4% was achieved on the Kaggle test set.

2 Relevant Work

There is much work done in the field of both image classification and proper neural network practices. In writing our code, we drew heavily upon the ideas voiced by P.Simard, D.Steinkraus, and J.Platt in *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis*[1]. The paper describes a set of concrete best practices for document analysis by neural network models. It specifically focuses on work with the MNIST dataset. The paper goes further than we did in its methods of preprocessing, describing the ways in which elastic transformations can provide improved accuracy. Another work which was integral to our exploration was D.Cires's *Multi-column Deep Neural Networks for Image Classification*[2], which discusses the rationale for using neural networks on image classification. Cires explains the ways in which GPUs have revolutionized researcher's ability to train complex models without waiting for incredibly long computational periods. Furthermore, we drew heavily upon the documentation provided by PyTorch. We used many of their built-in library functions to achieve our final accuracy[4]. To see the complete list of relevant works, please the references section on the final page.

3 Dataset & Setup

The training set consists of 40,000 64x64 sized images containing one or more handwritten numbers. Each of these images were labeled by the largest number (by area of bounding box) within that image. The test set contains 10,000 unlabelled images of the same size. In our work, we split the given dataset into a training and validation set (using a proportion of 8:2).

As a pre-processing step, we created a function which normalizes the entire dataset at once (as opposed to batch normalization). In doing so, it calculates the mean to center and standard deviation of all the images. We then apply this mean and standard deviation to normally distribute the images to create consistency among the activation function outputs of each level.

4 Proposed Approach

4.1 Our Model - CNN

In this project we chose to implement a convolutional neural network using the PyTorch package version 1.0, developed by Paszke et al[4]. A CNN is a variation of the neural network which has built-in invariance to certain variations of the inputs. In our case, an example of this is seen as our model generalizes number recognition irrespective of the number's position within the image. The CNN architecture uses characteristics such as local connectivity, parameter sharing, and pooling among the input layers to create this invariance. In our work, we utilized each of these characteristics as described below. To test each characteristic, either two or three trials were performed using different seed values (44, 121, and 379) for splitting the training set. The average of the test loss, training loss (defined below), and accuracy on the validation set between the trials was then taken.

4.2 Convolutional Layer

The primary purpose of a convolutional layer is to extract features from the input image. This is done by multiplying the values in the kernel (weights) with the pixel input values. Then, these multiplications are summed up to give the output value. This is done N times (depending on the kernel size and stride value explained below) to produce a filter matrix, also know as feature map.

4.2.1 Kernel Size

As previously noted, unlike a feed forward neural network, CNN architecture allows for local connectivity among input features. This means that rather than processing the image as a whole, it is processed in patches, called "kernels". The use of local connectivity allows us to extract high level, abstract information by combining information among pixels in a kernel. As such, kernel size has a great impact on the output of this model. We tested kernel sizes of: 3, 4 and 5.

4.2.2 Padding

Padding is used in CNNs to allow the application of kernels near an image's boundary. "Ghost pixels" of value 0 are added around the image boundaries. Without padding, corner pixels of images would hold less weight than their neighbouring pixels due to kernel formation. This creates a potential for data loss and higher inaccuracy of the CNN output. The number of added "ghost pixels" is referred to as the padding. In relation to our kernel sizes, we tested padding values of: 0, 1 and 2.

4.2.3 Stride

The stride value controls the amount the convolutional filter shifts around the input volume. We consistently used a stride value of 1 in all our models.

4.2.4 Layer Number

The number of layers in a neural network is related to the complexity of the function that wants to be represented. Specifically in convolution neural networks, added hidden layers allow features to become more high level and abstract. We experimented with different CNN architectures, using between 3 to 5 convolutional layers in our models.

4.3 Loss Function and Learning Rate

Neural networks are trained using an optimization process which updates the weights on each layer by back-propagation of errors. In calculating the errors of the model during the optimization, a loss function must be used. To trained our CNN, we used a cross entropy loss function to measure these errors, as defined in Appendix B[4]. Furthermore, we chose to use an stochastic optimizer called Adam to update the errors. This method takes as a parameter the learning rate. The learning rate is a hyper-parameter which controls

how weights are updated during training. The algorithm then updates the learning rate individually for each different weight from estimates of the first and second moments of the gradients [3]. When testing our model we tested with three potential learning rates: 0.0004, 0.0007, and 0.001.

4.4 Activation Function

Activation functions serve as thresholds to decide whether or not a neuron should be activated. In this project, we used two types: ReLU and linear functions.

4.4.1 ReLU Function

The main purpose of an activation function is to introduce non-linearity to the system. It is convention to apply them right after a convolutional layer since the convolutional layer computes linear operations. The nonlinear transformations create high levels of abstraction and help the algorithm perform more complex tasks. Moreover, they make back-propagation possible by supplying the gradients with the errors to update their weights and bias. The activation function that we used in our code is called Rectified Linear Unit (ReLU). This is the de-facto standard in deep learning due to the fact that this activation function only maps output positive values and ignores all negative inputs. This allows for faster training than other functions (for example, a sigmoid function).

4.4.2 Linear Function

This type of activation function is usually used at the end of the network and is known as the fully connected layer (FC). This layer takes an input volume (i.e. output of the last convolutional layer or pool layer) and outputs a vector with the same size as the classes that we need to predict (in our case, 10 different classes). Here the activation is proportional to the input.

4.5 Dropout

Dropout is an approach to regularization which prevents interdependent learning between neurons. During training, it randomly zeroes some input values given a probability p using a Bernoulli distribution. In all our models, we used dropouts after every activation function (linear and non-linear). We tested dropout probabilities ranging from 0.0 to 0.6 in steps of 0.1, as suggested by Paszke et al[4].

4.6 Pooling

Pooling, also known as downsampling, reduces the spatial dimension of the input volume. This lowers the computational cost and it is another way to control overfitting. In our code, we use 2×2 maxpooling. This means that it will apply a max filter on a 2×2 region with a stride of 2 (no overlapping regions).

4.7 Batch Sizes

The batch size impacts the CNN training both in terms of the time to converge and the amount of overfitting, i.e., smaller batch size yields faster computation (with appropriate implementations), but requires visiting more examples in order to reach the same error, since there are less updates per training iteration.[5] We experimented with training batch sizes ranging from 512 to 2048 in powers of 2 and used a validation batch size of 1024.

5 Results

5.1 CNN structure

[1]Sundaresan, Vishnu and Lin, Jasper. "Recognizing Handwritten Digits and Characters." Retrieved from CS231n in Stanford University. 2015

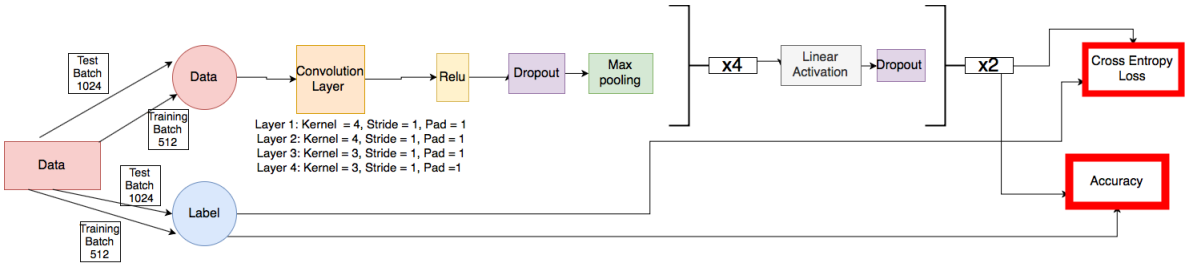

Figure 1: Our Architecture for Best Model[1]

Our first CNN model architecture consisted of a (conv → ReLU → dropout → maxpooling) × 3 ⇒ (FC → dropout) × 2). From this first model, the CNN structure and the hyperparameters were tuned one by one to create our final model. We found that a kernel of 5 and a stride of 1 on each convolutional layer gave the best results, with 87% accuracy on the validation set after 300 epochs. Next, we chose to try a deeper CNN with 4 convolutional layers instead of 3, as shown in Figure 1. After optimization, this model gave 92.8% on the validation set. Finally, we experimented with a 5 layered version of our model but it failed to outperform our previous best model.
In regard to our best model, we used a kernel size of 4 for the first two convolutional layers and a kernel size of 3 for the next two convolutional layers. Moreover, a padding of 1 was used on all convolutional layers to control the dimensionality and prevent data loss.

5.2 Hyperparameters

In tuning the hyperparemeters, we first began by searching for an appropriate learning rate. As mentioned above, we experimented with 3 different learning rates: 0.001, 0.0007 and 0.0004. Figure 2a shows that the test loss converges most rapidly using a learning rate of 0.001. Next, using this learning rate, we tested 3 different training batch sizes: 512, 1024 and 2048. Moreover, we can see in Figure 2b that a batch size of 1024 makes the test loss converge quickly as suggested by Radiuk in 2017 [6].

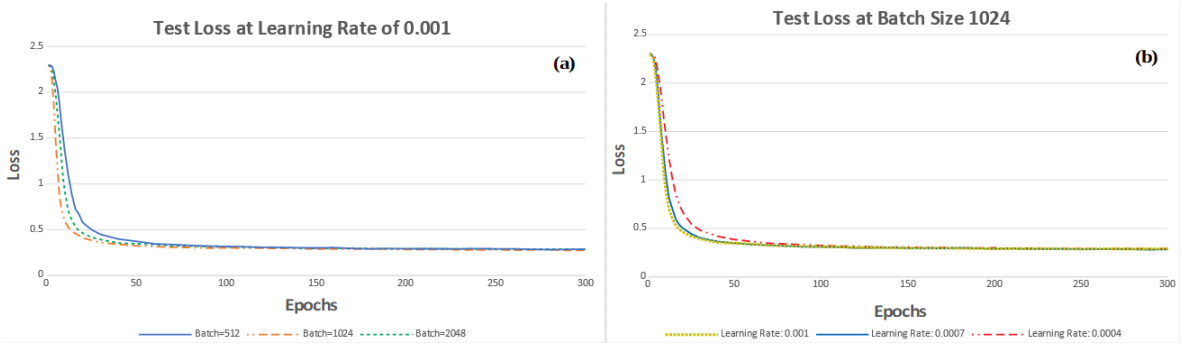

Figure 2: (a) Left: Batch size optimization. (b) Right: Learning rate optimization.

As previously described, we also tested different fractions of dropout: from 0.0 to 0.6 in steps of 0.1 and ran each of those for 20 epochs. We found that there is some increase in accuracy and a drop in the test loss before the trend reverses. The minimum loss was given at a probability of $p = 0.1$ and 0.2. To avoid overfitting, we chose $p = 0.2$ as our dropout probability.

6 Discussion & Conclusion

In this project, we were asked to implement a machine learning model that was able to analyze an image with multiple handwritten numbers and identify the one with the largest bounding box area. The dataset

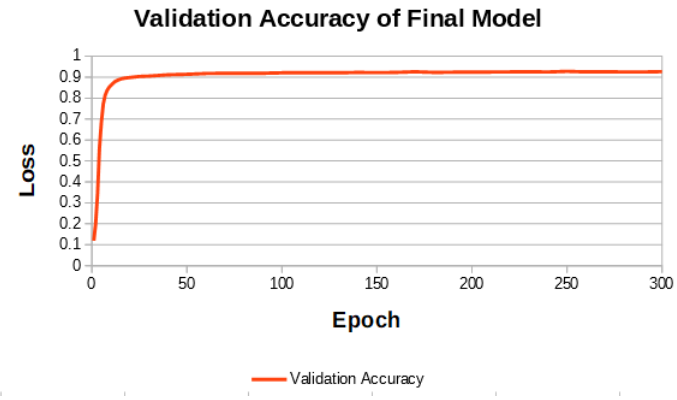

Figure 3: Final model with a maximum validation accuracy of 92.8% after 300 epochs

was taken from a popular dataset called MNIST (which has long been the standard for training handwritten number identification). We chose to implement a convolutional neural network because this model is traditionally successful with image classification. We experimented with a multitude of hyper-parameters, drawing deeply on the options available through the PyTorch package. Our highest accuracy was achieved by a model with padding = 1, stride = 1, dropout = 0.2, batch size = 1024, learning rate = 0.001, kernel size = 4 for the first two hidden layers and 3 for the next two hidden layers, and a total convolutional layer count of four. The epoch evolution of the validation accuracy of this model is shown in Figure 3, while the training and test loss are show in Appendix C. This model achieved a final accuracy of 93.4% on the Kaggle test set using 300 epochs. It is important to note that the training loss was higher than the test loss for the first 50 epochs because the dropout is activated only during training and gets deactivated during evaluation.

All the literature suggests that pre-processing in the form of elastic transformations would improve accuracy achieved by our model. For future work, we would like to experiment with the different methods of elastic transformation and data augmentation mentioned in Simard's *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis*. It would be interesting to see whether our model could also be scaled to identify the dominant color of a number with slight modification.

7 References

1. Simard, Patrice Y., Steinkraus, Dave., Platt John C. "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis." 2003. IEEE Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR) (2003).
2. Cires, Dan, Meier,Ueli, and Schmidhuber, Jürgen."Multi-column Deep Neural Networks for Image Classification." IDSIA-USI-SUPSI. 2012.
3. Kingma, Diederik P., Ba, Jimmy L."Adam: A Method for Stochastic Optimization." ICLR 2015.
4. Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam. "Automatic differentiation in PyTorch." NIPS-W. 2017.
5. Molchanov, Dmitry and Ashukha, Arsenii and Vetrov, Dmitry. "Variational Dropout Sparsifies Deep Neural Networks."arXiv preprint arXiv:1701.05369. 2017.
6. Radiuk, Pavlo M. "Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets." 2017 December 2017, vol. 20, pp. 20–24.
7. Nair, Vinod and Hinton, Geoffrey. "Rectified Linear Units Improve Restricted Boltzmann Machines." ICML.CS1 (2010).

Appendix A Statement of Contribution

The work was split fairly evenly among the group. The main architecture of our CNN model was written by Luis. John and Rebecca participated heavily in training and testing permutations of hyper-parameters. Rebecca took lead on writing the report but both John and Luis contributed large portions.

Appendix B Cross-Entropy Loss Function

The cross-entropy loss function is defined as

$$loss(x, class) = -log(\frac{exp(x[class]}{\sum_j exp(x[j])}) \tag{1}$$

Cross-entropy functions are typically used in classification tasks to quantify a classifier's ability to correctly categorize data examples. This function, as used by PyTorch, generalizes the normal cross-entropy formula to non-binary classification problems. This formula was given in the PyTorch documentation [4]

Appendix C Final Model: Loss Graph

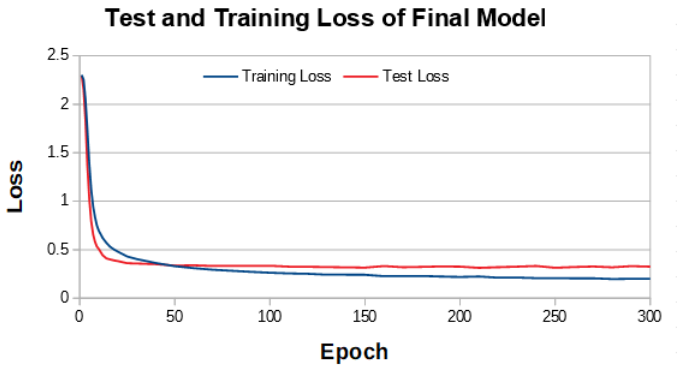

Figure 4: Loss on the final model after 300 epochs

Appendix D Dataset Example

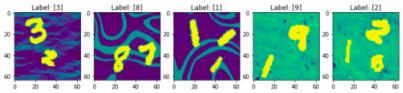

Figure 5: MNIST dataset example, plotted in color for greater readability. Taken from W. Hamilton, "Project 3: Description"