Application of Convolutional Neural Networks to the MNIST Dataset

Ziad Ali
Electrical and Computer Engineering
North Carolina State University
Raleigh, NC, USA
zaali@ncsu.edu

Amit Cudykier
Biomedical Engineering
North Carolina State University
Raleigh, NC, USA
acudyki@ncsu.edu

Ravindersingh Rajpal
Computer Science
North Carolina State University
Raleigh, NC, USA
rkrajpal@ncsu.edu

***Abstract*—A convolutional neural network is applied to the problem of classifying handwritten numerical digits in the MNIST dataset. Hyper-parameters for the network, including network structure, batch size, learning rate, and dropout rate were all trained using cross-validation and random search. The final model achieved an accuracy of 98.5% on the test set.**

I. Introduction

Convolutional neural networks (CNNs) are among the most ubiquitous machine learning frameworks used in image processing. Unlike conventional artificial neural networks, which use vectors to represent input data, CNNs are optimized to analyze data represented in a grid format [1]. The name "convolutional neural network" derives from the fact that a convolution operation is applied to a section of the input grid data at every layer within the network [2].

Convolution is a mathematical operation applied to two functions (x and w) to produce a third function (s) that is sometimes considered a "blend" of the original two functions:

$$s(t) = \int x(a)w(t-a)da \tag{1}$$

For discrete operations (such as with neural networks), the convolution operation is represented as:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2}$$

In a convolutional neural network, a matrix of weights called a "filter" scans over the pixel values of the input images and performs a convolution operation at each point to recreate a new image (shown in Figure 1). Multiple filters, each of which are tuned to elucidate specific visual features, are used to generate multiple output images in the next layer of the network. Once all of the filters have finished convolving the input, a subsampling operation known as "max pooling" is used to reduce the size of the output images by reducing adjacent pixel values to the single largest value within the group. More convolutional filter and max pooling operations are performed at subsequent layers until the initial image has been broken down into a collection of its key features which can then be fed into a neural network as a one dimensional vector. The neural network will then process the input features to, in most cases, classify the image into one of several predefined categories [3].

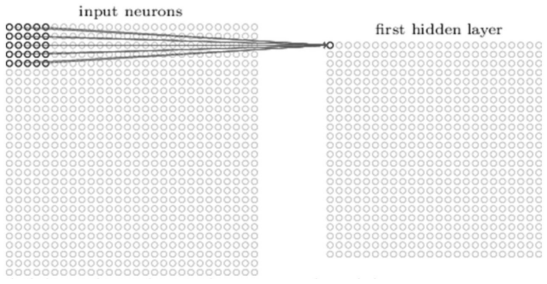

Fig. 1. 5x5 filter convolution and the resultant feature map

In this paper, we apply convolutional neural networks to the problem of classifying handwritten digits in the MNIST dataset. We discuss our approach to training the hyper-parameters for the network and discuss the efficacy of a CNN in comparison to a conventional neural network.

II. Method

While implementing a convolutional neural network, we have many hyper-parameters that can be tuned in order to enhance the accuracy of our model. For each hyper-parameter, we chose 'default' values to first initialize our network and then created a range around those values within which we searched for the optimal final value. Since there is more than one hyper-parameter, training for all possible combinations is not realistic; thus, we performed a random search across our chosen hyper-parameters to narrow down our search radius for each individual hyper-parameter. Once we had narrowed down our hyper-parameter values using a random search, we iterated over the remaining hyper-parameters systematically to pinpoint the ideal region of performance for each.

A. Random Search

The hyper-parameters we considered when training our model consisted of the number of feature maps at each layer of the CNN, the dropout rate for neurons in the neural network that analyzes data after the CNN, the batch size for the training process, and the overall learning rate. We decided early on that the structure of our CNN would consist of two convolutional

layers, one fully-connected hidden neural network layer (with a number of neurons equivalent to the final number of feature maps multiplied by the size of the maps), and one output layer. We did not make the number of layers within our network a hyper-parameter because we observed relatively quickly that high (>97%) validation accuracy values could be achieved without making the network deeper.

We set the number of feature maps at each convolutional layer and the batch size for the training process as discrete random variables. To perform our random search, we chose one of the values from Table I for each hyper-parameter according to a uniform distribution. We set the dropout and learning rates as quasi-continuous random variables as seen in Table II - while they were technically also discrete because there was a limited amount of values that could be chosen, the difference between successive values for each hyper-parameter was small enough so as to somewhat simulate a continuous random variable. Values were selected using a uniform distribution.

TABLE I
DISCRETE HYPER-PARAMETERS

Hyper-Parameter	**Values**			
Number of Feature Maps	(32, 64)	(64, 128)	(16, 32)	(64, 32)
Batch Size	64	128	256	-

TABLE II
QUASI-CONTINUOUS HYPER-PARAMETERS

Hyper-Parameter	Minimum	Maximum	Step
Dropout Rate	0.0	0.5	0.05
Learning Rate (10^x)	-4	-2	0.1

We split the MNIST training set of 55,000 images into 45,000 training images and 10,000 validation images. We created a CNN using TensorFlow for each randomly-selected hyper-parameter combination and trained it for 10 epochs using the training images. We tested 50 different hyper-parameter combinations and, afterwards, assessed which combinations achieved the highest validation accuracy values during training. We considered all ten validation accuracy values generated for each combination (one per epoch), since some models converged faster than others and some began overfitting before training had finished.

After determining which of our discrete hyper-parameter values seemed to perform best, we set those values constant, set the learning rate constant, and swept over a small range of dropout values to determine the optimal dropout rate. We then set dropout constant and swept over a small range of learning rates. The dropout and learning rate ranges were set by observing which dropout and learning rate values performed best in the random search.

Lastly, once every hyper-parameter value had been set, we tested the model using three different activation functions: reLu, tanh, and sigmoid. We selected the best-performing activation function and then retrained our model over the entire training dataset (no validation set).

III. RESULTS

As described in the previous section, we collected validation data from the random search and plotted it to have a clear idea of how the CNNs perform with respect to different hyper-parameter combinations for the model. We initialized our model to use the ReLU activation function. Figure 2 displays three different training and validation accuracy curves for three different random hyper-parameter combinations, while Figure 3 displays three different training and validation loss curves.

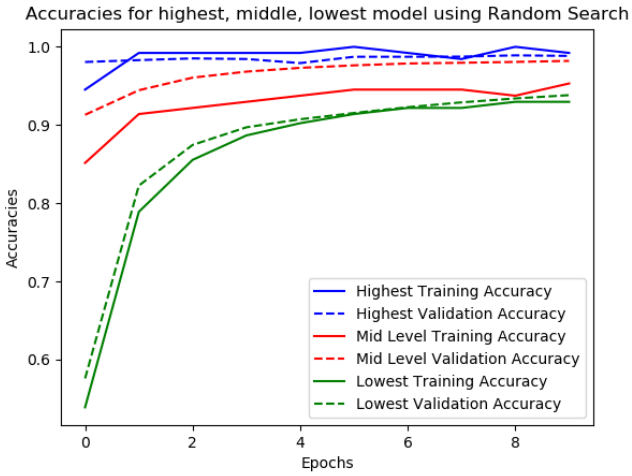

Fig. 2. Random search accuracy results. Training and validation accuracy values over 10 epochs plotted for the best and worst performing hyper-parameter combinations (as well as one with average performance).

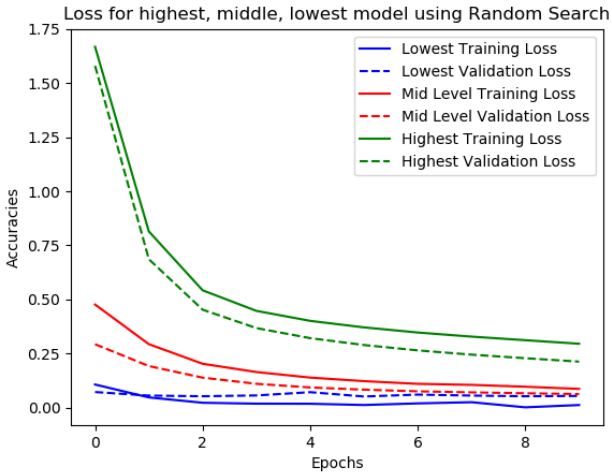

Fig. 3. Random search loss results. Training and validation loss values over 10 epochs plotted for the best and worst performing hyper-parameter combinations (as well as one with average performance).

After studying the plots, we decided to fix the number of feature maps of the first layer to 32 and the second to 64. We

also set the batch size to 128. These two values corresponded to the values present in the most accurate model as well as half of the top 10 most accurate models assessed during random search. We then searched through dropout and learning rates.

First, we evaluated the effect of a varying dropout rate while the learning rate was kept constant at 1e-3. The range for the dropout rate was changed to 0.1 - 0.3 with a 0.01 step to correspond with the highest performing dropout rates seen during random search. The validation set accuracy with respect to dropout is shown in Figure 4 and the validation set loss with respect to dropout is shown in Figure 5. The accuracy and loss values plotted are the maximum and minimum achieved, respectively, during training over 10 epochs. As shown, the maximum validation set accuracy was achieved at a dropout rate of 0.1.

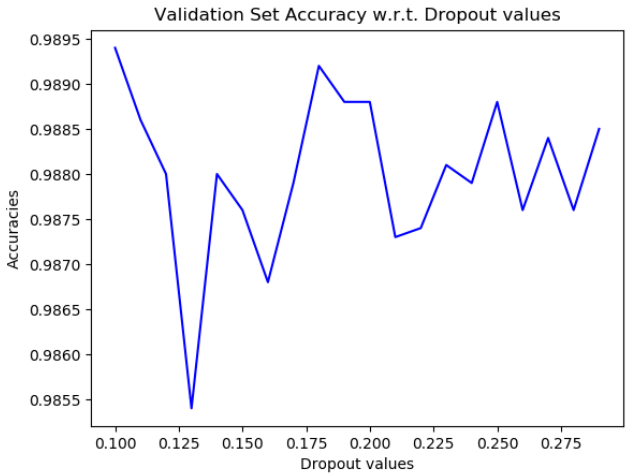

Fig. 4. Validation set accuracy with respect to dropout.

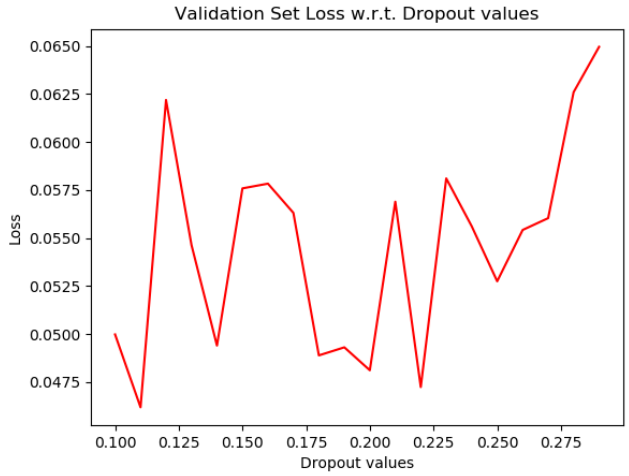

Fig. 5. Validation set loss with respect to dropout.

Next, we kept dropout constant at 0.1 and searched to find the optimal learning rate within the range of 1e-3 to 1.1e-2 with a step of 3e-4. While a learning rate of 3e-3 performed best for the model, we later determined that this learning rate was too high for a CNN using the sigmoid activation function to converge, so we reduced it to 1e-3 (almost equivalently high-performing) to test different activation functions.

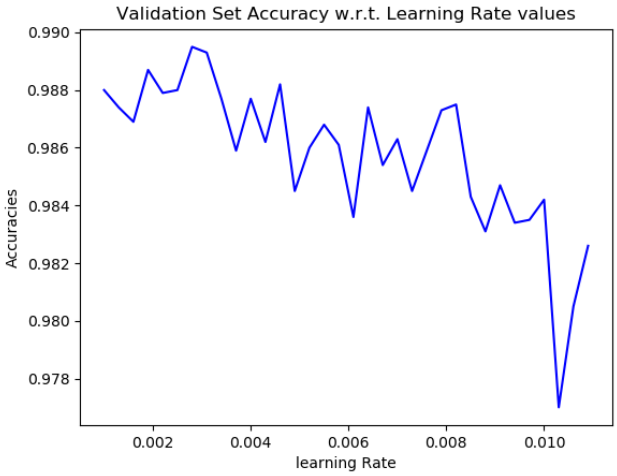

Fig. 6. Validation set accuracy with respect to learning rate.

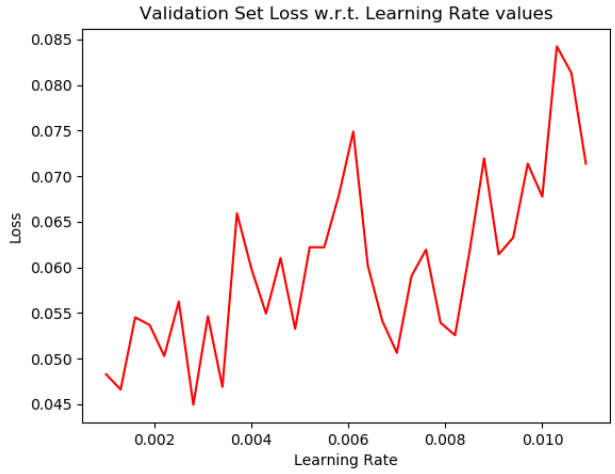

Fig. 7. Validation set loss with respect to learning rate.

After studying the plots for different dropout and learning rate values, we decided on all of the hyper-parameters except the activation function.

Feature Maps	[32, 64]
Dropout	0.1
Learning Rate	0.001
Batch Size	128

Lastly, we tested the model after keeping the hyper-parameters mentioned above with respect to different activation functions, namely ReLU, sigmoid, and tanh. Figure 8 shows validation accuracy over 10 epochs for the different

functions, while Figure 9 shows validation loss. While tanh did converge faster than ReLU, the ReLU CNN had a slightly higher validation accuracy, so we decided to use it in our final model.

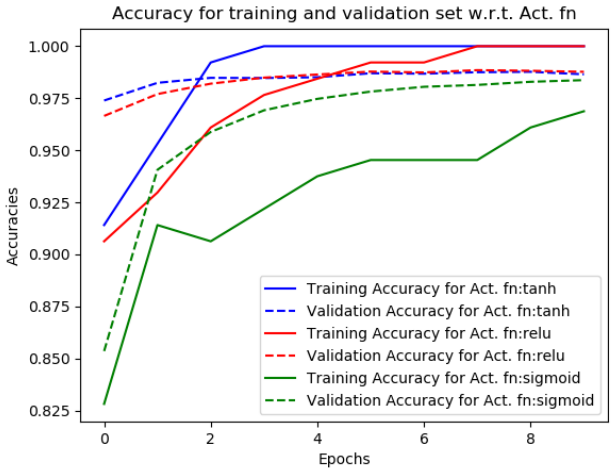

Fig. 8. Training and validation accuracy with respect to activation function (ReLU, sigmoid and tanh).

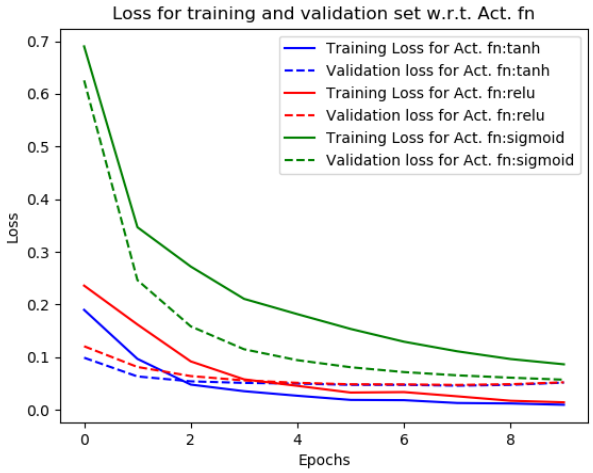

Fig. 9. Training and validation loss with respect to activation function (ReLU, sigmoid and tanh).

After searching through possible values of hyper-parameters and studying all of our plots, we decided on the final set of hyper-parameters for our CNN.

Conv. Layers	[32, 64]
Dropout	0.1
Learning Rate	0.001
Batch Size	128
Activation Function	ReLU

We also plotted the learning curve of the final model (with the aforementioned hyper parameters) to have a better understanding of which epoch value to stop training the model at. Figure 10 plots training and validation accuracy for the final model and Figure 11 plots training and validation loss. We decided to stop training the model after epoch 5. We then trained the CNN with the full training set (training + validation) for 5 epochs with the aforementioned hyper-parameters and arrived at a final accuracy of 98.5% and a loss of 0.04321.

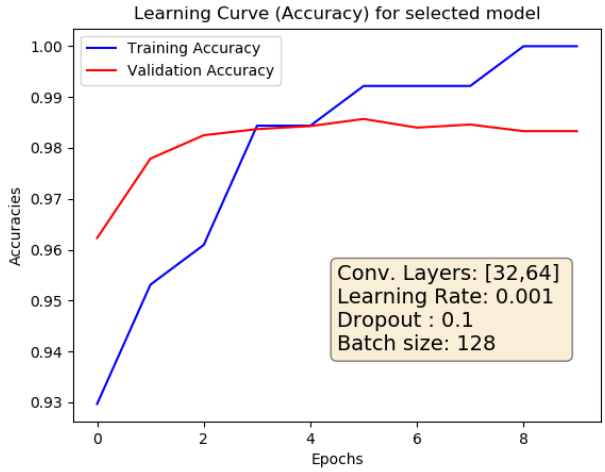

Fig. 10. Training and validation accuracy for the final combination of hyper-parameters.

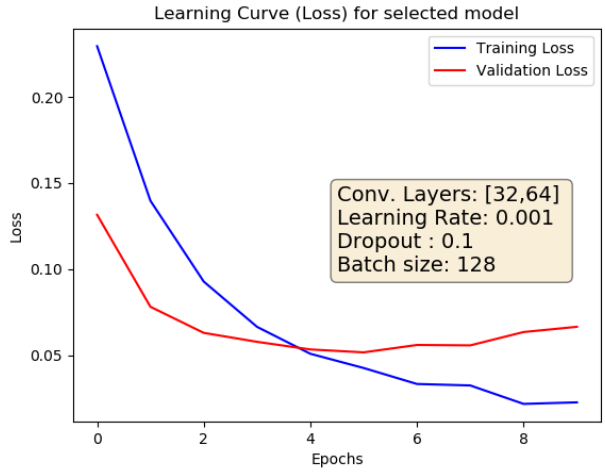

Fig. 11. Training and validation loss for the final combination of hyper-parameters.

IV. Conclusion

Convolutional neural networks are powerful machine learning tools that are especially adept at analyzing and classifying images. They operate by convolving a multi-dimensional filter over a subsection of grid-like input data to extract ”features” from the image. The weights within these filters are learned

throughout the training process to enable more accurate extraction of features that highly correlate with certain classifications. Subsampling (pooling) operations are applied after each convolution to reduce the size of the input data, until eventually the grid-like data can be flattened and fed directly in to a standard neural network. The neural network's weights and biases are then learned normally to minimize the network's loss function, and dropout can be added to the network to mitigate potential overfitting.

We developed a convolutional neural network using the TensorFlow machine learning framework to classify handwritten digits from the MNIST dataset. Our CNN had two convolutional layers, one fully-connected neural network layer, and an output layer. The number of outputs for each convolutional layer, the dropout rate for the fully-connected neural network layer, the batch size for the training process, and the overall learning rate were all trained as hyper-parameters. Our network achieved an accuracy of 98.5% on the test data.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016.

[2] Convolutional Neural Networks (CNN): Step 1- Convolution Operation. SuperDataScience - Big Data — Analytics Careers — Mentors — Success, SuperDataScience, 17 Aug. 2018.

[3] Convolutional Neural Networks (CNNs / ConvNets). CS231n Convolutional Neural Networks for Visual Recognition, Stanford, May 2018.