MiniProject 3 - Modified MNIST*

Sam Cleland (260675996), Kareem Halabi (260616162) and Matthew Lesko-Krleza (260692352)

***Abstract*— The work attempted during this project was to be able to classify the digit that occupies the most space (defined by largest bounding box) in an image that may contain multiple digits. This is a twist to the original MNIST dataset as there are multiple digits within one image versus classifying a single digit. We tackled the problem by developing a 4 layer convolutional neural network, which yielded a 96.9% accuracy on Kaggle. Analysis of our design choices such as the number of layers, dropout rates, length of training, number of convolutions and activation functions will be discussed in detail.**

I. Introduction

Handwriting recognition is the ability of a computer or device to interpret handwritten input from sources such as printed physical documents, pictures and other devices [1]. From a user's perspective, applications for this include developing algorithms to convert handwritten notes to digital text automatically [2]. For this task, we trained a convolutional neural network by experimenting with different architectures. This involved varying the number of layers, activation functions and the dimension / number of convolutions being performed at each layer. Our best performing model consisted of 4 layers with batch normalization and ReLU activation at the end of each layer.

A. Preliminaries

Neural Networks (NN) are systems that learn how to classify the input they are given correctly. This is done by stacking multiple models in the form of hidden units and training them jointly. They can also be referred to as Deep Neural Networks (DNN) if they have one or more hidden units.

Convolution is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other, can be visualized as sliding one function across the other and the output being the multiplication of the two functions at each time step.

Convolutional Layer is a layer that has a filter with specific dimensions that "slide"/convolve with batches of the data according to the filter's size. This allows for the layer to extract patterns from the input.

Convolutional Neural Networks (CNN) are deep neural networks that use convolutional layers.

Epochs are the number of iterations during neural network training.

Activation Functions are used to indicate if a neuron should be "activated or not" based on a non linear transformation of the output at the previous layer. Popular transformations include the sigmoid function ($\sigma = 1/(1 + e^{-x})$) or Rectified Linear Unit (ReLU $= \max(0, x)$).

*Presented by Group 90 for Applied Machine Learning (COMP 551)

Dropout regularization associates a probability of setting a particular node to 0 for each node in a particular layer, a technique useful to prevent overfitting [3].

Max Pooling is the idea of taking the maximum of different elements within a layer, similar to the convolutional filter where we slide across the input with an n by n pooling filter and perform some non linear operation with that n by n patch of the image. This is done to reduce dimensionality of hidden layers, resulting in higher level feature abstraction.

Batch Normalization is a technique to improve training efficiency by normalizing data for each training batch at each layer of the network. The mean and variance are independently calculated for each dimension [4].

Softmax maps vectors of values to a vector of probabilities. Useful when applying a softmax to the last layer of a NN as it is compatible with cross entropy loss and max likelihood.

Cross Entropy Loss Function is a measure of how many bits an encoding will require if we use information from the neural network's output compared to the actual predicted values. Cross-entropy loss increases as the predicted probability diverges from the actual label.

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent for scaling the learning rate for each parameter based on statistics of the history of gradient descent. For example, if a parameter wasn't updated very much in a previous iteration, increase the update strength for that particular parameter for the next iteration [5].

Early Stopping refers to terminating a training routine once validation error has reached a minimum to avoid overfitting.

Law of Diminishing Returns is a concept that refers to a point in which a benefit gained (in NN context, model accuracy) is less than the effort invested (in NN context, training time)

II. Related Work

An interesting finding on a related work for the MNIST dataset is a paper written on a multi-column CNN for image classification [6]. The paper demonstrates a new architecture approach by combining multiple deep neural network's layers to form a Multi-Column CNN. The performance increase is significant and is highlighted by comparing a single-column CNN's error rate of 0.40% to a Multi-Column CNN's error rate of 0.23%. Another interesting finding is one that demonstrates how Recurrent Neural Networks provide a valid alternative to CNNs [7]. In this paper, the RNN called ReNet performs at a Test Error rate of 0.45%, which is

highly comparable to its competing CNNs with Test Error rates of 0.40% [8]. Finally, we'll be referring to several papers such as that on *Wide Residual Networks* [9] and on *Training longer, generalize better* [10]. Significant increases in performance on the test set were seen even when validation error didn't change over long training durations.

III. Dataset and Setup

In this project we have been provided with a dataset of 40,000 training images each with a label representing the largest digit in the image between 0 and 9. Each image has a resolution of 64 x 64 pixels where each pixel has a single grayscale channel with a value between 0 and 255.

As this project is an adaptation from the original MNIST recognition paper [11], there are a two key differences between the original dataset and the one we have been provided with. First is the 64 x 64 resolution of our images is larger than the original 28 x 28 pixels, likely due to the fact that the resolution may be too low to fit multiple distincly identifiable digits. In addition, it appears that some noise has been added to the background of the images in our provided dataset which is not present in the original images. A comparison between samples of the two datasets is shown in Figure 1.

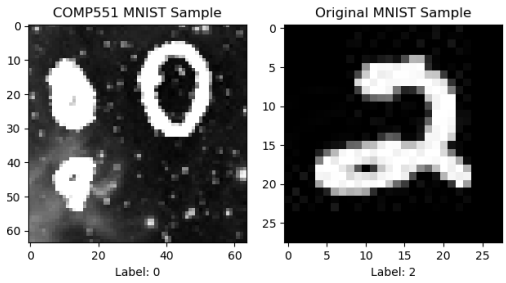

Fig. 1. Comparison of Original MNIST sample (right) and the COMP551 MNIST sample (left)

Our only form of preprocessing was to normalize the pixel values to a mean and standard dev of 0.5 as this yielded better results than the raw pixel values.

IV. Proposed Approach

A. Libraries Used

The PyTorch DNN framework was used for designing and training DNNs [12]. Scikit Learn's metrics.accuracy_score function was used to compute classification accuracy [13]. Numpy, Pandas, and Pickle were used for loading and manipulating data [14]. Finally, matplotlib's pyplot library was used for plotting data and results [15]. You'll see many of PyTorch's functions, such as Conv2D(), BatchNorm2D(), MaxPool2D(), Dropout(), Linear(), CrossEntropyLoss(), and Adam(). Conv2D() applies a convolution on a 2D input channel and outputs signals by a given output dimension. BatchNorm2D() applies a Batch Normalization on a 2D input given an input dimension size. MaxPool2D() applies Max Pooling on a 2D input given a kernel size. Dropout() randomly sets a value to 0 (drops) by a given probability. Linear() applies a linear function on an input by a given input dimension and outputs signals by a given output dimension. CrossEntropyLoss() applies a loss to a given output for classification. optim.Adam() applies the Adam algorithm for stochastic optimization.

B. Architecture

It was known when starting this project that CNNs are the *de facto* models for computer vision tasks. Nevertheless, we did some research to explore whether other high performing solutions existed for the MNIST dataset. Interestingly, Recurrent Neural Networks posed a viable alternative to Convolution Neural Networks [7]. However, given our lack of experience in neural networks and desire to learn more about convolutional layers, we decided to opt for a CNN architecture.

We took inspiration from a successful model submitted for the MNIST public competition on Kaggle as shown in Figure 2 [16].

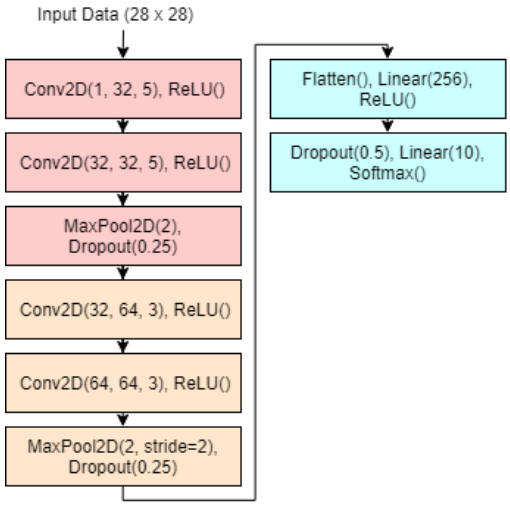

Fig. 2. Inspired CNN Architecute

This architecture consists of an input layer with two 2D-Convolutional filter layers with ReLU activation followed by one MaxPool2D and one Dropout layer. The hidden layer is identical to that of the input layer with the exception that the channels for the convolutional filters are of 32 by 64 and 64 by 64 respectively. The fully connected layer consists of a flattening filter, followed by a linear function layer with ReLU activation and a dropout filter, and ends with a linear function layer with softmax activation.

We drew inspiration from that model since it has a 99.7% accuracy on the MNIST competition. The architecture was adjusted by adding two additional layers, each with two 2D-Convolutional filters each with channels of (64x128),

(128x128), (128x256), and (256x256) respectively. However the Softmax layer was removed as the performance degraded when it was included. We used Cross Entropy (nn.CrossEntropyLoss()) as our loss function and Adam (optim.Adam()) for optimization. These criterion and optimization functions were selected because CrossEntropyLoss is popular for classification problems and Adam is a fast and popular optimizer.

The architecture is illustrated in Figure 3. Each layer is denoted by a single color, and composed of one or more blocks. For the sake of brevity, multiple filters may be included within a single block. Please note that the Flatten() filter is a call to PyTorch's reshape(size, -1) function that flattens an array from 2D to 1D where size is the current size of the image within the network.

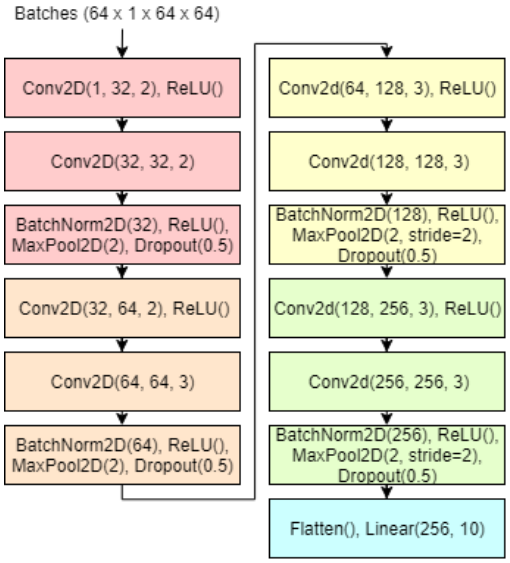

Fig. 3. CNN Architecture

We could afford to add these additional layers as our input image dimensions are quite low at 64x64 pixels, and because we used Kaggle's online kernels and GPU instances for training [17]. However, since additional depth was added, the kernel sizes had to be decreased from 5x5 to 2x2 or 3x3, otherwise we would get a 'CUDNN_STATUS_BAD_PARAM' error describing that computation was attempted on NULL values. This meant that the convolution kernel was too large and reducing the image size within the network to 0. We wanted the first two convolution layers (red and orange colored layers) to extract low level features, hence the small kernel dimension and stride values of 2 and 1 respectively. We increased the convolution kernel sizes from 2 to 3 and the MaxPool stride from 1 to 2 in the 3rd and 4th convolutional layers (yellow and green colored layers), because we believed this design would allow us to extract high level features from rich low level features extracted from the previous layers. We used ReLU as an activation function because it helps speed up gradient descent as opposed to a tanh or sigmoid function. Furthermore, since we're only using positive numbers to describe pixel intensity, we don't lose precision by setting negative values to 0 which would effectively produce 'dead neurons' thereby making ReLU a great choice for the activation function. We trained on batches of 64 images as it is an asequately sized batch that is a multiple of 2, making it possible to train faster on a GPU.

Out of curiosity, we experimented with this architecture at different depths and dropout rates. Therefore, we trained multiple CNN models each with different depths, either using three or four 2D-Convolution layers. We believed that increasing depth would increase the complexity of our model and potentially increase overfitting. Additionally, we suspected that the model with four 2D-Convolution layers would only perform slightly better than a model with three 2D-Convolution layers on the validation set as highlighted by the law of diminishing returns. Furthermore, we wanted to test to see how increasing dropout would achieve a more generalized model. We hypothesized that increasing dropout from 0.1 to 0.5 would increase the training loss but increase performance on the validation and test sets. Our results are summarized within the Results Section V.

V. Results

A. Variable Dropout and Depth Results

We can see the results of our dropout and number of layer experiments within Table I. The notation is the following: $< dropout\ value > D < number\ of\ layers > L$. So a model characterized as 0.10D 3L has a dropout of 0.10 for each dropout filter and a total of 3 convolution layers. Each model was trained for 20 epochs on 90% of the training set The Validation Score (higher is better) is the classification accuracy score on the 10% of the training set left out during training. The Loss metric (lower is better) is the model's final loss value after its last training epoch. In bold you'll find the best performing metric for its respective column.

Model	Validation Score	Loss
0.10D 3L	0.9345	0.0355
0.25D 3L	0.9367	0.1316
0.50D 3L	0.9320	0.1793
0.10D 4L	0.9385	**0.0055**
0.25D 4L	0.9483	0.0357
0.50D 4L	**0.9490**	0.4598

TABLE I

CNN Architecture Validation Score and Loss Metrics on Modified MNIST with Varying Dropout and Number of Layers

A quick glance in Table I shows that the 0.50D 4L model performs best on the validation set. Most notably, this model also has the highest final loss out of all experiments, with a value of 0.4598. We can also compare models 0.10D 4L, 0.25D 4L, and 0.50D 4L and their metrics to realize that as the dropout rate increases, Loss may increase but so do their Validation Scores. We can also compare 0.50D 3L with 0.50D 4L and realize that as we've increased depth, the Validation Score increases by roughly 1.7% which is significant enough to be considered as a viable design choice to increase performance.

Model	Epochs Trained	Test Score
0.50D 4L	250	0.9410
0.50D 4L	500	**0.9693**

TABLE II
0.50D 4L MODEL TEST SCORES ON MODIFIED MNIST

B. Test Set Results

We chose the 0.50D 4L model's architecture and parameters for the Kaggle public competition. Our best model's test set scores can be seen within Table II. In bold you'll find our Kaggle leaderboard accuracy. Their losses across multiple epochs can be viewed with Figures 4 and 5.

The 0.50D 4L model was trained on the entirety of the training set in two separate instances. We believed that given more data, the model would perform even better on the test set. One was trained for 250 epochs and the other for 500 epochs. We thought that training a model for any longer than 250 epochs wouldn't help us achieve a significant increase in performance, however Table II suggests otherwise.

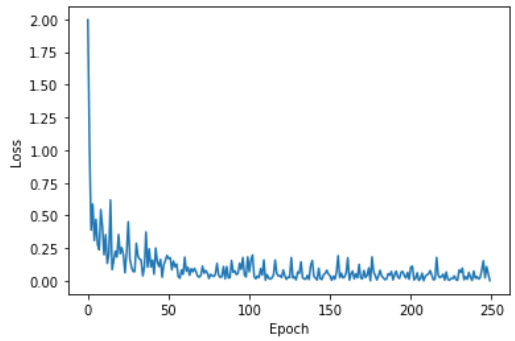

Fig. 4. 0.50D 4L Model Loss Across 250 Epochs of Training

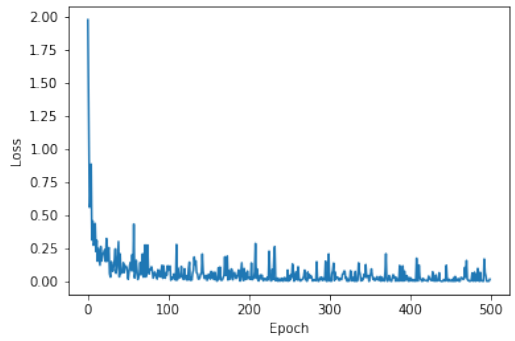

Fig. 5. 0.50D 4L Model Loss Across 500 Epochs of Training

When trained for 500 epochs, our model achieves a Test Score of 96.93% whereas the model trained for 250 epochs achieves a Test Score of 94.10%. This is interesting because Figures 4 and 5 show that after roughly 100 epochs, our loss stabilizes suggesting that there is little to gain in performing additional rounds of training. It's common to perform "early-stopping" to prevent overfitting, but our results suggest that it may be valuable to train for long periods even when loss isn't decreasing. A recent paper on long training periods suggests that "good generalization can result from extensive amount of gradient updates in which there is no apparent validation error change and training error continues to drop, in contrast to common practice" [10].

VI. DISCUSSION AND CONCLUSION

To conclude, we've experienced how dropout can achieve a more generalizable model, as seen when comparing the validation performance of our models with dropout rates of 0.1 and 0.5 confirming our hypothesis on that matter. Furthermore we've seen the law of diminishing returns in effect in neural network depth. After 20 epochs of training, we saw our models with three convolution layers perform with scores of roughly 93% and the models with four convolution layers perform with scores of roughly 94%. Finally, contrary to the popular practice and belief in "early-stopping", our test set results suggest that training for significantly longer durations could be a sound design choice for increasing performance even when there is no change in validation score or loss. This behavior is visible on the training curves of some state-of-the-art networks, such as Wide Resnets [9]. However, there is no theoretical understanding of this phenomenon [9] and further investigation is possible.

VII. STATEMENT OF CONTRIBUTIONS

Kareem: Wrote Dataset and Setup section, helped with finding citations for intro definitions and review of entire report

Sam: Wrote abstract and introduction sections and helped with analysis of neural network architecture.

Matthew: Wrote code for loading data, training neural networks, and making predictions on the validation and test sets. Wrote the proposed solution, results, and discussion sections.

REFERENCES

[1] N. N. B. A. S. Norhidayu binti Abdul Hamid, "Handwritten recognition using svm, knn and neural network," accessed: 2019-03-15.

[2] N. C. Hudges, "How handwriting recognition technology improves productivity on-the-go," 2016.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[6] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *CoRR*, vol. abs/1202.2745, 2012. [Online]. Available: http://arxiv.org/abs/1202.2745

[7] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. C. Courville, and Y. Bengio, "Renet: A recurrent neural network based alternative to convolutional networks," *CoRR*, vol. abs/1505.00393, 2015. [Online]. Available: http://arxiv.org/abs/1505.00393

[8] P. Simard, D. Steinkraus, and J. Platt, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. proceedings of the 7th international conference on document analysis and recognition," *Scientific Research*, vol. abs/1202.2745, 2003. [Online]. Available: https://doi.org/10.1109/icdar.2003.1227801

[9] S. Zagoruyko and N. Komodakis, "Wide residual networks," *CoRR*, vol. abs/1605.07146, 2016. [Online]. Available: http://arxiv.org/abs/1605.07146

[10] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," *arXiv e-prints*, p. arXiv:1705.08741, May 2017.

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

[12] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–, [Online; accessed ¡today¿]. [Online]. Available: http://www.scipy.org/

[15] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[16] "Introduction to cnn keras - acc 0.997 (top 8%)," https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6, accessed: 2019-03-15.

[17] "Kaggle kernels documentation," https://www.kaggle.com/docs/kernels, accessed: 2019-03-15.