CSCI 865 Project Report: Classify Handwritten Digits using Kaggle MNIST Dataset

Ryan Chui
Rochester Institute of Tech, NY

1. Introduction

The goal of this project is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively from a dataset size of 60000 square with the size of 28 by 28 pixel grayscale images. This report presents the implementation and result of a classical convolutional neural network (CNN) using TensorFlow. Other methods that I had explore is Fully Connected Neural Network (DNN) for measuring and comparing the effects of using different activation functions and number of dense layers. The training and testing accuracy using the CNN architecture with regularization slightly outperformed without regularization for the MNIST Dataset.

2. Motivation & Background

Handwritten digits are not always of the same size, thickness, orientation, and position. There are many real-world applications involve the use of online handwritten recognition such as processing mortgage payment by signature, bank checking amounts, numeric entries in forms filled up by hand. The needs to recognize handwritten digits has increased over year since the digital revolution. As a result, our goal was to implement a pattern classification method to recognize the handwritten digits provided in the MNIST dataset of images of handwritten digits from 0-9. The problem that I predict that I would face in this digit classification problem was identify any similarity between digits like 1 and 7, 5 and 6 and 3 and 8, and 8 and 9 etc. Additionally, people can write digits in different ways, like '1' as 'l', and similarly, 2 may be written as '□' or 'Z'. Finally, the uniqueness and variety in the handwriting of different individuals also influences the formation and appearance of the digits.

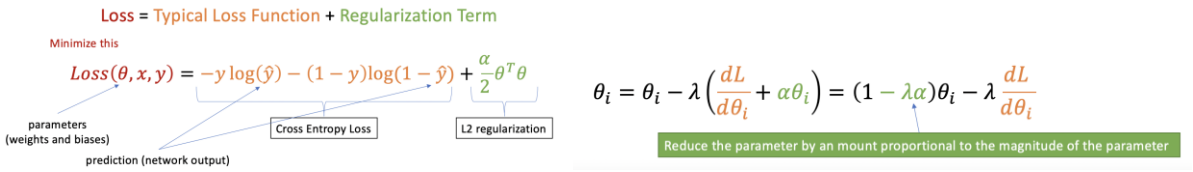
3. Related Work

Handwriting recognition has already achieved impressive results using shallow networks [1–10]. Many research papers have been published with research detailing new techniques for the classification of handwritten numerals, characters and words. In recent years, the convolutional neural networks have been effectively used for handwritten digit recognition and primarily for benchmark MNIST handwritten digit dataset. Most of the experiments achieved high recognition accuracy more than 98% or 99% [11]. The high recognition accuracy of 99.73% on MNIST dataset is achieved by combining multiple CNN's in an ensemble network [12].

4. Proposed Methodology

Classification of images is used in various applications, ranging from facial recognition to self-driving cars. ConvNets are current state-of-the-art models for object classification. In this exercise, we propose to classify digits using hyperparameters optimization methods and regularization techniques with CNN. *We will investigate the performance of un-regularized and regularized neural network*. For regularized neural network, the problem of overfitting when training neural networks can be addressed with regularization. Regularization methods like weight decay provide an easy way to control overfitting for large neural

network models which leads to better generation, and it's known as L2 regulation. The idea of a L2 regularization is by adding a term to the error function used by the training algorithm. The additional term penalizes large weight values and the two common error functions used in neural network are squared error and cross entropy error. The output from the neuron is a=σ(z), where z=$\sum_j w_j x_j$+b is the weighted sum of the inputs. We can then define the cross-entropy cost function for this neuron by $C = -\frac{1}{n}\sum_x [y ln a + (1-y) ln(1-a)]$ where n is the total number of items of training data, the sum is over all training inputs, x, and y is the corresponding desired output. As a result, our aim is to minimize the loss function to enhance the accuracy of the model for better predictions, and we need to take the derivative of this new loss function to see how it affects the updates of our parameters



Before regularization, we first implemented the backpropagation algorithm to compute the gradients for the parameters for the un-regularized neural network. For un-regularized neural network, we will specifically investigate the cost function. Suppose a cost function is given as

$$C(\hat{\theta}) = -\ln P(\mathcal{D} \mid \hat{\theta}) = -\sum_{i=1}^{n} y_i \ln[P(y_i = 0)] + (1-y_i)\ln[1 - P(y_i = 0)] = \sum_{i=1}^{n} \mathcal{L}_i(\hat{\theta}).$$

The last equality means that we can interpret our cost function as a sum over the loss function for each point in the dataset $L_i(\hat{\theta})$ The negative sign is to minimize a positive number, rather than maximizing a negative number. If $\hat{x}_i$ is the i-th input (image), y_{ic} refers to the c-th component of the i-th output vector $\hat{y}_i$. The probability of $\hat{x}_i$ being in class c will be given by the SoftMax function:

$$P(y_{ic} = 1 \mid \hat{x}_i, \hat{\theta}) = \frac{\exp((\hat{a}_i^{hidden})^T \hat{w}_c)}{\sum_{c'=0}^{C-1} \exp((\hat{a}_i^{hidden})^T \hat{w}_{c'})},$$

which reduces to the logistic function in the binary case. The likelihood of this C-class classifier is now given as:

$$P(\mathcal{D} \mid \hat{\theta}) = \prod_{i=1}^{n} \prod_{c=0}^{C-1} [P(y_{ic} = 1)]^{y_{ic}}.$$

Again, we will take the negative log-likelihood to define our cost function $C(\hat{\theta}) = -\log P(\mathcal{D} \mid \hat{\theta}).$

Once the gradient is computed, we then train the neural network by minimizing the cost function using gradientdescent optimizer with dropout. The dropout procedure is like averaging the effects of a very large number of different networks since different networks will overfit in different ways, and ideally the net effect of dropout will be to reduce overfitting. This can force the network to have redundant representation, preventing co-adaptation of features. Our proposed architecture of network is in below with

regularization, and an ANN visualizer is created and attached in the file by using python's graphviz library.

- (Input) -> [batch_size, 28, 28, 1] >> Apply 8 filter of [5x5]
- (Convolutional layer 1) -> [batch_size, 28, 28, 8]
- (ReLU 1) -> [?, 28, 28, 8]
- (Max pooling 1) -> [?, 14, 14, 8]
- (Convolutional layer 2) -> [?, 14, 14, 16]
- (ReLU 2) -> [?, 14, 14, 16]
- (Max pooling 2) -> [?, 7, 7, 16]
- [fully connected layer 3] -> [1x128]
- [ReLU 3] -> [1x128]
- [Drop out] -> [1x128]
- [fully connected layer 4] -> [1x10]

5. Image preprocessing steps and Classification Techniques

MNIST database (Modified National Institute of Standards and Technology database) contains a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image into a 20x20 pixel bounding box and anti-aliased, which introduced gray-scale levels. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. The data has already been split into training, validation, and test sets, which makes them easy to use. (LeCun et al., 1998b) The MNIST dataset is provided by Keras, and the shape of X_train is (60000, 28, 28) with each image being 28 x 28 resolution. The shape of X_test is (10000, 28, 28). CNN requires input shape to be in specific format, thus reshaping the input is required. Since all images are in grayscale, the number of channels is 1. If they were color images, the number of channels would be 3 (R, G, B). We've rescaled the image data so that each pixel lies in the interval [0, 1] instead of [0, 255]. It is a good idea to normalize the input so that each dimension has approximately the same scale. Next, we need to one-hot encode the labels on testing data for Y_train and Y_test, as an integer will be converted to an array which contains only one '1' and the rest elements are '0'.

A. Un-regularized neural network model (Simple Multi-layer Perceptron)

The pixels in the 28X28 handwritten digit training images are flattened to form an array of 784 pixel values, from decompressing a 2D array into 1D vector by using 'Flatten'. We set an input layer with 784 neurons, a first hidden layer (linear) of 250 neurons and a second layer of 100, with an output layer with 10 neurons since we are given 10 categories classification task. We process 100 images at a time in each iteration that is why we set the batch size as 100, although it can be any other number less than the total number of images. The weights are initialized with small values distributed around zero, drawn from a uniform or normal distribution. The bias weights $\hat{b}$ are often initialized to zero, but a small value is added to this model 0.01, ensuring all neurons have some output which can be backpropagated in the first training cycle. Next, we

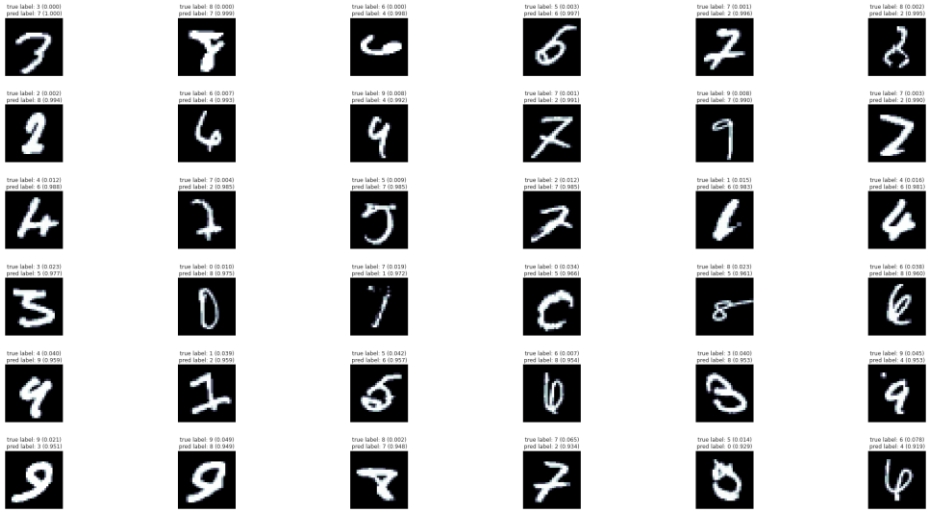
define a helper function to define a network, and our activation being used is ReLu with a softmax cross-entropy**.** Notice that sigmoid works better for only binary classification problem, and we have a multi-label classification problem and thus SoftMax loss function is being used to handle classification with more than two classes. Additionally, we set the ADAptive Momentum estimate (Adam) equal to 0.01 as the optimization here. It is a stochastic gradient descent algorithm that can update the learning rate adaptively by itself for each parameters and it is an upgraded optimization algorithms from Ada Grad. In short, we used backpropagation to adjust the weights of neurons by computing the gradient of the cost function. It starts from the output layer and the errors are propagated back to the first layer, so that neurons adjust their weights in a way that reduces the error from previous iteration. The number of iteration (num_iter) is set to 10.

B. Regularized neural network model using CNN

Next, We implemented a simple Convolutional Neural Network regularization with Dropout. The input image is 28 pixels by 28 pixels. The first dimension is the batch number of the image, and can be of any size (so we set it to -1). The second and third dimensions are width and height (28 by 28), and the last one is the image channels which is is 1 for grayscale. The Size of the filter/kernel is 5x5, and we need 32 different feature maps meaning 32 different filters are applied on each image. The output of the first convolution layer would be 28x28x32. We then apply ReLu activation function f(x) = max(0, x) and max pooling with kernel size 2 x2 before completion of first layer. For Layer 2, we then applied the same techniques as layer 1, but the image filter has increased from 32 to 64. After second layer is completed, we need to convert them to a flat array with size of 128, and applied a dropout layer before measuring the error at a softmax layer. The dropout layer is extremely important because this can reduced the network from overfitting.

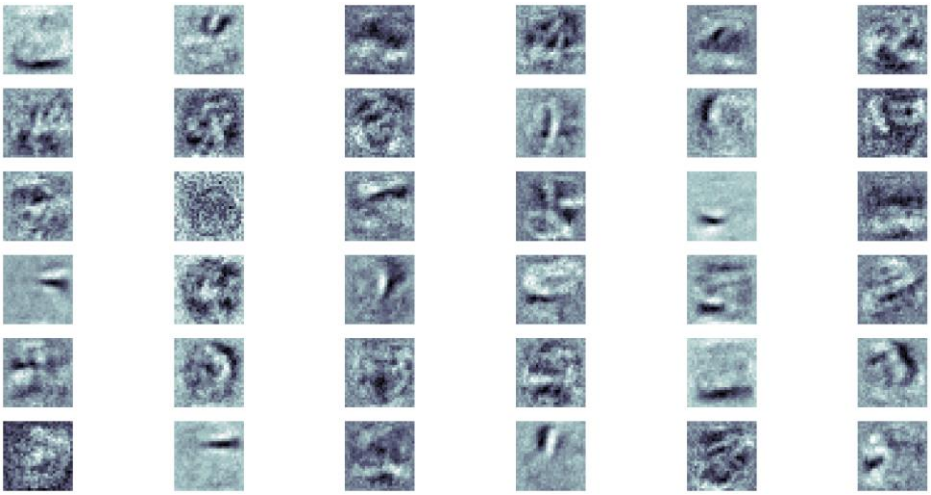
6. Experimental Setup & Performance Evaluation

A. Latent representations (filters) you generated for classification



Above 36 images are the digits that are being incorrectly predicted by the model. A lot of these digits are irregular so it's difficult for the model to do well on these. It is also

possible that the model is overfitting on the training set - that there is a common pattern in handwritten threes within the training set but it's not the pattern we want our model to learn. Here is the latent representation that is being used for the classification.



B. Prediction of test error and How the prediction is arrived for the test error?

To measure the performance of our network, we evaluate how well the test data has never seen before. We measure the performance of the network using the accuracy score. The accuracy is as you would expect just the number of images correctly labeled divided by the total number of images. A perfect classifier will have an accuracy score of 1, where $\text{Accuracy} = \frac{\sum_{i=1}^{n} I(\hat{y}_i = y_i)}{n}$, I is the indicator function, 1 if $\hat{y}_i = y_i$ and 0 otherwise. Here is the test error per 10 epoch.

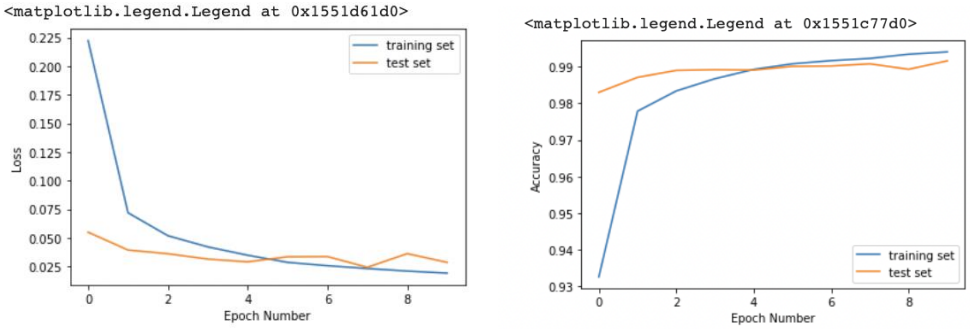
```
Epoch: 01 | Epoch Time: 0m 26s
        Train Loss: 0.181 | Train Acc: 94.43%
         Valid Loss: 0.126 |  Valid Acc: 96.05%
Epoch: 02 | Epoch Time: 0m 26s
        Train Loss: 0.139 | Train Acc: 95.68%
         Valid Loss: 0.096 |  Valid Acc: 97.03%
Epoch: 03 | Epoch Time: 0m 26s
        Train Loss: 0.121 | Train Acc: 96.24%
         Valid Loss: 0.093 |  Valid Acc: 97.10%
Epoch: 04 | Epoch Time: 0m 26s
        Train Loss: 0.107 | Train Acc: 96.62%
         Valid Loss: 0.099 |  Valid Acc: 96.97%
Epoch: 05 | Epoch Time: 0m 26s
        Train Loss: 0.099 | Train Acc: 96.88%
         Valid Loss: 0.086 |  Valid Acc: 97.47%
Epoch: 06 | Epoch Time: 0m 26s
        Train Loss: 0.092 | Train Acc: 97.09%
         Valid Loss: 0.072 |  Valid Acc: 97.83%
Epoch: 08 | Epoch Time: 0m 26s
        Train Loss: 0.080 | Train Acc: 97.47%
         Valid Loss: 0.093 |  Valid Acc: 97.28%
Epoch: 09 | Epoch Time: 0m 26s
        Train Loss: 0.078 | Train Acc: 97.56%
         Valid Loss: 0.063 |  Valid Acc: 97.97%
Epoch: 10 | Epoch Time: 0m 26s
        Train Loss: 0.071 | Train Acc: 97.76%
         Valid Loss: 0.066 |  Valid Acc: 98.08%
Test Loss: 0.056 | Test Acc: 98.24%
```

The model has 222,360 trainable parameters, and achieved nearly 98% accuracy on the test set. The overall idea is that we first clear the gradients calculated from the last batch, pass our batch of images, 'x', through our model to get predictions, `y_pred`. Then we calculate the loss and accuracy between our predictions and the actual labels. Next, we calculate the gradients with respective to each parameter and update the parameter by using the Adam optimizer. As a result, during each epoch running from 1 to 10, we calculate the training loss and accuracy, followed by the validation loss and

accuracy. We then check if the validation loss achieved is the best validation loss we have seen. If so, we save our model's parameters.

C. How to deal with overfitting?

There are couple ways to deal with overfitting problem. The common ones are L1 regularization, dropout, and artificially increasing the training set size. By using a regulared neural network with CNN using keras, we see that the training accuracy and validation accuracy are as high as 99.7% and 99.08% as compared to what is reported without regularization. The validation loss is optimal at 5 epoch with 0.029 and numbers start to pick up after.



7. Conclusion

From the MNIST dataset, we see the effect of using regularization can force the network learn small weights, while all other things being equal. We see the relative importance of the two elements of the compromise depends on the value of λ, such that when λ is small, we prefer to minimize the original cost function, but if λ is large, we prefer small weights. Additionally, this report has reported some digits model's mistakes of prediction in generating fake digits. Both un-regularized and regularized network have been carried from the study.

8. Reference

1. Choudhary, A.; Ahlawat, S.; Rishi, R. A neural approach to cursive handwritten character recognition using features extracted from binarization technique. Complex Syst. Model. Control Intell. Soft Comput. 2015, 319, 745–771.
2. Choudhary, A.; Rishi, R.; Ahlawat, S. Handwritten numeral recognition using modified BP ANN structure. In Proceedings of the Communication in Computer and Information Sciences (CCIS-133), Advanced Computing, CCSIT 2011, Royal Orchid Central, Bangalore, India, 2–4 January 2011; Springer: Berling/Heildelberg, Germany, 2011; pp. 56–65.
3. Cai, Z.W.; Li-Hong, H. Finite-time synchronization by switching state-feedback control for discontinuous Cohen–Grossberg neural networks with mixed delays. Int. J. Mach. Learn. Cybern. 2018, 9, 1683–1695. [CrossRef]
4. Zeng, D.; Dai, Y.; Li, F.; Sherratt, R.S.; Wang, J. Adversarial learning for distant supervised relation extraction. Comput. Mater. Contin. 2018, 55, 121–136.
5. Long, M.; Yan, Z. Detecting iris liveness with batch normalized convolutional neural network. Comput. Mater. Contin. 2019, 58, 493–504. [CrossRef]

6. Chuangxia, H.; Liu, B. New studies on dynamic analysis of inertial neural networks involving non-reduced order method. Neurocomputing 2019, 325, 283–287.
7. Xiang, L.; Li, Y.; Hao, W.; Yang, P.; Shen, X. Reversible natural language watermarking using synonym substitution and arithmetic coding. Comput. Mater. Contin. 2018, 55, 541–559.
8. Huang, Y.S.; Wang, Z.Y. Decentralized adaptive fuzzy control for a class of large-scale MIMO nonlinear systems with strong interconnection and its application to automated highway systems. Inf. Sci. 2014, 274, 210–224. [CrossRef]
9. Choudhary, A.; Rishi, R. Improving the character recognition efficiency of feed forward bp neural network. Int. J. Comput. Sci. Inf. Technol. 2011, 3, 85–96. [CrossRef]
10. Ahlawat, S.; Rishi, R. A genetic algorithm based feature selection for handwritten digit recognition. Recent Pat. Comput. Sci. 2019, 12, 304–316. [CrossRef]
11. Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009) "What is the best multi-stage architecture for object recognition?" In: Proceedings of IEEE 12th International Conference on Computer Vision (ICCV) 2146-2153.
12. Ciresan, D. C., Meier, U., Masci, J., Gambardella, M. L., Schmidhuber, J., Flexible, High-Performance Convolutional Neural Networks for Image Classification, In proceedings of Twenty-Second International Joint Conference on Artificial Intelligence, 1237-1242, 2011.