COMP-551-002 Project 4
Find the "Largest" Digit in Modified MNIST Dataset

Jeffrey Tsui
260611355
Kaggle Team: Spicy Mémés
Department of Electrical Engineering
McGill University
jeffrey.tsui@mail.mcgill.ca

Frank Ye
260689448
Kaggle Team: Spicy Mémés
Department of Electrical Engineering
McGill University
frank.ye@mail.mcgill.ca

I. Introduction

The goal of the Kaggle competition was to classify the "largest" digit in an arbitrary sample image, using different classifiers on a modified version of the MNIST dataset. A baseline linear SVM, feed forward neural network trained with backpropagation, and classifier of the team's choice were evaluated on the given dataset.

Given 40 000 labeled training instances, 10 000 labeled validation instances, and a test set, we have found that CNNs perform better on image classification compared to the Feed Forward Neural Network and Baseline Linear SVM Classifier. The report detailed below is meant to clarify all the results of the code written for the assignment, and to justify classifier designs and values obtained.

II. Feature Design

Apart from classifier design, feature preprocessing and a priori insight on the data helps improve overall accuracy. Shown below is a raw sample in the dataset.

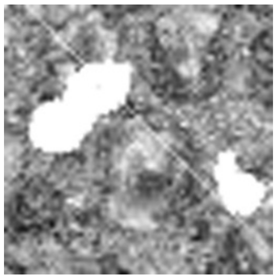

Fig. 1. Raw Input Sample Image from Training Dataset

A. Noise Reduction Using Thresholding

From visual inspection of a few images in the dataset, it can be seen that the digits in each image are white, while the background in each image is composed of other distinct colors. Hence, instead of analyzing information on 3 channels per image (RGB), all pixel values above 251 (nearly white) were forced to white, while all pixels below that value were set to black, clearly reducing background noise.

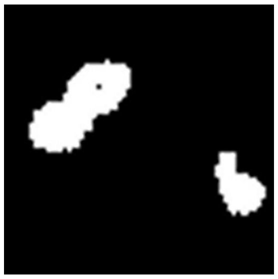

Fig. 2. Sample Image Processed for Noise Reduction with Thresholding

B. Digit Extraction Using Largest Bounding Squares

In order to further clean the image data, regions of interest were defined to contain only the largest digit in each image, in terms of the largest bounding square. This criterion was defined by the creators of the dataset. After finding the region of interest, the picture is transformed to only contain a centered largest digit.

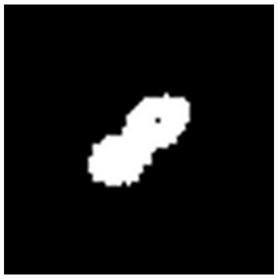

Fig. 3. Sample Image Processed for Noise Reduction with Thresholding and Largest Bounding Square Digit Extraction

C. Digit Rotation and De-skewing

At this point, the largest digit extracted is still unaligned to the y-axis due to rotation and unprocessed for other distortions.

To correct for the misalignment, the digit was first rotated over a range of $-45°$ to $45°$. The angle at which the bounding rectangle fit to contain the digit has the smallest width is used to rotate for y-axis alignment. The assumption for an aligned digit is that its length is larger than its width.

In attempts to treat possible perspective distortions, deskewing was applied to the rotationally corrected digits. Deskewing does this by computing the second moments of inertia of the digit pixels and shears the whole digit by horizontally shifting the lines so that the principal axis is vertical.

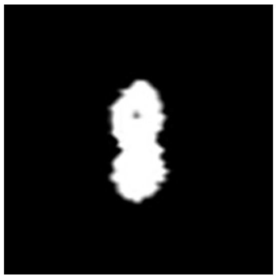

Fig. 4. Sample Image Processed for Noise Reduction with Thresholding, Largest Bounding Square Digit Extraction and Rotation with De-skewing

III. Algorithms

The main algorithms used in the classification problem are the linear SVM as the baseline learner, feed forward neural network trained by backpropagation, and the algorithm of choice was the CNN.

A. Linear SVM

Consider the perceptron, as a simple linear classifier that maps sign($w^T x$) to , where the boundary is defined by sign($w^T x$) = 0. By randomly initializing weights w_j, while any training samples x_i remain incorrectly classified, we update the weights using the formula $w_j \leftarrow w_j + \alpha\ y_i x_i$, where α is a learning-rate.

We can then once again use gradient-descent techniques to measure and minimize error, i.e. for correctly labeled examples the error would be 0 and for incorrectly labeled examples, the error would describe by how far $w^T x$ is on the wrong side of the boundary.

Hence, a linear SVM is a perceptron for which we choose w / the hyperplane such that the margin between the classes is maximized.

B. Feed Forward Neural Networks

Recall the sigmoid function σ(x) = $\frac{1}{1+e^{-w.x}}$, providing a softer threshold than then perceptron unit mentioned above. Given σ(x) is differentiable, it is possible to derive a gradient descent rule to train.

We then define a feed forward neural network by creating multi-layered networks of sigmoid units described above, i.e. neural networks consist of a collection of non-linear activation functions, arranged in layers.In general, the network consists of one input layer, one output layer and k-1 hidden layers.

The structure is such that neurons at layer k become the inputs to the neurons at layer k+1 (forward pass). This "feed forward" network has the predicted output at its output layer. In order to update the weights in each layer and improve prediction accuracy, we use gradient descent methods and the simple notion of the chain rule in order to update each neuron weight, after running a forward pass and computing the correction for each weight using backpropagation.

In fact, different gradient descent techniques can be used, from stochastic (iteratively train individual samples), to batch (all samples) to mini-batch (subset of samples). Thus, the neural network is able to express many functions.

C. CNNs

Whereas feed-forward neural networks do not take into account the structure of the data, CNNs (Convolutional Neural Networks) take 3D tensors as input in the input layer (in the context of RGB images in computer-vision). Then, each intermediate layer uses differentiable functions in order to output a tensor, which is used as an input for the next layer.

In the context of image classification, CNNs are more efficient than feed-forward neural networks due to the use of local receptive fields, parameter sharing and pooling. The concept of features' local receptive fields revolves around data features only depending on a small region of interest in the image. Then, parameter sharing states how local features should be treated the same independently of where they are found in the image. Finally, pooling, in brief, states how the region in which a feature occurs is not as important as if it occurs in an image.

In addition, the term "convolutional" in CNN comes from the convolution formula, heavily used in signal processing applications. In fact, we define $x(t)*g(t)$ as $\int x(a)g(t-a)dt$. As such, if we define $x(t)$ as the input and $g(t)$ as a kernel of learned parameters, if we apply the kernel to various regions of the image, this would be the equivalent of performing a convolution of the kernel on the image. Hence, since there are many kernels that are learned, convolutions of various kernels are applied to the image. In popular literature, the term "filter" is also commonly used to describe a different kernel, whereas different regions of the image are described as "receptive fields".

Hence, convolutional neural networks generally involve convolutional, pooling and fully connected layers. In brief, by having a sequence of a convolutional layer, followed by a pooling layer, we essentially look for important features in the image, then subsample and see if the feature occurs anywhere in the image. In fact, as we go deeper into the network, it is desired to consider features that are more prominent (cover more area) in the image. Finally, the output layer of a CNN would describe the probability of each class occurring depending on the input.

Similarly to feed forward neural networks, training for CNNs occurs via backpropagation consisting of: a forward pass of training data, a loss function evaluating error, a backwards pass, and finally, a weight update. In the methodology section of the report, more hyperparameters, such as learning rate, batch-size, number of training epochs, and optimizers will be discussed (gradient techniques using first order information only, Nesterov's momentum, Adam gradient method, etc.)

IV. Methodology

A. Training/Validation Split

The commonly occurring ratio of 80/20 for the training/validation split was tested at first, but in the chase of the CNNs, we used a 85/15 split in the training process. This choice was justified by the relatively large size of the dataset, allowing both splits to be able to represent most of the variance in data. In addition, running standardized tests proved the efficiency of the split, as shown below.

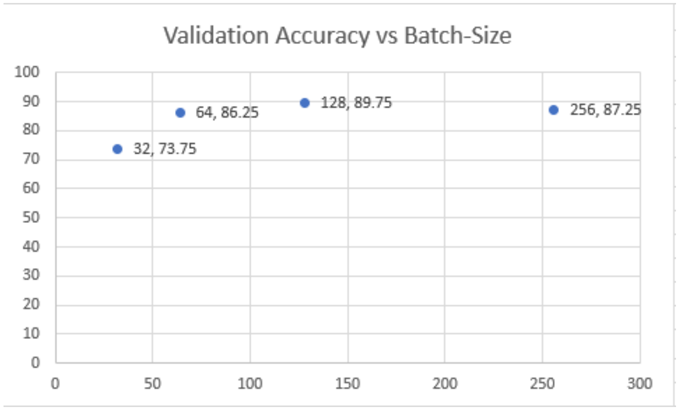

Fig. 5. Test graph showing final validation accuracy vs train-test split. Note that the CNN model was run for 6 epochs, with 800 steps/epoch, and a default RMS propagation gradient descent technique implemented using Keras.

B. Regularization Strategy

1) Linear SVM: Due to the purpose of the linear SVM, regularization was not used on our baseline linear learning model.

2) Feed Forward Neural Network: In order to avoid overfitting, instead of directly using regularization techniques, we observed the training and validation accuracy curves during the training process. This was due to a lack of insight on what type of regularization was best.

3) CNN:

Dataset Augmentation: Data augmentation is a way we can reduce over-fitting on models, where the amount of training data is increased by performing geometric transformations on the existing dataset to artificially "create" more information and thus improved generalization. During the training process of our CNN, instead of feeding the exact preprocessed images as inputs, a combination of geometric transformations including rotations ($-8°$ to $8°$), horizontal/vertical translations (0 to 5 pixels), shearing (0 to 0.3 rads) and scaling (0% to 8%) were applied. In addition, the creation of noisier data used during the training phase helps the classification model generalize better.

Dropout Layer: A deep neural network uses layers of non-linear hidden units between the inputs and output. Feature detectors are learned by incoming connections from different layers that enable the model to predict the correct output. Dropout works by probabilistically removing a neuron from designated layers during training or by dropping certain connections. The idea behind dropout is to prevent complex co-adaptations between neurons and "force" them to not rely on each other by making them work with a random subset of neighboring neurons. A dropout layer with probability 0.1 was applied after the 512 fully connected layer.

C. Optimization Tricks

1) Linear SVM: Due to the purpose of the linear SVM, optimization was not necessary on our baseline linear learning model.

2) Feed forward Neural Networks: No optimization tricks were used during the hyper-parameter tuning of the Feed Forward Neural Network because manual grid search of parameters was implemented.

3) Convolutional Neural Networks: In the case of CNNs, apart from tuning parameters, optimization tricks were used in order to design a higher performance network. In fact, convolutional layers were fed into other convolutional layers, then pooling/ subsampling layers were added in succession.

As for the non-linear activation function, we used 'ReLU' instead of sigmoid functions, which would optimize the network by reducing the likelihood of the gradient vanishing during the training process.

Furthermore, in order to accelerate training times, a callback function was used to reduce the learning rate, if a certain metric did not improve for a number of "patience epochs". We chose to reduce the learning rate by a factor of 2, if for 3 epochs, the accuracy of the model did not improve, with a minimum learning rate of 0.00001.

D. Setting Parameters

1) Feed Forward Neural Networks: In order to optimize the performance of our Feed Forward Neural Network, a standardized testing method was used to evaluate a range of different hyper-parameters. This was done more extensively in the Convolutional Neural Network section due to the computational speed being less of a limit there. However, for the Feed Forward Neural Network two main hyper-parameters were tuned: Number of Nodes per hidden layer and Number of Hidden Layers.

We applied a standardized testing method where all model parameters were held constant while exploring the effect of varying just one parameter. Below is a visual representation of tuning the number of hidden layers with 256 nodes per hidden layer over a duration of 10 epochs. One notable observation is that the validation accuracy drastically increases when there are 4 or more hidden layers.

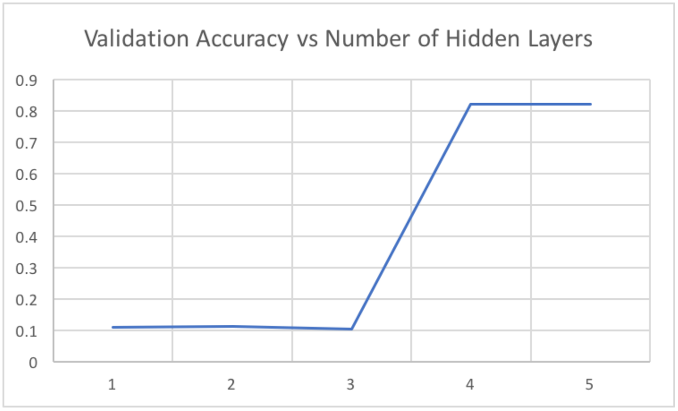

Fig. 6. Test graph showing final validation accuracy vs number of hidden layers.

Similarly, we applied the same test method to evaluate the optimal number of nodes per hidden layer with a standardized 4 hidden layer model trained over 10 epochs. A notable finding is that the validation accuracy steadily increases until 256 nodes per hidden layer and then drastically decreases past that.

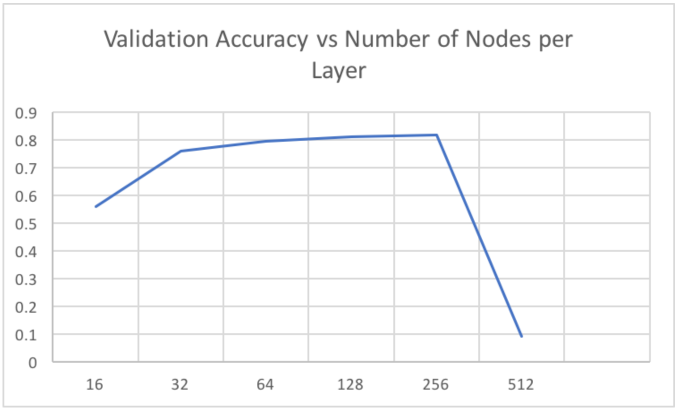

Fig. 7. Test graph showing final validation accuracy vs number of nodes per hidden layer.

2) Convolutional Neural Networks: In order to optimize the performance of our CNN, a standardized testing method was used to evaluate various parameters' importance in optimizing the network. The method involved holding all model parameters constant, while varying only one factor. Thus, using training and validation accuracy as metrics, we were able to find a direct correlation between the given parameter and its effects on the performance of the CNN.

For example, in order to test the effect of batch-sizes on the final accuracy, we ran a CNN model for 6 epochs, with 500 steps/epoch, for various batch-size values in order to choose one for the digit classification. The optimal batch-size value was found to be 128.

Similarly, we applied the test method to evaluate the optimal number of steps per epoch, this time using a batch-size of 32 as a constant, as seen in Figure 3. The number of steps

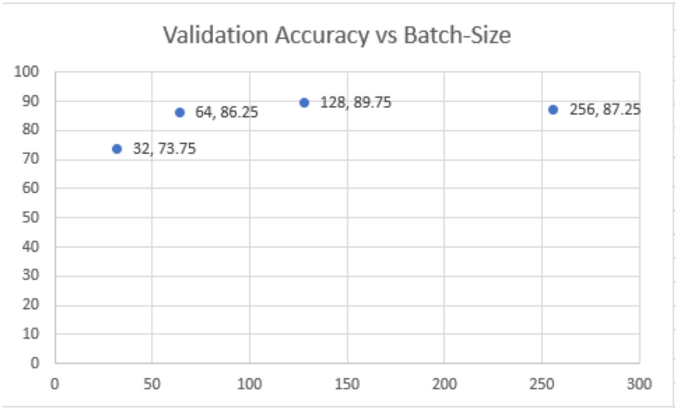

Fig. 8. Test graph showing final validation accuracy vs batch-size. Note that the CNN model was run with a default RMS propagation gradient descent technique implemented using Keras.

per epoch is defined as the number of subdivisions of the training samples used, and is a parameter that we tweaked using Keras. Hence, for the optimal CNN model, we used 2000 as the best number of steps per epoch during training.

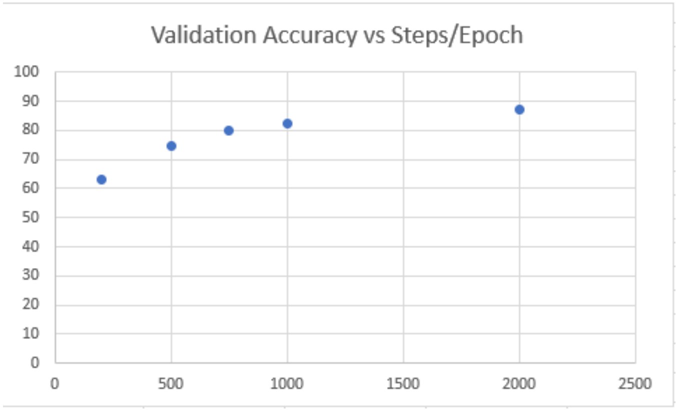

Fig. 9. Test graph showing final validation accuracy vs batch-size. Note that the CNN model was run with a default RMS propagation gradient descent technique implemented using Keras.

Finally, various gradient descent techniques were tested for the same CNN model, holding every other parameter as a constant in order to fairly evaluate the best optimizer for our purposes. The best results were empirically achieved using the Nadam, i.e the Adam gradient descent optimization using Nesterov's momentum, as opposed to traditional momentum. Since the Adadelta optimizer performed with similar validation accuracy, it was also tested on the final CNN model. However, we chose to continue with Nadam, as it performed very slightly better on the Kaggle test set.

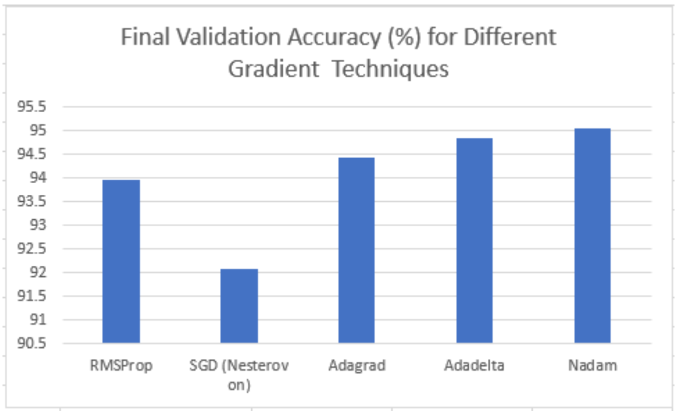

Fig. 10. Test graph showing final validation accuracy for different optimizers implemented and tested using Keras

V. Results

A. Modified MNIST Dataset Results

Amongst the three classifiers implemented, the CNN performed the best on the test set by far, showing a 95.3% test accuracy. This was followed 81.8 % , achieved by the feed forward neural network, and 11.30 % , obtained by the linear SVM.

Shown below is a training plot of the Feed Forward Neural Network model over 20 epochs.

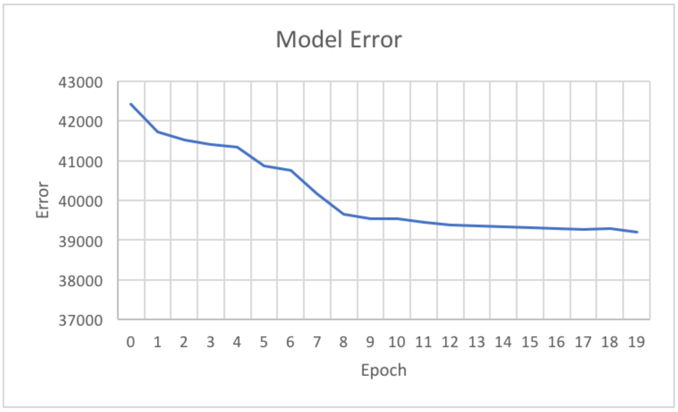

Fig. 11. Training Error of the Feed Forward Neural Network after training for 20 epochs

Shown below is a training plot of the CNN model over 10 epochs on the final preprocessed dataset.

Moreover, shown below is the training plot of the same CNN model, but this time on the raw data.

As shown, we can see drastic improvements just from the preprocessing mentioned in the feature design section of the report.

As explained in the methodology section, standardized tests were written in order to determine the most important hyper-parameters affecting the performance of the CNN. From our empirical results, we have concluded that batch-size, the number of steps per epoch, and the choice of gradient descent techniques all affected the overall performance of the CNN up to different degrees.

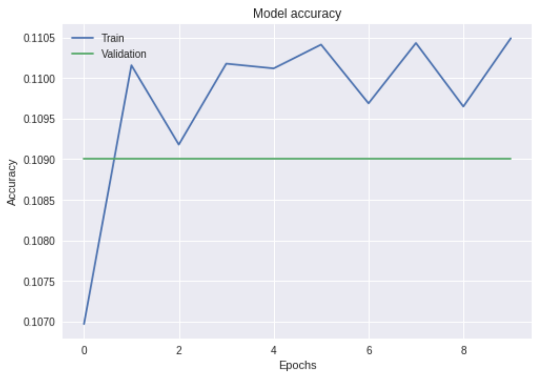

Fig. 12. Training and Validation Accuracy of the Optimized CNN Model on Raw Image Data

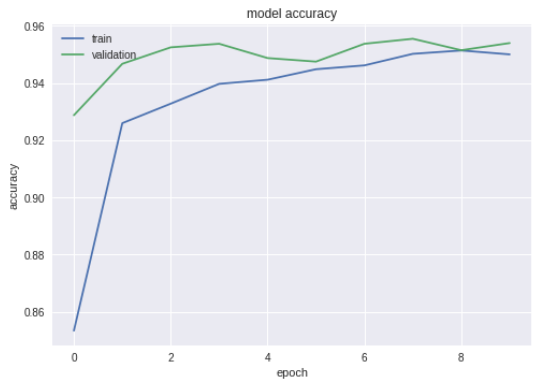

Fig. 13. Training and Validation Accuracy of the Optimized CNN Model on Preprocessed Data

As such, we have built set the optimal batch-size value to 128, the number of steps per epoch to 2000, and we use Nadam as the gradient optimizer technique.

In fact, large discrepancies in performance were found depending on the batch-size. The worst validation accuracy of 73.75% was due to a small value of 32, while the highest validation accuracy of 87.25% was found when using the optimal value of 128 (Note: this was run on fewer epochs, for faster training. These tests were run to quickly evaluate optimal values).

Moreover, the worst accuracy was found for 200 steps per epoch, at 63% , while the best results came from 2000 steps per epoch, also at 87.25%. This drastic difference lead us to believe that steps per epoch has a great impact on CNN training and performance.

Finally, the choice of optimizers also showed a great difference in performance. For example, on our light CNN model, stochastic gradient descent offered a validation accuracy of 92.07% , whereas the Nadam optimizer offered 95.05% , hence the choice.

NOTE: all values in the results are shown from the graphs in the parameter choice subsection in methodology.

VI. Discussion

A. *Methodology Advantages/Limitations*

1) Advantages: The advantages in the test methodology lies within systematic choice of hyper-parameter values. The standardized tests easily helped visualize the importance of each parameter, especially in the case of the CNN.

Our time allocation was mainly focused towards the CNN, which, from the slides, was said to perform significantly better than both feed forward neural nets and linear learners.

However, on the feed forward neural network, we were able to methodically test each parameter separately. This lead to experimental accuracy of up to 81%.

In addition, the feature preprocessing done on the raw image data significantly improved classifier performance. This was demonstrated from test results using different preprocessed features on the same CNN model.

2) Limitations: However, without constant access to powerful GPUs, computational resources were limited. Furthermore, the linear learner and the feed forward neural networks were not adapted to use GPU, hence runtimes are very slow. In addition, while looking for optimal hyper-parameters, perhaps the range of values we searched on did not contain the best hyper-parameter. Hence, we could improve our test methods by searching on a larger grid of test values.

For the SVM classifier, we could have improved it by making it a non-linear SVM (adding a kernel). In the case of the feed forward neural network, we could have experimented with a wider range of layers, nodes/layer in order to increase classification accuracy.

B. *Areas of Future Work*

Areas to be improved upon are, for example, our knowledge in optimization, which would help us understand hyper-parameters contained within the different gradient descent techniques. In fact, upon the Keras documentation 's recommendation, we mostly stuck to default values. Hence, we had little knowledge of how the learning rates were being effected, or if first order or both first order or second order information was being used to find descent directions during the gradient search.

Moreover, doing more work related to deep learning could help us understand how to optimize network structure itself, e.g. the number of filters needed per convolutional layer, the dropout rate needed, the subsampling layer, the number of hidden layers required, etc.

VII. Statement of Contributions

The authors, Jeffrey Tsui and Frank Ye, have both taken direct roles in the three classifier implementations, methodology and model evaluations, feature preprocessing, as well as the composition of the report.