COMP 551 - Mini Project 3 Multi-label Classification of Image Data

David Castonguay (260804528), Marco Guida (260803123), Mathew Marques (260804167)
ECSE and Computer Science McGill University

Abstract—**Utilising a multilayer perceptron, we propose a model for a convolutional neural network (CNN) to identify up to 5 digits of an image from a modified MNIST dataset. An algorithm was implemented to find and extract the individual digits of an image. Then, a CNN model was architected and trained using these extracted images of the individual digits. The model consists of 4 convolutions layers, each with a ReLU and max pool layer, that then gets flattened as an input to 3 linear layers, reducing to a softmax output of 10 classes. A final implementation of our model using 25 epochs resulted in a test accuracy of 99.523% in the Kaggle competition.**

I. Introduction

The task: For this project, we were tasked with detecting and classifying multiple digits embedded in an image of a modified MNIST dataset. The first sub-task was to determine the number of digits in each image, and further where each of those digits were located so they can be extracted and made into individual images for the purposes of training of our model.

We first focused on finding a way to split the digits in a given image of the modified MNIST dataset. With this, we could later train a model and predict unseen data for individual digits. Our first attempt was iterating over the width of the image to find black columns (indicating separation of digits) and taking the average the average of the every group of nearby black columns to extract an image of each individual digit. This method achieved a 0.15% error rate on the training dataset.

Though our first attempt was fine, we found an even more performant approach using Canny Edge detection with the ”findContours” function to find the centers and radius of each digit, as illustrated in Figure 1. Due to multiple detections (i.e. multiple cluster centers) on the same digit we had to group the centers that only correspond to a single digit. With the average radii of each digit, we were able to determine that a distance of 6 pixels between the current digit center to the previous center is the optimal threshold to differentiate between digits. Finally, we are left with the final number of digits in the image which we pass to a helper function that creates the separation of each digit. Through our visualization of the data we realized that once we know the number of digits in the image we can easily create fixed 12x12 boxes to separate the digits. This implementation was a significant improvement to the first method as it achieved an error rate of 0.044% on the training set.

We chose to use a Convolutional Neural Network (CNN) as our deep neural network model to classify the digits in images. The details of our architecture of our model will be discussed later in the Results section. Furthermore, we also experimented with using the original MNIST dataset for training, but did not realize any significant improvements in our model as we still achieved approximately 99.5%. This is probably because the original dataset is built off the MNIST dataset so there is no increase in variance to help our model with unseen data.

II. Dataset

The dataset used is a modified MNIST dataset where each sample in the dataset is an image containing up to 5 digits as oppose to just a single digit. If the number of digits in the image is less than 5, than the remaining labels are associated with a special class called ”no-digit” (i.e. class ”10”). The modified dataset has 56,000 samples. The only pre-processing done the dataset was splitting each image into 12x12 images of a single digit so that our model trains on single digit images.

III. Results

A. Model Architecture

A selection of pre-existing CNN purposely designed for MNIST datasets are available through certain python packages (e.g. the VGG-19 architecture). Though the VGG-19 may seem like an interesting choice, we decided to follow our own research through the number of convolution layers and number of linear hidden layers, in order to intercept the best resultant accuracies.

The model started with three linear hidden layers as a baseline, and one convolution layer. The amount of linear layers will not change. In fact, the linear layers are meant to reduce the output of the convolution (e.g. 32, 64, 128, etc.) to 1000. Then another layer to 100, and a final softmax layer to 10 classes. Though the linear layers remain unchanged, we will observe the changes through the amount of convolution layers. The search started with the singular convolution layer receiving a singular input channel which then transferred to 32 channels. The convolution layers will always have a kernel size of 3 and stride of 1. Furthermore, at each convolution layer, the output of the 32 channels are then fed to a ReLU activation function, followed by a max pooling setup with stride of 2. For such model, the accuracy after 1 epoch is visible in Figure 3. The search continued with 2 convolution layer, first layer with output of 32 channels and second layer with input of 32 and output of 64 layers.

Then, the search continued to 6 convolution layers. With the search completed, the model was selected to have 4 convolution layers, where the accuracy was the greatest. The model properties are available in Table I. With this particular model, we were able to obtain the best performance. Though, through this model are multiple hyper-parameters. These hyper-parameters can also be modified in order to maximise the performance of the model, on top of its current architecture.

B. *Optimizer and Hyper-Parameters*

For our model's optimizer, momentum stochastic gradient descent was selected. Considering the large size of the dataset, we wanted an optimizer that converged fast to the minima (hence stochastic) and that ensured that, on average, it is always trending in the correct direction to the minima (hence momentum). Past experience with momentum SGDs was also a considerable factor. For the choice of momentum and learning rate for the optimizer, separate grid searches were conducted. Momentum was tested for values between 0.6 and 0.95, and learning rate was tested for values between 0.0001 and 0.01. For the best model, we found a momentum of 0.9 and a learning rate of 0.001 to be the best. Figures illustrating our choice for these two can be seen in Figure 4 and Figure 6, respectively.

Batch size was also an important hyper-parameter to evaluate. Batch size was crucial as it defined how many iterations the network will need to train on for each epoch. Larger batch sizes resulted a faster algorithm but weaker accuracy, whereas, smaller batch sizes computed each epoch very slowly but had an overall increased training and validation accuracy [1]. Figure 5 illustrates the search performed. We decided that a batch size of 17 would be optimal for our model as it is closest to 16, the highest performing batch size in the search, and it also evenly divides the total number of individual digits in the dataset.

C. *Training Epochs*

The choice of number of epochs is critical to the performance of a neural network. Too few epochs may be terminated too quickly to properly fit the data, but too many epochs may over fit the data and perhaps minor improvements in performance on a test set. We observed the performance of our best model (see Table I) for different epochs and report the results, as illustrated in Figure 2. It can be seen that the training and validation accuracy quickly increased within the first 15 epochs, and then stagnated at an accuracy of approximately 99.9%. Beyond 25 epochs, the plot begins to trend lower, and extrapolating the curve beyond 30 epochs sees the accuracy decreasing due to overfitting [2]. Consequently, we deduced that the best epoch for our model was 25 (even though other epochs between 15-25 could have perhaps performed similarly).

IV. Discussion and Conclusion

As explored in the results, we applied multiple grid searches to discover a performant model architecture (e.g. convolution layers search) with the best combination of hyper-parameters that resulted in a final test accuracy of 99.523% in the public Kaggle competition. We explored how individual components of our model impact how accurately it predicted the samples in the modified MNIST dataset. However, this was all conducted under the assumption that we use a simpler CNN as the backbone of our model. Perhaps with other more intricate convolutional neural network, like VGG-19 with its larger network of 16 convolution layers and 5 max pool layers, our model would be able to make deeper connections between features in the image of a digit [3]. This could have allowed for certain patterns of digits that were consistently wrongfully identified in our current model to be predicted correctly, resulting in an even better test accuracy.

V. Statement of Contributions

All: Report and Hyperarameter Testing David: CNN Marco: CNN Mathew: Extract digits

References

[1] J. Brownlee, "How to control the stability of training neural networks with the batch size."
[2] B. Carremans, "Handling overfitting in deep learning models."
[3] A. Kaushik, "Understanding the vgg19 architecture."

Appendix

Hyper-Parameter	Value
Convolution Layers	4
Convolution Layer 1 Output Channel Size	32
Convolution Layer 2 Output Channel Size	64
Convolution Layer 3 Output Channel Size	128
Convolution Layer 4 Output Channel Size	256
Linear Hidden Layers	3
Linear Hidden Layer 1 Output Size	1000
Linear Hidden Layer 2 Output Size	100
Linear Hidden Layer 3 Output Size	10
Max Pool Stride	2
Convolution Stride	1
Convolution Kernel Size	3
Momentum	0.9
Learning Rate	0.001
Epochs	25
Batch Size	17

TABLE I
BEST MODEL HYPER-PARAMETERS

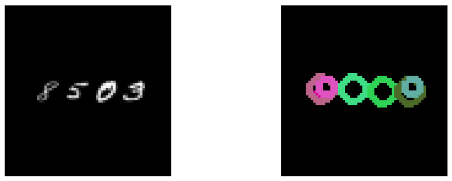

Fig. 1. Final implementation to split up digits by finding digit centers.

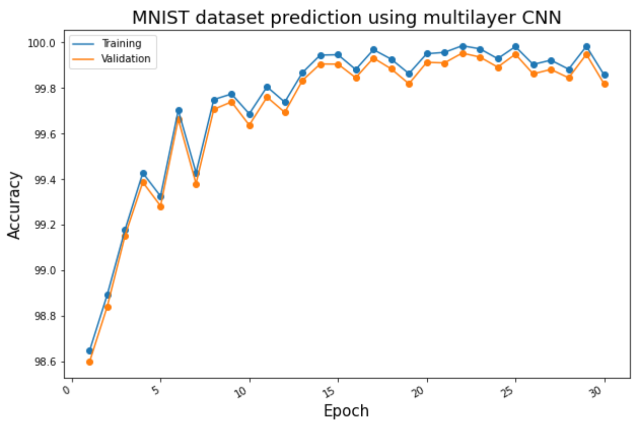

Fig. 2. Training and validation accuracy (percentage) gradually stagnates as the number of epochs increases.

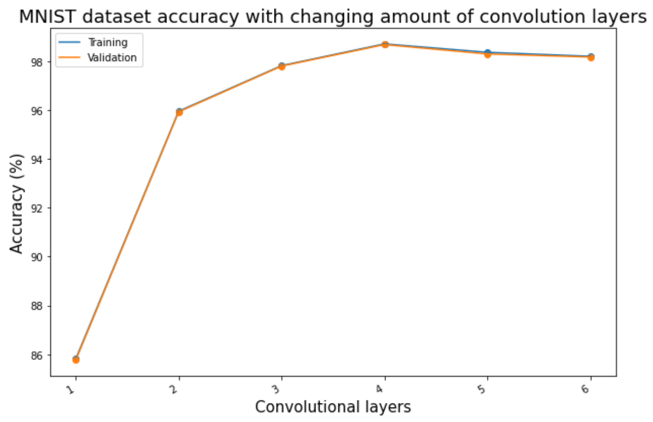

Fig. 3. Training and validation accuracy (percentage) gradually stagnates as the number of convolution layers increases (epochs = 1).

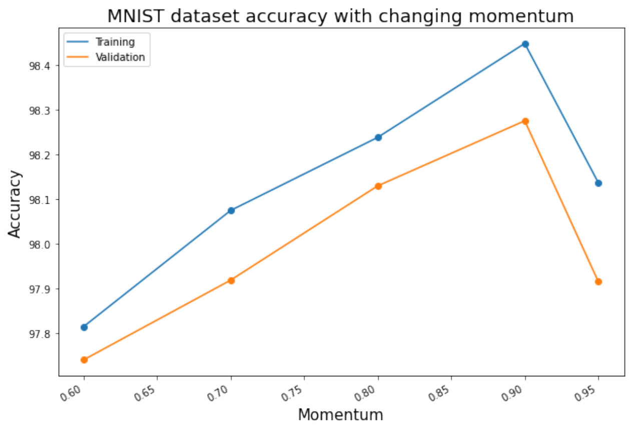

Fig. 4. Training and validation accuracy (percentage) peaks at a momentum of 0.9 (epochs = 1).

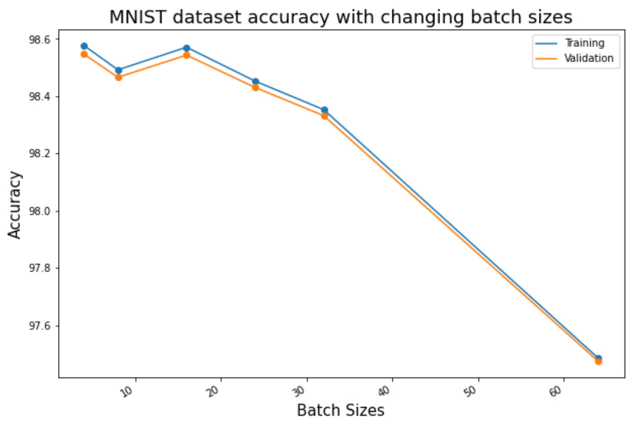

Fig. 5. Training and validation accuracy (percentage) peaks at a batch size of 16 (epochs = 1).

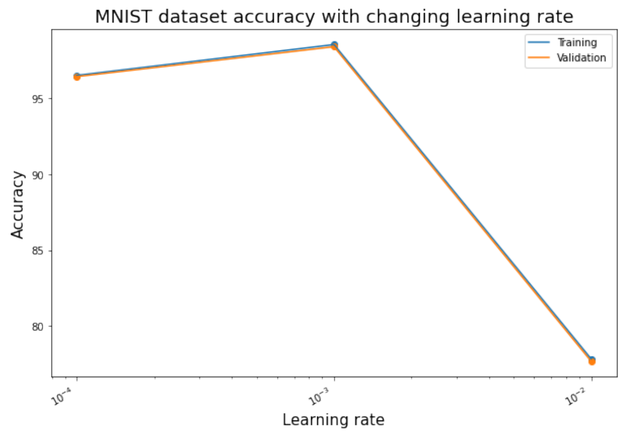

Fig. 6. Training and validation accuracy (percentage) peaks at a learning rate of 0.001 (epochs = 1).