Convolution Neural Network architecture for modified MNIST dataset

Navpreet Singh
Department of Electrical and Computer Engineering
McGill University

Yu-Cheng Cho
Department of Electrical and Computer Engineering
McGill University

Pouriya Alikhani
Department of Computer Science
McGill University

***Abstract*—Publicly available image repositories such as ImageNet and MNIST have greatly contributed to the advancement in convolutional neural networks, CNN. MNIST database is a collection of handwritten digits frequently used for prototyping image recognition pipelines. Here we are given a modified version of MNIST where each image contains two digits; our task is to detect the digit occupying more space. We performed image preprocessing: we removed the image background, then we selected the digit with the greatest bounding-box and placed it in the center of a new plot; this procedure effectively eliminates the other digit. We employed a classical CNN structure while tuning batch size, optimizer, activation function, and learning rate decay to increase the accuracy of our model. In our best performing model, we employed Adam optimizer, batch size of 128, ReLU as activation function, and learning rate initialized at 0.001 and reduced to half after three epochs without improvement. Such a model obtained an accuracy of 96.11% on the validation set and 95.5% on the test set.**

I. Introduction

MNIST database is a collection of handwritten digits first introduced by LeCun et al. in 1998 to study image segmentation via convolutional neural nets, CNN [1]. MNIST is commonly used for training in image processing and machine learning, so far as today it is considered a benchmark for prototyping many newly created pipelines, particularly in image recognition. Recent advances in deep neural network and machine learning have resulted in the exhaustion of MNIST and has opened the field for newer versions of MNIST [2].

Finding patterns in the given data is an important focus in machine learning; in image recognition, this effort translates to predicting the existence of some specific object in a given image. As a discipline image recognition is at the intersection of computer vision and machine learning. The recent success in deep convolutional neural networks is responsible for much of the success image recognition [3].

We are provided with a modified version of MNIST where each image contains two digits and our task is to distinguish the image that occupies more space. We used OpenCV to remove the background from the normalized images and convert the image into a binary image where there are two possible values for each pixel. Then we used findContours option form OpenCV to find the digit with the largest bounding-box. Our training data is composed of 64 by 64-pixel grayscale images: each pixel is in the range [0, 255] which is indicative of its brightness. To obtain the trained model we used a common CNN obtained from Google Tensorflow tutorials such that the sequence of layers is composed of **[INPUT – CONV – RELU – POOL – CONV – RELU – POOL – FC – FC - SOFT]** where CONV is a convolutional layer, RELU is an activation layer, POOL denotes maxpooling, FC is a fully connected dense layer and SOFT is the output layer [16]. To further improve the performance of the model, we tuned the following hyperparameters: batch size; optimizer; activation function; and learning rate decay. In our best model we have emplyed Adam optimizer, batch size of 128, ReLU activation function, and learning rate initialized at 0.001 and decaying to half after 3 epochs without improvement; such model obtained an validation accuracy 96.11% and testing accuracy 95.5%.

II. Related Work

Girshick et al. created an object recognition pipeline in three phases: detection of a set of proposal regions that are presumed to contain an object; feeding the proposal regions through multiple convolution layers to extract features of a fixed length; and feeding the features to a class-specific linear SVM to verify the presence of the expected object in the proposal region [7]. The main benefit to this approach is that it enables the proper segmentation and localization of the image objects.

State-of-the-art performance on the ImageNet challenge can be obtained using a conventional architecture, ConvNet, with a substantially increased depth, up to 19 weight layers [5]. Although much of the recent success in CNNs is attributed to high-performance computing systems, e.g. GPU, and deep networks, it is argued that a simple CNN with a little parameter tuning can obtain results comparable with that of the state-of-the-art models [4][5].

Domain guided dropout is shown to produce a more robust data representation for mixed datasets with portions of dataset sampled from different distributions; however, we have strong reasons to believe that our pre-processed data belongs to one domain which makes random dropout a suitable approach to establish a more robust model that is less prone to overfitting [6].

III. Dataset and Setup

Dataset: The dataset used here was a MNIST database (Modified National Institute of Standards and Technology database). The dataset had been modified and the images contain more than one digit and the goal was to find which number occupies the most space in the image.

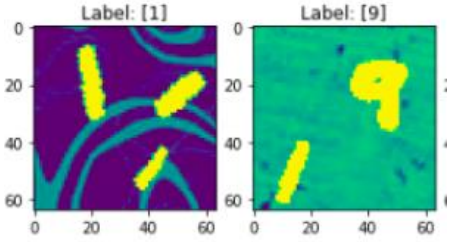

Figure 1: Example of the training images with the associated labels

The database contains 40,000 training images and 10,000 testing images, each of size 64x64. The 40,000 training

images have a label associated with them and this label is the target to be predicted for the test images.

Pre-processing: As shown in figure 1, the images contain a lot of background noise, which can harm our model. OpenCV library was used to preprocess the images before inputting them into the neural network. OpenCV is an open source computer vision and machine learning software library.

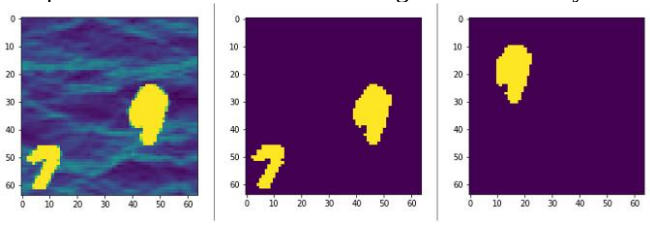

Figure 2: Preprocessing - Raw Image converted to binary removing all the background noise. The image is then cropped around the bigger digit and converted to a 64x64 image with zero padding around it.

Using OpenCV the images were first converted into binary images, shown in figure 2, using a very high threshold of 0.7 (all the images were normalized initially by dividing them by the max value of the pixels i.e. 255). The *findContours* option of the OpenCV was used to get the contours of the 2 or 3 digits in the image and the image was cropped around the digit with the biggest bounding box. The cropped image was padded with zeros to get the original size of 64x64 to maintain consistency.

IV. PROPOSED APPROACH

The problem of classifying the digits in this modified MNIST dataset was tackled with a simple approach. We started with an already published model and built upon it to achieve the best accuracy results. There were many hyperparameters that could be tuned, but we did not exhaust all the options as the list is very long. We tried to work with different optimizers, the size of the batches, the activation functions and the learning rate. The process is described in detail in the next section.

A. A standard CNN architecture

CNNs is a sequence of layers and every layer of a CNN transforms one volume of activations to another through differentiable function. The field of digit recognition using convolutional neural networks has already been exhausted by acedemic research. There are already established CNN architectures that perform very well on recognizing digits in an image. Our idea was to select one such existing model and tweak it to get the best performance. As such we decided to use the model given by Google in their TensorFlow tutorials [16]. We used three main types of layers to build the architecture. We stacked these three layers to form our CNN architecture. The sequence of the layers is as follows: **[INPUT – CONV – RELU – POOL – CONV – RELU – POOL – FC – FC - SOFT],** where CONV is the convolutional layer, RELU is the activation function, POOL stands for maxPooling, FC stands for the fully connected dense layer and SOFT is the output layer which gives the probability of the input for all of the classes. Figure 3 explains the described network in a pictorial form with the names of the layer on the right.

B. Identifying hyper-parameters

The baseline model was designed for the original MNIST dataset but must be modified to get good accuracy on our dataset. One approach is to make the network deeper and the other approach is to tune the hyperparameters of the network. We talk about the hyperparameters in this section.

i) Size of the mini-batch

According to Professor Andrew Ng, in his online course on Neural Networks, choosing a correct batch size is not a trivial problem [9]. There can be two extremes to the case. First is to take the entire dataset for each gradient step. Our training dataset has 40000 images and using all the data for gradient descent is very computationally rigorous. The advantage of this approach is that we don't waste time in going in the wrong direction and can reach the local optima easily. Second case, which is very computationally efficient, is to take one sample for each gradient descent step. The disadvantage of this approach is we may wander around the local optima for some time before proceeding towards it. But in some cases, single sample can help in getting better results because it takes many more iterations compared to full batch. The middle ground can be found by trying different values of the batch size, and with trial and error, find the best batch size. We tried different values of batch size, ranging from 32 to 512. After running 50 epochs with each of these values, we found out that batch size of 128 gave the best results.

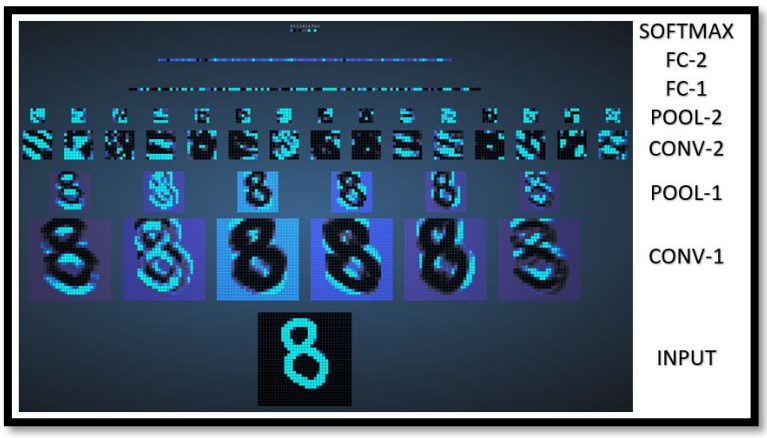

Figure 3: Visualization of a 2D Convolution Neural Network [8]

ii) Choice of the activation function

The activation functions are the decision units at each neuron which take a set of inputs and define its output for the next layers. Three commonly used functions, ReLU, tanh, and sigmoid [13], are implemented and compared by fixing the model's parameters. We use two main criteria to select the activation functions: (1) the best prediction accuracy we can achieve before fine-tuning its hyperparameters, and (2) the number of epochs that the model requires to converge (or reach the point of overfitting). For a better comparison of each active functions, we ran the model for 100 epochs.

iii) Learning rate reduction

If we are using a fixed learning rate, our model may converge to the minima, but may wander around the minima instead of reaching it. This may be because of the noise in our mini-batches. But if we reduce our learning rate to a smaller value, our model may still oscillate around the minima, but now it will be closer to the minima as compared to the larger learning rate. As shown in the figure below, color blue denotes a large learning rate and is oscillating around the minima, but never converging to it. However, if we start with the same learning rate shown in green and reduce it to a lower value at a further stage, it will oscillate around local minima but will be much closer to it than the blue line. [15]

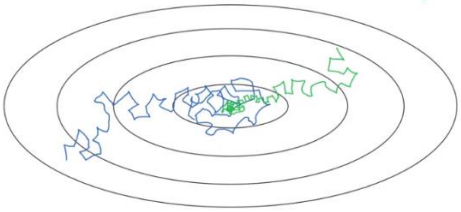

Figure 4: Illustration showing different learning rate

V. RESULTS

A. The baseline

Implementation of the presented Convolutional Neural Network gave us an accuracy of about 95%. The model was run for 50 epochs and the results show no sign of overfitting. This inference was made based on figure 5 which shows that the error for the training set and the validation set keeps getting smaller even after 50 epochs.

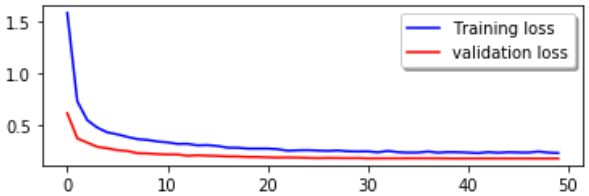

Figure 5: Loss on training and validation set for 50 epochs

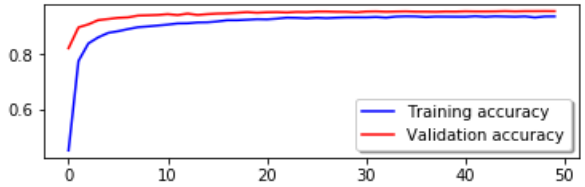

Figure 6: Accuracy of the prediction for the training and validation set after 50 epochs.

B. Hyper-parameter tuning

i) Size of the mini batch:

We ran experiment on the size of the mini-batch, varying it from 32 to 256. As explained in the previous section, fixing the size of the batch to obtain the maximum accuracy is a trial and error problem and the results are evident from the figure below. Although there is a very small change in the accuracy with different sizes of the batch, we fixed the batch size to 128 as it gave the best accuracy.

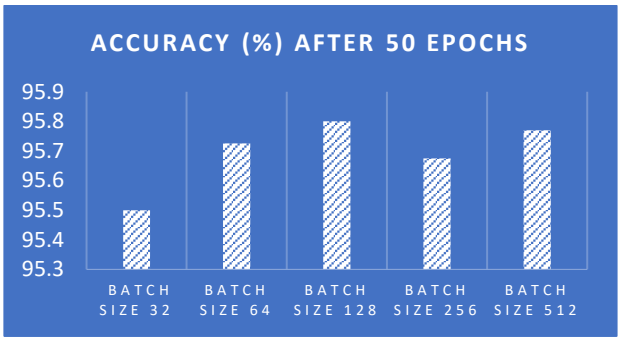

Figure 7: Accuracy for the validation set for different sizes of the batches.

ii) Choice of Optimizers:

We tried five different versions of the optimizers. The figure below shows the accuracy obtained for different optimizers. The lowest accuracy (87.6%) was obtained from using Stochastic Gradient Descent without any momentum, followed by using the value of momentum = 0.1 (88.25%). A little improvement was seen by activating the Nesterov option, improving the accuracy to 89.925%. As explained in the previous section RMSprop shows better results (95.47%) compared to SGD and it is evident from the figure below. And since Adam combines the pros of SGD and RMSprop, using Adam gave us the best accuracy of 95.775% and this helped us to settle on a choice of the optimizer.

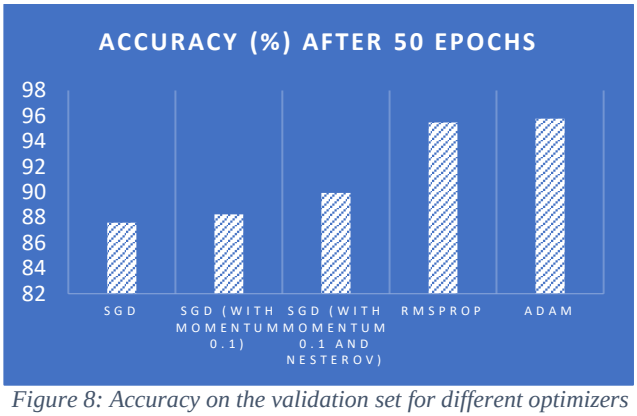

Figure 8: Accuracy on the validation set for different optimizers

In terms of time taken to converge, SGD takes the greatest number of epochs to converge whereas Adam optimizer takes just a few epochs to converge to an accuracy of about 95% (shown in the figure below).

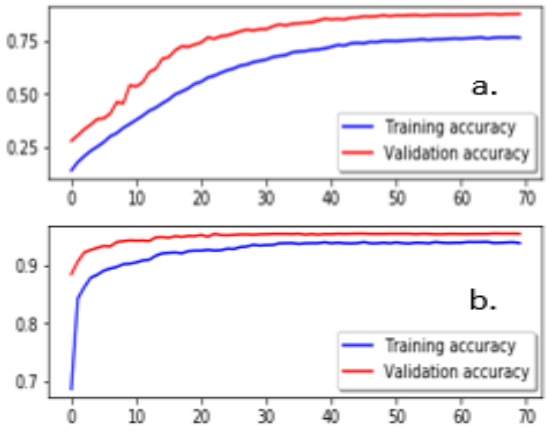

Figure 9: Comparison of epochs taken to converge by SGD vs Adam

iii) Choice of Activation function:

From the results, a sigmoid function was not an ideal activation function for our model due to its slow rate of convergence. A test on sigmoid function using 500 epochs was conducted; however, the model did not converge and gave us the best accuracy of 90.62%. The ReLU and tanh functions both converge within 100 epochs and the ReLU showed better accuracies. Thus, the ReLU was selected as the choice of activation function for our model.

Table 1: Comparison of different activation functions

Activation Function	ReLU	tanh	Sigmoid
Prediction Accuracy	94.35 %	94.10 %	87.53 %

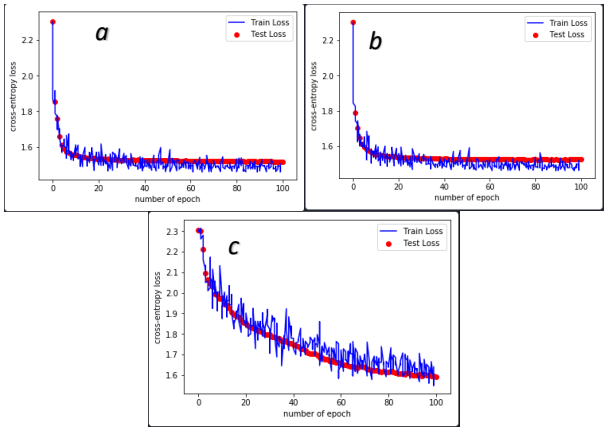

Figure 10: The training and test (validation) loss when using a) ReLU, b) tanh, and c) sigmoid as the activation function.

iv) Learning rate reduction:

We started with a learning rate of 0.001 and reduced it to 0.5*learning rate, if we saw no improvement after 3 successive epochs. We also set a lower bound on learning rate to 0.00001, which is done to make sure that our model does not take indefinite amount of time trying to converge to the minima. The implementation of this strategy gave us the value of learning rate as 1.5625e-05, i.e. initial learning rate of 0.001 was reduced to half 6 times.

ImageDataGenerator: We used Keras Image Preprocessing option to augment our training images. The ImageDataGenerator [12] class helps in generating a batch of images data with real-time augmentations. The data is looped over (in form of batches). From a huge set of parameters, we played around with just four parameters, i.e. zoom range, width shift range, height shift range and rotation range, with the rest set to default.

Architecture with the best hyperparameters: Starting with a baseline model (summarized in the table below), we tweaked the hyper-parameters to the ones computed from the experiments run in the last section and observed an improvement in the accuracy. We increased the number of epochs from 50 to 70 and still observed no overfitting (see figure 11). The final accuracy on the validation set was 96.11%. And on running the model on the test set, the Kaggle submission gave us an accuracy of 95.5%. The table below summarizes our model with the details of each layer, it's output shape and the number of parameters in each layer.

Table 2: Summary of the CNN model

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(64, 64, 32)	832
max_pooling2d_1 (MaxPooling2)	(32, 32, 32)	0
dropout_1 (Dropout)	(32, 32, 32)	0
conv2d_2 (Conv2D)	(32, 32, 64)	51264
max_pooling2d_2 (MaxPooling2)	(16, 16, 64)	0
dropout_2 (Dropout)	(16, 16, 64)	0
flatten_1 (Flatten)	(16384)	0
dense_1 (Dense)	(512)	8389120
dropout_3 (Dropout)	(512)	0
dense_2 (Dense)	(256)	131328
dropout_4 (Dropout)	(256)	0
dense_3 (Dense)	(10)	2570
Total params		8,575,114

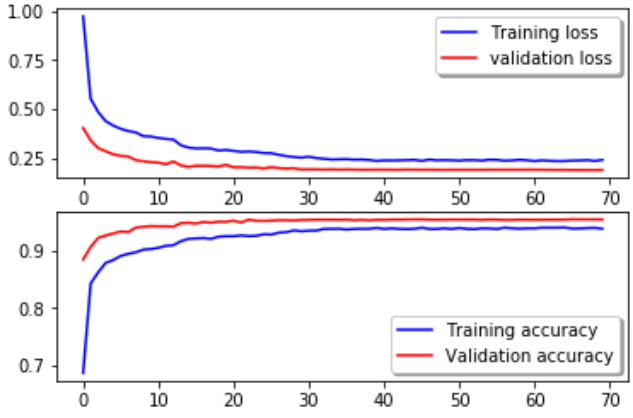

Figure 11: The loss and accuracy on the training and validation set for 70 epochs with the best hyper-parameters

Figure 12 shows the confusion matrix which presents the number of labels presented correctly and incorrectly. The most common errors were misclassifying 5 as 6, 6 as 5, 4 as 9 and 9 as 4. These errors were expected because the images were not very high resolution and we believe the dropouts at each layer may have caused these errors.

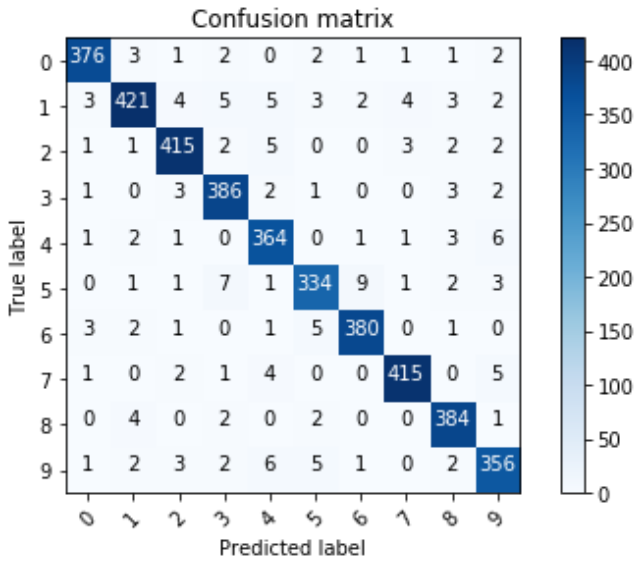

Figure 12: Confusion matrix showing the predicted and the original label

VI. CONCLUSION

We would like to conclude by saying that Convolutional Neural Networks are very efficient at classifying images with a very high accuracy and a fast convergence rate. Although an accuracy of 96% is acceptable, but the model can be further improved by both, working on the pre-processing of the images and by making the model more complex. Working on these suggestions in the future can help in improving the accuracy.

VII. STATEMENT OF CONTRIBUTION

All three members contributed equally to the project. Yu-Cheng Cho was responsible for exploring the activation function, optimizers and CNN architecture. Navpreet Singh was responsible for the data pre-processing CNN architecture, learning rate reductions and size of Mini-batch. Pouriya Alikhani was responsible for literature survey and CNN architecture.

REFERENCES

[1] LeCun, Y. Botou, L. Bengio, Y. Haffner, P. 1998. Gradient-Based Learning Applied to Document Recognition.

[2] Xiao, H., K. Rasul, and R. Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms:1–6.

[3] He, K., X. Zhang, S. Ren, and J. Sun. 2015. Deep Residual Learning for Image Recognition.

[4] Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification.

[5] Simonyan, K., and A. Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition:1–14.

[6] Xiao, T., H. Li, W. Ouyang, and X. Wang. 2016. Learning Deep Feature Representations with Domain Guided Dropout for Person Re-identification:1249–1258.

[7] Girshick, R., J. Donahue, T. Darrell, and J. Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition:580–587.

[8] http://scs.ryerson.ca/~aharley/vis/conv

[9] https://www.coursera.org/learn/deep-neural-network/home/

[10] https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

[11] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[12] https://keras.io/preprocessing/image/#imagedatagenerator-class

[13] https://keras.io/activations/

[14] https://keras.io/optimizers/

[15] https://www.coursera.org/learn/convolutional-neural-networks/home/welcome

[16] https://www.tensorflow.org/tutorials/estimators/cnn?fbclid=IwAR3SQAaI7p40gG83AzgXUvYKubL4F3_qpSGZhJe-TL-iIsA1-T7fjSLDwjg