COMP551-McGill University miniProject3

Kianoosh Ojand-260894514, Dara Shahriari-260715981, Negar Hassantabar-260939318

November 2019

Abstract–MNIST is a database of handwritten data. In this work, our goal was to train a machine learning model on monochrome images containing handwritten digits from MNIST database on a background of different shapes of curves, lines, and dots. They came paired with their respective classes, the largest digit in each image. To reach this goal, we converted the feature vectors of the image data to numerical features and trained several classifiers on deep neural network and convolutional neural network models. Our best result was obtained using ResNet (Residual neural network) with an accuracy of 96.9%.

1 INTRODUCTION

In this work the main task was to design a machine learning model to predict the maximum digit in the images. Our dataset consisted of 50000 images with different backgrounds. For the prepossessing part, we vectorized, normalized, and centred the image data. In the next step we implemented different neural network algorithms. We examined a simple convolutional Network, a deep convolutional network, and a ResNet structure, obtaining the accuracy rate of 79%, 95.1% and 96.9%. Although the validation accuracy in our algorithms was increasing, the rate was so low that an improvement was not possible for hours of extra runtime. In the following sections, we provide a brief background of the related research in this area and explain the pre-processing in detail. In the section 3, we explain the implemented algorithm in more detail. Finally, in section 5 and 6 we provide the results and conclude.

2 RELATED WORKS

Handwritten digit recognition is of great importance in the realm of machine learning. It is used in many applications such as bank cheque analysis, post mail sorting and handwritten form processing. There are many algorithms written to classify digits with low computational costs [1]. Due to their distinguishing ability, fast training, and fast classification time neural networks are generally used for this type of classification problem [2]. When considering what type of neural network to use, based on the criteria of accuracy, training time, memory requirements, etc., the convolutional neural network (CNN) is the clear choice.

The Convolutional neural network (CNN) works properly because of its high capability to learn shapes with varying size [3]. The ability of the CNN's generalization highly depends on the architecture of its network. There are three kinds of layers which can be used in a convolutional neural network: "convolution, pooling, and fully connected layers". Fully connected neural networks usually have so many parameters that a restriction criteria is required for the connecting layers to reduce computational costs and avoid over fitting [4, 5]. In addition, the depth of a neural network is generally results in better performance. However, the vanishing gradient problem makes it difficult to train a deep network. In ResNet, the problem is solved by adding shortcut connections with identity functions to the network[6]. Thus, with such structures we are able to train deeper network because the gradient is preserved with the identity function through the network layers [7].

3 DATASET AND SETUP

Our data set for this project is a modified version of the original MNIST database. Each of the images in our modified dataset contain three digits 0-9 over varying backgrounds. The images are encoded as a list of rows where each index represents a gray scale intensity. These images are serialized as a file of objects that can only be reconstructed by pickling.

Before this data is fed to a neural network, the data needs to be pre-processed. Pre-processing consisted of 3 steps: flattening the rows of each image into a single list of pixels, reshaping the dimensions of the images to work with a CNN, and vectorizing

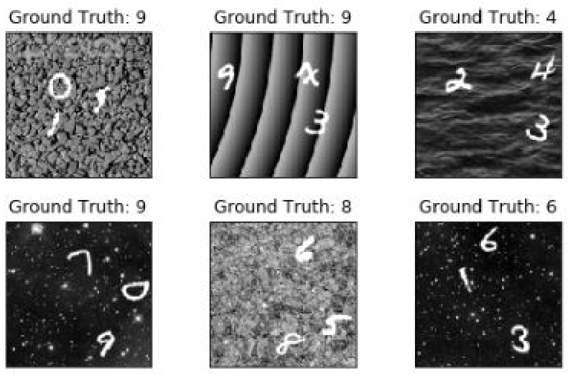

Figure 1: Visualization of the data

the y input values.

4 PROPOSED APPROACH

4.1 Simple CNN with No Validation

Our first model was not successful and suffered from heavy over fitting. We used a very basic 5 layer CNN with little no data augmentation, meaning that the model's learning was specific only to the training set. We also had not split the train data into a validation set, thus after 30 or so epochs our train accuracy was close to 0.99. Testing on a validation set soon revealed that this accuracy was heavily inflated, and that our real accuracy was closer to 0.11.

4.2 Simple 2-layer CNN

In this model, we used a very simple convolutional neural network that reads single channel images (128 x 128). The model had 2 convolutional layers with a kernel size of 5 and a Maxpool layer. The output of the convolutional part of the model is sent to a fully connected neural network which has one hidden layer. In this part of the work, we added a dropout to the network in order to let it learn more important features and avoid overfitting. The activation function used in this model is Relu and log_soft was applied to output. For the optimizer, we used optim.SGD and for the loss function we used nll_loss. It's worth mentioning that we tried out different optimizers and loss functions for this topology but the above mentioned structure gave us the best result: 80% on validation. This model was developed by Pytorch in google Colab.

4.3 Six hidden deep CNN

In this model, we created 6 hidden layer CNN, containing a layer of Convolution using the Relu activation function. This model also had Poolmax and Dropout layers, which we used to avoid overfitting, in addition to a dense layer using the Softmax activation function. In our model, we used Adam as the optimizer and Categorical Crossentropy as the loss function. We split our data into two sets, 40,000 for the train data and 10,000 for the validation set.

4.4 ResNet

One of the approaches we took in this project was adopting some famous image classification models like VGGNet, ResNet, AlexNet, etc. ResNet, based on the original network developed by Kaiming He[6], became our main focus. Also, for the part related to the defining a Resnet Class, we used an existing work[8]. In this approach, firstly, we read the raw data.Then by using `train_test_split` in sklearn library we split the training data into training and validation. We then normalized the data by dividing by 255. Finally we pass the processed data to ResNet in order to do the classification. It's worth mentioning that the original ResNet was developed for 224 x 224 size pictures. In our case, the pictures were 128 x 128. We were posed with two approaches. One was to change characteristics of different layers of ResNet in order to make it fit our images size, which was difficult for us to do. So we tried a simpler method: we changed the dimensions of our input images by using the F.upsample() function, importing "import torch.nn.functional as F" to our python code. The goal here is to move from 128 x 128 to 224 x 224. Re-scaling all images at once was not possible for our hardware. So instead of reshaping the data set all at once, we made our `train_loader` and `val_loader` using original size images. Then, through trial and error, we realized that it was better to resize the input data in each batch right before sending them to training or validation process. This model was developed using the google colab service and developed in Pytorch.

5 RESULTS

In this section we will propose our results based on our proposed models. It should be mentioned that we got best results from Resnet and after it from Six Hidden Deep CNN.

5.1 Resnet results

For this model we used the following settings shown in Table 1

Table 1: ResNet settings

Parameters	numerical values
Learning Rate	0.06
Maximum number of epochs	40
Momentum	default value
Batch size	64
Optimizer	Optim.SGD
Loss Function	CrossEntropyLoss

According to the setting in Table 1 we obtained results shown in Table 2. For this method, we used Adadelta optimizer and we got 0.958 for accuracy in the Kaggle then we changed it to SGD with learning rate of 0.06 and we got a better result is 0.969. We should mention that for this method we tried the model without normalization of the photos but the results were worse so we decided to normalize them for better results. In addition, we tried to normalize the data by subtracting from min value and dividing by maximum value, but results didn't change a lot from the current normalization technique.

Table 2: ResNet Results

Parameters	numerical values
Max num of epochs	40
training loss	8.24e-05
validation loss	0.194
each epoch duration	79.53s
running time in total	3181.47s
padding and filtering	default values

Figure 2 shows the loss function behaviour of training and validation loops in this model and we can see that till the last epoch, both of them are decreasing and it means that model is not suffering over-fitting.

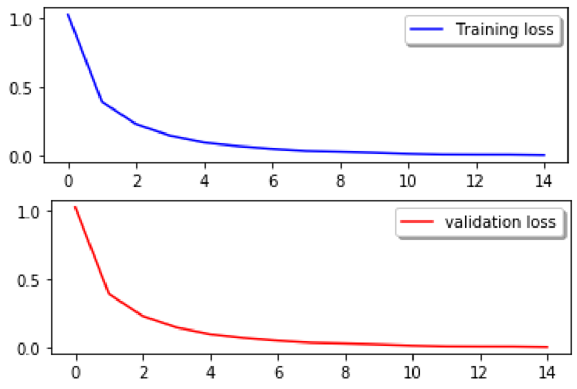

Figure 2: Training loss vs Validation loss

5.2 Simple 2 layer convoultional model results

This was the first model that we developed and it gave us our first result in Kaggle, it consist of two convolutional layers with kernel size of 5 for each of them and two poolmax filters with kernel size of 5 for each convolutional layer. For this model we used original single channel images with size of 128 x 128. For this model we used the setting shown in table 3

Table 3: simple 2 layer convolutional model settings

Parameters	numerical values
Learning Rate	0.03
Maximum number of epochs	24
Momentum	0.4
Batch size	64
Loss Function	F.nll_loss
Optimizer	optim.SGD

Based on the settings in Table 3 we obtained the results shown in Table 4. For this method we tried CrossEntropy loss function but it was not giving good results. We tried using F.nll_loss function and we got the best accuracy for this model. In this model setting Momentum helped a lot to reach convergence point.

Table 4: Simple 2 layer Convolutional model Results

Parameters	numerical values
mean Training loss	0.824
Validation loss	0.6599
validation accuracy	79%
each epoch duration	30.15s
total training time	723.75
padding and filtering	nothing was added

5.3 Six Hidden Deep CNN

For this model we used normalization as preprocessing section and use setting that shown in table 5.

Table 5: 6 hidden deep convolutional neural network setting

Parameters	numerical values
Learning Rate	0.0008
Max number of epochs	200
Momentum	Default Value
Batch size	100
Loss Function	CategoricalCrossentropy
Optimizer	Adam

The obtained result shown in table 6.

Table 6: 6 hidden deep convolutional neural network result

Parameters	numerical values
Training loss	0.1067
Validation loss	0.1916
validation accuracy	95.49%
each epoch duration	53s
total training time	10600s
padding and filtering	Same and default value

We tried different learning rates, batch sizes, and filtering sizes but the optimum solution based on validation accuracy is shown in table 5.

As we can see in Figure 3 both validation and training accuracy increase, so we can assume that the model is not over-fitting on our train data.

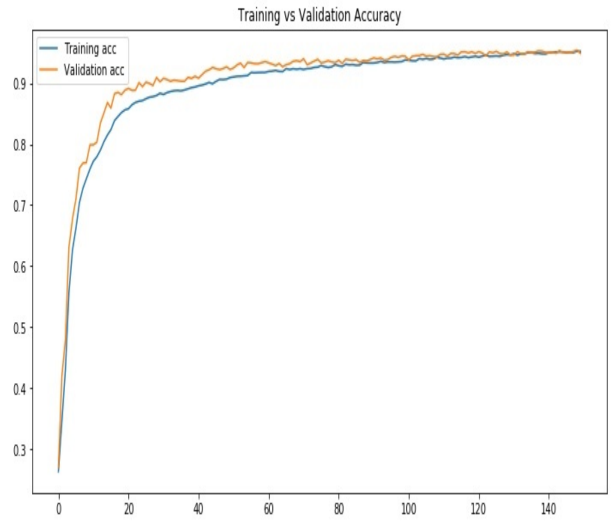

Figure 3: Training vs Validation accuracy

6 CONCLUSION

In this project, we tried using data augmentation on some of our models, but it was not helpful. The lack of improvement may have been attributed to our original data having numbers with different angles and positioning. A case where this was especially noticeable was when the model had to differentiate between 6 and 9.

As shown in our results, we learned that choosing an optimizer and loss function without ample testing was not a good idea. We also learned that using momentum can be very useful in reaching high levels of accuracy, all while using less epoch iterations. It is worth noting, however, that the use of momentum raises the likely-hood of the model not converging.

We tried implementing VGGNet and Alexnet from existing models on the internet, but, unfortunately, we couldn't find a good implementation that yielded sufficient results. Because we were able to find a good implementation, we decided to stick with Resnet [8]. While implementing Resnet, we realized that we had two choices. One was to go deep inside Resnet's code and change its characteristics, including padding, kernel size, and stride, etc. This was a risky decision because it requires full knowledge of the model's inner workings. The other option was to restructure the format of the images to ensure compatibility with the model. We opted with the latter. In this project, we tried Resnet18, Resnet34, Resnet50,Resnet101 and Resnet152, with Resnet18 providing us the best result. We realized that higher complexity doesn't always correlate with better results. In fact, our results depended primarily on the distribution of our data, and the only way we were able to understand which model was superior was via examination.

7 STATEMENT OF CONTRIBUTIONS

Kianoosh developed Resnet model and Simple 2 layer convolutional model and helped in writting the report, Negar developed 6 hidden deep convolutional neural network and helped in writting the report. Dara developed two models but unfortunately his models didn't have good accuracy so we didn't include them in the report and also he helped in writting the report and final editing.

References

[1] Vineet Singh and Sunil Pranit Lal. Digit recognition using single layer neural network with principal component analysis. In *Asia-Pacific World Congress on Computer Science and Engineering*, pages 1–7. IEEE, 2014.

[2] Dejan Gorgevik and Dusan Cakmakov. An efficient three-stage classifier for handwritten digit recognition. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 4, pages 507–510. IEEE, 2004.

[3] Yann LeCun, LD Jackel, Leon Bottou, A Brunot, Corinna Cortes, JS Denker, Harris Drucker, I Guyon, UA Muller, Eduard Sackinger, et al.

Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia, 1995.

[4] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[5] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1131–1135. IEEE, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Zhuxin Chen, Zhifeng Xie, Weibin Zhang, and Xiangmin Xu. Resnet and model fusion for automatic spoofing detection. In *INTERSPEECH*, pages 102–106, 2017.

[8] Francesco Zuppichini. Residual network: Implementing resnet, Jul 2019.