Modified MNIST Image Classification

COMP 551 - Mini Project 3

Surya Kumar Devarajan (260815492)   Ameya Bhope (260849407)
Anindita Golder (260839315)

March 2019

Abstract

The MNIST dataset is one of the standardized datasets that is widely used in Machine Learning Research [1]. In this project, we developed a model that would be able to identify the digit with largest area from the modified MNIST images. Each image consists of multiple digits of varied size. The models implemented were Convolution Neural Networks (CNN), VGG5, and ResNet with different model parameters. The project was in form of a in class competition on the Kaggle website. We achieved an accuracy of 97.78% on validation set and 96.73% on the Kaggle test set.

Keywords— Machine Learning, Deep Learning, Convolutional Neural Networks, Classification, Bounding Box, ResNet, VGG

1 Introduction

As described in and explained in [1], the MNIST Database is actually a subset of a much larger database, the NIST database, which consists of images made up of handwritten letters and numbers while the MNIST database consists of only handwritten digits. The MNIST Dataset is easily accessible and hence is one of the most commonly used datasets in neural networks and deep learning models for classifications. The NIST database is not as compatible and can be used to represent much more complex classification tasks and along with the feasibility of adding more tasks such as author identification and so on.

The modified MNIST Dataset considered for the project consists of images which have multiple handwritten digits. The objective of this project involved developing a deep learning model that would be able to identify the largest digit (according to the area occupied by the digit in the image) from a modified MNIST image.

The original dataset considered for the project was 40000 images for the training set and 10000 images for the test set. The 40000 images in the training set were split into 36000 images as the training set and 4000 images as the validation set to measure the performance of the models that were developed.

The first step was to develop the models that could be tested. We first started with a 4 layer 3x3 CNN with Dropout Regularization and a Reduced Learning Rate. Model 2 consisted of a 4 layer 3x3 CNN with Dropout Regularization and a Learning Rate Scheduler. Next we implemented a VGG5 with Reduced Learning Rate and VGG5 with a Learning Rate Scheduler. Another model consisted of implementing a ResNet model. Lastly, we implemented the Long Short-Term Memory model. It was found that the VGG5 with the Learning Rate Scheduler performed the best. Based on the results obtained using the validation set (an accuracy of 97.78%, we used the best performing model (VGG5 with Learning Rate Scheduler) to predict the digit with largest area for the test set and uploaded our predictions on Kaggle achieving an accuracy of 96.73% which is similar to the validation set accuracy.

2 Related Work

The image classification task with the MNIST Dataset is widely practiced. For this project, we referenced some of the well-known models for image classification to use for our project.

In [2], the implementation of a fast CNN to work with 3 datasets: the MNIST, NORB and CIFAR-10 on a Graphics Processing Unit (GPU) was proposed and the best test error for each of the dataset was measured. All the models consisted of a pre-processing layer where the images were pre-processed followed by the convolutional layers with maxpooling. The final layer was a fully connected layer converting the output of the topmost convolutional layer into a $1D$ feature vector. The best performing model gave an accuracy of 0.35%, 2.53%, 19.51% with the MNIST, NORB and CIFAR-10 Datasets respectively. The experiments were run on a GPU resulting in an increase in the computation speed.

Taking the CNN a bit further, [3] discussed the impact of the depth of the CNN on the accuracy of the model for image recognition tasks. The model consisted of multiple $3x3$ convolutional layers followed by max-pooling layers after which we have 3 fully connected layers. The final layer used softmax activation. After training and validating the different models with different depths, the best model obtained an error of 7.1% and a test error of 7.3% using an ensemble of 7 models. This model was submitted under the name "VGG5" and secured 2^{nd} place in the ILSVRC-2014 Challenge.

In [4], the formation of a residual learning method was proposed where the layers of a deep neural network were re-arranged as residual functions with a reference to the layer inputs. The model consisting of 152 layers was tested on the ImageNet Dataset achieving an error of 3.57% winning the first place in the ILSVRC-2015 classification task. The models were also tested with the CIFAR-10 Dataset and the COCO Dataset. The best performing model gave an error of 6.43% on the CIFAR-10 Dataset and improved the performance relatively by 28% on the COCO Dataset where the baseline considered was the VGG5 Network.

All of the above models were discussed and summarized in [5]. The above models became our baseline for the project task on the modified MNIST Database based on what was discussed in [5].

3 Dataset and Setup

The dataset used for this project was curated from MNIST. The dataset contained images, with each image containing multiple digits.The total dataset contained 50000 samples. It was split into training and test datasets with 40000 and 10000 samples respectively. Modeling the neural net on the 40000 training samples, we were supposed to predict the digit with the largest bounding box area in each test sample. For evaluation of various models we used k-fold cross validation to take different samples from the training dataset to be used as the validation set. The model which performed the best on the validation set, was used to predict the digits on the test set.

The training dataset images were passed through an array of pre-processing steps. This was done primarily by using the *OpenCV* library which contains several programming functions aimed at real time computer vision [6]. We converted all the raw images from greyscale to black and white as it made the digits stand out from the background. Post this, for each digit within the sample, we found the bounding box i.e. a simple boundary which encapsulates the digit accurately. Thus, we select the digit with the largest bounding box area. In the following step, we re-sized the image by interpolation and get the scaled image by padding. After this, the images were normalized, which is a process that changes the range of pixel intensity values. The size of the original image was $64x64$, which, after reshaping became $28x28$. This was done for facilitating the input to the convolutional neural nets. Padding of the digits was also done, so that the digit was more distinct from the background. The labels associated with each image were *one-hot encoded* to convert them from categorical data type to numeric data type. We next passed the images through various data augmentation techniques. First, we accounted for the rotations the digit might have in the test set by training our model with a number of rotations of the same digit and class label. This value is in degrees which randomly rotates the digits in the range 0 to 180. Another parameter here was the random shift, which accommodates the non centered digits. Further we used width shift and height shift that are ranges (as a fraction of total width or height) within which we randomly translate pictures vertically or horizontally. We also use shear range for shearing transformations. These data augmentation features are implemented by using the ImageDataGenerator provided by Keras for pre-processing. This leads us to use fit_generator for training our models. The processed training set was split further into training and validation datasets, with validation dataset being 10% of the training set.

4 Proposed Approach

After pre-processing each image, we implemented a number of models with the objective of achieving the best accuracy. We implemented various architectures of Convolutional Neural Networks (CNN) as they act as automatic feature extractors. The issue with a regular feed forward neural network is that it does not scale well with images. For instance, an $200x200$ RGB image would have 120000 weights for the neurons in the first layer, which could lead to overfitting, enormous space and time complexity. In contrast, a CNN considers the fact that the input is an image and the layers would have neurons arranged in 3 dimensions: width, height and depth[1]. A CNN effectively uses adjacent pixel information to effectively downsample the image first by convolution and then uses a prediction layer at the end[7]. We created the CNN's using the keras framework.

We first tried with two layers having two convolutions each, max pooling layer and a dropout layer. We fixed the kernel size as 3 and also the stride length as 2; however, we varied the number of output filter size. The activation for the convolutional layer was rectified Linear Unit (ReLU). We also experimented with dropout rates setting it to 0.25 and 0.5 and found 0.25 to be performing the best. We also tried Batch Normalization, but it gave inferior performance compared to Dropout of 0.25. In the end, we added a flattening layer and a fully connected layer of 2000 nodes and a dropout layer with a dropout of 0.25. Finally a softmax layer was used to classify into one of the ten digit classes. The callback used was ReducedLROnPlateau which reduces the learning rate once the metric stops improving. We ran this for 10, 30, 50 and 100 epochs. For the second model we used the same CNN architecture with callback as LearningRateScheduler, which schedules the learning rate. We compare the performance of various optimizers on the validation set accuracy and also the validation set loss. They are presented in figure 1. Validation set accuracy for all the models implemented have been listed in Table 1.

To further improve the accuracy, we implemented a variant of the CNN inspired by *VGG5* model[3]. It explores very deep convolutional networks for image recognition and it performed extremely well in the ImageNet competition. We implemented the VGG5 with both the callbacks, as mentioned above. It was implemented on the basis of the Caffe Model provided by the authors. We also implemented a version of ResNet. As the neural networks go deeper, they become difficult to train. ResNet is a residual learning framework that eases the training of networks. The layers are reformulated in a way in which they learn residual functions with reference to layer inputs, instead of learning unreferenced functions. Another model we implemented was the LSTM (Long-Short Term Memory), which is a artificial recurrent neural network architecture[8]. It tries to combat the disadvantage of any new model, which starts solving every problem, with no previous knowledge. LSTM's make use of the loops in them, which allows the knowledge to persist.

Finally, we ensembled all the models, but the model that gave us the best performance on Kaggle was the VGG5 with Learning Rate Scheduler.

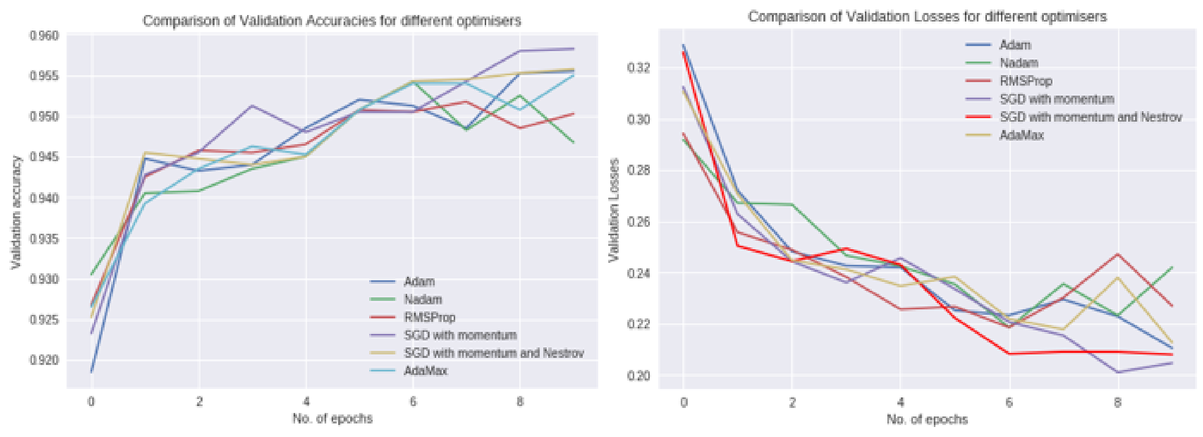

Figure 1: Validation accuracies and Validation losses across optimizers

[1]Here, depth refers to the depth of activation volume and not the depth of entire network

5 Results

In this section we make an analysis of all the models we executed. We also report our results by executing the top performing model in the testing data set on the Kaggle in-class competition.

Model	Callback	Validation Accuracy	Time per epoch
CNN	ReduceLROnPlateau	96.23	17 seconds
CNN	LearningRateScheduler	96.38	17 seconds
VGG5	ReduceLROnPlateau	97.52	37 seconds
VGG5	LearningRateScheduler	97.78	66 seconds
ResNet	ReduceLROnPlateau	97.60	36 seconds
LSTM	ReduceLROnPlateau	95.15	60 seconds
Ensemble		97.70	15.3 seconds

Table 1: List of models executed

We found out that correct pre-processing significantly alters the validation set accuracy results. We first tried scikit-learn image processing library and got 95.76% accuracy on the VGG5 Learning Rate Scheduler Model. To further improve the pre-processing we used OpenCV python library[6]. We executed a variety of models, tuning different hyperparameters for example using several optimizers, varying the dropout rates, using different activation functions like ReLU, sigmoid, tanh. We found the best hyperparameters by performing a k-cross validation with setting k as 10. This is done using sci-kit learn's GridSearchCV class [9]. We obtained our best result by ensembling all the models. For ensembling, we used a simple *argmax* function which maximizes the training weights from each of the models. Although the performance of all the models was roughly similar around the 96.5% range for a standard 50 epochs on the validation set, we achieved the best accuracy of 96.73% on Kaggle using the VGG5 with the Learning Rate Scheduler.

Figure 2 shows that the training set vs. validation set accuracy converges. Similarly, we also see that the losses converge too. The confusion matrix diagram, shows the relationship between the actual and predicted values, for each of the 10 digits[2].

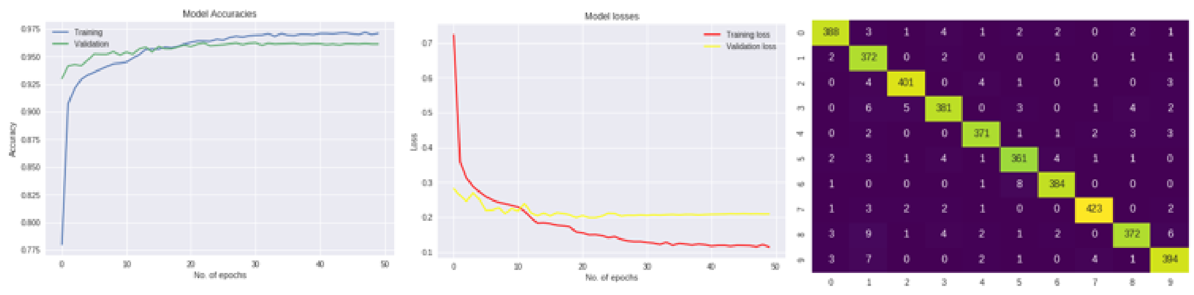

Figure 2: Training vs. Validation Accuracy (Left), Loss(Middle), Confusion Matrix (Right)

6 Discussion and Conclusion

In this project, we classified the handwritten modified MNIST dataset using different deep-learning algorithms. We followed the pipeline approach for our implementation: 1. Analyze the dataset 2. Preparing the dataset 3. Create the model 4. Compile the model 5. Fit the model 6. Evaluate the model 7. Summary. The approach we used was quite effective in producing efficient results, however we can further improve our accuracy by adopting a more efficient pre-processing scheme. We learned that

[2]Some graphs and analyses were avoided because of the limitation of space, but they can be found in the Jupyter notebook provided

even with sophisticated models like VGG5 and ResNet, the accuracy is quite similar to a simple CNN model. This indicates that not only the model/architecture selection, but pre-processing is an extremely crucial part of the building top-performing models. One of the limitations we faced was the limited computation power on Google Colab. In the future we would like to attempt to experiment with various deep learning architectures and hope to improve the accuracy of the model. We implemented our models using Keras. PyTorch framework which allows more flexibility and customization, might give better results. Further, FastAI, which runs on top of PyTorch might also help in achieving better performance.

7 Statement of Contributions

We all three contributed to the project quite equally. We exchanged our ideas about the implementation of various tasks and also worked together to write the report.

References

[1] J. Tapson G. Cohen, S. Afshar and A. V. Schaik. Emnist: Extending mnist to handwritten letters. *Conference on Neural Networks (IJCNN)*, 2017.

[2] Jonathan Masci Luca M. Gambardella Jurgen Schmidhuber Dan C. Cires¸an, Ueli Meier. Flexible, high performance convolutional neural networks for image classification. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[3] Karen Simonyan Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[4] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. 2015.

[5] William L. Hamilton. Comp 551 - applied machine learning, lecture 15: Convolutional neural nets, Feb 2019.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] 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.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.