Modified MNIST

COMP 551 - Group 11

Beatrice Lopez 260654565
Minh Anh Nguyen 260671180
Amanpreet Walia 260834477

March 2019

Abstract

The image analysis prediction challenge of this project sought to identify the numeric digit which occupies the most space in images in a dataset of handwritten digits. The MNIST dataset is a classic dataset of handwritten numeric digits from 0-9. In this project, we worked with a Modified MNIST dataset where each image contains more than one digit, each of a different size. The aim was to find the network with the highest accuracy and best generalization, yet is easy to set up and train. Our work in this project involved pre-processing the data and training different convolutional neural networks including ResNet, VGGNet as well as using techniques such as spatial pyramid pooling and optimizing with ADADELTA and RMSProp. We found that our 18-layer ResNet implementation gave the highest accuracy on the final test set on Kaggle with an accuracy score of `97.533`%.

1 Introduction

Handwritten digit recognition has many direct applications, including recognizing license plates, zip codes for mail sorting, bank cheque amounts and numeric entries in tax forms. In research, it is used as an important benchmark for computer vision.

For many handwritten digit recognition tasks, the original MNIST dataset[1] is largely used for training various image processing systems. In this project, we worked with a modified version of the MNIST dataset to explore the applications of deep neural networks in computer vision and image analysis prediction.

The Modified MNIST dataset consists of 50,000 grey-scale 64 x 64 pixel images with each containing more than one digit. For training and testing, 40,000 images are used as training set and 10,000 for testing set. This project sought to predict the digit which occupies the most space in each image i.e. the digit with the largest bounding box. Every image in the dataset comes with an associated label that identifies this digit, and the 4096 pixels that make up the image.

Our work involved pre-processing the data and training various networks. For pre-processing, we used normalization, image stacking in three channels, reshaping, one-hot encoding for the labels and splitting the training set to make a validation set, image augmentation like random rotations and random shifts. For training, our approach was to replicate top-performing neural networks to achieve state-of-the-art results. The networks we implemented include two convolutional neural networks (CNN) of 4 layers taken from previous Kaggle competitions, as well as replications of ResNet [2] and VGGNet [6]. Overall, we found that all of these models performed well, given sufficient training examples, time, and compute power. However, ResNet consistently performed better than any of the other networks, converging faster than any other network. After fully training the network on approximately 40 epochs, our ResNet implementation achieved an accuracy score of `97.533`% on the Kaggle leaderboard.

2 Related Work

There has been extensive research conducted on the image classification task in general, and MNIST digit recognition in particular. Below are some networks from notable research papers we attempted to replicate in this project.

1. **AlexNet**, proposed by Krizhevsky et al. [2012], was the deep CNN that started the revolution of image classification. It has 60 million parameters and 650,000 neurons, and consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax.

2. We implemented **VGGNet** by Simonyan and Zisserman [2015], a CNN of 16 layers, 138 million parameters, and only 3x3 convolution filters. It employs max-pooling and a final ReLU activation.

[1] `http://yann.lecun.com/exdb/mnist`

3. We also explored **Spatial Pyramid Pooling (SPP-Net)**, a pooling strategy proposed by He et al. [2015], which allows networks to generate a fixed-length representation for datasets of any image size, with state-of-the-art results and faster training speeds.

4. He et al. [2015] recognized the difficulty in training deep convolutional neural networks, and presented a residual learning framework called **ResNet**. It addresses the degradation problem of training accuracy by explicitly letting layers fit a residual mapping, instead of hoping each few stacked layers directly fit a desired underlying mapping. ResNet was able to achieve super-human accuracy. Motivated by this, we replicated a ResNet with 18 layers, which became our best-performing network.

In addition to these networks, our research led us to the optimization method **ADADELTA**, a per-dimension learning rate method for gradient descent, proposed by Zeiler [2012]. ADADELTA dynamically adapts over time using only first order information and thus has very little computational overhead beyond normal stochastic gradient descent. We found this optimization technique generally improved the performance of our models as compared to SGD with decaying learning rate.

3 Dataset and Setup

The Modified MNIST training set consists of 40,000 grey-scale 64 x 64 pixel images with an associated label of digits from 0 to 9 for each image, while the test set contains 10,000 of such images without labels. Each image contains multiple digits of different sizes and a background consisting of noise. Some images were also transformed by being rotated or translated.

For each of our networks, our pre-processing included normalization to have a common scale and faster convergence, reshaping, and one-hot encoding for labels. We also found that cleaning the images by removing the background noise generally improved performance.

For certain networks, we used data augmentation to generate transformed images, using random rotations, random zooming, random vertical and horizontal shifts, random vertical and horizontal flip, and ZCA whitening[2] to decorrelate features by transforming data in such a way that its covariance matrix is the identity matrix. This allowed us have more diverse data to train our networks. Figure 1 below visualizes some examples of the transformations we used: random rotation, random horizontal shift, and a combination of rotation and horizontal shift.

In addition to that, we tried background noise removal using handcrafted methods in OpenCV. This leads to better convergence time as network won't have to learn the the background is unimportant for classification, however our final accuracy did not change significantly. For training VGGNet, we stacked up image in one channel to three channels (RGB) to match the input dimensions of the network.

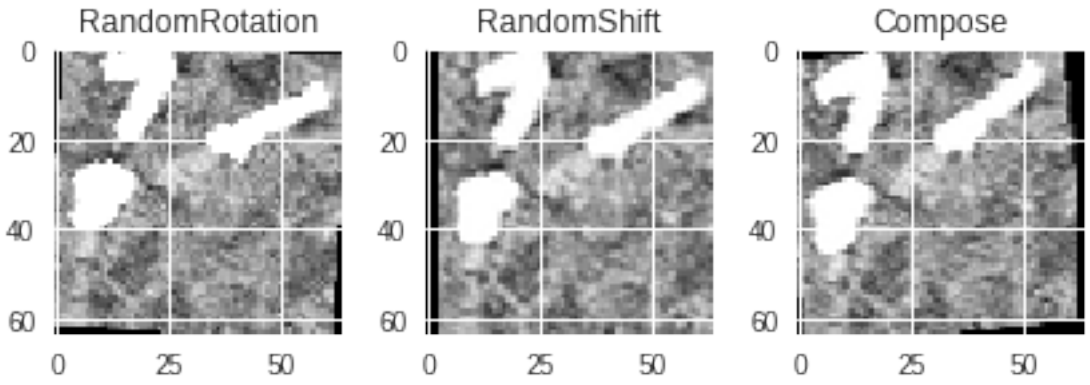

Figure 1: Examples of image transformations: 1) Rotation 2) Horizontal Shift, and 3) Both

4 Proposed Approach

Our overall approach was not to implement our own CNNs from scratch, as we thought it would be time-consuming and ineffective in regards to finding the best network. Instead, we focused on finding and replicating the best-performing networks in previous Kaggle competitions for MNIST dataset as well the state-of-the-art networks in recently published research papers on digit recognition and image classification.

[2] `https://martin-thoma.com/zca-whitening/`

1. Our first network, **CNN-6**, was implemented with Keras and replicated from a Kaggle notebook[3]. The network consists of 4 convolutional layers and 2 dense, fully-connected layers. Each convolutional layer uses ReLU as the activation function, proposed by Agarap [2018]. The kernel filter matrix of each layer is applied to the whole image. It then has 2 max-pooling layers, which down-sample each image by picking the maximum value of every 2 neighboring pixels. This not only reduces computation cost, but also helps to reduce overfitting. Dropout regularization is utilized, which randomly chooses a set of neurons to be ignored during the training phase. This leads to a network that is capable of better generalization and is less likely to overfit the training data.

2. Our second network, **CNN-7-BN**, was replicated from a PyTorch implementation from another Kaggle kernel[4]. It contains 4 convolutional layers, followed by 3 fully-connected layers. Each convolutional layer has kernel size 3. Similar to CNN-6, this network employs ReLU as the activation function, max-pooling,and dropout regularization. However, CNN-7-BN uses batch normalization by Ioffe and Szegedy [2015] at every convolutional and fully-connected layer, which normalizes the input of hidden layers, allowing us to have faster training times and much higher learning rates.

3. We implemented VGGNet from Simonyan and Zisserman [2015] by loading a pre-trained model, which greatly eased the setup and training process. Our network, known as **VGG-16**, has 16 layers: 13 3x3 convolutional layers, followed by 2 fully-connected layers and an output layer with softmax activation of 4096 nodes. Although it is generally inferior in performance to ResNet, VGG-16 is capable of classifying 1000 labels for Imagenet dataset, we expected it to perform well in classifying our 10 digits.

4. Our final and best performing network, **ResNet-18** was based on ResNet with 18 layers by He et al. [2015]. A single model of ResNet achieved 4.49% top-5 validation error, while an ensemble of 6 ResNet models achieved 3.57% top-5 validation error. In regards to the architecture, ResNet is not too different from VGGNet as it also uses mostly 3x3 convolutional layers. However, ResNet starts with a 7x7 convolutional layer, and ends with a global average pooling layer and a 1000-way fully-connected layer with softmax activation. The brilliance behind ResNet is reformulating the layers as learning residual functions with reference to the layer inputs, which allows for much deeper networks and higher accuracy.

In addition to this, we also attempted implementing a pooling strategy to detect objects at various scales, like implementing **SPP-Net** He et al. [2015] as well **SVHN** Goodfellow et al. [2013] to build a classifier by modifying the last few classification layers; however those models didn't work well as computation time per epoch was too large and convergence to good accuracy was not reachable soon.

5 Results

Each network was evaluated using prediction on the validation set. We set aside 10%, i.e. 4000 images, for the validation set. We thought this is large enough for us to have a fair evaluation of the generalization of the network, while still having an ample amount of training examples to train the network.

Table 1 below lists the neural networks we implemented, each with its accuracy on the validation set, accuracy on the Kaggle test set, and average runtime per epoch.

Neural Network	Validation Accuracy	Test Accuracy	Runtime per epoch
CNN-6	0.9138	0.91100	30-40 minutes[5]
CNN-7-BN	0.9210	0.93733	1.5 minutes
VGG-16	0.9609	**0.97433**	22 minutes
ResNet-18	0.9728	**0.97533**	5 minutes

Table 1: Neural networks with validation accuracy, test accuracy and runtime

Overall, all of the neural networks we implemented performed well, with accuracies of at least 90%. We submitted each of these networks to the Kaggle leaderboard and found that they generalized well, each giving a final test accuracy similar to or higher than their respective validation accuracy. Although CNN-6 and CNN-7-BN performed reasonably well, we found that VGG-16 and ResNet-18 significantly and consistently outperformed them.

CNN-6 did not perform as well as we expected, as it achieved 0.99 accuracy in the original MNIST Kaggle competition. We attributed this to working with a different MNIST dataset and not training long enough, due to time constraints. Depending on the batch size, CNN-6 took 30-45 minutes per epoch.

[3] `https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6`

[4] `https://www.kaggle.com/juiyangchang/cnn-with-pytorch-0-995-accuracy`

[5] Runtime varies with batch size

CNN-7-BN was well-trained, since each epoch only took 86 seconds on average. We tested it at 60 epochs and 120 epochs, but also found that it did not perform as well as we expected.

We implemented VGG-16 with the intention of simply using it as a baseline for our networks. However, VGG-16 exceeded our expectations with a score of 0.97433 on the final test set. We think this is because VGG-16 is powerful and able to classify 1000 image labels, but in this project we are using it for a less complex task of classifying 10 labels.

Our single-model ResNet-18 performed very well, which was what we expected. With a test accuracy of 0.97533, this was our best network and put us in the 18^{th} position in the Kaggle leaderboard. Given more time, we would like to make an ensemble of multiple ResNet models. We believe this would give us state-of-the-art accuracy.

It is worth mentioning that we implemented and trained AlexNet and SPP-net. However, they had relatively poor performance so we will not discuss them in detail here.

6 Discussion and Conclusion

We learned that the performance of a network depends not only on the architecture of the network, but also on the nature of the image data and what pre-processing tasks were applied to the data.

For pre-processing tasks, we find that removing background noise improved validation accuracy by 1-2% for most of the networks. The image transformations (e.g. random rotations, random flips, etc.) seemed to improved performance.

Regarding runtime, we concluded that batch normalization significantly reduces training time, as CNN-7-BN trained much faster than CNN-6 and the one major difference between their architecture is the presence of batch normalization. We also noticed that generally, a larger batch size correlated with longer runtime and more computational power - but not necessarily higher accuracy.

For optimization, we worked with both ADADELTA and RMSProp and found that generally ADADELTA performed better with exception of CNN-6. With ADADELTA, we observed that accuracy often fluctuated between epochs, but overall accuracy increased and convergence reached sooner.

Regarding the models, ResNet-18 performed the best, followed by VGG-16, which is what we expected, as they are architecturally much more extensive networks. Our two custom CNNs did not perform as well, but still gave 91-92% accuracy.

We extensively trained CNN-7-BN for 120 epochs, recorded in Figure 2, expecting overtraining to occur. To our surprise, validation loss did not increase and validation accuracy did not decrease. We believe overtraining was prevented in CNN-7-BN due to batch normalization, dropout regularization, and max-pooling.

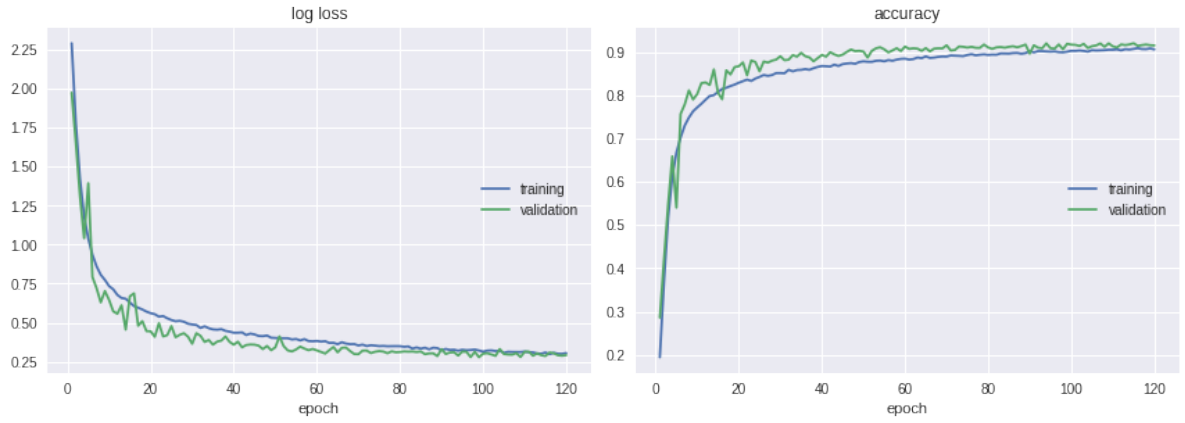

Figure 2: Batch normalization prevents overtraining CNN-7-BN

Possible directions for future investigation include training deeper ResNet models, such as ResNet-34 or ResNet-50. We would also like to perform ensembles, such as boosting CNN-6 or CNN-7-BN to reduce variance, and stacking on multiple ResNet models to minimize loss and achieve state-of-the-art accuracies.

7 Statement of Contributions

Each team member read research papers, studied various kernels on Kaggle, and implemented at least one CNN. In particular, Walia worked on ResNet-18 and VGG-16, Nguyen on CNN-6 and CNN-7-BN, and Lopez on CNN-7-BN. Each member also trained on the different networks implemented by the team and thus gained familiarity with all models. Walia took the lead in project design and method, while Nguyen and Lopez took responsibility over the report.

References

[1] Abien Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay D. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *CoRR*, 2013.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *IEEE Transactions On Pattern Analysis And Machine Intelligence*, volume 37, pages 1904–1916, 2015.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 2012.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556v6*, 2015.

[8] Matthew Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.