Machine Learning in Practice

Classifying MNIST using various methods

Alex Suciu Brigel Pineti Laurens Kuiper Ankur Ankan Luca Parolo
Rick Dijkstra Tristan de Boer

March 2018

1 Introduction

The seven of us together make up the team "**The Abstracted Boars**", and this is our progress report for the first part of the *Machine Learning in Practice* course. The aim of this course is to become familiar with using machine learning methods on real-world problems by entering a competition on Kaggle.com. For the first part of the course we decided to enter a beginner's competition, because not all of us have much experience.

In this report we discuss the challenge, the different approaches that we tried, the results they yielded, and the final ranking that we achieved on Kaggle.

2 Challenge: classifying MNIST

The MNIST database consists of 70,000 grey scale handwritten digits and their associated labels, written by approximately 250 different writers. The original set is split into a training set of 60,000 images and a test set of 10,000 images. For the Kaggle challenge set is split differently: the training set has 42,000 images, while the test set has 28,000. The size of the images has been normalised to dimensions of 28×28 pixels.

3 Approach

Many classification methods yield low error rates on the MNIST dataset. We decided to split up and try as many of them as possible to expand our knowledge before attempting to achieve our best possible result. In this section we describe the methods that we have explored. We first describe our pre-processing methods before discussing the different classifiers.

3.1 Augmentation

Research on image classification has shown improved results when using data augmentation on MNIST and various ImageNet datasets [2,5,6,11]. Data augmentation increases the size of the data set through creating new data from existing data by editing the latter in various ways. If done correctly, augmentation can give a more robust model by preventing overfitting. We explored two general augmentation approaches: augmenting the entire training beforehand set to create an improved data set (offline), and augmenting the images during training (on-the-fly). We used both built-in functionality from `Keras`, and the `imgaug` library to augment the images.

3.1.1 Offline Augmentation

Common augmentation techniques include horizontal/vertical flips (not useful for this challenge), slight random rotations, elastic deformations, and adding gaussian noise. A less common technique was used in the DropConnect paper [11]: random 24×24 cropped patches were created from the original images, then scaled back up to 28×28, before applying other augmentations. This method increases the size of the data set drastically, but it yielded a lower accuracy compared to using no augmentation at all in our early tests, so we decided to drop this method.

Due to memory restrictions on *Google Colab* (a shared notebook environment with GPU's) we could only increase the data set size threefold for our final result (the entire set had to fit in memory). Gaussian noise did not seem to improve our results much, so we decided to use just the following set: {unaugmented + rotated + deformed}. The augmented images were used for training only.

3.1.2 On-The-Fly (Online) Augmentation

On-the-fly augmentation augments the images for every batch that is fed to the model. The images are not saved, therefore this consumes much less memory. An advantage of this method this is that it is possible to keep training the model with images that are augmented in different ways every time. However the downside is that training the model is much slower because augmentation (especially elastic deformation) takes time.

Based on the results from the offline augmentation we also used the same augmentations on the on-the-fly augmentation.

3.2 Classifiers

3.2.1 Convolutional Neural Network

We started off by implementing LeNet-5 [8], a simple network consisting of two convolutional layers, both followed by max pooling, followed by a 1024 neuron fully connected layer, after which we apply a dropout of 0.5 before softmax classification. As expected this gave good results: CNNs excel at this. However, there was much room to improve.

DropConnect [11] is an alternative to dropout, where edges instead of activations are randomly set to 0. We modified LeNet-5 to incorporate a dropconnect ratio of 0.5 instead of dropout, and saw a small boost in accuracy.

Fractional Max Pooling [4] is a version of max-pooling where the pooling factor is allowed to take on non-integer values. We replaced the max-pooling layers in our modified version of LeNet-5 with random overlapping pooling layers with a pooling factor of $\sqrt{2}$ and saw another small boost in accuracy.

3.2.2 Random Forest & Gradient Boosting

We also tried a Random Forest classifier, which achieved an accuracy of 93.81%, and Gradient Boosting [1], applied with with softmax classification and a learning rate of 30% on a tree with depth 10, which resulted in an accuracy of 97.11%. We stopped experimenting with these methods due to obtaining much better results using others.

3.2.3 Network in Network

Network in Network [9] replaces the convolution layers in the network with MLP on each pixel of the image. We tried to improve the results presented in the paper by using batch normalization and adding gaussian noise. The paper reports an error of 0.45% on MNIST dataset but we were unable to recreate their results and got a higher error rate in our implementation. Adding Batch Normalization and Gaussian Noise to the network did improve the results a bit.

3.2.4 Binary Connect

Binary Connect [3] stochastically sets the values of all the weights in the network to either $+1$ or -1 after each batch while training. Since Keras doesn't have the functionality to change the weights on the GPU while training, our implementation of Binary Connect was incredibly slow and training took a couple of hours per epoch. We were not able to get any substantial results and discontinued working on this method.

3.3 Pre-trained models

Our team coach encouraged us to try out pre-trained models and fine-tune them on the MNIST dataset. This approach is known as *transfer learning.* We used Keras to import both the VGG-16 and VGG-19 networks [10], which are pre-trained on thousands of ImageNet images. Both networks have a minimum input size of 48×48 pixels and expect three colour channels. By scaling our images up to 56×56 and copying the grey scale channel three times both networks accepted the MNIST images.

We tune the network to our problem by adding fully connected layers. We tried different variations of layers but eventually stuck to just one fully-connected 512-neuron layer with ReLU activation and a softmax layer, because worked well for us, and has worked well before in a competition [7]. We experimented with completely freezing the layers in VGG (untrainable), unfreezing the last three layers, and unfreezing every layer. The best results were had with unfreezing the last three layers.

3.4 Ensemble Learning

The transfer learning approach gave us the best results out of the approaches that were tried. We decided to ensemble different fine-tuned pre-trained models for our final result. The difference in the models was created with a fivefold split of the dataset each with a different validation split, and augmenting the training splits. Five VGG-16's were trained, each on a different split, for 15 epochs, as well as five VGG-19's. Different combinations of these were ensembled by adding up their predictions on the test set.

The ensemble of five VGG-16's outperformed any combination of ensembles with VGG-19's.

4 Results

Our results are presented in Table 1. We abbreviate Network in Network to NiN. Our VGG-16 fivefold ensemble performed best, yielding an error of 0.4%.

Table 1: Results

Method	Result (error%)
NiN (3 mlp layers)	1.41%
NiN (3 mlp layers + Dropout)	1.23%
NiN (3 mlp layers + augmentation)	2.06%
NiN (4 mlp layers + Batch Normalization + Gaussian Noise)	1.16%
Random Forest	6.19%
Gradient Boosting	2.89%
LeNet-5	1.00%
LeNet-5 DropConnect	0.95%
LeNet-5 DropConnect + FMP	0.92%
LeNet-5 + Augmentation (offline)	2.015%
LeNet-5 + Augmentation (on the fly)	2.034%
VGG-19, frozen, 2x 128 dense, dropout:	2.81%
VGG-19 unfrozen, 512 dense	1.14%
VGG-19 last 3 unfrozen, 512 dense	0.9%
VGG-16 fivefold ensemble	**0.4%**
VGG-19 fivefold ensemble	0.61%
VGG-16 fivefold + VGG-19 fivefold ensemble	0.44%

5 Conclusion

The ensemble of pre-trained models gave us our best results. With an accuracy of 99.6% we've reached position of 207 out of 2246 on the Kaggle leaderboard (last checked April 16th).

What is interesting is that augmentation often decreased our accuracy when used in simple networks (LeNet-5 and NiN), while it gave great results with the deeper convolutional pre-trained networks (VGG). As for the other classification methods that we have tried: while they were interesting and some of them gave good results, they had problems such as being hard to implement, taking much too long to train, or generally being less suitable for this challenge. Therefore we will likely be using pre-trained models earlier on for the next challenge.

We would like to add that the fully labeled MNIST data set is available, and that it is possible to completely overfit on it. Therefore it should come as no surprise that the top of the leaderboard for this challenge is littered with (close to) 100% scores.

Individual contributions

Everyone worked on LeNet-5 using a tutorial at the start, to learn.

- Ankur and Brigel worked on BinaryConnect and Network in Network.
- Laurens and Tristan worked on offline augmentation, improving the LeNet-5 baseline, pre-trained methods, and ensembles.
- Luca and Alex worked on Random Forest, XG-Boost, and ensembles.
- Rick worked on on-the-fly augmentation and improving the LeNet-5 baseline.

GitHub project link:
https://github.com/alexthe2nd/machinelearningin practice

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[2] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 3642–3649. IEEE, 2012.

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015.

[4] Benjamin Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014.

[5] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.

[6] DMJ Klep. Data augmentation of a handwritten character dataset for a convolutional neural network and integration into a bayesian linear framework. 2016.

[7] Kyung Mo Kweon. How far can we go with mnist?? https://github.com/kkweon/mnist-competition.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[9] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[11] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.