Image Classification: Modified MNIST

Team Kineserne

Chuan Qin	chuan.qin2@mail.mcgill.ca	260562917
Tobias Engelhardt Rasmussen	tobias.rasmussen@mail.mcgill.ca	260810689
Guanqing Hu	guanqing.hu@mail.mcgill.ca	260556970

Introduction

The object of this project is to derive a machine learning model that could classify the arithmetic operation results of images. The training data, including labels, is 50,000 images consisting of two handwritten digits and an arithmetic operation. Both the training data and test data were downloaded from Kaggles's Modified MNIST competition page. [1]

This paper is structured as follows. In Section Related work, relevant papers and online resources were referenced and discussed. Data pre-processing and feature design are presented in Section Problem representation. Section Algorithm selection and implementation describes three main algorithms that have been implemented, including Logistic Regression (LR), Neural Network (NN), and Convolutional Neural Network (CNN) as well as their optimization. This section also briefly introduces how Google Compute Engine has been set up with graphics cards to greatly speedup learning. Detailed results and discussion can be seen in Section Testing and validation and Discussion.

The CNN model implemented in this paper reaches 98.70% accuracy on the test set, ranking first in Kaggle competition.

Related work

Convolutional neural network is the main focus of this project, for its giving the highest performance when dealing with the given problem. This section mainly presents related works of CNN.

The LeNet architecture was regarded as pioneering architecture of CNN and firstly introduced by LeCun et al. in [1]. Each convolutional layer of this model had 3 parts: convolution, subsampling(pooling), and nonlinear activation. Fully connected layer and Gaussian connection layer came next before output. This model wass designed to identify the digit in 32x32 images and the accuracy reached over 99%.

There are some successful models ranking top 2 in recent years' ILSVRC (ImageNet Large Scale Visual Recognition Challenge). AlexNet[2], the championship of ILSVRC 2012, brought up some new technical points. Firstly, they used ReLU as their activation function instead of sigmoid because ReLU does not have gradient vanishing problem that sigmoid and tanh function have. Secondly, they used dropout layer to randomly turn off certain amount of neurons to avoid overfitting. They also used data augmentation like cropping and flipping a 256x256 image to a 224x224 one, which increased the size of train data by a factor of 2048 (32x32x2). GPU was also used to accelerate their computation. In that year's competition, their error rate was only 15.4% which was an astounding improvement compared with the runner up which achieved an error rate of 26.2%.

VGGnet was developed by researchers of Google Deepmind and Visual Geometry Group of Oxford University[2]. It created a 19 layer CNN that strictly used 3x3 filters with stride and pad of 1, along with 2x2 maxpooling layers with stride 2. Their use of only 3x3 sized filters was quite different from AlexNet's 11x11 filters. The combination of two 3x3 conv layers had an effective receptive field of 5x5 but decreased the number of parameters and used two ReLU activation instead of one. The number of filters doubled after each maxpool layer. This reinforced the idea of shrinking spatial dimensions, but growing depth. This model got runner up in ILSVRC 2014 with an error rate of 7.3%

[1] https://www.kaggle.com/c/comp551-modified-mnist/data

Problem representation

In this project, the training data is given in a csv file containing 50,000 lines. Each line has 4096 pixel values, representing a 64x64 image. Each image consists of two handwritten digits and an arithmetic operation. The operation letter could either be 'A' for add or 'M' for multiple. The label of each image is the operation result, for example, the label of Figure 2 is 6 as the result of six times one. Test data is given in the same format containing 10,000 images.

Both train and test data were converted to PNG image format before further processing in order to take advantage of PNG's light weight and visualization. The size of the data set decreased by 10 times in PNG format and thus data can be loaded faster.

Noise Filtering

All the images in the given training and test sets have some noises points and unrelated background textures (Figure 2). We came up with a straightforward idea that removing the background and noise and changing the image to black and white would benefit the training. To achieve this, a threshold was calculated for each image based its pixels' mean value. Then each pixel value was reset to either 1 or 0 based on its above or below the threshold. An image-base filtering threshold improves our filtering results. Two examples can be seen in Figure 1.

Please note that, these filtering images have been used only in logistic regression and neural network implementations, but not convolutional neural network. Because convolutional neural network is able to extract image features on its own. The noise and background textures even improve the performance of image augmentation which will be discussed in the next subsection.

Figure 1: Images before and after filtering

Image Augmentation

Data is the key to deep learning. It is extremely hard to train a good model with small amount of data. Augmentation is a very efficient way to increase the size of image data by generating modified copies of the existing ones, which allows learning with smaller data set and reduces overfitting.

Augmentation modifies existing images by changing their brightness, width, height and applying mirror effect, zooming, and rotation. In this project, Keras image augmentation library has been used for a quick implementation of this feature. It generates augmented images using random combinations of augmentation parameters mentioned above in their predefined ranges. Please noted that, mirror effect is not applicable for text related applications like this project, but it is very useful for object related problems, like Kaggle's famous cat and dog classification.

An example of image augmentation can be seen in Figure 2 and Figure 3. These examples were generated using a rotation range of 15 degrees, width shift, height shift, and zoom ranges below 0.1, and a shear range below 0.3. Rotation range was set to a small value because the raw images have already been applied a maximum 45-degree random rotation. The other parameters are also set smoothly in order to prevent loss of information (e.g. cutoff of letters). The background texture actually provides image augmentation a larger 'inventing space'.

Figure 2: Raw image example

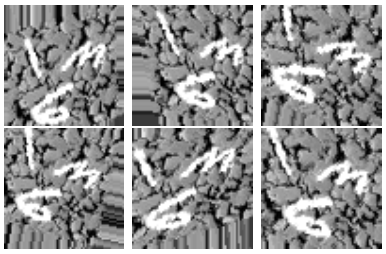

Figure 3: Images generated from augmentation

Algorithm selection and implementation

The implemented solution mainly consists of three stages. Firstly, raw data has been pre-processed using the methods mentioned in Section Problem representation. Following that, a classifier is selected, trained, and optimized. Finally, predicts of test data are computed using the optimized classifiers.
Table 1 summarizes the performance of the three classifiers we chose to approach the problem. Logistic regression algorithm was implemented using Python sklearn library and is used as a baseline. A feed-forward neural network was implemented from scratch as a practice to better understanding it. Then, a convolutional neural network was trained and optimized using Python Keras library.
Among these three algorithms, CNN gives surprisingly good performance, a 98.7% accuracy on test set, which ranks No. 1 in Kaggle's competition. Its architecture, training, and fine tuning methodologies can be found in Subsection Convolutional Neural Network (CNN).

Algorithm	Validation Accuracy (%)	Test Accuracy (%)	Running Time (min)
LR	3.28	-	180
NN	11.0	12.4	11
CNN	97.64	98.70	60

Table 1: Performance of algorithms summary

Logistic Regression

The implementation of logistic regression is straightforward. The idea is to train several linear classifiers and go with the one that gives the highest confidence.
This project is a 40-class classification problem. So firstly, linear Classifier Zero is trained based on if the label of a image is 0 or not. Then thirty-nine more such classifiers are trained. During prediction, each of the classifier generates a probability of how possible the image belongs to it, then, the one that gives the highest probability becomes the final prediction.

Neural Network

A fully-connected NN was implemented first. It is both a baseline of CNN and a good practice to better understand feed-forward neural network.
Pre-processed black and white training data (described in Section Problem representation) was used to simplify the computation.
After loading training data and splitting it into training and validation set, two major functions *"genNN"* and *"predNN"* were called for NN training and evaluating. The first function is capable of customized number of layers, number of nodes in each hidden layer, training rate, number of epoch and batch size. The *"predNN"* function uses the weight model generated from *"genNN"* to make predictions.

The *genNN - function* is hard-coded to use sigmoid activation function. The feed forward algorithm was implemented based on Equation 1 and 2 where $\sigma 1$ is the sigmoid function on each element in the vector. The output of the previous layer is the input of the next layer.

$$O_0 = \sigma(X \times W_0) \tag{1}$$

$$O_i = \sigma(O_{i-1} \times W_i) \tag{2}$$

The back-propagation was implemented based on Equation 3 where Y is the desired label. The program finds the change in the weights for each example and accumulates it based on user defined batch size.

$$\delta = O \cdot (1 - O) \cdot (Y - O) \tag{3}$$

The weight changes are computed using corrections multiplied by a learning rate (Equation 4) and accumulate for the whole batch. And the weights only get updated when the batch size is reached.

$$\delta_i = O_i \cdot (1 - O_i) \cdot (\delta_{i-1} \times W_i) \tag{4}$$

In terms of training the neural network, there were 4 hyper parameters to be determined by cross validation. The data set was split into 45000 training samples and 5000 validation samples. The number of epochs (iterations) and the learning rate were tuned simultaneously. It was observed that the error rate was minimized with a learning rate of $\alpha = 0.05$ after 11 iterations. These parameters ware trained on a network with one hidden layer and 12 hidden units. Next parameter is the number of hidden layers. Based on cross validation, 1 hidden layer achieves the highest accuracy at 11%. Two or more hidden layers do not improve the accuracy. The number of hidden units in each layer should be around $\log_2(4096) = 12$ [4] and some testing shows that the

accuracy does not improve with more neurons. Table 2 shows the validation accuracy corresponding to different number of nodes in the hidden layer where 12-node hidden layer performs the best.

Table 2: The validation accuracy after training a different number of nodes, in one hidden layer and with $\alpha = 0.05$

Nodes	6-8	9	10	11	12
Val. Acc.	10.74	10.78	10.8	10.72	11.0

13-15	16	17	18	19
10.74	10.82	10.76	10.6	10.6

Convolutional Neural Network (CNN)

Based the result of feedforward neural network, we realize that it failed to solve this problem for two main reasons. One factor that restricts fully connected network is its huge amount of parameters to train. For example, with an input dimension of 64x64, a 1000-node hidden layer will generate 4 million parameters, which is too enormous to converge. The second reason is from the nature of image. Most pixels in the image only have close relation with their neighbors, but much less and even none correlation with the non-neighbor ones. It means that most weight values trained in fully connected network are redundant.

Convolutional neural network is a better choice because it considers only a few neighbors around each pixel, which makes the training more focusing and efficient.

Our CNN implementation starts with architecture design, followed by hyper parameter tuning and data augmentation, with fine tuning at the end.

The final architecture can be seen in Figure 6, including four convolution blocks and one fully-connected neural network at the end. This architecture is a simplified version of the well-known VGGNet architecture.

It took quite a certain time of tuning until we get this final architecture work. The first version we built consists of only 2 convolutional layers, 2 max-pooling layers, and two dense layers. This simple CNN architecture reaches about 22% test accuracy, which, as expected, performs better than logistic regression and the neural network implemented from scratch. But this accuracy is obviously too low, comparing with most MNIST related problems where more than 90% could be achieved.

After comparing with several successful CNN implementations, we modified some critical hyper parameters that limited our CNN's performance, including the pool size of maxpooling layers which should have been set to 2x2 instead of 3x3. And our convolutional layers should have one zero padding to keep the size of its output and input the same. This modification produced a milestone accuracy at about 79%.

The next milestone tuning happened when we followed the architecture of VGGNet which uses three convolutional layers and one maxpooling layer as a layer group or a convolution block. In the modified model we used 3 such layer groups and increase the batch size from 128 to 256. This modification improve our prediction accuracy to 97%.

And the third remarkable improvement was from our using of data augmentation. Details of how image augmentation works can be found in Section Problem representation. This technique enabled our model to be trained on more data and effectively reduced overfitting. The test accuracy increased slightly to 98.5% after this modification.

Further improvement became harder. [3] mentioned a fine tuning method that have further improved their project performance from a well-trained VGGNet model. The idea is to freeze most of the well-trained layers and use smaller learning rate to keep training the layers that are close to the output layer. Following this method, the first three convolution blocks were frozen and we used SGD optimizer instead of Adam optimizer as what [3] suggests, with a learning rate that 10 times smaller than before. Please note that data augmentation was still used in fine tuning. This modification improved our test accuracy further to 98.7%.

It is worth mentioning that the training of our CNN model is impossible without using GPU to accelerate. Google Compute Engine (GCE) was set up for our project and successfully decreased training time by a factor of 100. Since this topic is not quite related to our CNN model, detailed discussion and setup can be seen in Appendix.

Testing and validation

This section presents detailed analysis results of the NN and CNN algorithm.

Neural Network

Validation was done on one tenth of the dataset. The accuracy on the validation set itself was not more than around 11 %, which must be due to the simplicity of the architecture, and the complexity of the problem. Validating on the 10000 example test set, would give a public score of 12.4%, which corresponds a lot to result achieved when validating on the validating part of the training set, but it still very low with respect to the results of the convolutional neural network.

CNN

Using the optimized architecture and hyper parameters, the training accuracy of CNN regarding each epoch can be seen in Figure 4. It is quite impressive that image augmentation successfully prevent overfitting. The validation accuracy remains higher than training accuracy in most of the epoch.

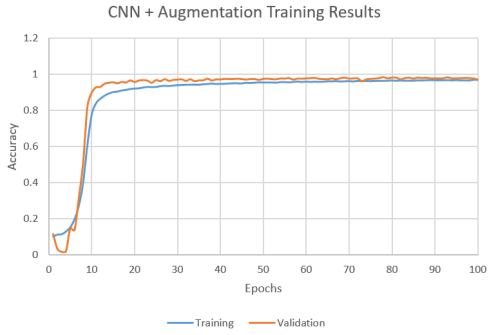

Figure 4: CNN training results with augmentation

Fine tuning results can be seen in Figure 5. Two trending line were plotted to avoid confusion. As mentioned in Seciton Algorithm selection and implementation, fine tuning was able to improve the test accuray from 98.5% to 98.7%.

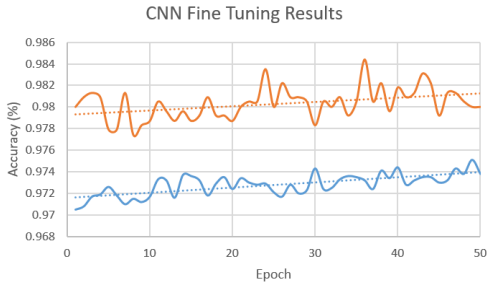

Figure 5: CNN fine tuning results

Discussion

When working with data in the format of pictures, each pixel is highly dependent on the neighboring pixels. This might explain both the logistic regression and the normal feed-forward neural network do not give the best predictions of the test set. The convolutional neural network enables different hidden units in the hidden layers to feed information to each other, i.e. now the network also knows when pixels are close together, or are sitting together in different ways that the network can learn. This means that this model in theory should be better at doing the problem presented in this assignment, which is also what is observed through testing. Writing digits and letters in pictures are different but always follow aproximately the same structure, which is why a good learner would have to know the link between the different pixels making up the symbol. The convolutional network can do this, because it links multiple units in the same layer. This must be why this algorithm works so much better than the simpler ones.

Statement of contributions

Chuan was in charge of Google Cloud Engine environment setup, convolutional neural network architecture design, validating, and data pre-processing. Tobias was in charge of implementing the feed-forward neural network from scratch and the corresponding testing and validation. Guanqing was in charge of image augmentation, CNN fine tuning, and implementation of logistic regression baseline. The work in the report is equally distributed between the 3 team members. We hereby state that all the work presented in this report is that of the authors.

References

[1] Y. LeCun, L. Bottou (1998). Gradient-based Learning Applied to Document Recognition. [online] Available at: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf [Accessed 13 Nov. 2017].

[2] Adit Deshpande, (2017). The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3). [online] Available at: https://adeshpande3.github.io/ade

shpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html. [Accessed 13 Nov. 2017].

[3] F. Chollet, (2016). Building powerful image classification models using very little data. [online] Available at: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html [Accessed 13 Nov. 2017].

[4] J. Pineau, (2017). COMP 551 slides from October 30th 2017.

Appendices

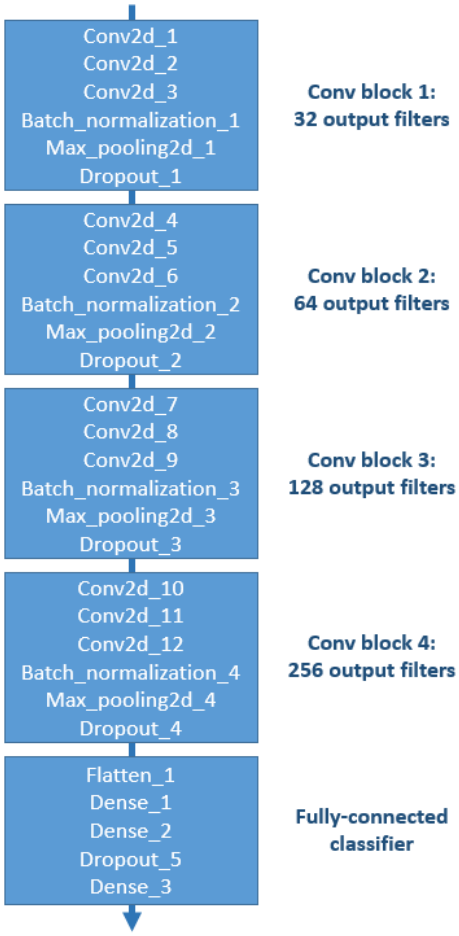

Figure 6: CNN architecture

Computing Engine Setup

GPU accelerated computing uses GPU with CPU together to accelerate the process of deep learning. The advantage of GPU is it has thousands of cores to process parallel workloads efficiently while CPU has only several cores (normally 4 to 8) optimized for sequential serial processing. So GPU works much better in handling multiple tasks simultaneously.

Google Cloud Compute Engine (GCE) has been chosen to provide GPU computing acceleration. GCE is part of Google cloud service that provides customized virtual machines (VMs). We setup several similar VMs for our project for efficient validation and testing. Each VM was initialized with Nvidia Tesla P100, 16 vCPUs, 64GB memory, and 256GB solid disk. Correct versions of GPU driver and libraries are also critical. Detailed setup steps can be found in our project folder.

Before we decided to use GCE, only CPU was used to train our first CNN model consisting of 4 convolutional layers and 2 fully-connected layers. It took around 400 seconds per epoch and thus several hours to finish a 50-epoch training. This high time cost made test and validation very difficult.

GCE with GPU, however, increases our computation speed by a factor of 100. The same task started to take only 4 seconds for each epoch and a 50-epoch training can be done in only several minutes. Fast and large amount of testing and validating become much more feasible.