Deep Learning Assignment 1

Haichao Wu
Center for Data Science
New York University
New York, NY 10011
hw1551@nyu.edu

Raochuan Fan
Center for Data Science
New York University
New York, NY 10011
rf1711@nyu.edu

Peimeng Sui
Center for Data Science
New York University
New York, NY 10011
ps3336@nyu.edu

1 Model Architecture and Configuration

The model architecture and configuration we used is a slightly modified version of convolutional network provided in the sample code. The input to our ConvNets is a fixed-size $28 * 28$ single channel images of hand-written numbers. The image is passed through a stack of two convolutional layers of $5 * 5$ kernels. The convolution stride is fixed to 1 pixel. Spatial pooling is carried out by five max-pooling layers, which follow the conv layers. Max-pooling is performed over a $2 * 2$ pixel window, with stride 1.

2 Techniques

2.1 Dropout

Dropout provides a way of approximately combining exponentially many different neural network, similar to ensembling methods. Applying dropout to a neural network can be visualized as sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout, with a probability p. A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all. At test time, we can use a single neural net without dropout, scaling the outgoing weights of unit by p at test time. By doing this scaling, 2^n networks with shared weights can be combined into a single neural network to be used at test time.

We set the possibility of dropout p = 0.5. In this case, when doing forward part of network, half of weight has been set to 0. The activation of hidden layer become sparse. As for mean activation, the weight is skewed, more close to zero for most weight, which prevent over-fitting for the model. For model without dropout, the weights spread more evenly.

2.2 Augmentation

We perform rotate, skew, affine, and randomcrop separately to the MNIST dataset. The reasons for doing this are avoid overfitting and try to mimic as many different styles of handwriting of digitals as possible.

When doing rotation, for each original image, we generate a random angle between $(-maxAngle, +maxAngle)$, where maxAngle is a predefined hyper-parameter. Then we rotate the

image by this random angle.

When doing skew, for each original image, we will generate a random number called distance. Then we will either skew the image to the right or the up by this distance. An example is in Figure 1.

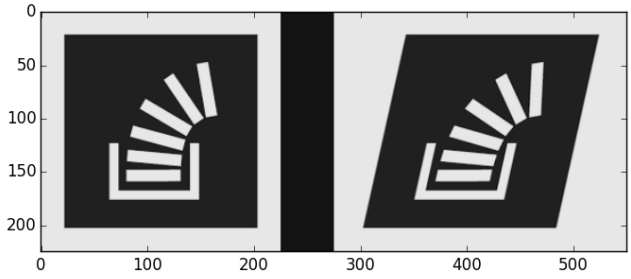

Figure 1: Example of Skew

When doing affine, for each original image, the idea is similar to skew, and an example is in Figure 2.

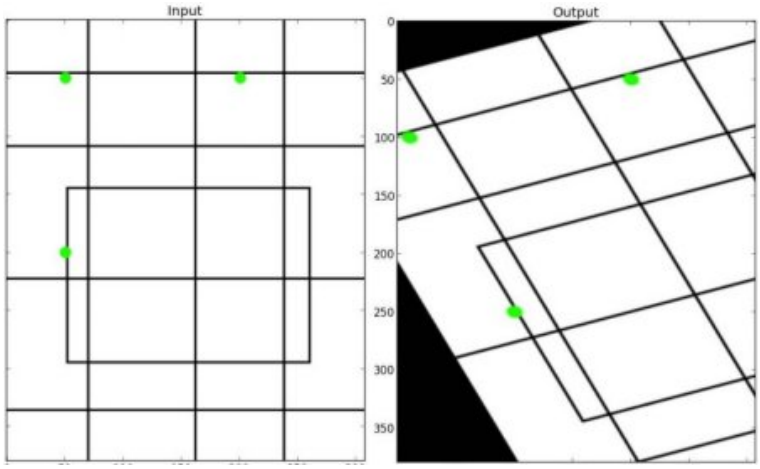

Figure 2: Example of Skew

For crop, we're using the default RandomCrop class of torchvision.transfrom, which crops the given PIL.Image at a random location. Then we rescale the image to the original size.

2.3 Semi-supervised Techniques - Pseudo Label

Pseudo label are target classes for unlabeled data as if they were true labels.

During each epoch for the unlabeled data, we first used the model from last epoch to provide them pseudo labels, which are recalculated every weights update. The overall loss function is

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L(y_i^m, f_i^m) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L(y'^m_i, f'^m_i) \tag{1}$$

Where n is the number of mini-batch in labeled data for SGD, n' for unlabeled data, f_i^m is the output units of m's sample in labeled data, y_i^m is the label of that, f'^m_i for unlabeled data, y'^m_i is the pseudo-label of that for unlabeled data, $\alpha(t)$ is a coefficient balancing them.

The entire training process is divided into three phases: training only labeled data, training only unlabeled data and training all data. For these three phases, $\alpha(t)$ increasing, and it is 0 for the first phase.

2.4 Some other Techniques

Some other techniques we also tried (See the model training code for implementation details):

1. Dropout and DropConnect
2. Momentum method to accelerate the gradient descent step
3. Three different initialization: Kaiming, Xavier and pre-training Kmeans

3 Experiment

In this section we will introduce how we trained our model and show our results for the Kaggle Competition.

3.1 Training With Labeled Data

As mentioned before, we used four kinds of data augmentation separately. An example is in Figure 3. Data Augmentation proved to improve our model performance a lot. We can see this clearly from Figure 4, comparing training and validation errors with and without using augmented data.

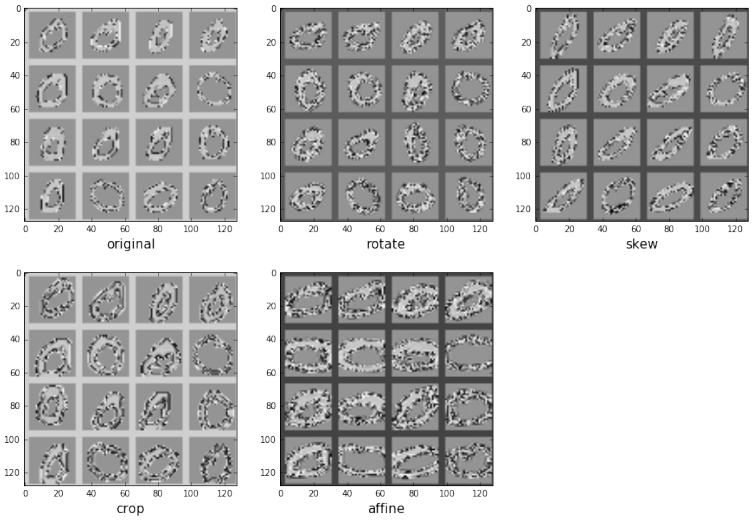

Figure 3: Example of Skew

The test accuracy is about 98.70% when implement augmentation, while it is about 97.20% without augmentation for 150 epochs.

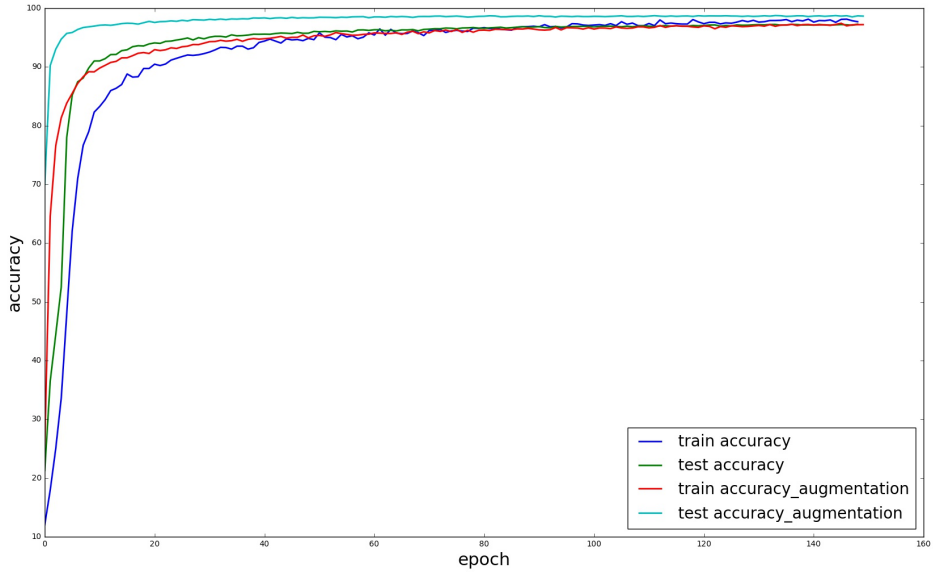

Figure 4: Augmentation Effect

3.2 Training with All Data

We use pseudo-label techniques described in part 2.3.

1. We trained 20 epochs used only the 3000 labeled images plus 12000 augmented images transformed from the labeled data. This phase is to stabilize our classifier to predict a relatively reliable label for unlabeled data. We used SGD optimizer with learning 0.01.
2. The second phase begin with getting involved the unlabeled data. During each epoch for the unlabeled data, we first used the model from last epoch to provide them pseudo labels, which are recalculated every weights update.

   In this phase, α in equation (1) is growing from 0 to 3.1 during the training process, as we become more and more confident on the pseudo labels. Another trick we found useful to improve the final model accuracy is let the training epoch loop more on the labeled data to serve as a self correction. In our final training process, during every epoch of looping through all the unlabeled data, we loop through the labeled data for 7 times. This phase lasts for 20 epochs. We used SGD optimizer with learning 0.01.
3. During the final phase, we did exactly the same as in phase 2, except that we fixed α to be 3.1. This is the best value we get from the fine tuning. By monitoring the validation error, the entire process converged after 100 epochs, as you can see from Figure.5.

From Figure 5, we compared three models together, one using only labeled data without augmentation, one using only labeled data with augmentation, and the third one using all labeled and unlabeled data. The test accuracy is above 99.30% for 150 epochs.

To submit our final Kaggle submission, we repeated the training process for ten times and ensemble ten models to get the final prediction. All of our trained models can be downloaded through this link: https://drive.google.com/drive/folders/0B8EoX4shjz25M2lzaTFVRHhCaEE?usp=sharing. One can download it with his NYU Id login.

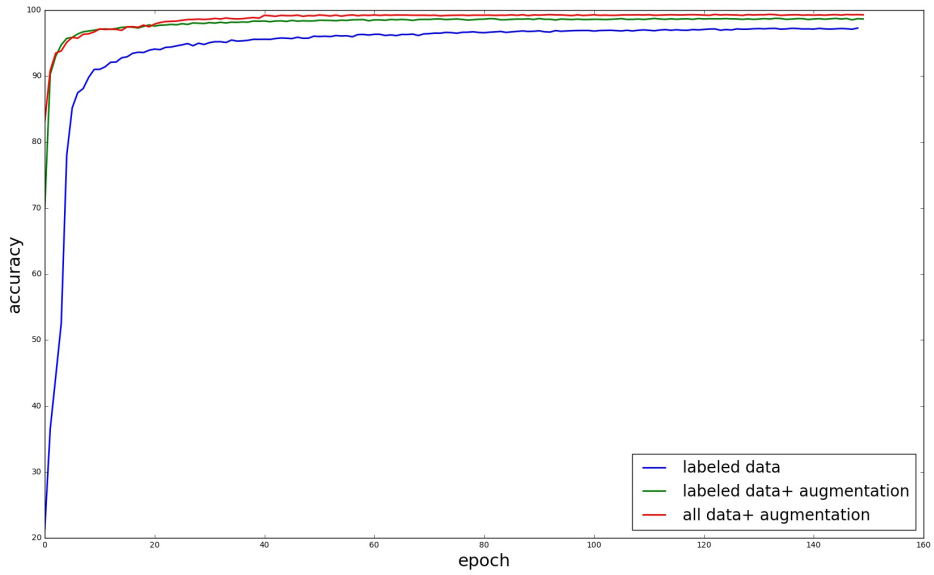

Figure 5: Augmentation Effect