Deep Learning and Practice
Lab 3: CNN on Cifar-10

李韡
0556157
fm.bigballon@gmail.com

Introduction

As we know, traditional convolution neural network looks like 3x3 conv + ReLU, but in this Lab, we will try to build NIN (Network in Network) network architecture.
(NIN: 3x3 conv + ReLU + 1x1 conv + ReLU + 1x1 conv +ReLU)

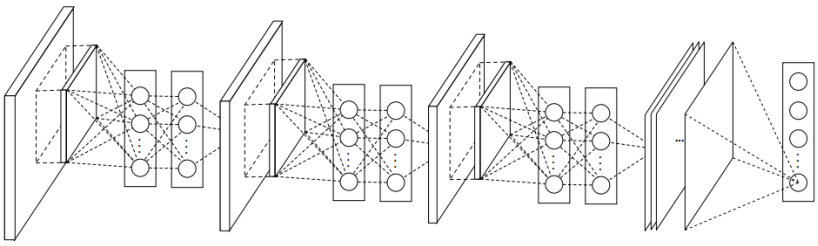

Figure 1: The overall structure of Network In Network

We will train it on Cifar-10 dataset, the CIFAR-10 dataset consists of 60000 32×32 color images (RGB) in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

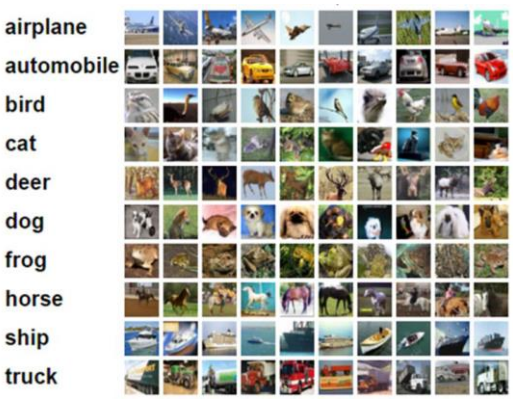

Figure 2: The CIFAR-10 dataset

I using the **Keras** (A higher-level API based on Tensorflow) to implement the model. And I just use the following combination to train the model. **(Assume Data preprocessing called DP, Weight Decay called WD, Weight initialization called WI, Nesterov momentum called NM, Data augmentation called DA)**

- ***Model 1.*** **NIN + DA + DP + WD + WI + NM + Dropout0.5**
- ***Model 2.*** ***Model 1*** **without data augmentation**
- ***Model 3.*** ***Model 1*** **without data preprocessing**
- ***Model 4.*** ***Model 1*** **without data preprocessing & data augmentation**
- ***Model 5.*** **New All convolution model**

Experiment setup

Model 1: NIN + DA + DP + WD + WI + NM + Dropout0.5

Architecture Details: **ConvPool-CNN** is the same as TA's architecture. And according to STRIVING FOR SIMPLICITY: THE ALL CONVOLUTIONAL NET, I changed the **ConvPool-CNN** model to the **All-CNN** architecture for ***Model 5***.

Table 1: All-CNN and ConvPool-CNN

ConvPool-CNN	**New All-CNN**
5x5 conv. 192 pad = 2 ReLU	3x3 conv. 96 ReLU
1x1 conv. 160 ReLU 1x1 conv. 96 ReLU	3x3 conv. 96 ReLU
3x3 max pooling, stride = 2	3x3 conv. 96 ReLU with stride = 2
5x5 conv. 192 pad = 2 ReLU	3x3 conv. 192 ReLU
1x1 conv. 192 ReLU 1x1 conv. 192 ReLU	3x3 conv. 192 ReLU
3x3 max pooling, stride = 2	3x3 conv. 192 ReLU with stride = 2
3x3 conv. 192 ReLU 1x1 conv. 192 ReLU 1x1 conv. 10 ReLU Global averaging Softmax	

- **Training hyperparameters:**
  - Method: SGD with Nesterov momentum momentum=0.9, nesterov=True
  - Data preprocessing: Color normalization
  - Weight initialization: stddev = 0.01 (for first conv layer) 0.05 (for others)
  - batch size: 128 (391 iterations for each epoch)
  - Total epochs: 164
  - Loss function: cross-entropy
  - Initial learning rate: 0.1, divide by 10 at 81, 122 epoch
  - Weight Decay: kernel_regularizer=keras.regularizers.l2(0.0001)
  - Dropout: 0.5

- **Data augmentation:**
  - Translation: Pad **4** zeros in each side and random cropping back to 32x32 size
  - Horizontal flipping with probability 0.5

Model 2: use ***Model 1*** without Data augmentation
Model 3: use ***Model 1*** without Color normalization
Model 4: use ***Model 1*** without Data augmentation & Color normalization

Model 5: All convolution for ***Model 1*** and ***Model 3***

Result

we comparison the ***Model 1*** **and** ***Model 2*** **(*Model 1* without** data augmentation**)**

■ **Final Accuracy (test error)**

Model 1	*Model 1* without **data augmentation**
90.49%(9.51%)	**89.14%**(10.86%)

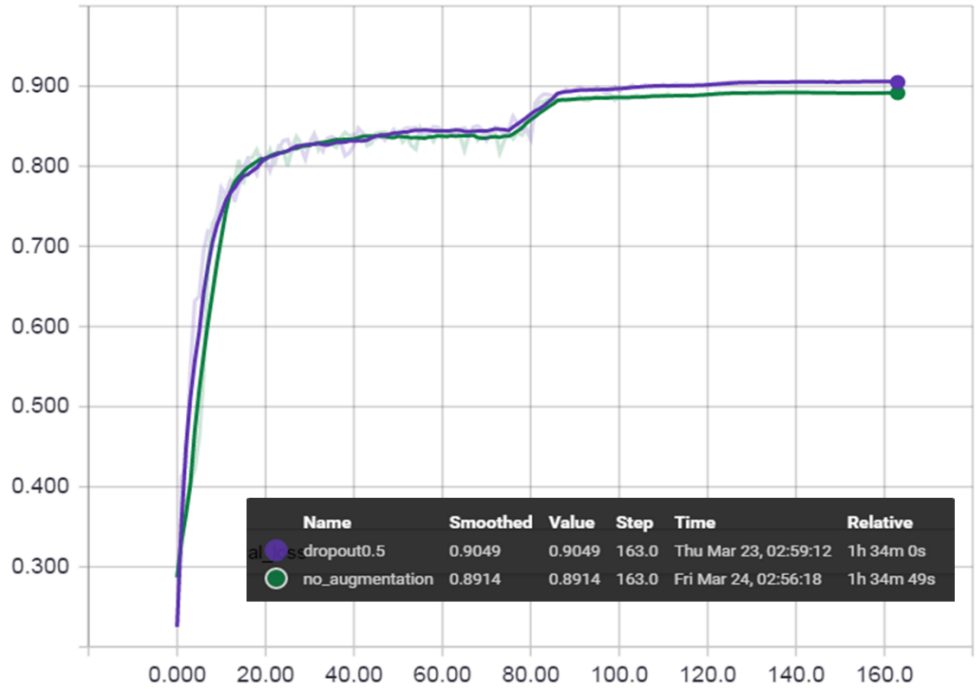

Figure 3: Test accuracy curve for *Model 1* & *Model 1* without data augmentation

- **Training loss curve:**

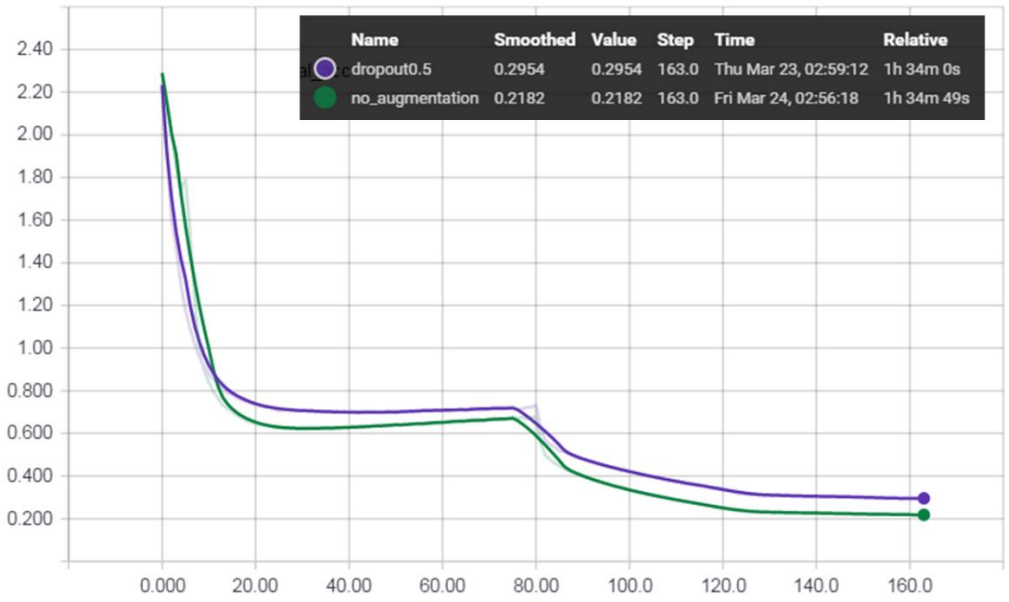

Figure 4: Training loss curve for *Model 1* & *Model 1* without data augmentation

- **Test loss curve**

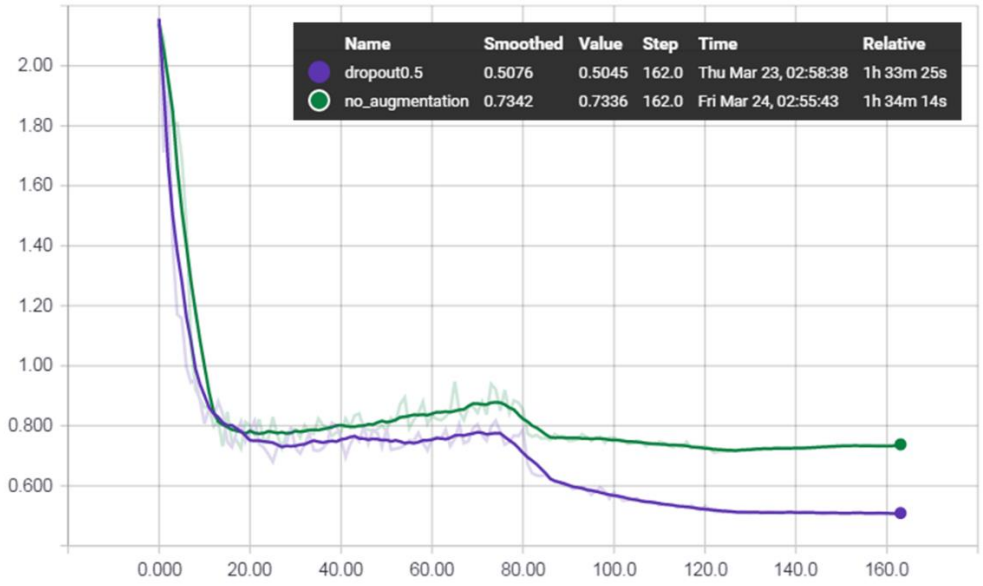

Figure 5: Test loss curve for *Model 1* & *Model 1* without data augmentation

We got the **final accuracy 90.49%** with Data augmentation and **89.14%** without Data augmentation. It is obvious that Data augmentation is important for training. According to the **Training loss curve** and **Test error curve**, we can learn that Data augmentation can help reduce the manual intervention required to developed meaningful information and insight of business data, as well as significantly enhance data quality.

Other experiments & Discussion

■ Color normalization Test

I trained other three model based on ***Model 1***, and we can see that the **Color normalization** trick provides about 4~5% improvement, it's cool.

Table 2: *Model 1*'s DP & DA test

Without data preprocessing:
Without data augmentation: **84.30%**
With data augmentation: **86.93%**
With data preprocessing:
Without data augmentation: **89.14%**
With data augmentation: **90.49%**

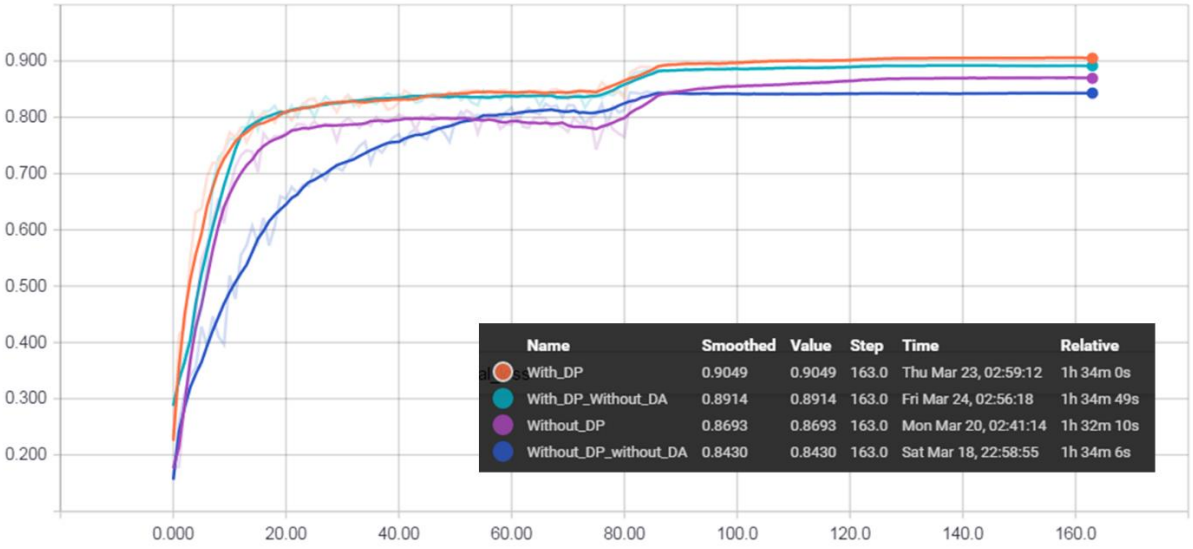

Figure 6: The accuracy for *Model 1*'s DP & DA test

■ SGD with/without momentum Test

We can see that SGD with momentum (86.93%) is better than without momentum (86.25%)

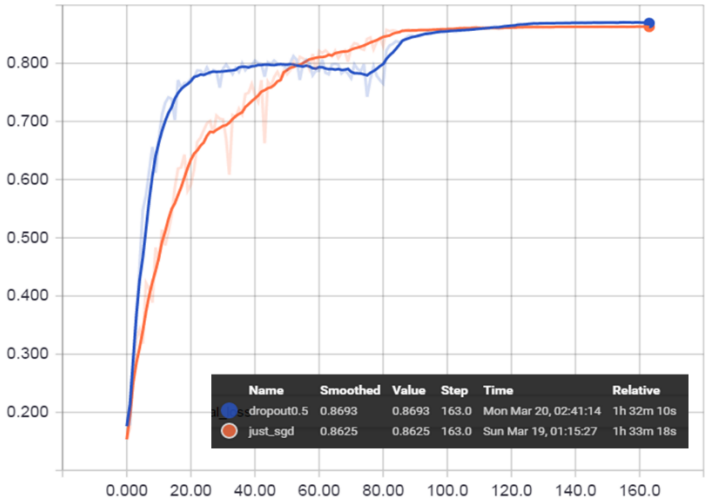

Figure 7: The accuracy for *Model 3* with/without momentum

■ **The effect of Dropout rate**

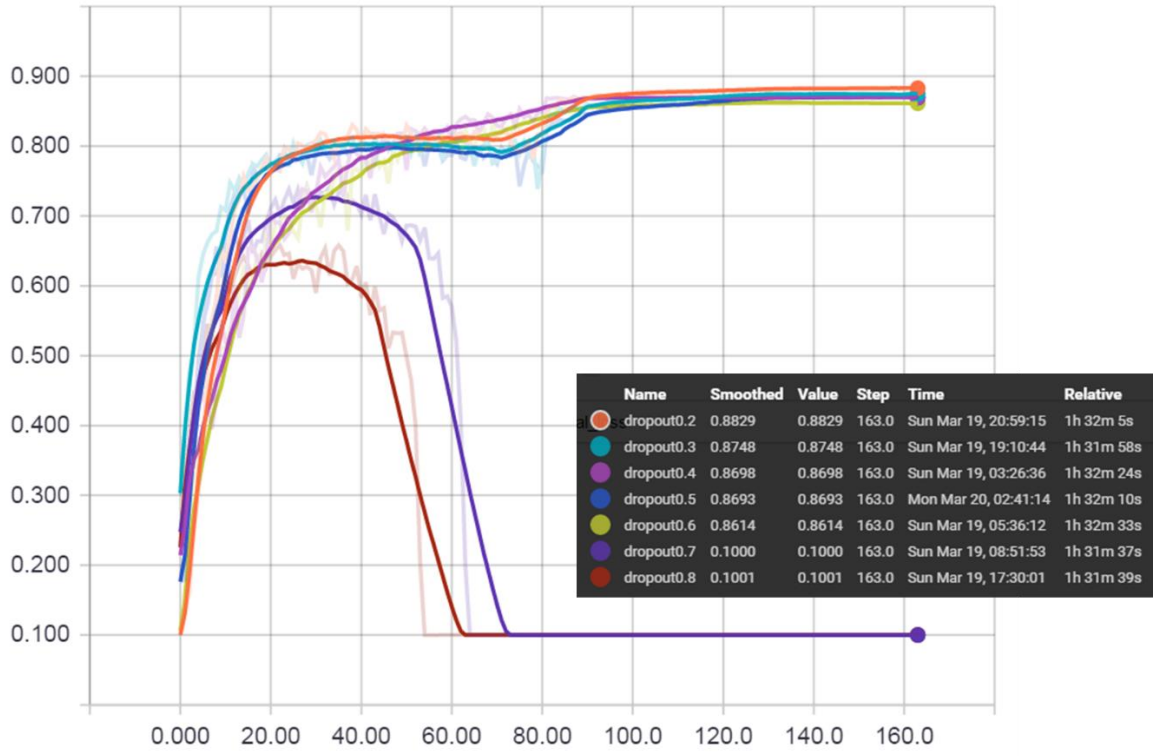

Figure 8: The accuracy for *Model 3* by using different dropouts

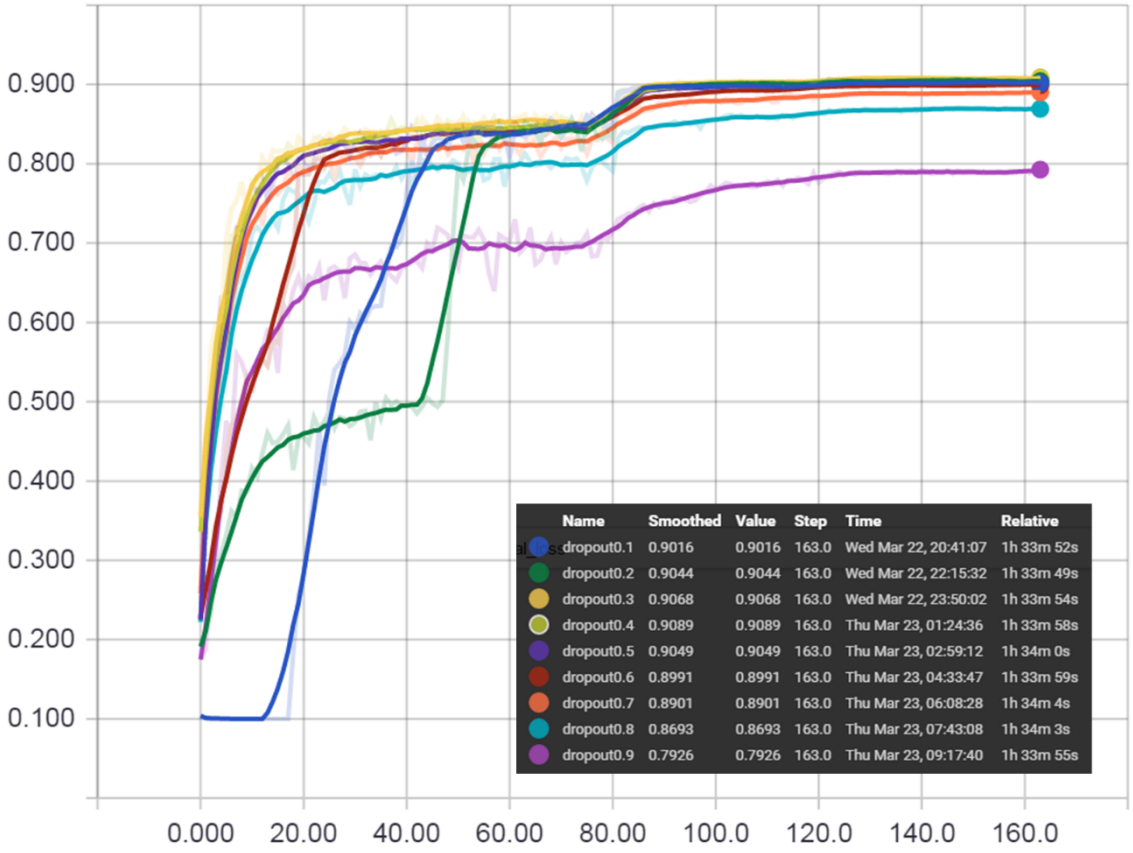

Figure 9: The accuracy for *Model 1* by using different dropouts

Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. From the Figure 5 and Figure 6, we can realize the value of dropout between 0.3 and 0.6 have a good performance.

However, if the value is too large, the model may become overfitting, in *Model 3*, if dropout's value is more than 0.7, training will be failed.

■ The All-CNN

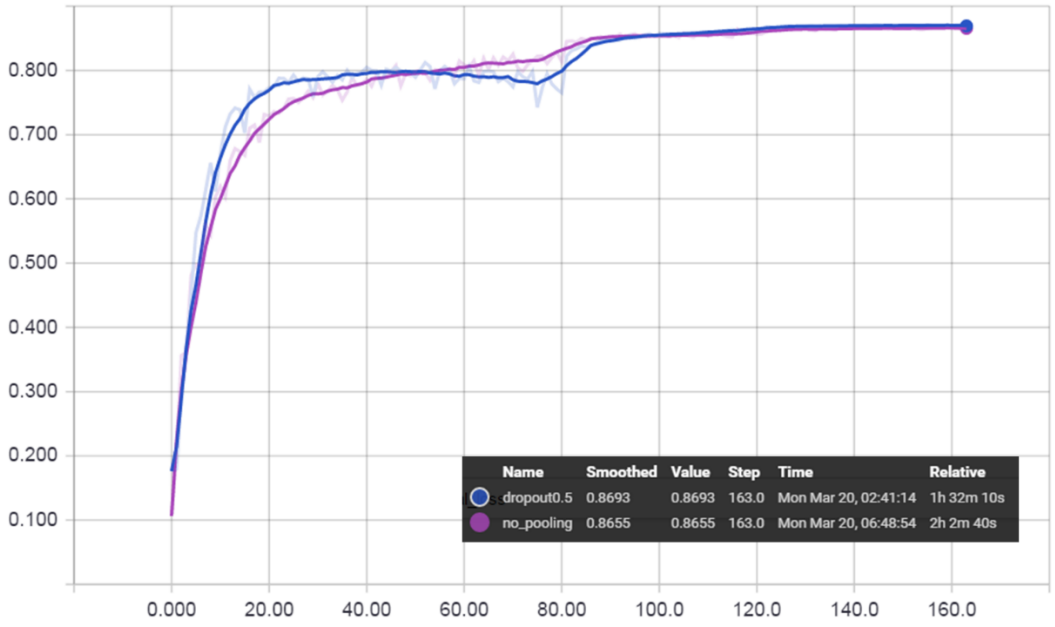

Figure 10: The accuracy for *Model 3* by using All-CNN

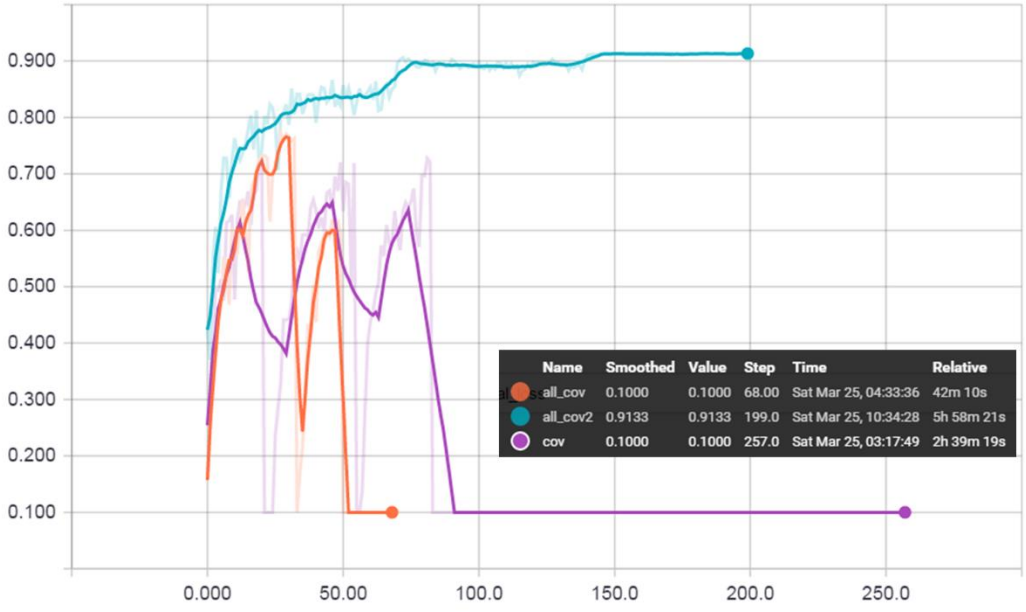

Figure 11: The accuracy for *Model 1* by using All-CNN

I used the **All-CNN** model which I mentioned earlier. And I got lots of trouble at the time of training, sometimes the model's accuracy will go up, sometimes will go down, finally, the accuracy will become 0.1 and training will be failed (just like orange line). I changed the initial learning rate from 0.1 to 0.05, but training still failed. Then, I used batch normalization to make it train fast, after 200 epochs' training (about 6 hours), I get the blue line. The accuracy of this model is **91.33%.**
While in ***Model 3***(***Model 1*** without Color normalization), it seems that **New All-CNN's** performance is no better than **ConvPool-CNN**. I'm not sure, maybe there are some bug in my code?

All in all, it's fantastic to learn deep learning and it's very interesting. I learned from the assignment that each parameter in CNN is important, how to design the architecture, how to initialize the weight, the learning rate and which activation to choose, all this problem is important, I still have a long way to go!