ASSIGNMENT 1
Object Classification with TensorFlow!

March 2, 2019

Name: Xinjia Yu
Andrew ID: xinjiay

Contents

0.1 Fashion MNIST classification in TensorFlow

Q 0.1: Both scripts use the same neural network model, how many trainable parameters does each layer have?

For conv2d1 layer, the number of parameters should be ((5*5 + 1) * 32 = 832). For conv2d2, the number of parameters should be ((5*5 * 32 + 1)*64 = 51264). For dense1, the number of parameters should be ((7 * 7 * 64) * 1024 + 1024 = 3212288). For dense2, the number of parameters should be (1024 * 10 + 10 = 11264).

Q 0.2: Show the loss and accuracy curves for both scripts with the default hyperparameters.

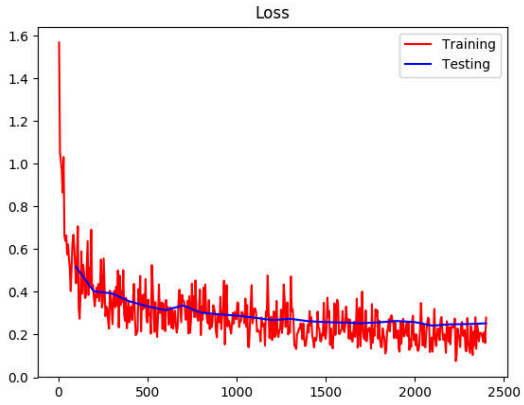


Figure 1: Loss for static graph

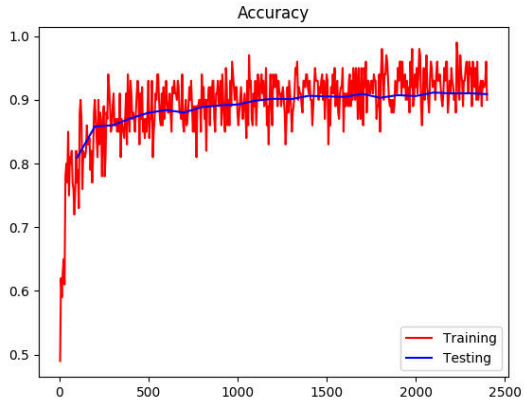


Figure 2: Accuracy for static graph

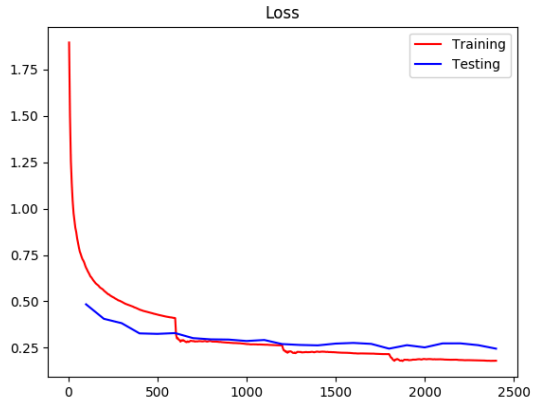


Figure 3: Loss for dynamic graph

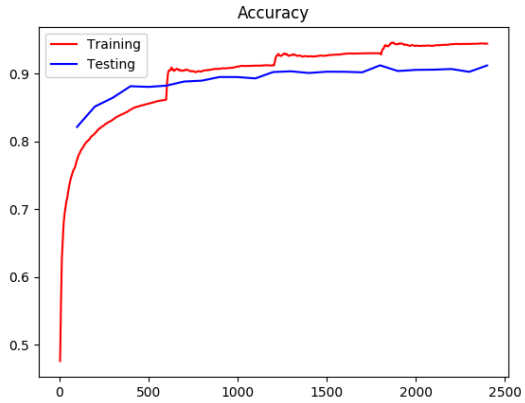


Figure 4: Accuracy for dynamic graph

Q 0.3: WHY DO THE PLOTS FROM TWO SCRIPTS LOOK DIFFERENT? WHY DOES THE SECOND SCRIPT SHOW SMOOTHER LOSS? WHY ARE THERE THREE JUMPS IN THE TRAINING CURVES?

The dynamic one is much more smoother than the static one. One possible reason that the dynamic graph could provide a smooth update is because the dynamic graph could achieve real-time update response towards the data while the static graph could only update the graph when the data has gone through the whole graph. When finished one epoch of training set, the training would repeat to feed the old data into the net and update the network. Since the net is more 'familiar' with the old training data, the performance could be improved comparing with the performance for the last epoch. That is why the jump number is 3 for 4 training epochs.

Q 0.4: WHAT HAPPENS IF YOU TRAIN THE NETWORK FOR 10 EPOCHS?

It is clear that, after 1000 iteration, there is a huge gap between the training loss and testing loss, which indicates that overfitting could happen after 1000 iterations.

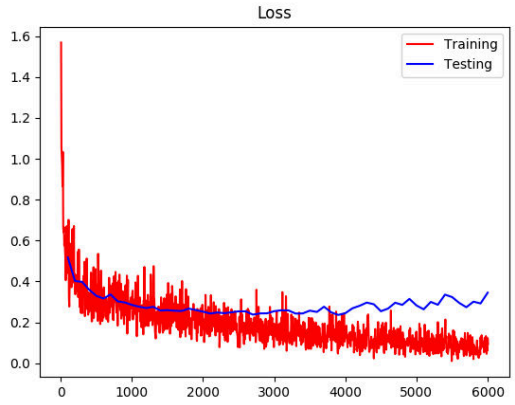


Figure 5: Loss for static graph

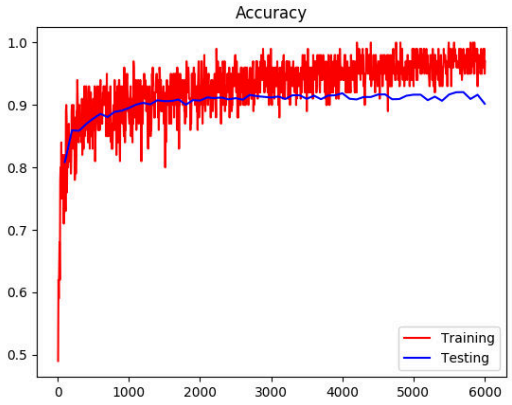


Figure 6: Accuracy for static graph

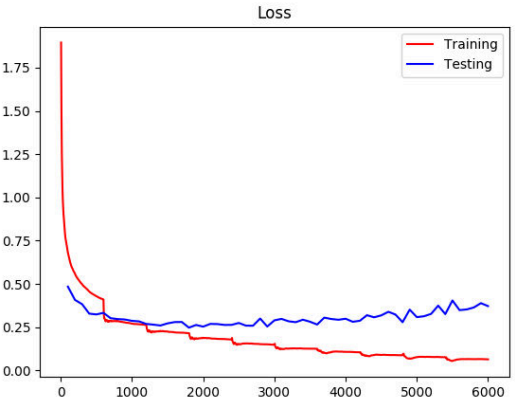


Figure 7: Loss for dynamic graph

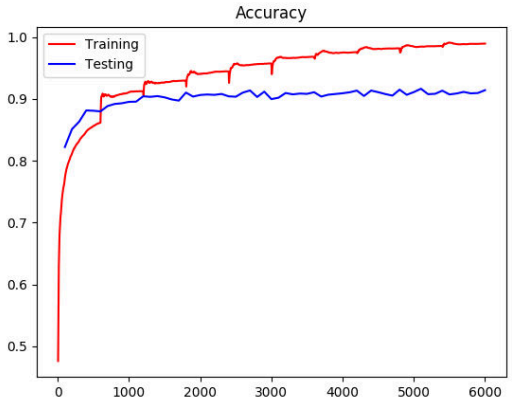


Figure 8: Accuracy for dynamic graph

0.2 Simple CNN network for PASCAL multi-label classification

Q 1.1 Show clear screenshots of the learning curves of testing MAP and training loss for 5 epochs (batch size=20, learning rate=0.001). Please evaluate your model to calculate the MAP on the testing dataset every 50 iterations.

The training loss, testing accuracy, gradient distribution and gradients histograms are plotted below.

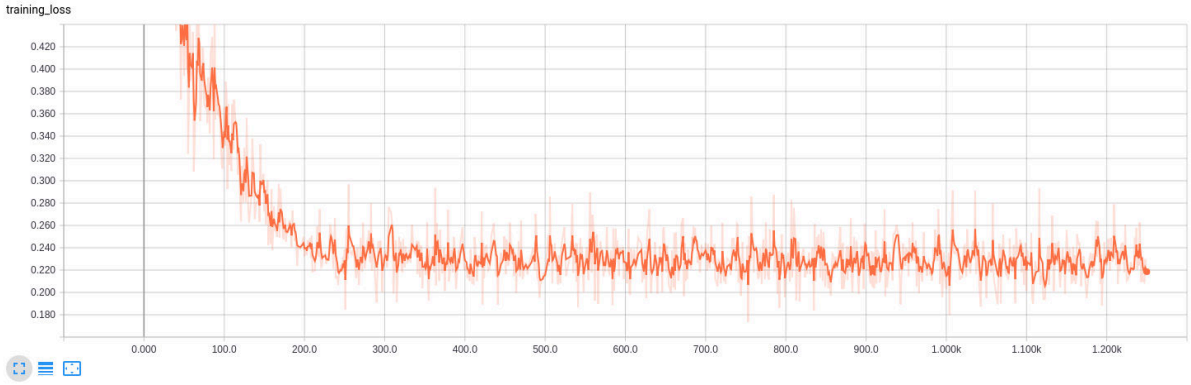


Figure 9: Loss for SimpleCNN

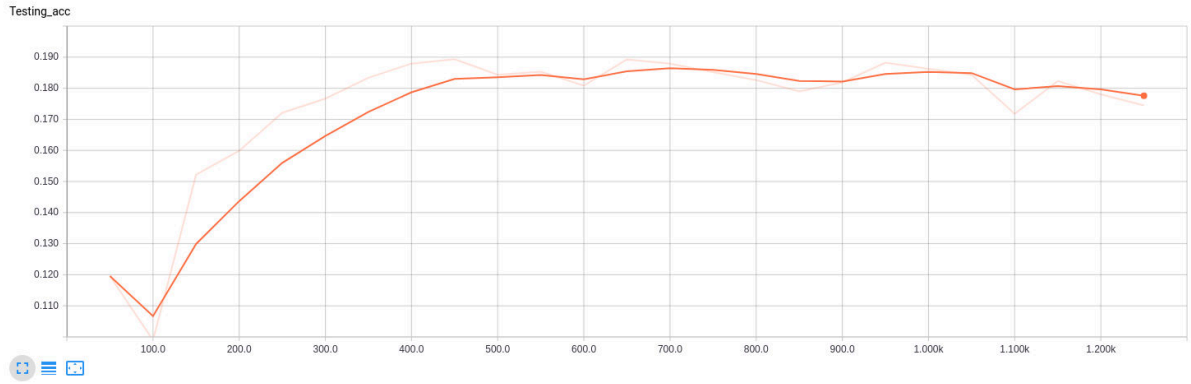


Figure 10: Accuracy for SimpleCNN

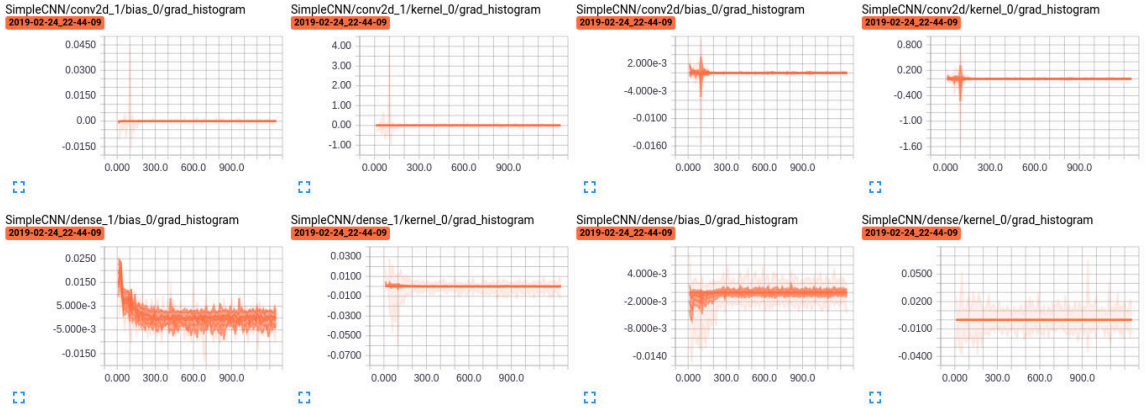


Figure 11: Distribution for SimpleCNN

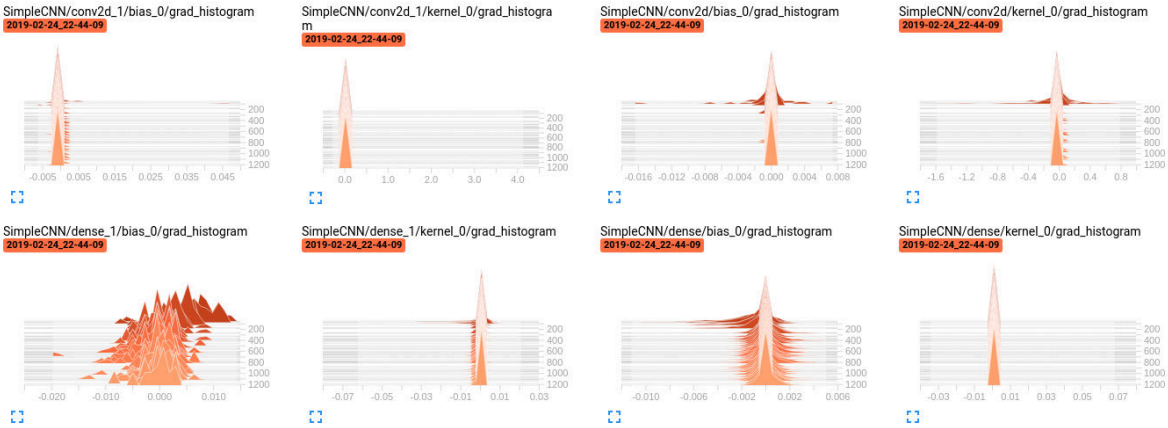


Figure 12: Histograms for SimpleCNN

0.3 CaffeNet for PASCAL classification

Q 2.1 Show clear screenshots of testing map and training loss for 60 epochs. Please evaluate your model to calculate the map on the testing dataset every 250 iterations.

The training loss, testing accuracy, gradient distribution and gradients histograms are plotted below. The hyper-parameters settings were set just as the question description. Training images were augmented by randomly cropping and testing data were augmeted by central cropping.

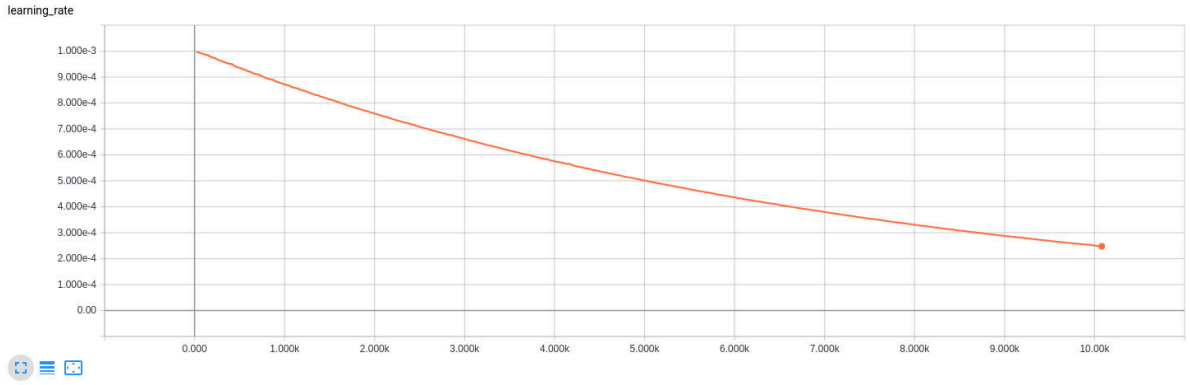


Figure 13: Learning rate for Caffenet

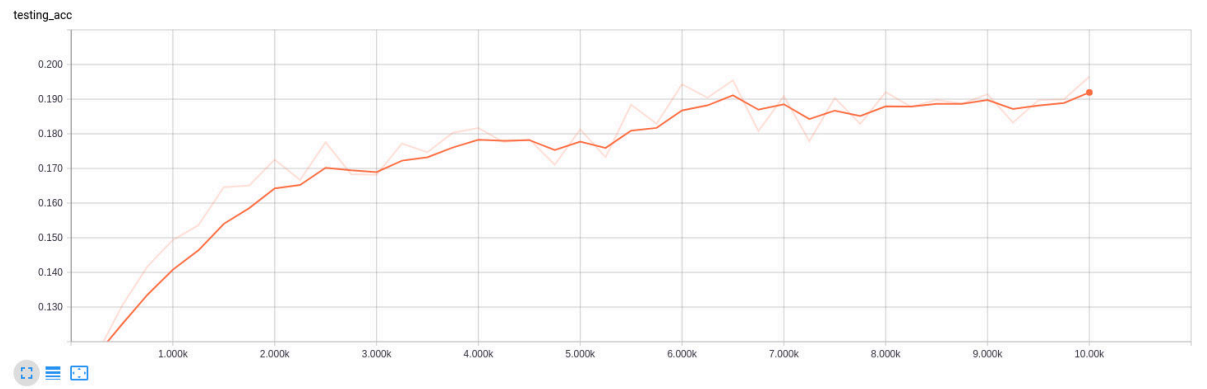


Figure 14: Accuracy for Caffenet

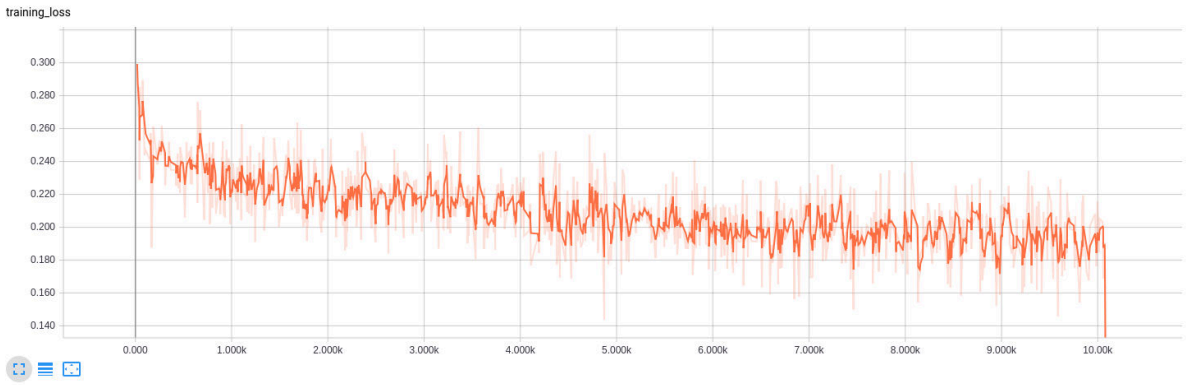


Figure 15: Loss for Caffenet

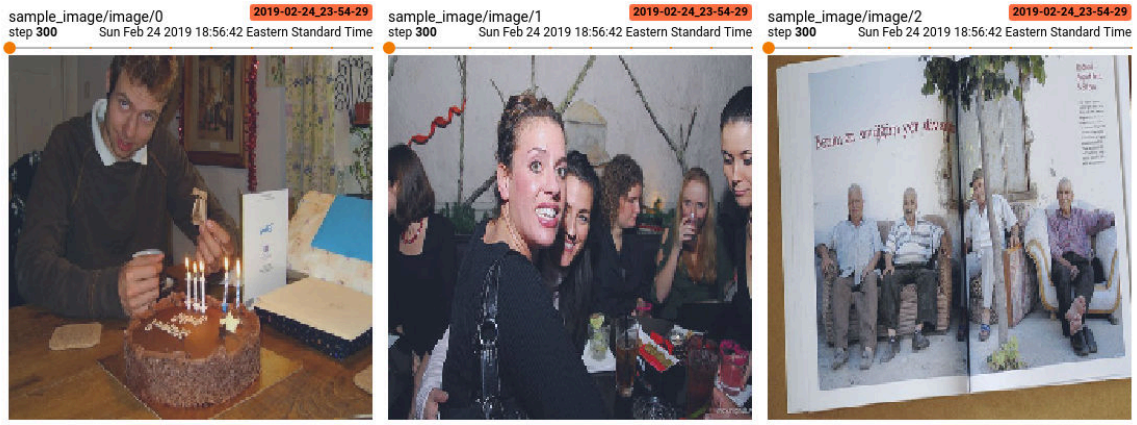


Figure 16: Images for 300 iteration

Figure 17: Images for 3700 iteration

Figure 18: Images for 10000 iteration

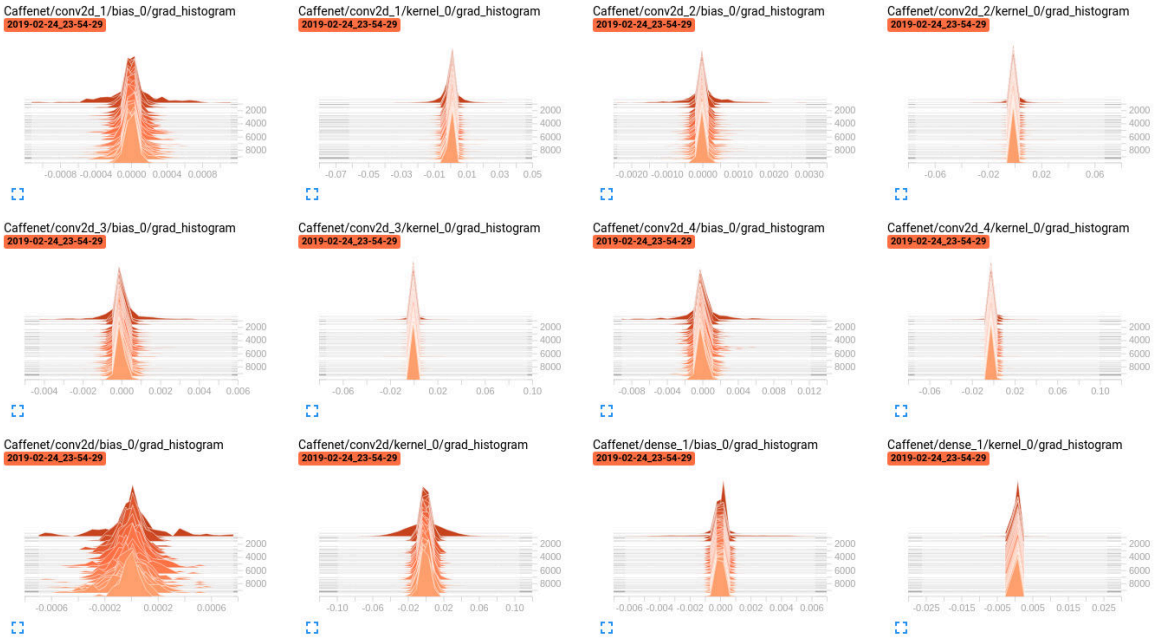


Figure 19: Gradients histogram for Caffenet

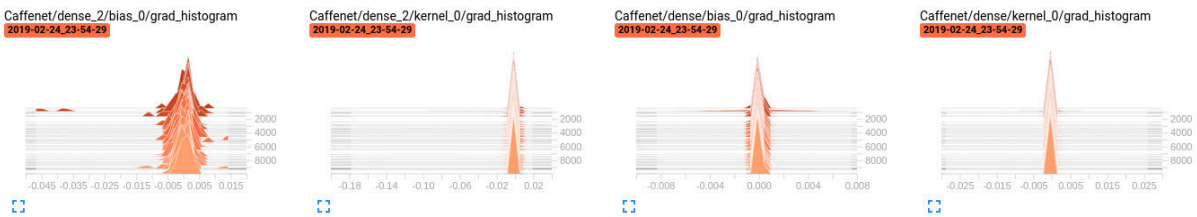


Figure 20: Gradients histogram for Caffenet

Figure 21: Gradients distribution for Caffenet

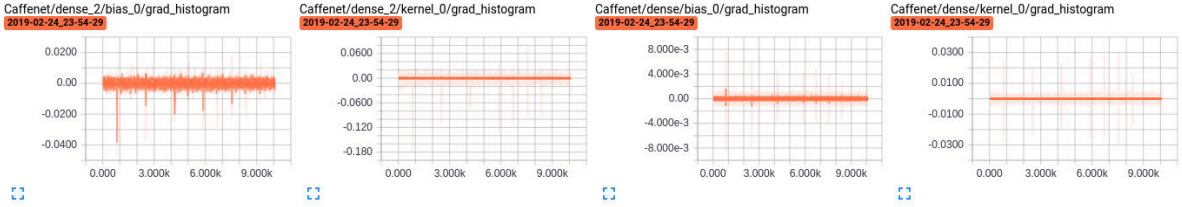


Figure 22: Gradients distribution for Caffenet

0.4 VGG-16 for PASCAL classification

Q 3.1 Add screenshots of training loss, testing map curves, learning rate, histograms of gradients and examples of training images from tensorboard.

The training loss, testing accuracy, training images, gradient distribution and gradients histograms are plotted below. The batch size is 20 and the number of total epochs is 20.

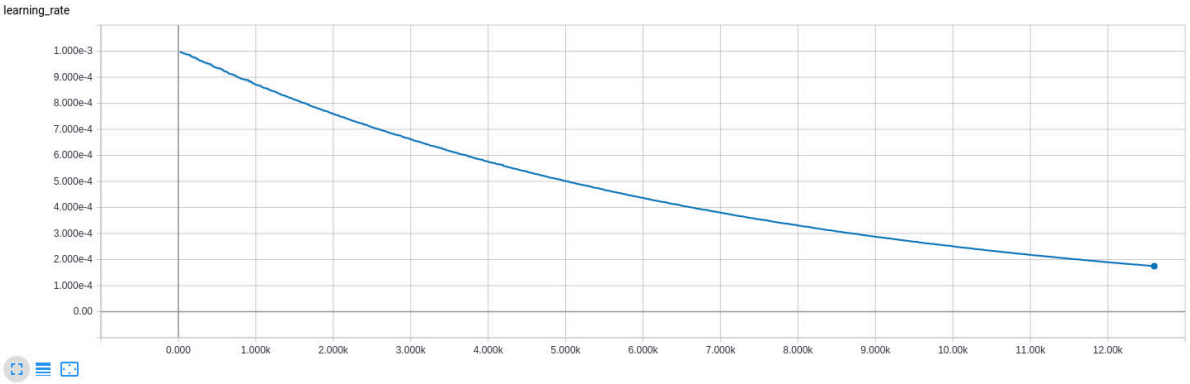


Figure 23: Learning rate for VGG

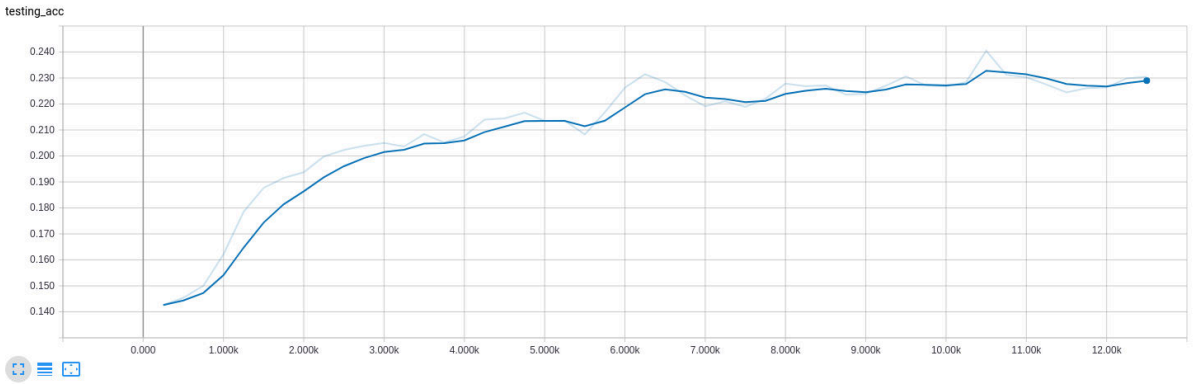


Figure 24: Test accuracy for VGG

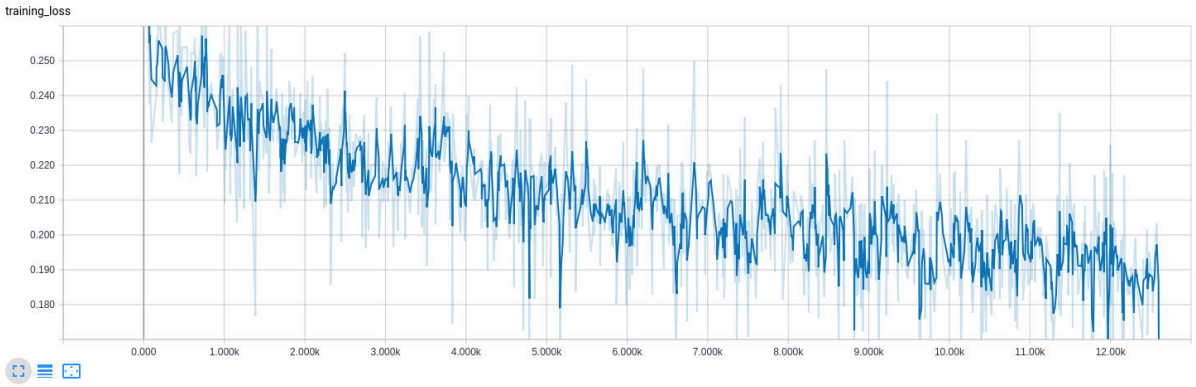


Figure 25: Loss for VGG

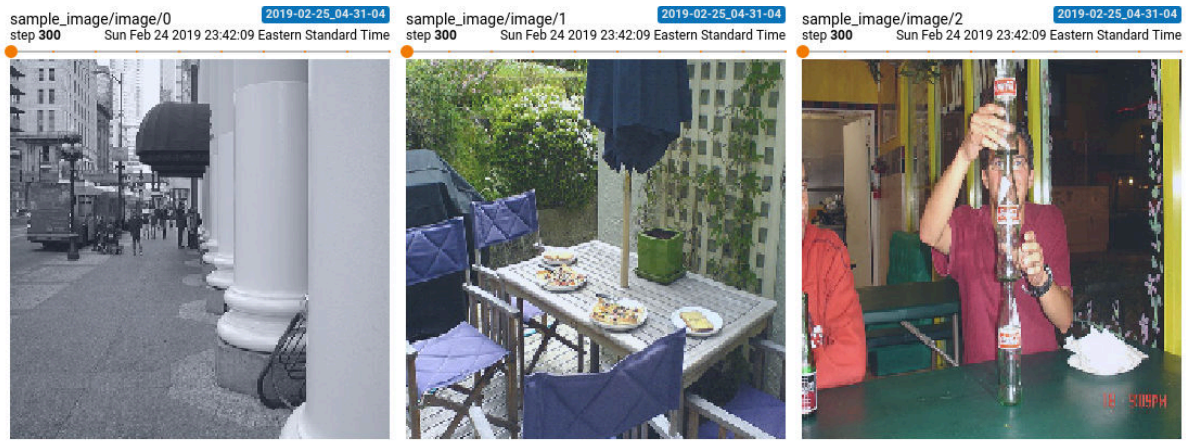


Figure 26: Images for 300 iteration

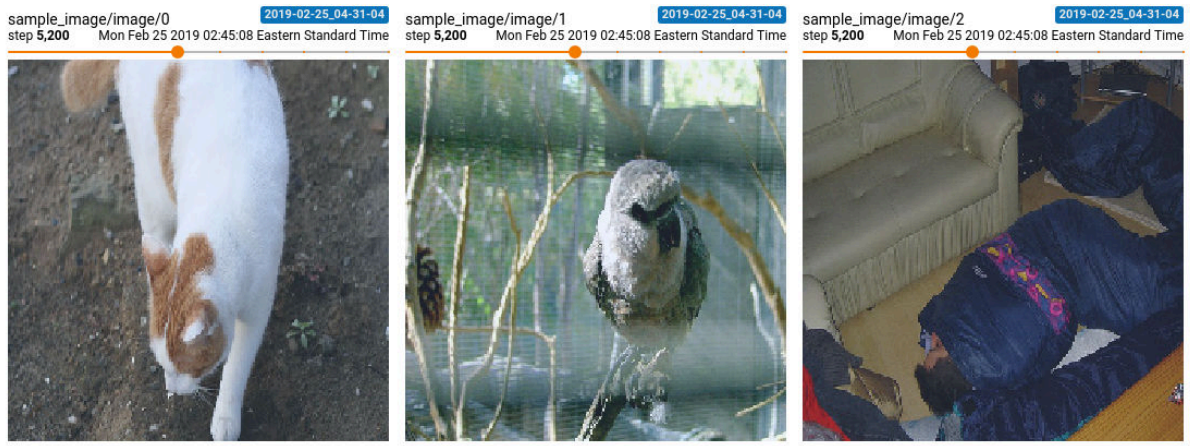


Figure 27: Images for 5200 iteration

Figure 28: Images for 1250 iteration

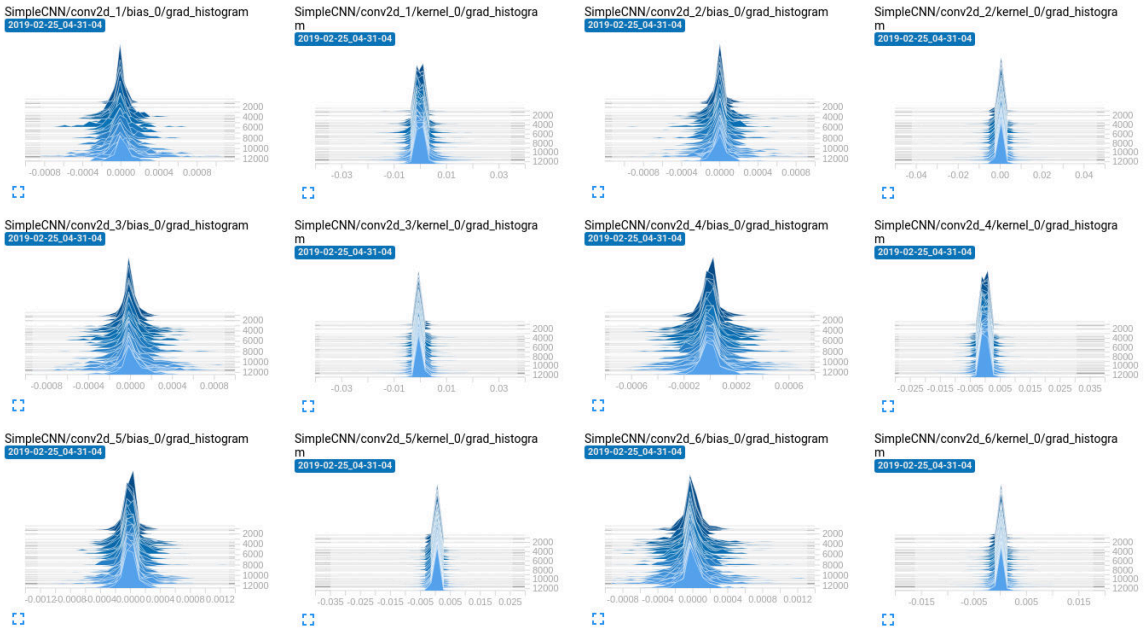


Figure 29: Histogram for VGG

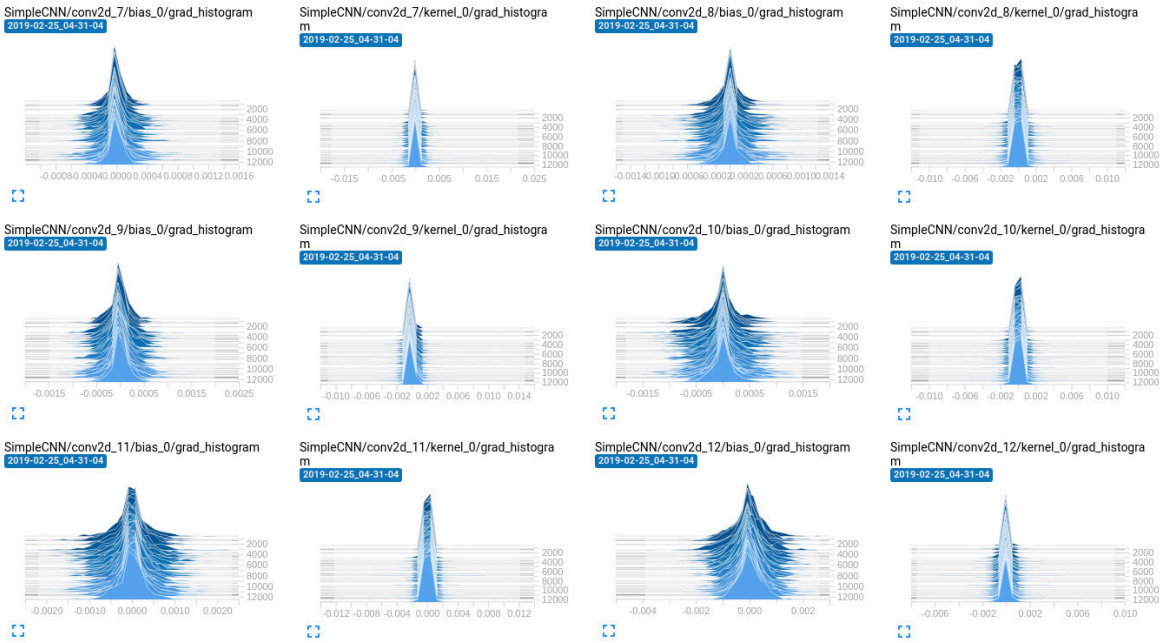


Figure 30: Histogram for VGG

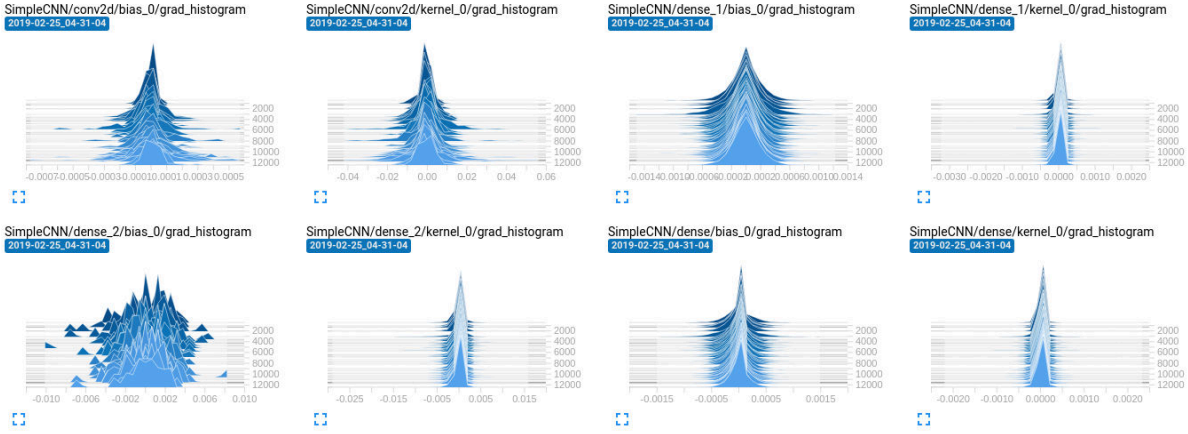


Figure 31: Histogram for VGG

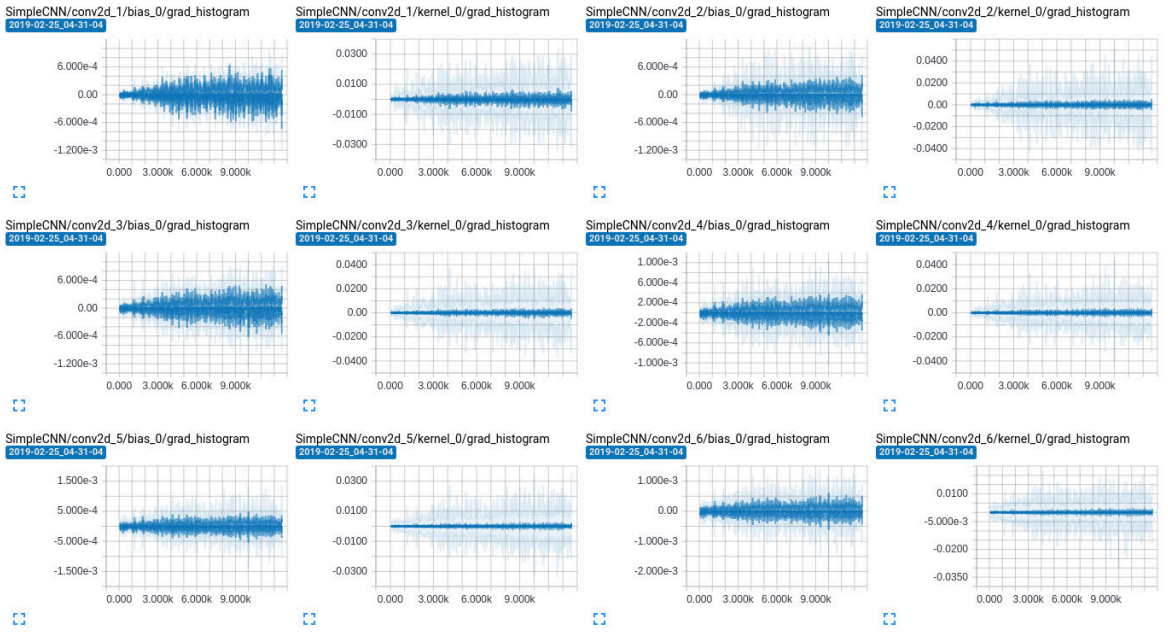


Figure 32: Distribution for VGG

Figure 33: Distribution for VGG

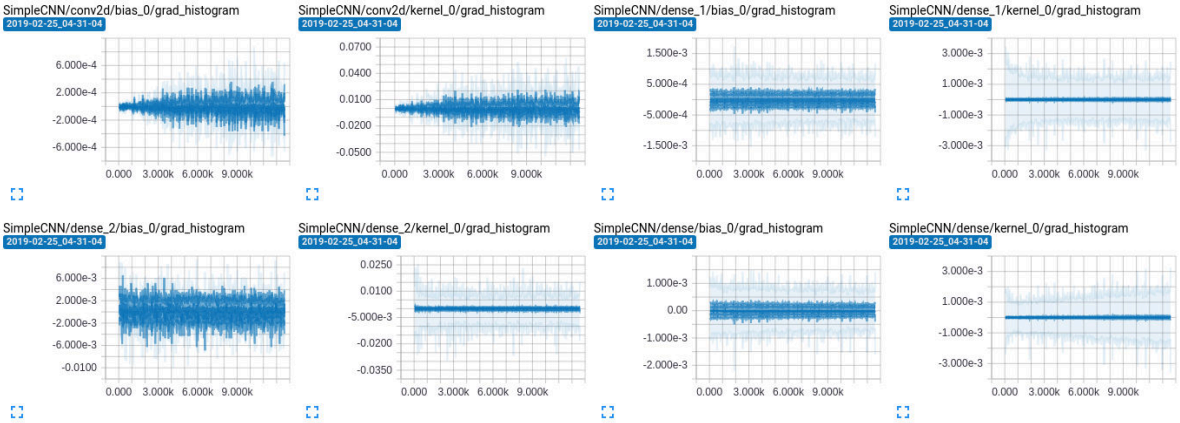


Figure 34: Distribution for VGG

0.5 Finetuning from ImageNet

Q4.1: use similar hyperparameter setup as in the scratch case, however, let the learning rate start from 0.0001, and decay by 0.5 every 1K iterations. Show the learning curves (training loss, testing MAP) for 10 epochs. Please evaluate your model to calculate the MAP on the testing dataset every 60 iterations.

Following the instruction, I loaded the pre-trained weights up to fc7 layer and initialized the fc8 weights trained from my scratch VGG network. It is obvious that the fine-tuned weight could significantly improves the performance of the VGG network.

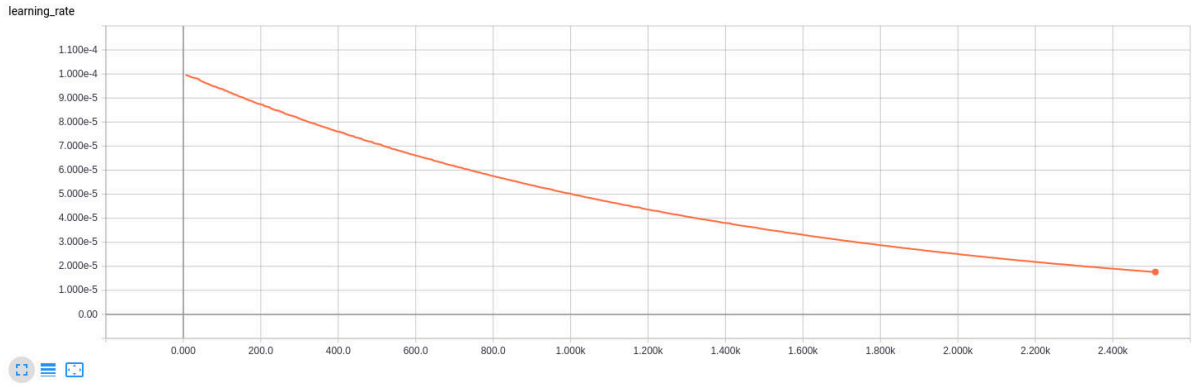


Figure 35: Learning rate for VGG

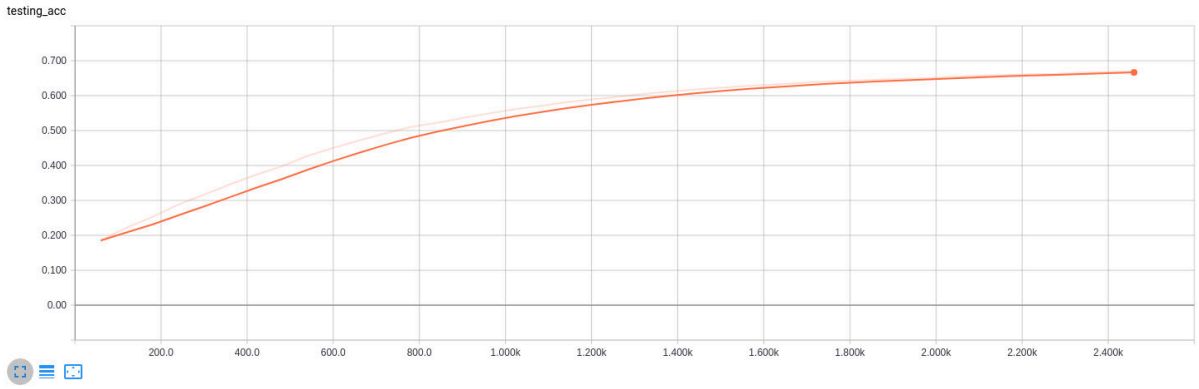


Figure 36: Accuracy for VGG

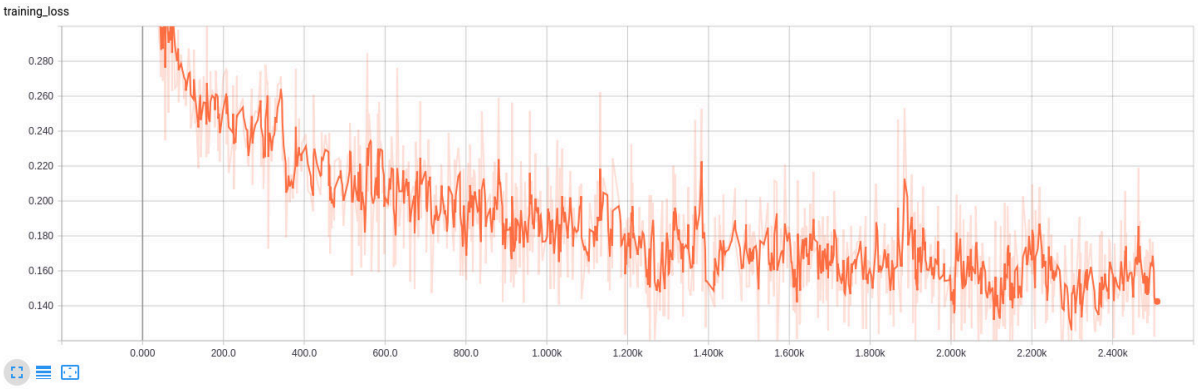


Figure 37: Loss for VGG

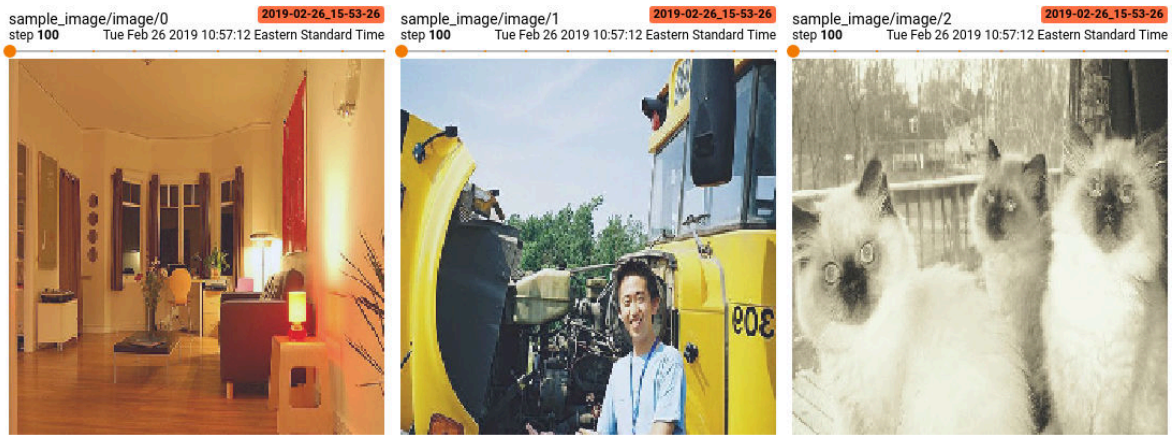


Figure 38: Image for 100 iteration

Figure 39: Image for 1100 iteration

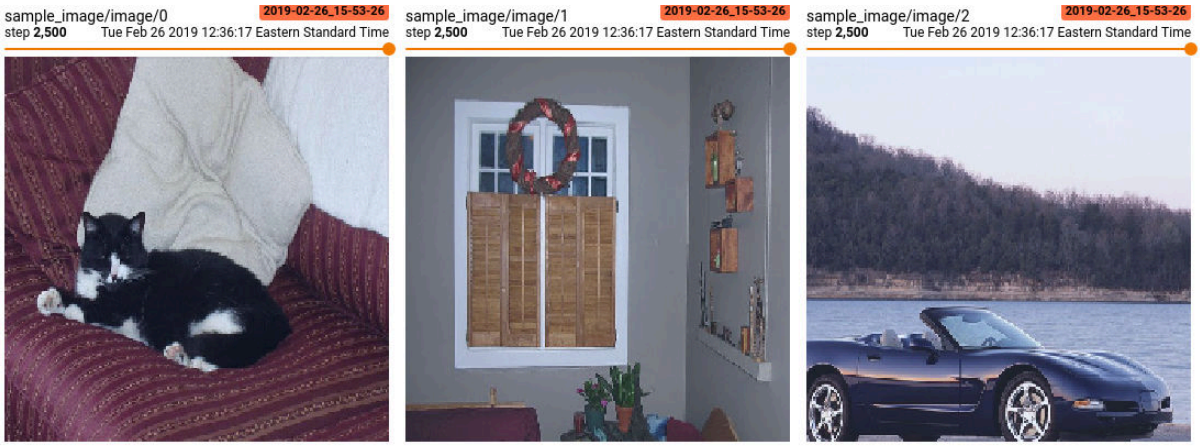


Figure 40: Image for 2500 iteration

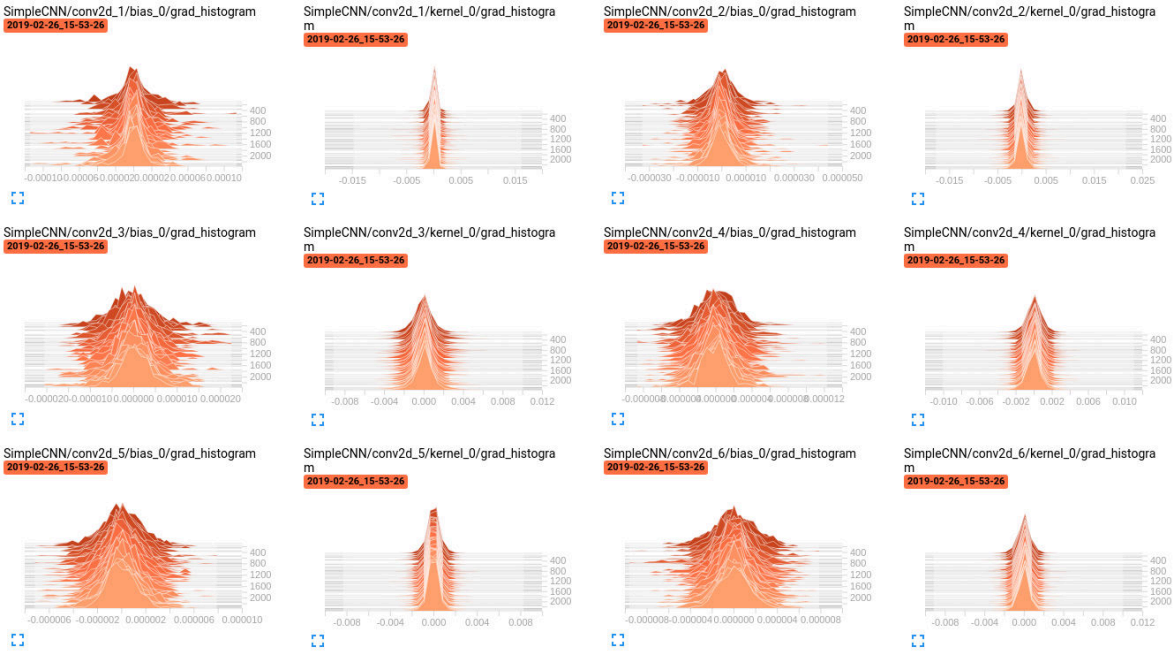


Figure 41: Histogram for VGG pretrained

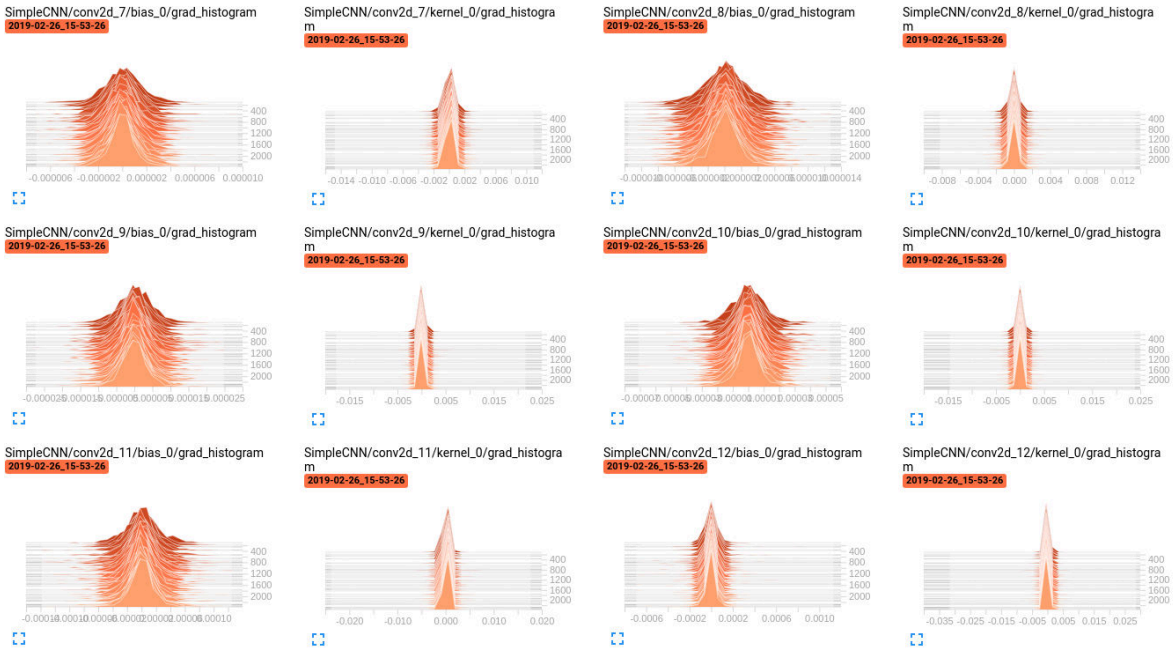


Figure 42: Histogram for VGG pretrained

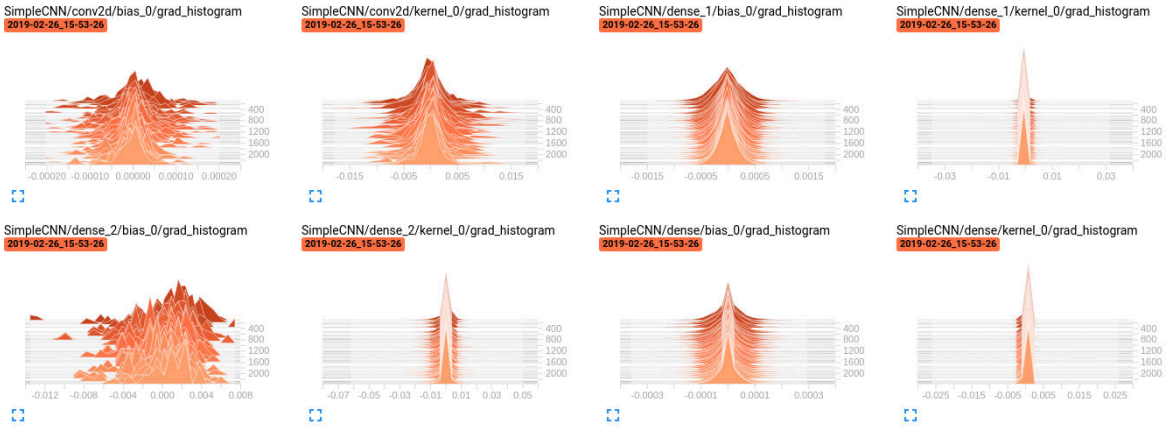


Figure 43: Histogram for VGG pretrained

Figure 44: Distribution for VGG pretrained

Figure 45: Distribution for VGG pretrained

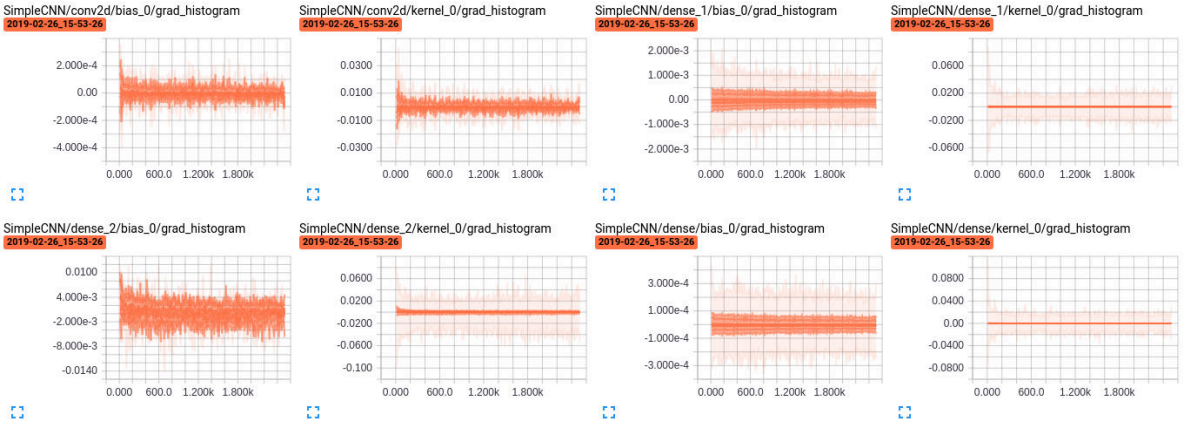


Figure 46: Distribution for VGG pretrained

0.6 Analysis

Q5.1: CONV-1 FILTERS

As it can be seen from the Fig.46, at the beginning stage of the training the filters were just totally random distribution. In the end of the training (e.g. 58th epochs), the filter seems to eventually converge to have some features as an edge detector.

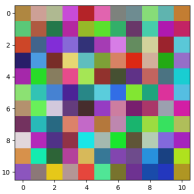
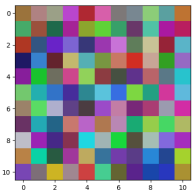
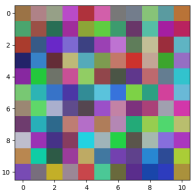
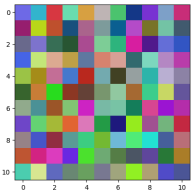
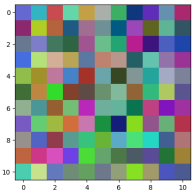
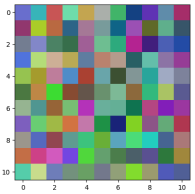
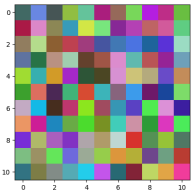
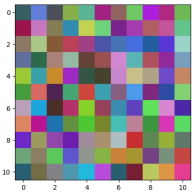
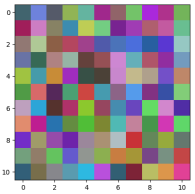
	0 epoch	30 epoch	58 epoch
Filter No.18			
Filter No.39			
Filter No.48			

Figure 47: CONV-1 FILTERS

Q5.2: NEAREST NEIGHBORS

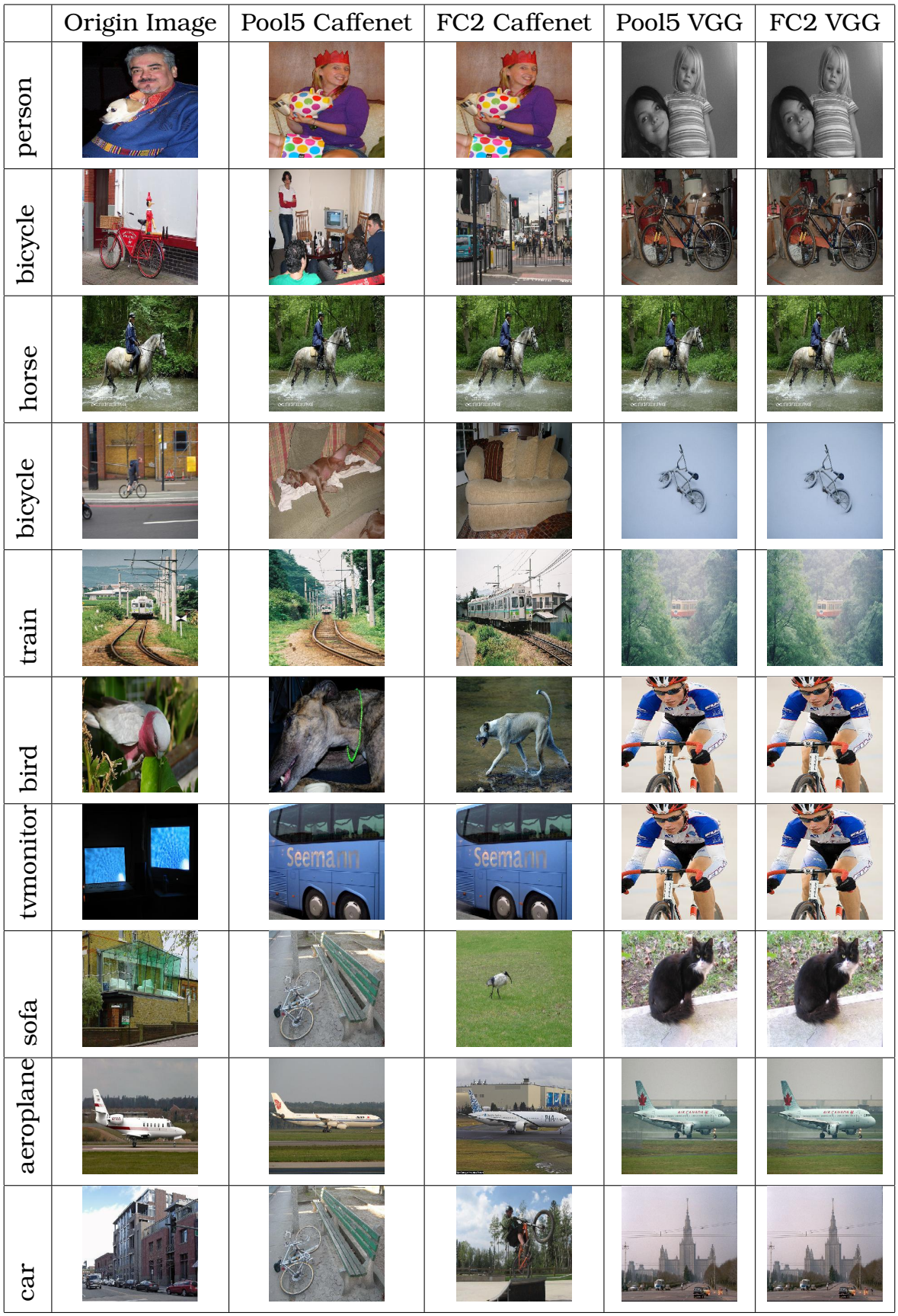


Figure 48: Nearest Neightbours

Q5.3: T-SNE VISUALIZATION OF INTERMEDIATE FEATURES

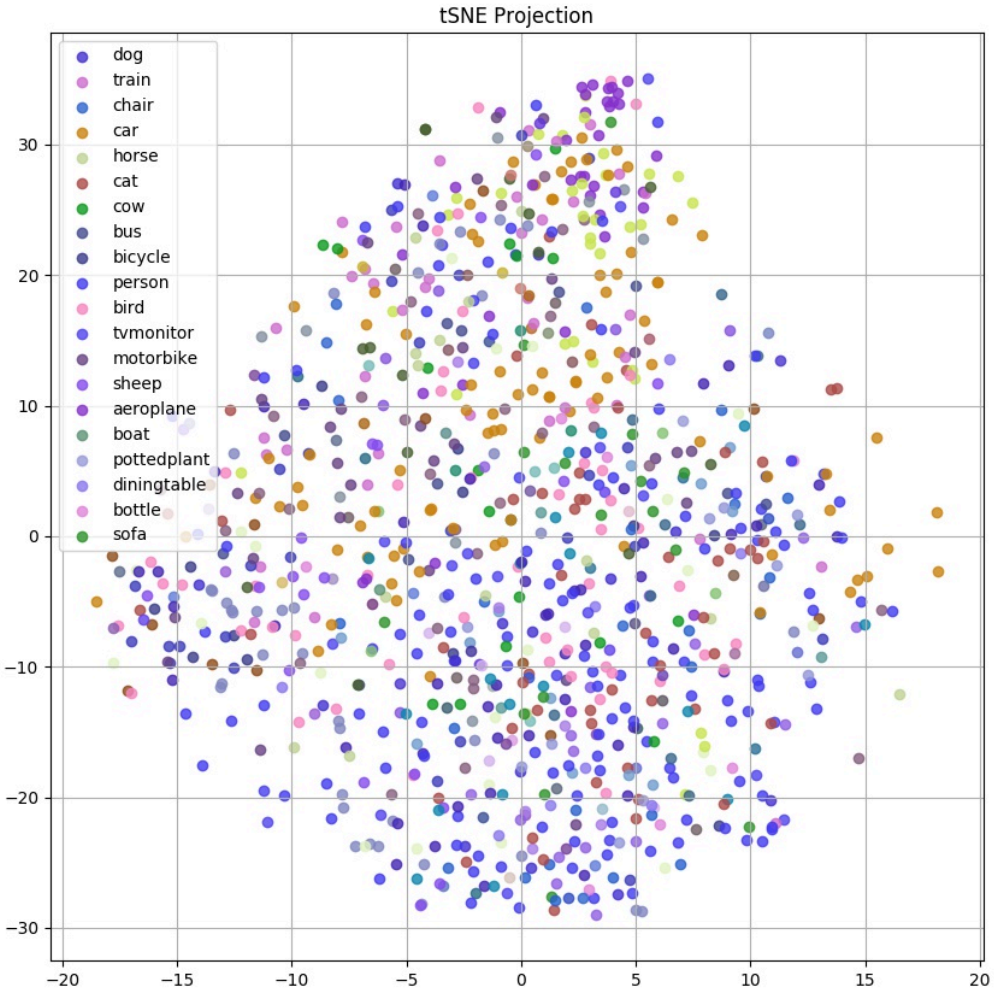


Figure 49: tSNE Plot

Q5.4: ARE SOME CLASSES HARDER?

	VGG (scratch)	VGG (fine tuned)
aeroplane	0.22208	0.72446
bicycle	0.24374	0.65485
bird	0.22670	0.68252
boat	0.23575	0.74635
bottle	0.19827	0.70997
bus	0.27387	0.61518
car	0.27646	0.63646
cat	0.27364	0.72001
chair	0.25000	0.63920
cow	0.22755	0.62207
dining table	0.21844	0.72092
dog	0.27500	0.69964
horse	0.24905	0.60439
motorbike	0.23983	0.67064
person	0.20138	0.59465
potted plant	0.28778	0.68584
sheep	0.22539	0.63477
sofa	0.24753	0.59944
train	0.20174	0.69628
tv monitor	0.26026	0.70030

Table 1: VGG (scratch) vs VGG (fine tuned)

As it can be observed from the table, I found that cow, sheep and horse are categories which were hard to be predicted. One possible reason is that these three kinds of images always appeared in the background with huge noise. And light condition varies a lot because those pictures are always taken outdoor. Another reason is that animals in pictures tends to have various posture (sit, lie and stand), which also increases the difficulty of predictions. On the contrast, boat, airplane and train images are always taken from the similar shooting angle and their shape are fixed (rigid body). Therefore, they tend to be easier to be predicted comparing with other categories.

Objects like dining table and bottle are improved a lot by using the pre-trained model. It is perhaps because that with the better network more detailed shape could be extracted

from the images and thus the performance could be improved a lot. For dining tables and bottles, they are usually have the complicated shape which could be difficult for the scratch model with weak filters.

0.7 Improve the classification performance

I tried to implement the mix-up augmentation technique to improve the performance of the VGG network. Following by the hyper-parameter settings given in the article, I have set alpha to be 0.2 to randomly mix up the images. But unfortunately, the mix-up technique could only reduce the performance of the network. I thought It could be the problem of alpha so I have taken one even more aggressive alpha (2). But when the alpha was set to be 2, the performance could be even worse with the final accuracy around 40%. How to fine tune the hyper-parameters for mix-up network is still one mystery for me right now.

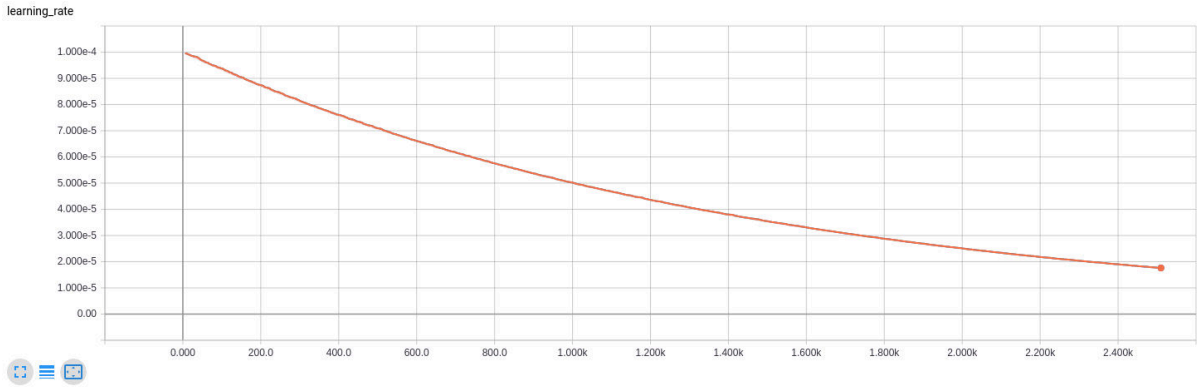


Figure 50: Learning rate for VGG

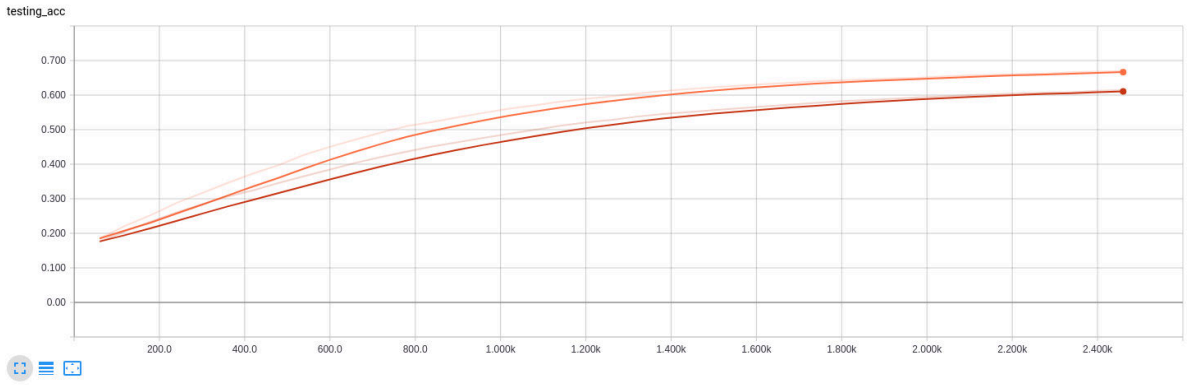


Figure 51: Accuracy for VGG

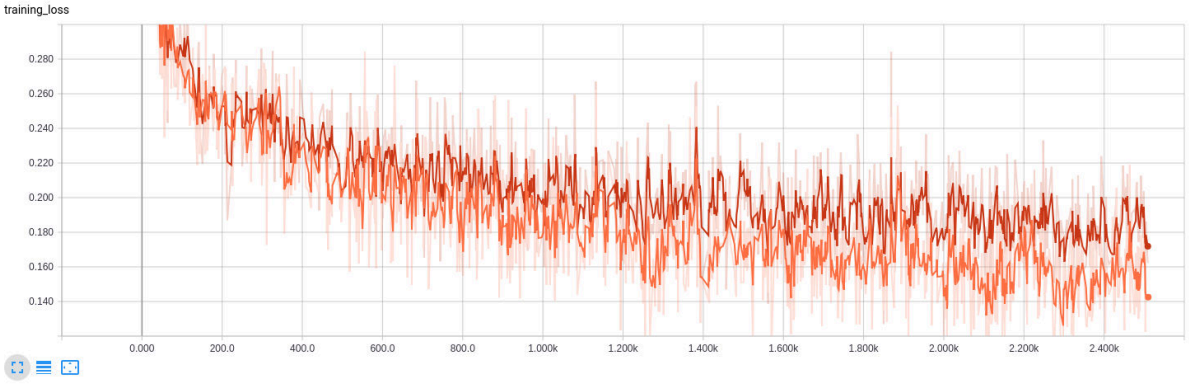


Figure 52: Loss for VGG

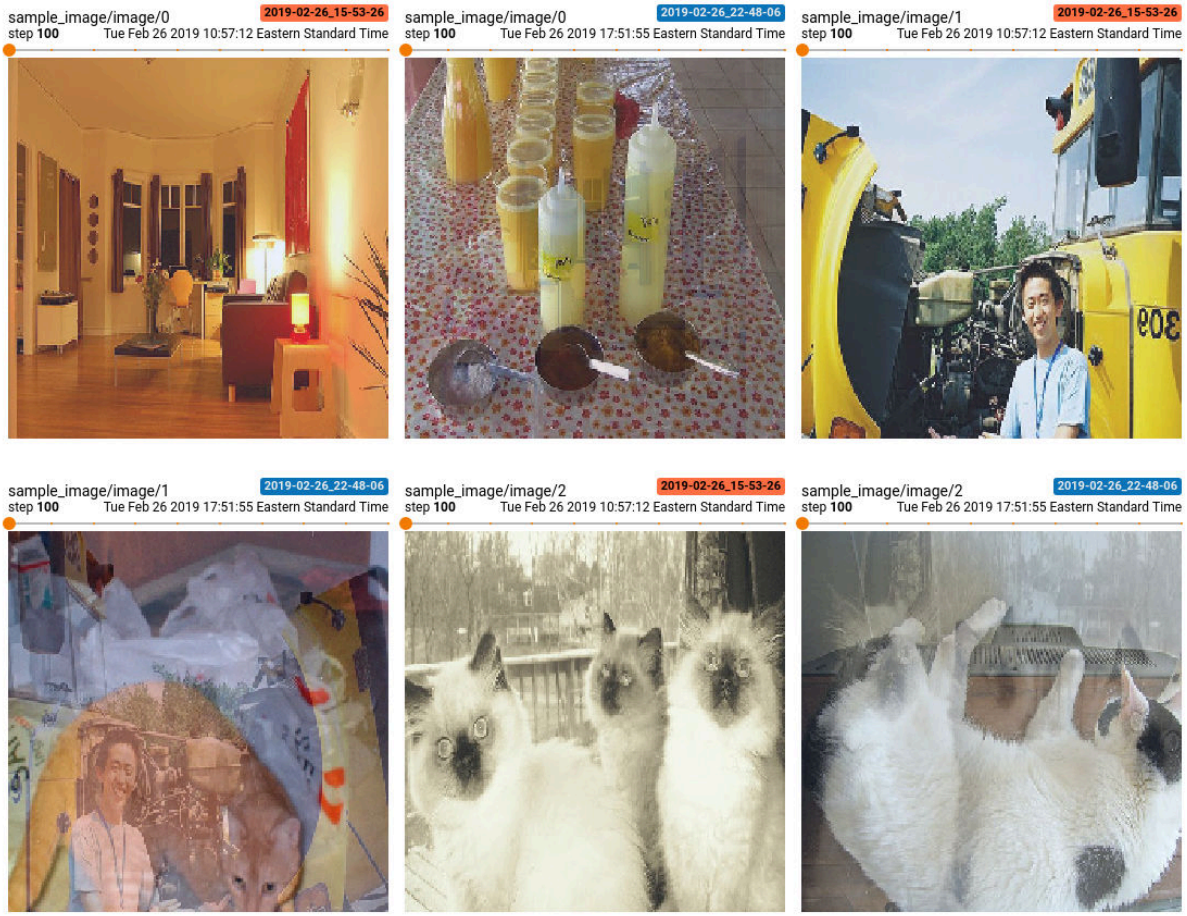


Figure 53: Images for 100 iteration

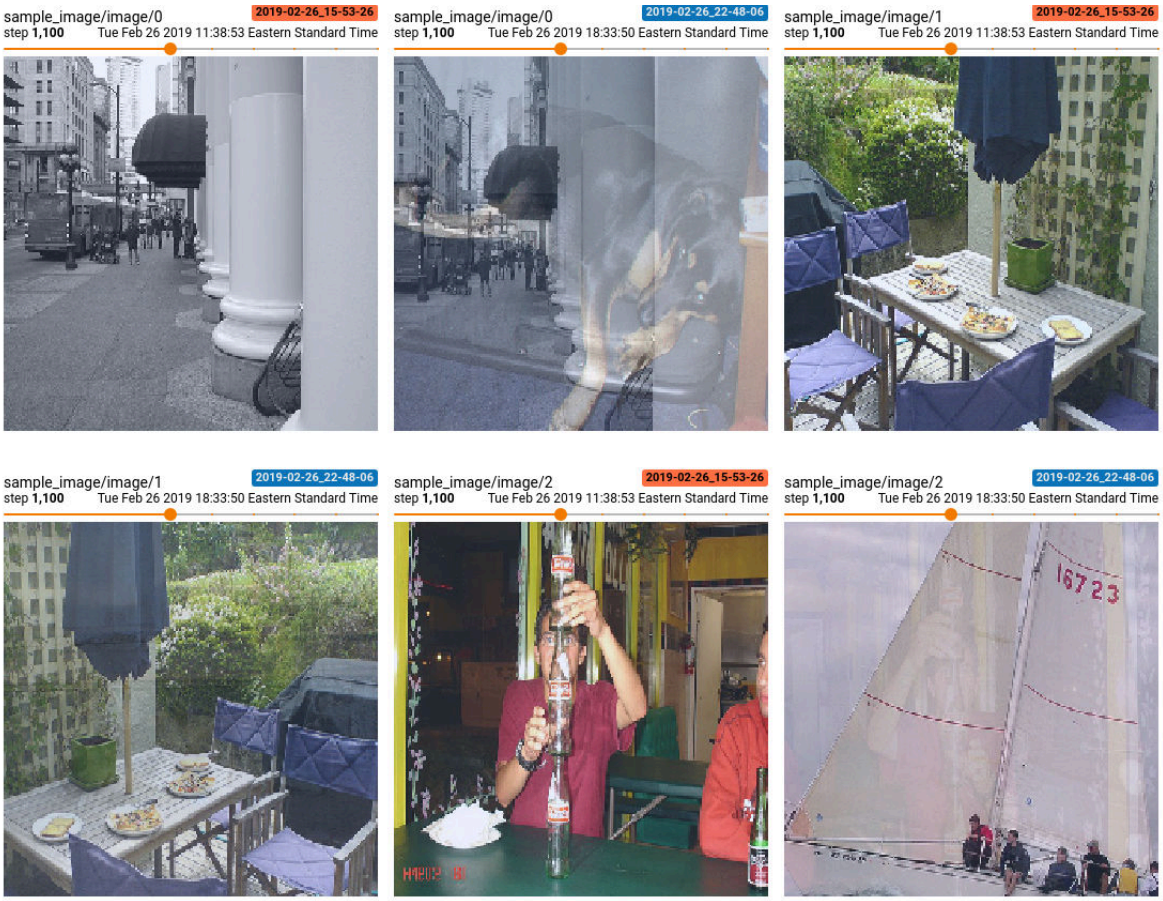


Figure 54: Images for 1100 iteration

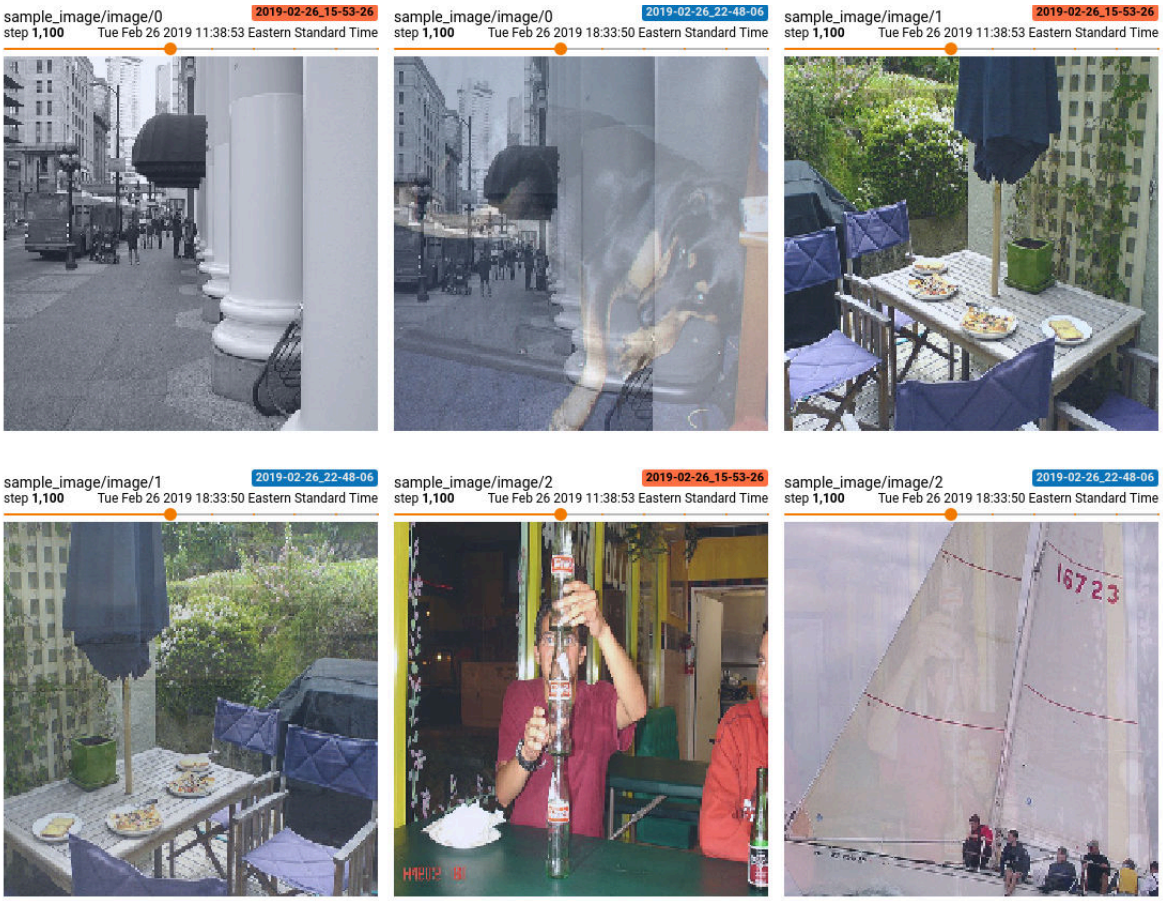


Figure 55: Images for 1100 iteration

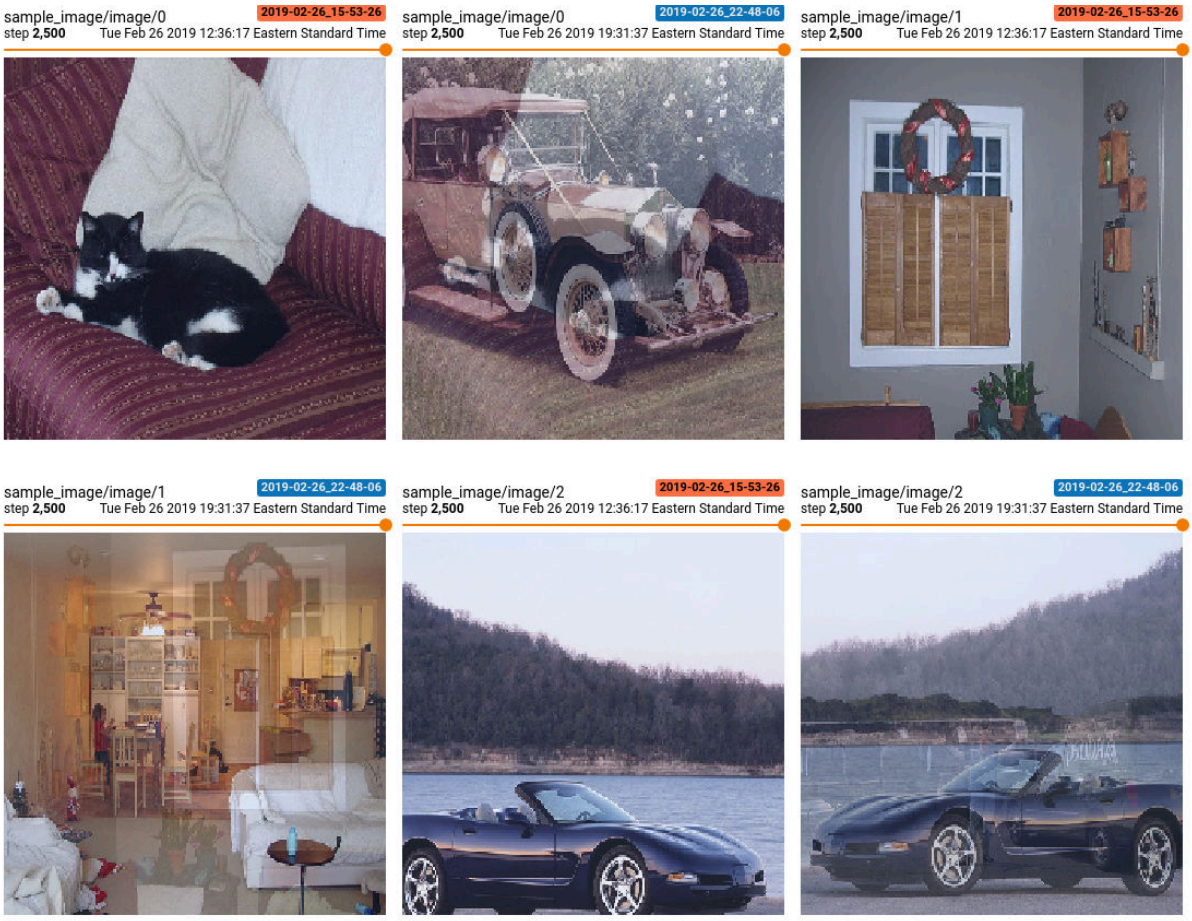


Figure 56: Images for 2500 iteration

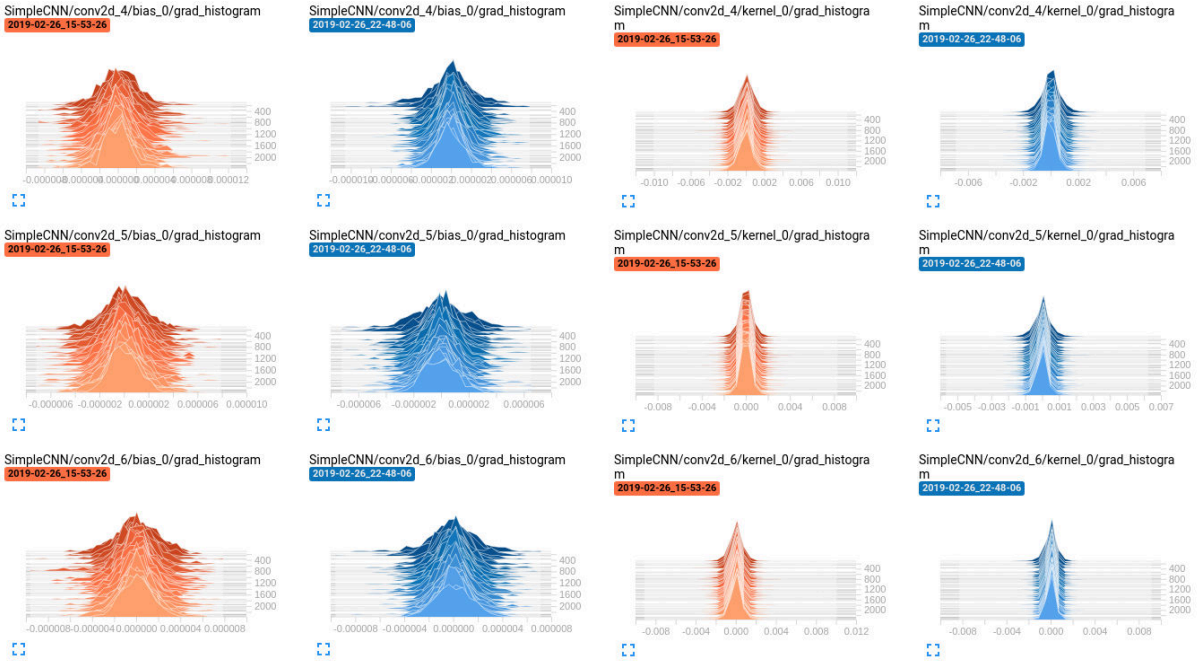


Figure 57: Histogram for VGG pretrained

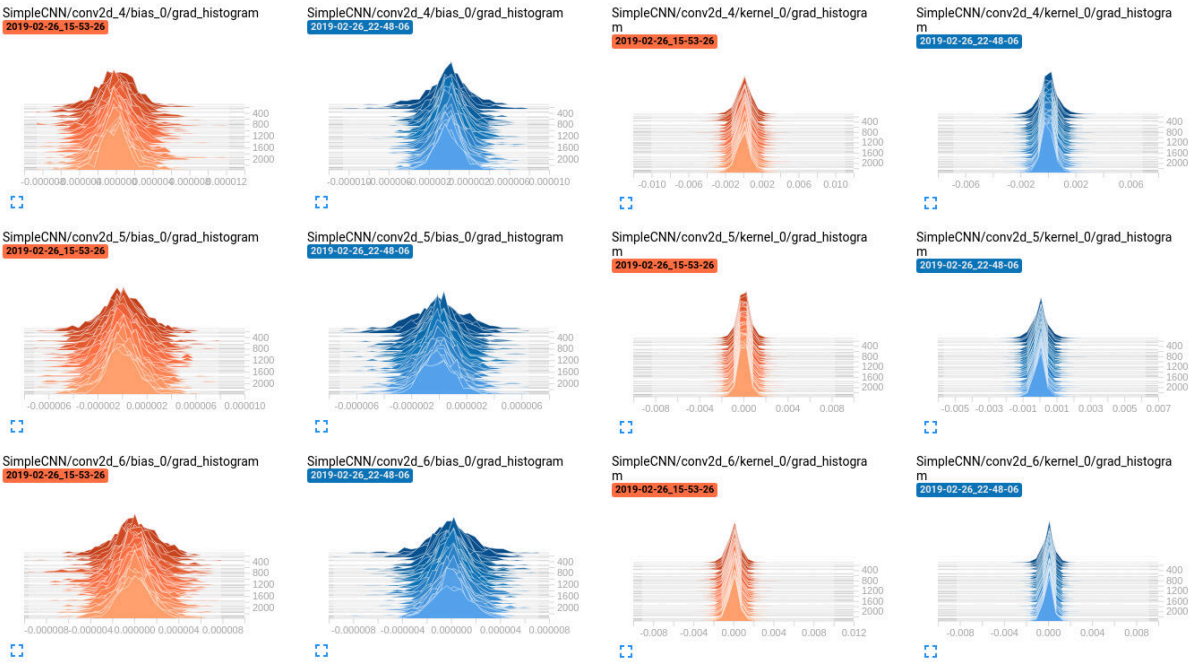


Figure 58: Histogram for VGG pretrained

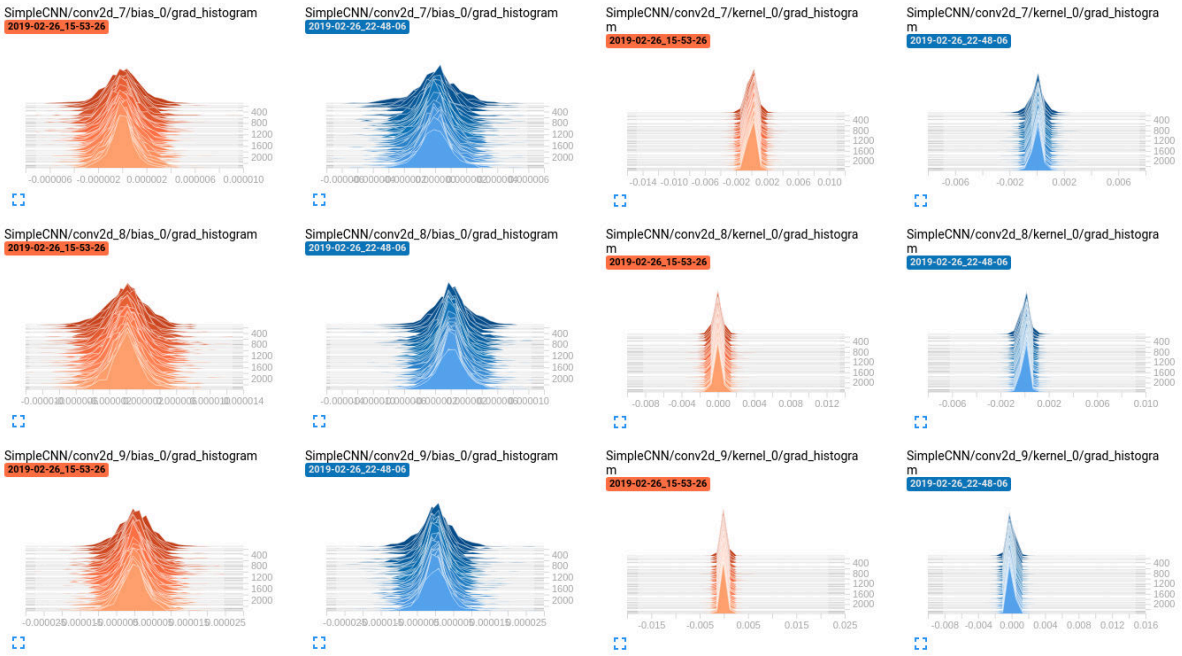


Figure 59: Histogram for VGG pretrained

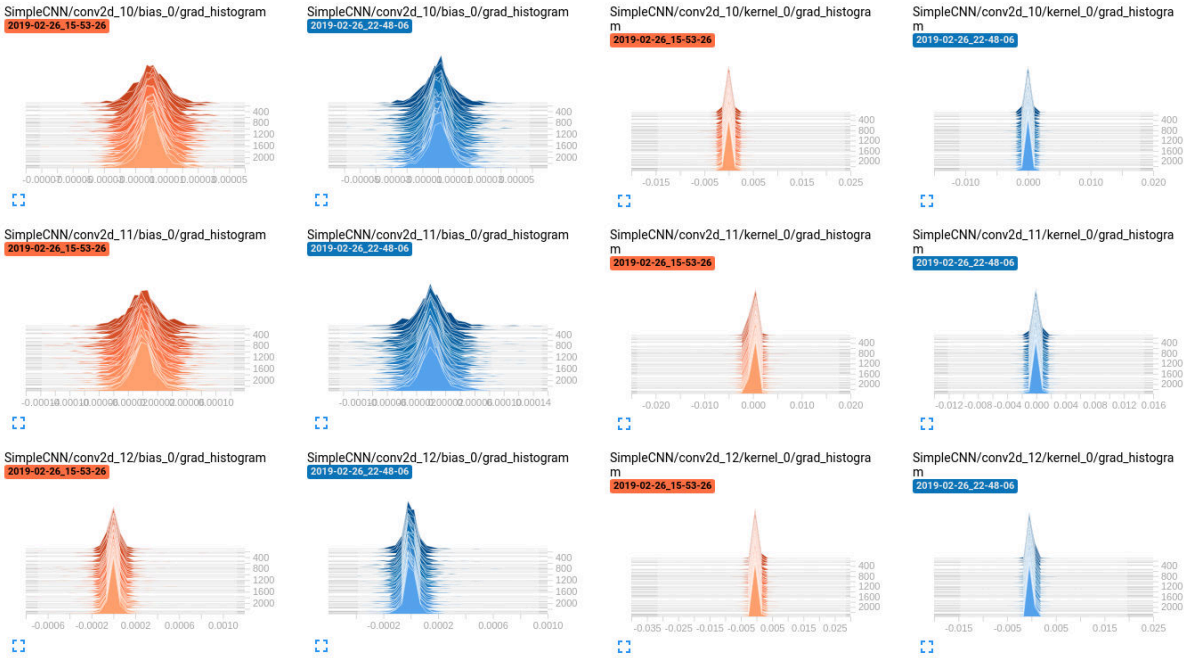


Figure 60: Histogram for VGG pretrained

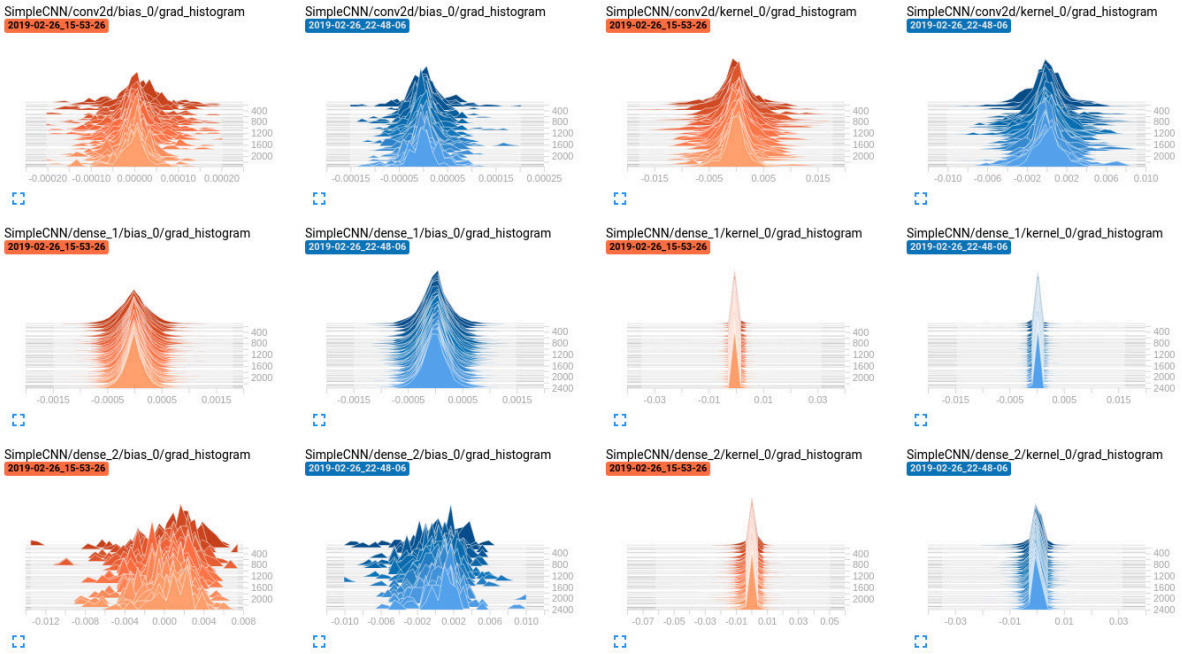


Figure 61: Histogram for VGG pretrained

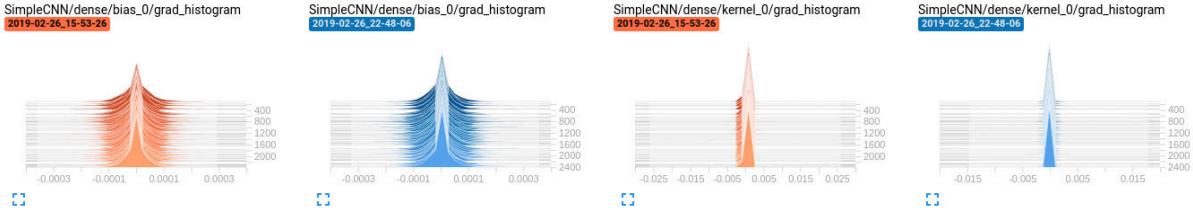


Figure 62: Histogram for VGG pretrained

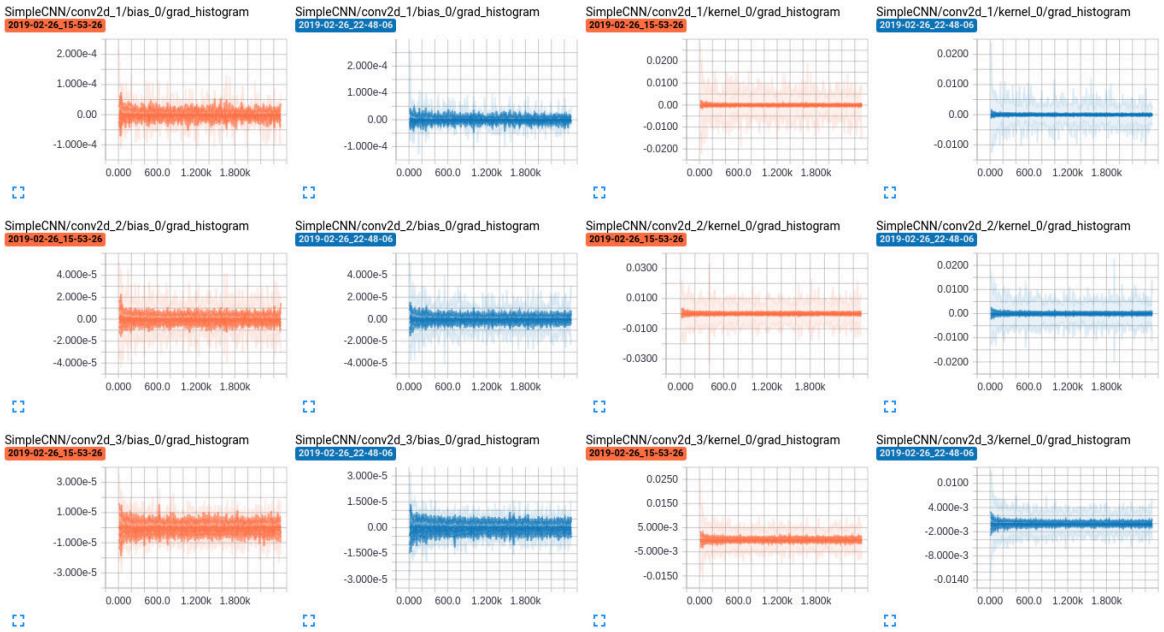


Figure 63: Distribution for VGG pretrained

Figure 64: Distribution for VGG pretrained

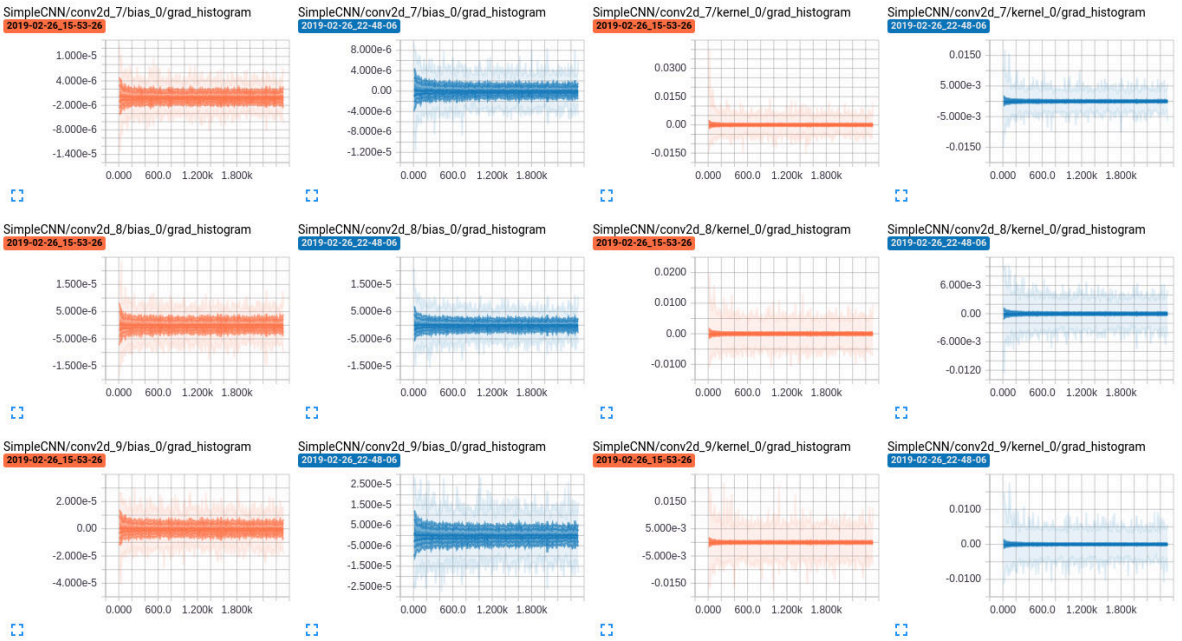


Figure 65: Distribution for VGG pretrained

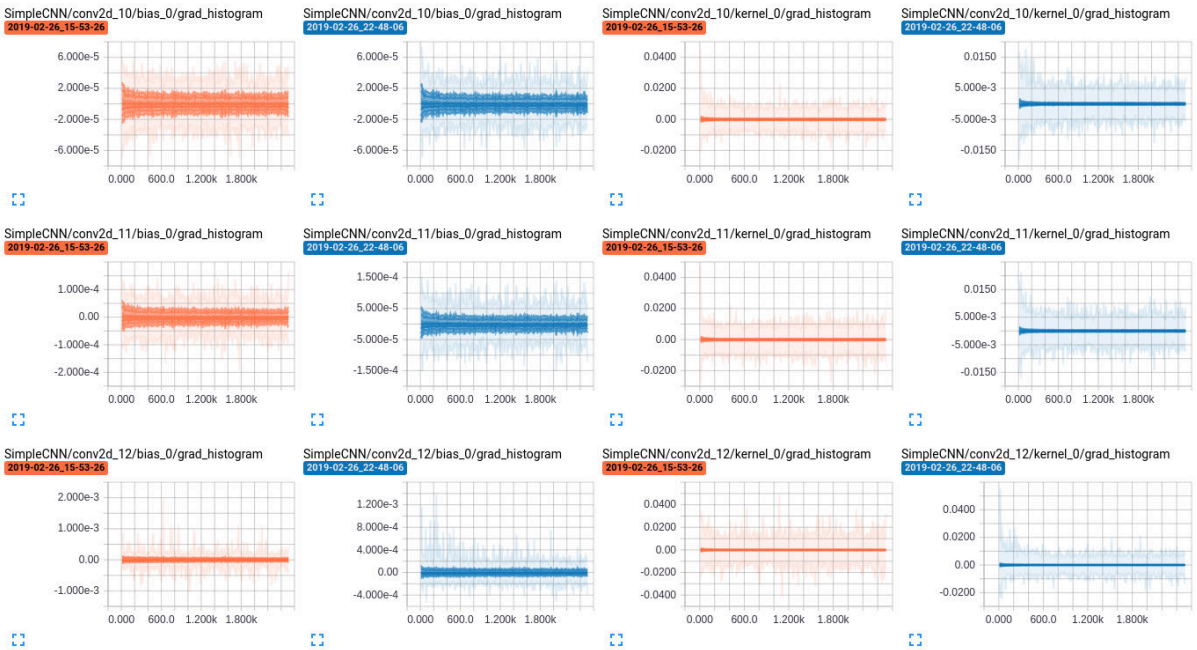


Figure 66: Distribution for VGG pretrained

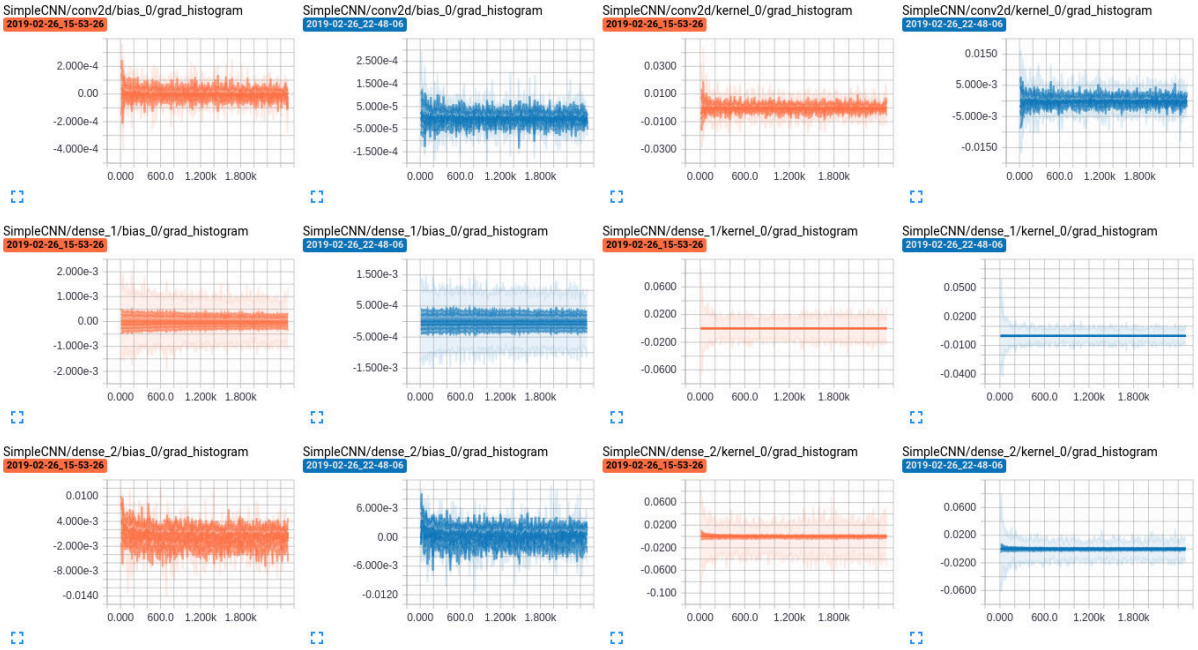


Figure 67: Distribution for VGG pretrained

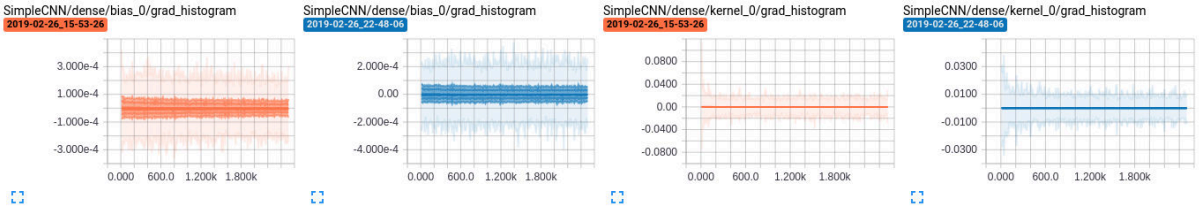


Figure 68: Distribution for VGG pretrained