Report for the Deep Learning Course Assignment 3

Arthur Bražinskas

1 Task 1

1.1 Main

The CNN model's graph is shown in fig. 1. When the model is trained for 10k steps(iterations) and all default hyper-parameters the best accuracy is around 75.7 % on the test set consisting of 10,000 images, and accuracy curves are shown in fig. 2b. We can see that by the end of training training accuracy reaches 100% and the gap between test and training accuracies widens, which is a sign of over-fitting. The loss curve is presented in fig. 3a, from which we can see that our model successfully converges by the end of the training by reaching the loss plateau.

When model is regularized with l2 norm over weights(0.01 strength) of fully connected layers, the performance improves a little, i.e. best test accuracy is 76.5 %. The accuracy and loss curves are shown in fig. 2a and 3b respectively. The reason why l2 regularization help is because our model is likely to over-fit training datasets (which is a common trend of complex models such as neural networks), and by regularizing our weights we're able to train a better generalizing model to unseen instances. Finally, one can notice that after regularization training accuracy "hit" 100% less frequently.

1.2 Feature extraction

In this subsection I describe experiments with features extracted from different layers. I've trained 10 one-vs-rest linear classifiers(linear SVMs) and report performance (accuracy, precision, and recall) on each layer together with t-SNE visualized features based on 5k test datapoints. Please note that training and test sets were the same due to our interest in separability of features and not the ability of simple classifier to generalize.

The obtained total/multiclass accuracies (were a final prediction is made by argmax over predictions of each classifier) on features of unregularized model were the following: 26.4 %, 43.2 %, 66.7 % for flatten, f1, and f2 layers respectively.
As one can see the deeper we go into the networks, the more discriminative features we obtain. This statement is also supported by the visualization of features presented in fig. 6 as different class images become more separated into segments. Since we use softmax that converts values obtained via dot-product into distribution over class labels, we should expect that representations (features) belonging to datapoints of the same class should have a small angle, i.e. should be in some common segment of space. It's easy to see why by considering the nature of dot product which is an angle between vectors scaled by their magnitudes.
Finally, I report per-class recall and precision in fig. 5. It's important to note that linear classifiers such as SVM(which has been used) do not operate well on unbalanced datasets, therefore the balancing step has been performed, and the number of data points for the target and the "rest" classes has been to be the same. One interesting observation we can draw is that some classes are less separable than others, for example class 2.

When features where extracted from the regularized model the total/multiclass accuracies have been the following: 28%, 47.6 %, and 65.4 for flatten, f1, and f2 layers respectively, and visualization of

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

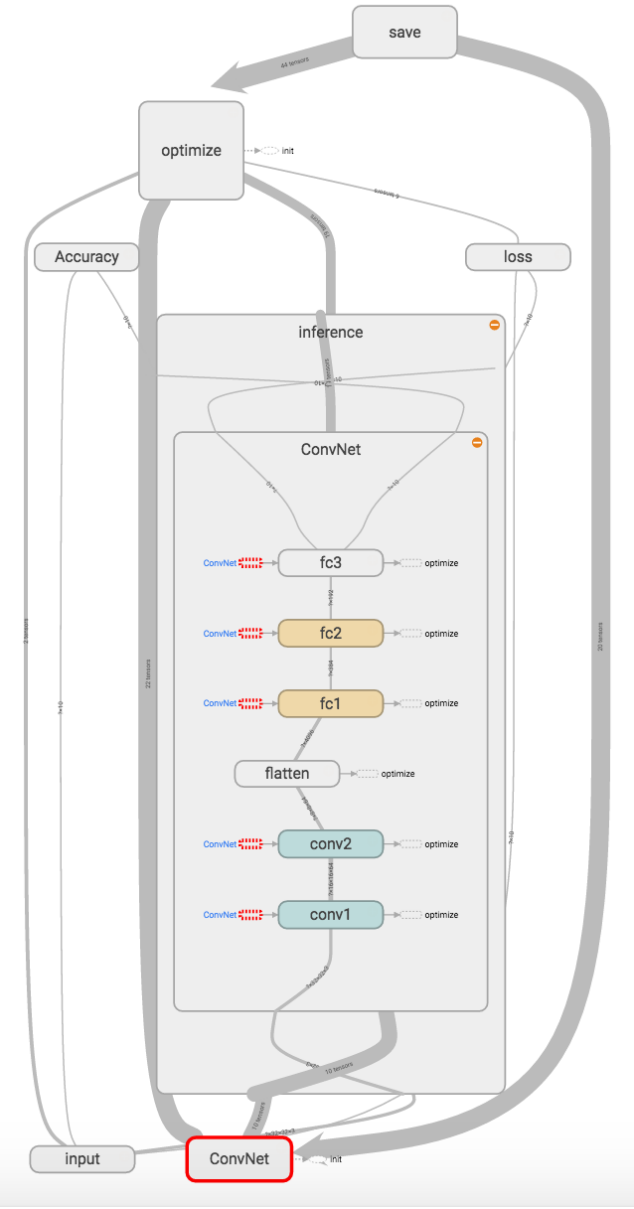
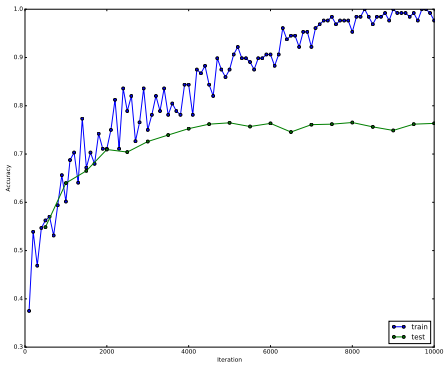


Figure 1: CNN Tensorflow graph

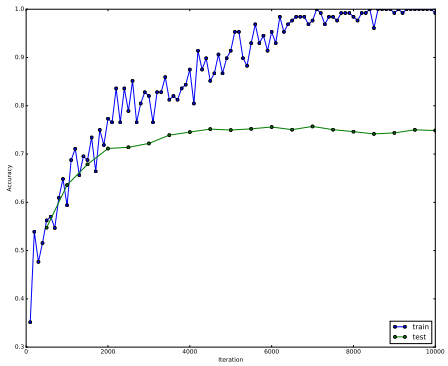
features is presented in fig. 7. In addition, precision and recall performances per class are shown in fig. 5. The regularized features help to perform multiclass classification but there are no clear signs that regularized features make every class more separable based on precision and recall.

2 Task 2

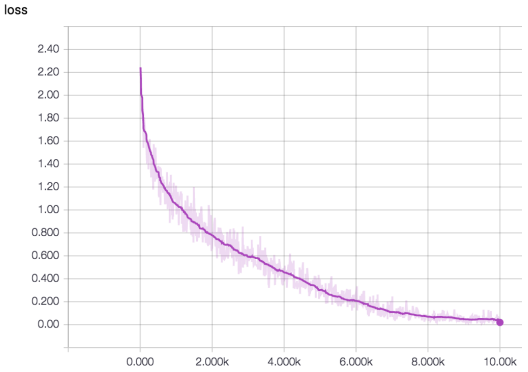
Two possible applications of such architectures are signature matching and passport face matching with database records on customs or airports. In the former case we're interested in knowing if two signatures are the same or not, and thus we train a NN to discriminate between similar and disimilar signatures, then we apply it in practice to avoid fraud. In the second case, we're interested in knowing if a person's pass photo match with photos in our database (assuming the same size, which is a reasonable assumption in the context of pass photos) of criminals, so we train a siamese NN.



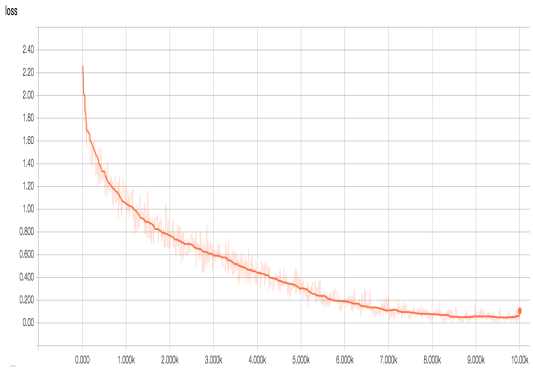
(a) Accuracy with regularization



(b) Accuracy without regularization



(a) Training loss without regularization



(b) Training loss with regularization

2.1 Feature extraction

After the model with default parameters has been trained for 50k steps(with default hyper-parameters), the total/multiclass accuracy was 63.5 %, and the precision/recall over classes is presented in fig. 10. As we can we from precision/recall and multiclass accuracy, we perform slightly worse on some classes and on multi-class accuracy then CNN f2 layer but still competitive, the explanation for that follows.

The loss curves are presented in fig. 8. As one can see from the loss curve the training loss looks a bit stochastic while validation loss looks smooth, and it could be caused by the nature of small batches on which loss is estimated.

The visualization of features is shown in fig. 9. As one can see from the visualization, the separation of images of different classes is much more meaningful than in case of CNN, we have small clusters instead of segments of points belonging to the same class, and this is likely to be caused by the difference in the discriminator, i.e. softmax vs contrastive loss. As stated previously in softmax we're projecting each representation of an image to a representation of a class just before the softmax normalization, and thus same class datapoints are expected to have a small angle similar to word2vec. In our current case, we are using l2 distance to guide our model in contrastive regime, and thus datapoints of the same class are expected to be nearby in space (which is technically a D-dimensional unit sphere as we do l2 normalization).

An interesting research question would be to experiment with kernelized classifier, such as radial ones or fitting a clustering algorithm because it betters fits the expected output of the model (i.e.

```
pr :0.731429, re: 0.786885 for class 0
pr :0.628272, re: 0.712871 for class 1
pr :0.581571, re: 0.751953 for class 2
pr :0.609568, re: 0.794769 for class 3
pr :0.624650, re: 0.879684 for class 4
pr :0.614286, re: 0.793033 for class 5
pr :0.702362, re: 0.908350 for class 6
pr :0.611852, re: 0.834343 for class 7
pr :0.766143, re: 0.871032 for class 8
pr :0.652824, re: 0.766082 for class 9
total accuracy is 0.264200
```

(a) Flatten layer

```
pr :0.743119, re: 0.829918 for class 0
pr :0.779439, re: 0.825743 for class 1
pr :0.610018, re: 0.666016 for class 2
pr :0.640299, re: 0.863179 for class 3
pr :0.662975, re: 0.826430 for class 4
pr :0.678808, re: 0.840164 for class 5
pr :0.749077, re: 0.826884 for class 6
pr :0.810680, re: 0.674747 for class 7
pr :0.804836, re: 0.924603 for class 8
pr :0.706147, re: 0.918129 for class 9
total accuracy is 0.432800
```

(b) F1 layer

```
pr :0.793241, re: 0.817623 for class 0
pr :0.723104, re: 0.811881 for class 1
pr :0.589457, re: 0.720703 for class 2
pr :0.681431, re: 0.804829 for class 3
pr :0.707120, re: 0.861933 for class 4
pr :0.752852, re: 0.811475 for class 5
pr :0.625544, re: 0.877800 for class 6
pr :0.829960, re: 0.828283 for class 7
pr :0.802708, re: 0.823413 for class 8
pr :0.889105, re: 0.890838 for class 9
total accuracy is 0.666400
```

(c) F2 layer

Figure 4: Regularized model's per class performance

```
pr :0.724070, re: 0.758197 for class 0
pr :0.624194, re: 0.766337 for class 1
pr :0.586268, re: 0.650391 for class 2
pr :0.622671, re: 0.806841 for class 3
pr :0.673986, re: 0.786982 for class 4
pr :0.622186, re: 0.793033 for class 5
pr :0.706690, re: 0.839104 for class 6
pr :0.618590, re: 0.779798 for class 7
pr :0.776786, re: 0.863095 for class 8
pr :0.621746, re: 0.791423 for class 9
total accuracy is 0.280800
```

(a) Flatten layer

```
pr :0.736937, re: 0.838115 for class 0
pr :0.758813, re: 0.809901 for class 1
pr :0.600000, re: 0.673828 for class 2
pr :0.628912, re: 0.849095 for class 3
pr :0.694309, re: 0.842209 for class 4
pr :0.633648, re: 0.825820 for class 5
pr :0.775472, re: 0.837067 for class 6
pr :0.796748, re: 0.791919 for class 7
pr :0.848197, re: 0.886905 for class 8
pr :0.730956, re: 0.879142 for class 9
total accuracy is 0.476000
```

(b) F1 layer

```
pr :0.835391, re: 0.831967 for class 0
pr :0.700787, re: 0.881188 for class 1
pr :0.600985, re: 0.714844 for class 2
pr :0.629261, re: 0.891348 for class 3
pr :0.774955, re: 0.842209 for class 4
pr :0.692429, re: 0.899590 for class 5
pr :0.721843, re: 0.861507 for class 6
pr :0.879208, re: 0.896970 for class 7
pr :0.898374, re: 0.876984 for class 8
pr :0.867804, re: 0.793372 for class 9
total accuracy is 0.654400
```

(c) F2 layer

Figure 5: Non-Regularized model's per class performance

clusters). Considering all that, the potential reason why we're getting slightly worse performance could be caused by not well suited classifier(linear).

3 Task 3

3.1 Feature extraction

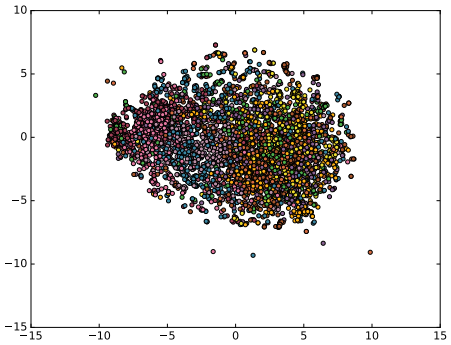
The Tensorflow graph is presented in fig. 13. During the training I've used default setup including Adam optimizer and 2500 steps due to constant problems with a long queue on SurfSara. Without training VGG layers the final test accuracy was 79%(see fig. 14e), which is higher then in the task1 (despite the difference the training steps). The reason for this is because we've used higher quality (discriminative) features of images that let us perform better classification, and which require more training and tuning to obtain on a primitive task like ours(10 classes only). Technically, we've used learned by VGG filters that were obtained on a more complex task, and over more iterations.

3.2 Retraining

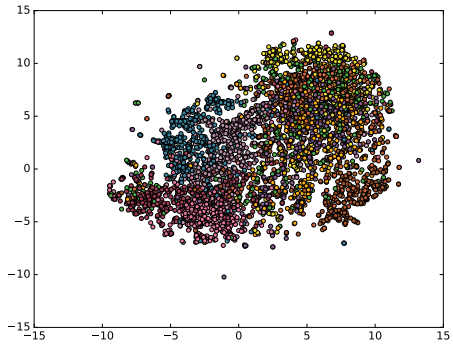
After training the VGG weights from step 0 the final accuracy was 86.2 %, which is much better than in the previous experiment, see fig. 14a, it's also interesting that training accuracy approached 100% by the end of the training. The corresponding training (based on batch) loss curve is shown in fig. 11, and histograms of VGG weights are shown in fig. 12 (only few weight histograms are shown to save space because the pattern is the same for all weights). As one can see from histograms the weights do not change over time that much, and biggest changes happen in the beginning of training. That can indicate that VGG filters are well suited for our task at hand.

3.3 Refining

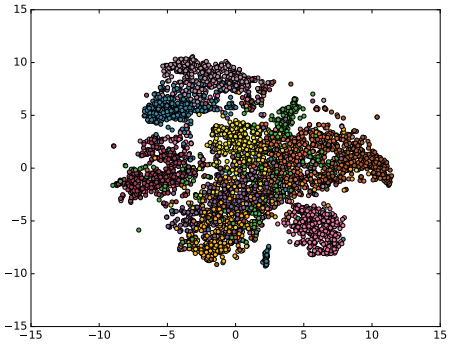
In addition, I've experimented with different starting points of training VGG weights, and the results are shown in fig. 14. The final test accuracies were: 85.6%, 85.06 %, and 83.8 % for refine after 100, 1000, and 1500 steps respectively. As one can see the sooner we start to train VGG weights the better is the final result. The reason for this could be that despite we initially get powerful filters we still can improve by adjusting them to our task, and the sooner we start, the more accurately those filters fit our image patterns.



(a) Flatten layer



(b) F1 layer



(c) F2 layer

Figure 6: Visualization of features from different layers (without reg)

3.4 Hypothesis

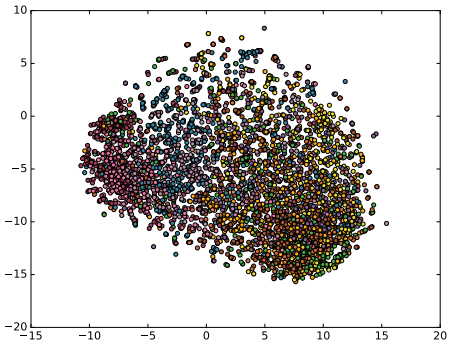
The first hypothesis is that the initial task's(which VGG weights are obtained) similarity with the subsequent one (cifar10) has a positive correlation with the final accuracy. The hypothetical reason is the following: say we train our initial model to classify 10000 types of cars, then it's unlikely it's going to generalize to classification of cats. But if we classify 10000 animals, we might be able to reuse it to classify cats and dogs. Unfortunately, this hypothesis is impossible to test due to time limitations.

A second hypothesis is that regularization of feed-forward weights can improve the performance, and the result of the experiment with joint training of all weights after 0 step with 0.00015(strength) l2 regularization is shown in fig. 14f, and the best accuracy has been 86.9%, which is slightly higher than without regularization.

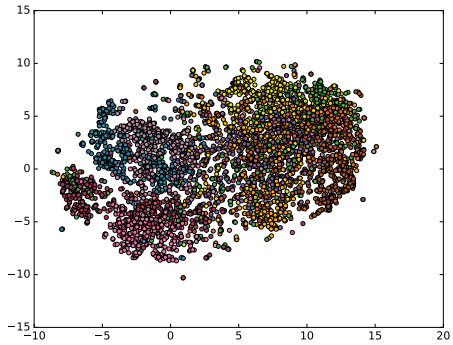
Third hypothesis is that if we would increase dimensions of the feed-forward layers, we would get better performance. I've tested this hypothesis with 512 and 256 dimensions of fc1, and fc2 layers respectively, and the final result was only 84.8 % and the training accuracy approaches 100% relatively faster, which can indicate that more neurons require some mechanisms for preventing over-fitting.

3.5 After deadline's extension

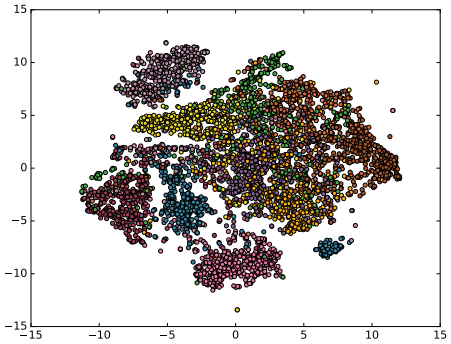
After the deadline has been extended, I've experimented with training the model for 10k steps to see the result. I've observed that in principle all setups regardless of the refinement converge to a similar accuracy (around 85- 86%), but what was stunning is that the model without training of VGG layers managed to get around 86% too! So one conclusion we can draw is that it's very beneficial to



(a) Flatten layer



(b) F1 layer



(c) F2 layer

Figure 7: Visualization of features from different layers (with reg)

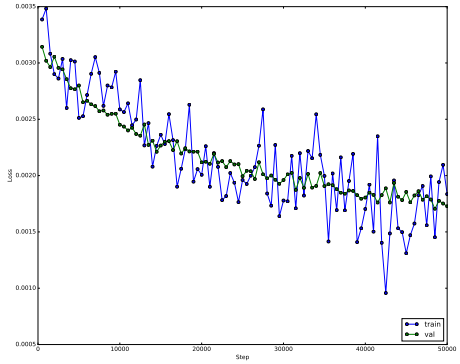


Figure 8: Siamese loss curve

perform updates as early as possible is there are not enough time resources, otherwise the model can find a way to learn necessary forward layers features to produce a decent fit given enough time.

4 Conclusion

In this project, we experimented with 3 different neural network architectures/setups. First was a pure CNN which let learning of shared parameters(filters) and that showed better results than feed-forward

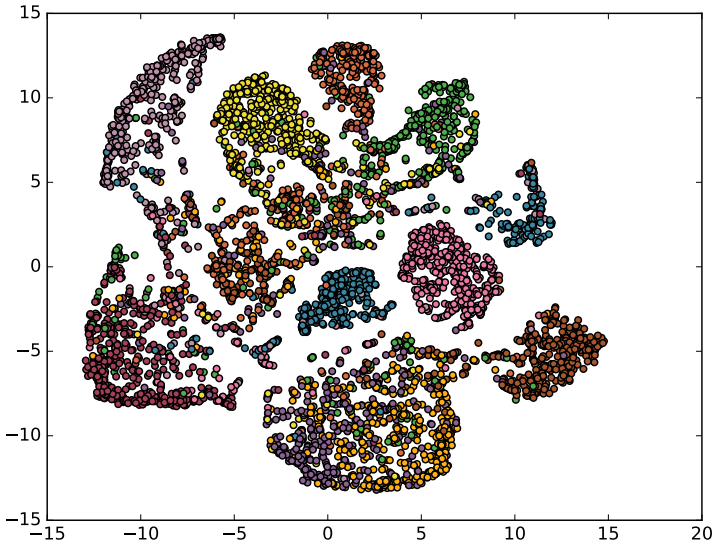


Figure 9: Visualized 5k test data points with siamese features

```
pr :0.892116, re: 0.881148 for class 0
pr :0.635200, re: 0.786139 for class 1
pr :0.639286, re: 0.699219 for class 2
pr :0.653631, re: 0.706237 for class 3
pr :0.658088, re: 0.706114 for class 4
pr :0.785567, re: 0.780738 for class 5
pr :0.764499, re: 0.885947 for class 6
pr :0.767010, re: 0.751515 for class 7
pr :0.661264, re: 0.809524 for class 8
pr :0.821862, re: 0.791423 for class 9
total accuracy is 0.635200
```

Figure 10: Precision/recall over classes of siamese learned features

neural networks in the previous labs on the same task. The second was Siamese architecture, which uses a different from softmax loss and capable of producing features that will cluster datapoints of the same class. Last one was transfer learning setup where we experiments with pre-trained weights on a different task and produced even better than pure CNN results. Experiments showed that the earlier we start training, the better results we obtain unless we're given a lot of time resources.

References

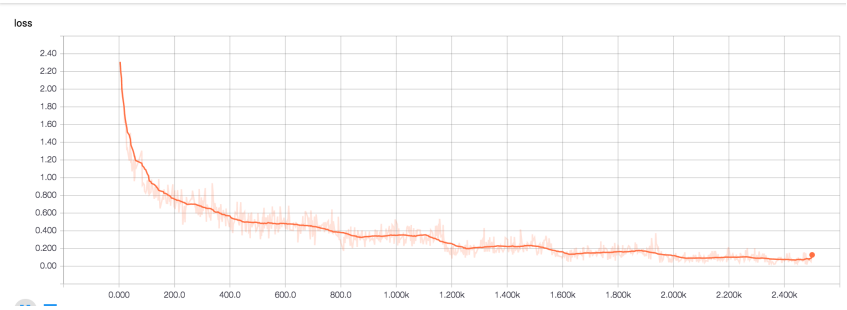


Figure 11: VGG joint training's loss

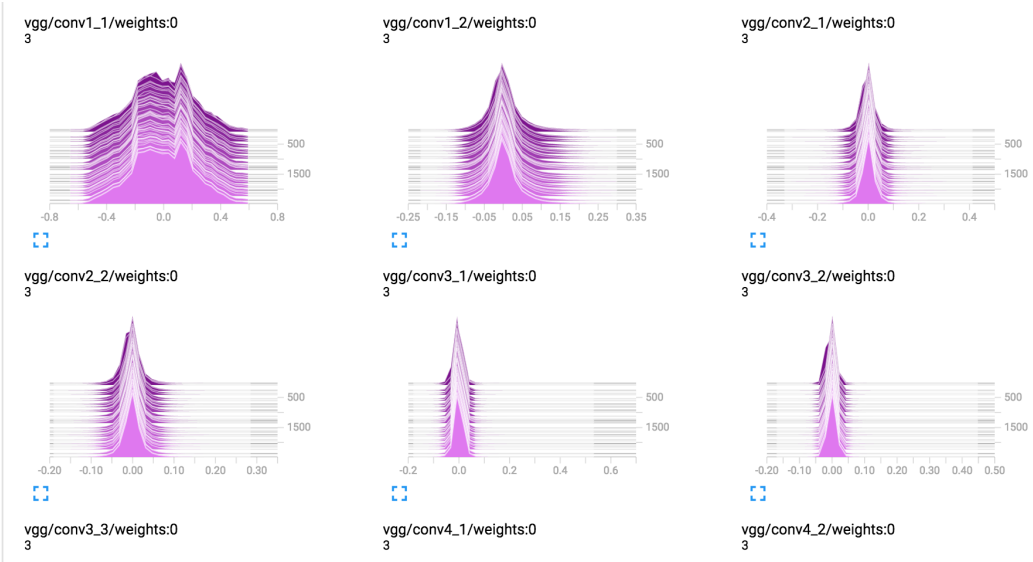


Figure 12: VGG joint training's hisograms

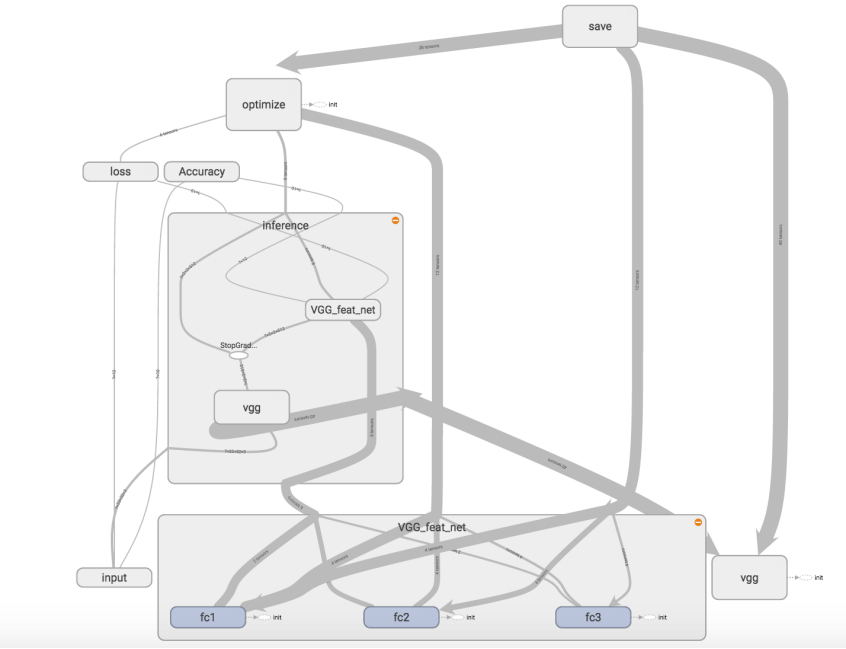
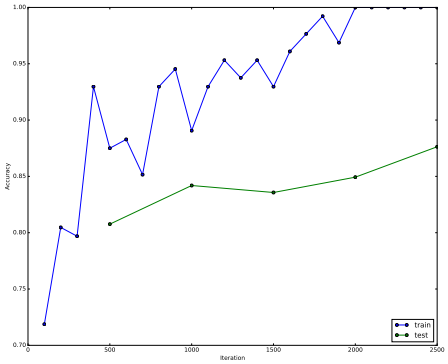
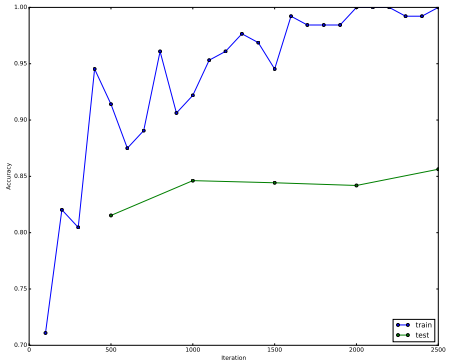


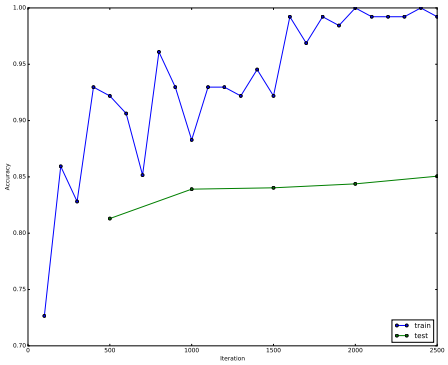
Figure 13: VGG Tensorflow graph



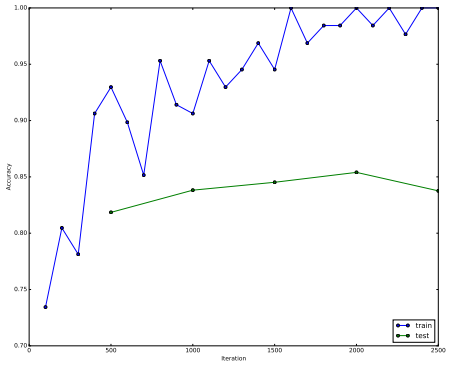
(a) Refine after 0 steps



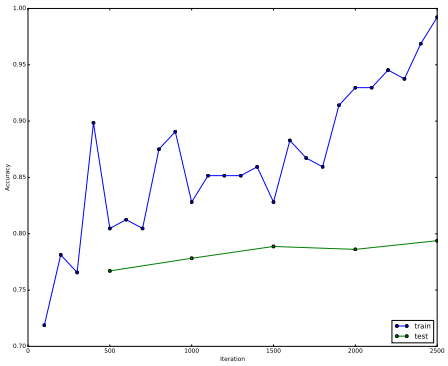
(b) Refine after 100 steps



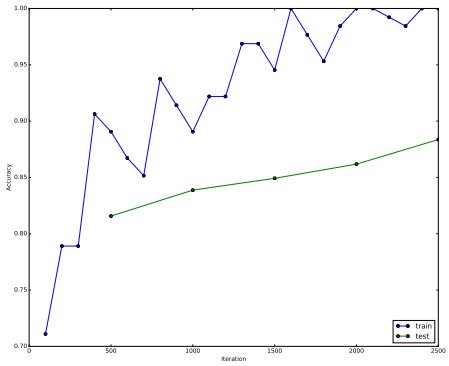
(c) Refine after 1000 steps



(d) Refine after 1500 steps



(e) No training of VGG layers(only feature extraction)



(f) Refine after 0 steps with l2 regularization

Figure 14: Accuracies of different models with VGG filters