CS5242 Project Report

Teekayu Klongtruajrok, Zheng ZhiJian

1 Accuracy Jump on Restart [Not important, can remove]

We used *tf.contrib.metrics.streaming_accuracy* to update accuracy. This function seems calculating running average. So the curve is smooth, but lower than actual value because of low values at the begining. When restart, the old values are discarded for calculating the average. See firgure.

2 Effect of learning rate [Please advice other explanations]

We used exponential decay learning rate. Initial learning rate is 0.0002, it is reduced to 70% every 2 epochs. We also tried another setting with initial learning rate of 0.001, reduced to 97% every 2 epochs. To our surprise, accuracy for the smaller learning rate increases much faster. E.g. after 4 epochs, we got 65% training accuracy for the smaller learning rate, but only 43% accuracy for the bigger learning rate.
We suspect it may due to the ADAM optimzer. Smaller learning rate give us more time to accumulate momentum, and thus easier to escape from the local traps.

3 Network Model Tested

	traing accuracy	validation accuracy	ImageNet Top-1 Accuracy
inception v4	99.75%	87.90%	80.2%
inception resnet v2	99.73%	88.39%	80.4%
nasnet large	84.73%	84.12%	82.7%

Our experiement results for Incpetion resnet v2 and inception v4 looks consistent with reported ImageNet accuracy. However, nasnet large doesnot work well on our expriement although it is reported to perform best on ImageNet. Perhaps

- it doesnot fit our dataset well.
- or maybe we need to finetune the hyperparameters.
- or maybe our implementation is not correct.

We tested the model again by modifying *slim.train_image_classifier.py* to process our dataset. The hyperparameters are also different from our code. It produces validation result of 82.60%. With this, we believe our implementation is correct.

As it is much slower to train nasnet large compared to inceptino net (about 4x slower). We give up it and focus on improving the inception resnet v2.

There is a big gap between training accuracy and validation accuracy. A clear indication of overfitting. We have tried the following the aleivate this problem:

- Regularization.
  We checked tensorflow implementation. Regularization is enabled by default with weight 1.0. Cross entropy loss is about 0.11 while regularization loss is about 0.50. We felt that it seems not meaningful to increase regularization weight.

- Dropout. Default dropout keep probability is 0.8. We reduced it to 0.5. But the result is worse. Then we realized that we may did it wrongly. To save some time, we start from our last trained model (dropout 0.8). But seems the model already 'memorized' our dataset, and it is not 'forgetting' even after we reduce the drop out.

- Reduce network model. This one is a bit complicated to us. We did not have time to explore.

- Freeze bottom layer variables. Low level features are more likely to be reusable. They are trained on ImageNet which has We tried 4 models: train the top layer only, the last two layers, the last four layers (the repeated inception network is considered as one layer) and the last six layers. The first 3 attempts improves very slowly, we stopped them as we think they suffer from underfitting. The last one improves faster, we are still training it. If it fails, we'll try traing the last 8 layers, i.e. left only the stem freezed.

- Feed more data.
  This should work. Once we fixed the hyperparameters, we can merge the validation set to training set.

4 Future Work

- Reduce network model. This one is a bit complicated to us. We did not have time to explore.

- SGD. Refer to the paper.

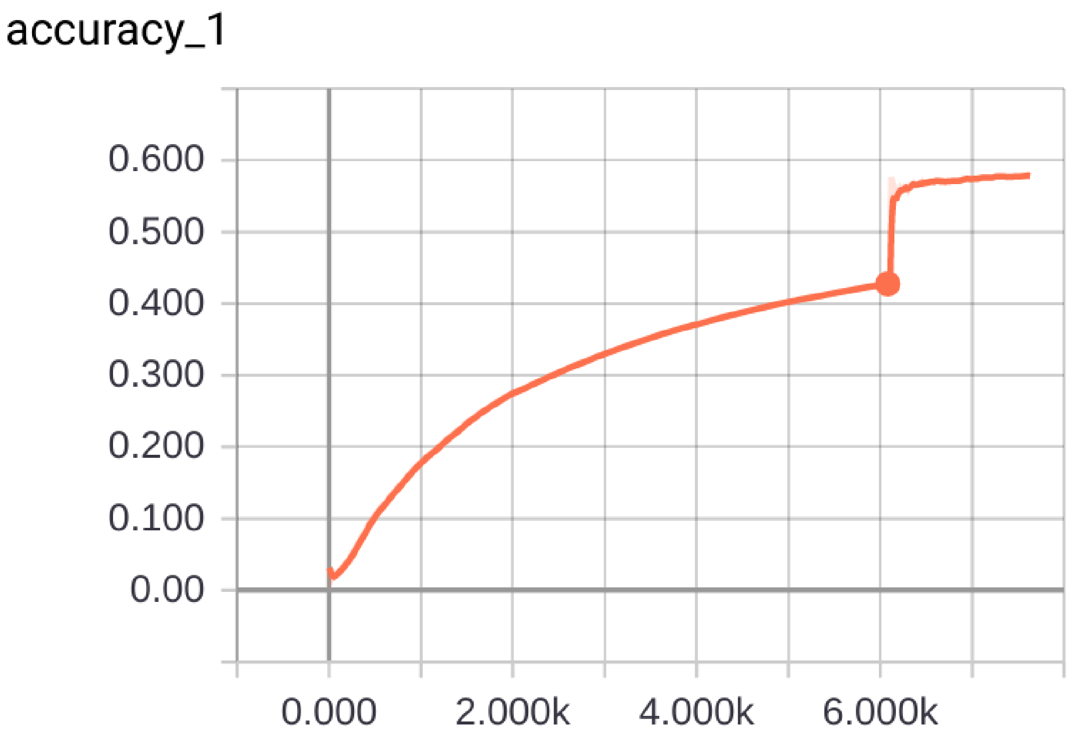

Figure 1: Accuracy Jump on Restart