Deep Learning Training on fasion mnist data

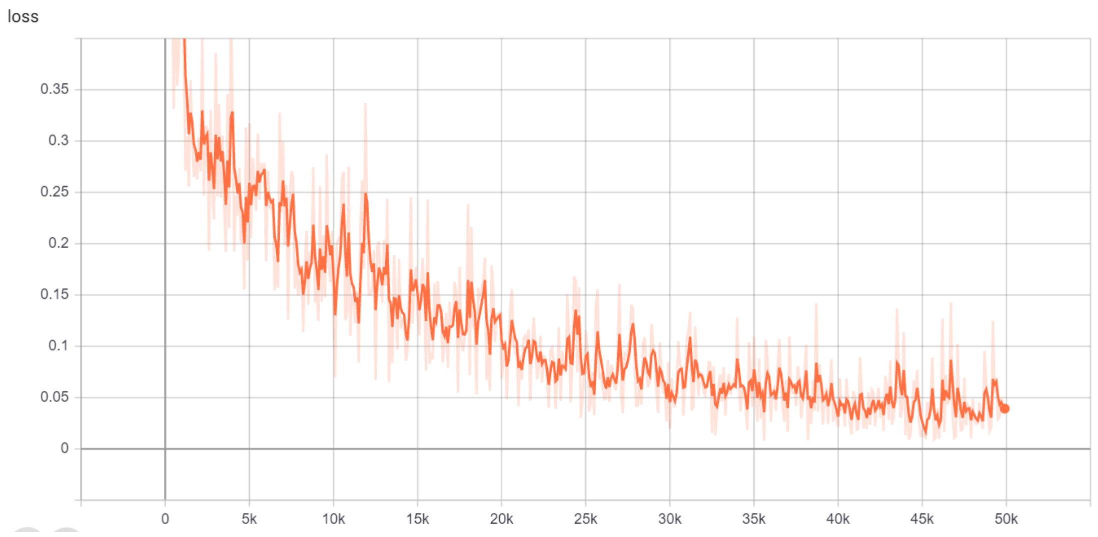
1. Build model:

Fasion mnist data has exactly the same image dimension and data size with original mnist data, but each image does look a bit more complex. Different classes could result from different shape related features in the images, so I choose CNN based neural network Lenet for training. I also did a bit of research online for benchmarks submitted by others, complex models do not show a lot of improvement compared to 2 layer CNN and cause a risk of overfitting.

2. Training:

In fasion mnist dataset, originally it has 60000 training dataset and 10000 test dataset, I use mnist data api in tensorflow, which further splits 60000 training dataset to 55000 for training and 5000 for validation. Here I train the model and optimize model weights using training datasets, and then evaluate model performance on test datasets. To avoid overfitting, I tune hyper-parameters by observing the metrics of both training and validation dataset.

First set iterations to large number, 50000 in my case. Then check tensorboard for loss movement over time, as following:



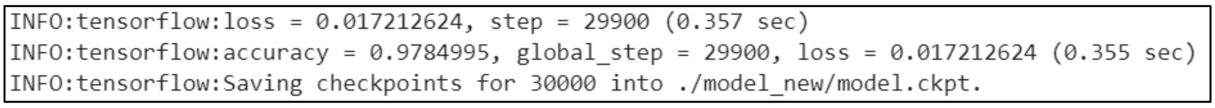
After 30000 iterations, training loss starts to look unstable, and loss reaches 0.05. So I decide to end training at iteration 30000.

(When using google colab's magic code for tensorboard at tensorflow versin 1.15.2, it does not work as expected. But tensorboard for tensorflow 2.x works just fine. Suspect it is due to environment setup at the backgound, a temporary fix is to modify "/usr/local/lib/python3.6/dist-packages/tensorboard/backend/application.py" to tackle the error "ValueError: Duplicate plugins for name whatif")

During hyper-parameters tuning, I experiment with different optimization method with the same learning rate. Observing training and validation accuracy result, I decide to use gradient descent. Then I also tried with different learning rate to deterimne the suitable value to be 0.01.

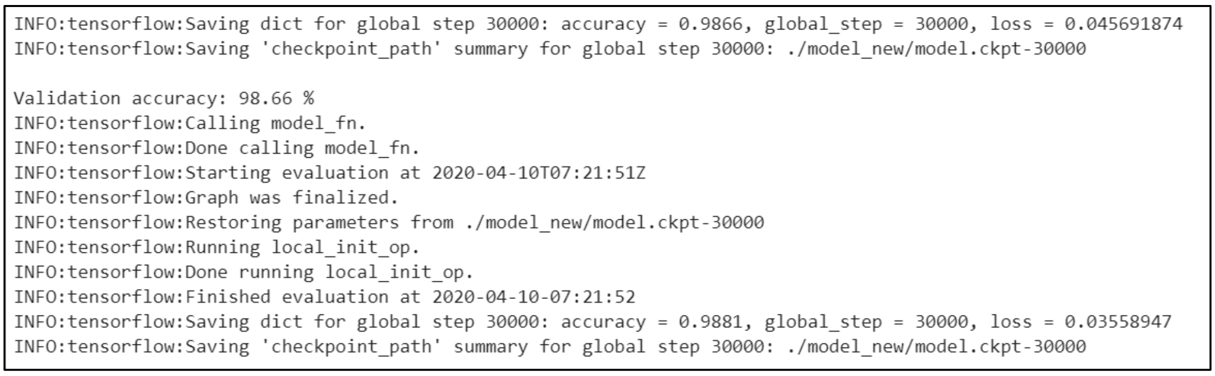
3. Results

After tuning, training results is loss at 0.017, with an accuracy of 97.84%,



```
INFO:tensorflow:loss = 0.017212624, step = 29900 (0.357 sec)
INFO:tensorflow:accuracy = 0.9784995, global_step = 29900, loss = 0.017212624 (0.355 sec)
INFO:tensorflow:Saving checkpoints for 30000 into ./model_new/model.ckpt.
```

While validation accuracy is 98.66%, and test accuracy at 98.81%. The numbers looks comparable, thus no overfitting should be present.



```
INFO:tensorflow:Saving dict for global step 30000: accuracy = 0.9866, global_step = 30000, loss = 0.045691874
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 30000: ./model_new/model.ckpt-30000

Validation accuracy: 98.66 %
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2020-04-10T07:21:51Z
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from ./model_new/model.ckpt-30000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2020-04-10-07:21:52
INFO:tensorflow:Saving dict for global step 30000: accuracy = 0.9881, global_step = 30000, loss = 0.03558947
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 30000: ./model_new/model.ckpt-30000
```

4. Future improvement

Since this assignment is mainly for testing of tensorflow programming skills, this is a basic illustration for what can be done. In future, data augmentation and more training paramters tuning can be an option to further improve model performance.