COMP90051 Project 2: report

Student: CHEN Zhengcheng

Student ID: 725439

Environment: Operating System: Ubuntu 17.04, Graphic Card: GTX 1060 with 6G memory, Framework: Tensorflow with CUDA v9.0 and cuDNN v8.0, Software: PyCharm with Anaconda which has python 3.6.2

First part is pre-process the dataset. Using the standscaler to fit the original dataset and transform the train set and test set to have 0 as mean value and 1 as standard deviation, which could keep the most features of the dataset and decrease the computing load. Next step is to reshape the original dataset and test dataset to 64 * 64 monochrome images to fit the CNN model. Then split the original dataset into train dataset to train the model and validation dataset to get holdout error. Also, apply one-hot encoding to label set, which could help the machine learning model to predict.

Second part is the CNN model. Sequential API in Keras package provides a dummy way to apply the convolutional layer to the CNN model. For machine learning, deep is better. So, I choose three set of layers and each set has three layers, two convolutional layers and one pooling layer (see fig 1). At first set, I applied 32 filters and 5 * 5 kernel, because each image is quite large and the side pixels are not helpful to do prediction, large kernel size could reduce computing load rapidly. At last two layer sets, I applied 64 filters and 3 * 3 kernel, which could extract useful features more precisely from the train dataset. Pooling layer picks the larger value from adjoin pixels, which could increase the computing efficiency and decrease the overfitting. At the last of the model is flatten layer and fully—connected layers.

Third part is setting several parameters like learning rate reduction. According to the Sebastian Ruder's article, An overview of gradient descent optimization algorithms(http://ruder.io/optimizing-gradient-descent/). I choose Adam as the model optimizer and a learning rate annealing schedule as the learning rate reduction. The parameter in these functions are token from their documents. About data augmentation, according to the Kaggle result, model with data augmentation has higher accuracy, however in my test train, model without data augmentation has higher accuracy (see fig 2). Steps usually related to the batch size, steps * batch size = train set instances. Epochs, larger is better, although the accuracy increases slowly after 6 epochs (see fig 4).

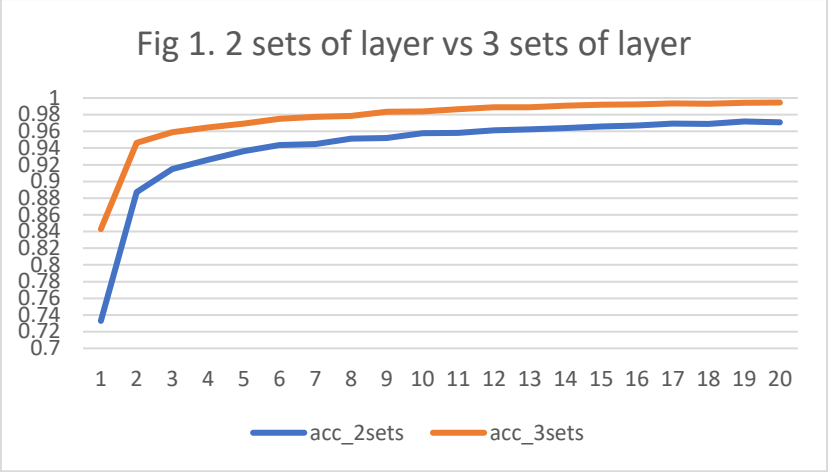
Fig 1. 2 sets of layer vs 3 sets of layer
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0.8
0.78
0.76
0.74
0.72
0.7
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
acc_2sets
acc_3sets

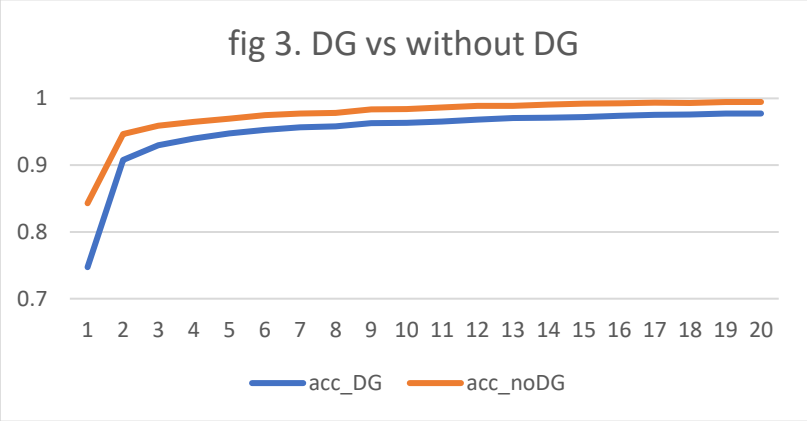
fig 3. DG vs without DG
1
0.9
0.8
0.7
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
acc_DG
acc_noDG

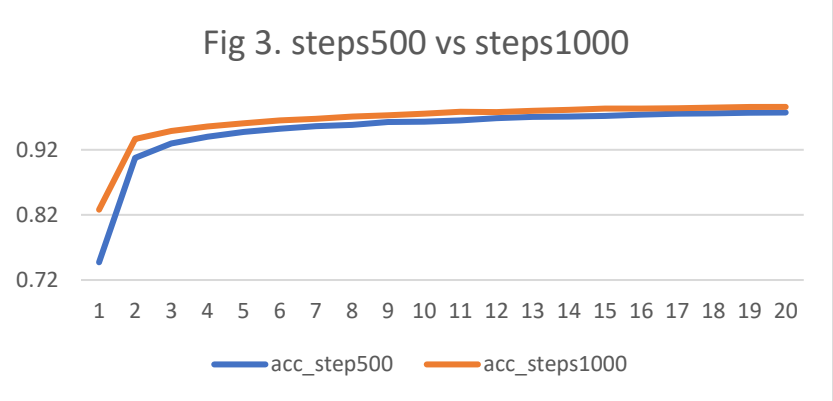
Fig 3. steps500 vs steps1000
0.92
0.82
0.72
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
acc_step500
acc_steps1000

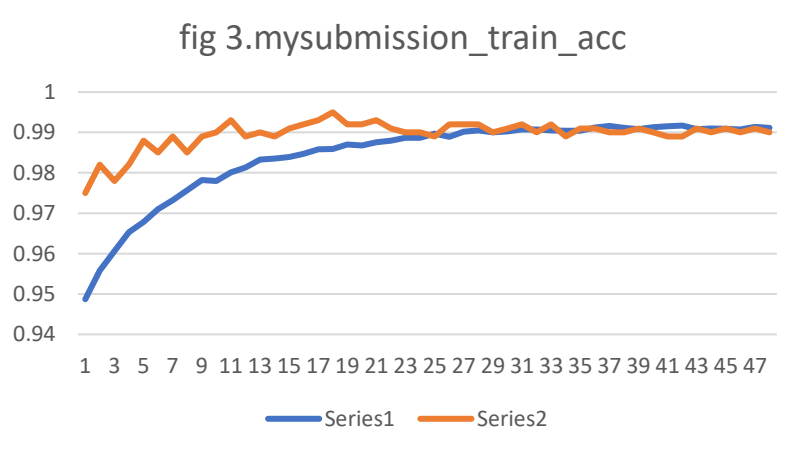
fig 3.mysubmission_train_acc
1
0.99
0.98
0.97
0.96
0.95
0.94
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
Series1
Series2