Student ID: R05944004

Name: Cheng-Wei Wu

hw 3
Image Classification
Report

Q1

In this supervised-learning part, I split the labeled data into two parts, the training set and the validation set. I picked 50 images from each class to be the validation data. Then train the CNN by training data.

CNN Structure:

Convolution → Activation(PReLu) → MaxPolling → SpatialDropout → BatchNormalization →
Convolution → Activation(PReLu) → AveragePooling → Dropout → BatchNormalization →
Convolution → Activation(PReLu) → AveragePooling → Dropout → BatchNormalization →
Flatten → Dense → Activation(tanh) → Dropout → BatchNormalization → Dense → Activation(softmax)
Optimizer: Adamgrad

Then its performance is about **64% ~ 66%** (based on the random pick of validation)

Q2

To increase the performance of the model trained by few images, I used "ImageDataGenerator" to scale, rotate or translate the images to produce "more" training data. And this model's accuracy is about 65% to 68%. Then I used this model to predict the "label" of un-labeled data. In each iteration, I predicted the un-labeled data and add the one whose confidence is higher than the probability threshold to the training set. After repeating such process few times, I could get the better model than in Q1. The structure of the initial model is the same with the re-train one.

CNN Structure (for middle model):

Convolution → Activation(PReLu) → MaxPolling → SpatialDropout → BatchNormalization →
Convolution → Activation(PReLu) → AveragePooling → Dropout → BatchNormalization →
Convolution → Activation(PReLu) → AveragePooling → Dropout → BatchNormalization →
Flatten → Dense → Activation(tanh) → Dropout → BatchNormalization → Dense → Activation(softmax)
Optimizer: Adamgrad

Then the final performance is about **68%~73%** (based on the random chose data)

Q3

I use de-noising auto-encoder and k-means algorithm to assist another semi-supervised learning method. At first, I trained a CNN auto-encoder model. And before the training process, I add some noise to the original images to increase the performance of reconstructing the images. The CNN structure is shown below:

CNN Structure:

Convolution → Activation(ReLu) → MaxPolling → Convolution → Activation(ReLu) → MaxPolling →

Convolution → Activation(ReLu) → MaxPolling (encoded layer end) →
Convolution → Activation(ReLu) → UpSampling → Convolution → Activation(ReLu) → UpSampling →
Convolution → Activation(ReLu) → UpSampling → Convolution → Activation(tanh)
Optimizer: Adamgrad

After training the auto-encoder, I could use its encoder part to encode the images to be some codes which may represent or store the important information of the original images. Then I encoded both the training data and test data to be an array of values(codes). Next, I used K-means algorithm to cluster the encoded training data, and tried to use the K-means model to predict the encoded test data. Since the index (class) of the K-means algorithm may not the same with the original images' real class index, I selected the index which appears the most frequently in each class (500 images) to be the real class index's "representation" in the K-means algorithm's clustering index.

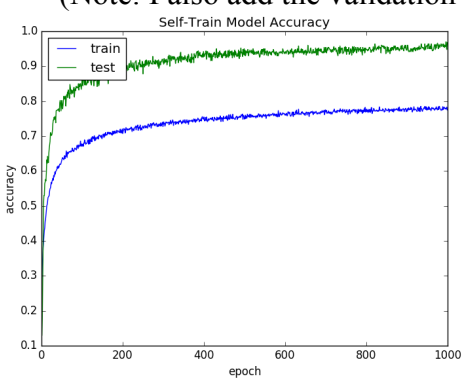
Then the final performance is only about **19%~21%** (based on the random chose data)

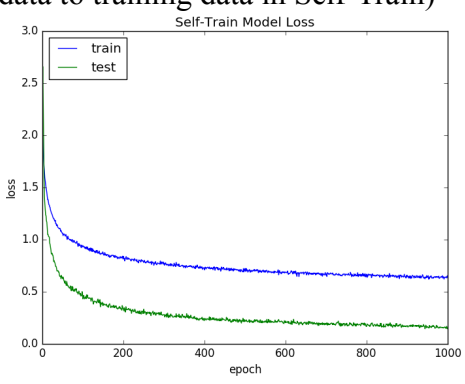
- # Q4

Since the size of Q1's training set is very small, it can't get good performance even it use the convolutional neural network. However, in Q2, to increase the basic performance of initial model, I used generator to produce more images to train the model. The reason why I want to increase its performance is because I think that if I could get better performance in initial model, I can predict the un-label data with higher accuracy which will also increase the final accuracy if I us these "labeled data" to train on the CNN model. Fortunately, it works as my expected. Although there may exist some error prediction, overall speaking, However, when I implement the auto-encoder (Q3), it get very low frequency (around 20%). I think the main reasons of this bad results could be the two, one is that the auto-encoder is not well implemented on this dataset, which means that its CNN structure or the pre-processing of the images is not suitable for these images. And the second reason is that since I used K-means algorithm to cluster the codes encoded by auto-encoder; nevertheless, each cluster's index is not the same with the real class index of the images belong to the cluster, For example, the image with real class index 7 is clustered to the cluster 1 in K-means. Worst of all, the error clustered images are not considered into the previous discussion. Based on these reasons, I think that the very low accuracy of auto-encoder method could have acceptable explanation. The last but not the least, to avoid the overfitting of the training data, I also adopt the dropout technique in my CNN structure. Also, the spatial-Dropout (for area wise dropout), batch-normalization (fix the output value of activation function) and average-pooling (reduce the effect of border noise) are included in my CNN structure to increase the model's accuracy.

- # Figures

(Note: I also add the validation data to training data in Self-Train)





(Auto-Encoder Model Loss)