School of EECS

CS519: Deep Learning

#3 Keras CIFAR-10 Image Classification

Professor: Fuxin Li

Kaibo Liu * 932-976-427 *

Mar. 06th, 2017

Question and schedule descriptions

1. Remove the dropout layer after the fully-connected layer. Save the model.
   Layer structure(Fig.1) and **loss/error curves**(Fig.2,3) are shown in this section.

2. Load the model from Q.1 as initialization to the training. Add another fully connected layer with 512 filters at the second-to-last layer (before the classification layer). Train and save the model.
   Layer structure(Fig.4) and **loss/error curves**(Fig.5,6) are shown in this section.

3. Try to use an adaptive schedule **Adam** to tune the learning rate.
   Loss/error curves(Fig.7,8) are shown in this section.

4. There are 5 ways of tunning for the model to make the validation error better.

   4.1. Tune the network from random initializations with the layer structure of Q.3, and in the following way:

   - Add dropout layer after both fully-connected layers dense(512).
   - Change the activation function for both fully-connected layers dense(512) from ReLu to Sigmoid.

   Layer structure(Fig.9) and **loss/error curves**(Fig.10,11) are shown in this section.

   4.2. Tune the network from random initializations with the layer structure of Q.4.1, and in the following way:

   - Change the number of hidden units of the second fully-connected layer from dense(512) to dense(256), before dense(10).

   Layer structure(Fig.12) and **loss/error curves**(Fig.13,14) are shown in this section.

   4.3. Tune the network from random initializations with the layer structure of Q.4.2, and in the following way:

   - Add local normalization before each activation function.

   Layer structure(Fig.15) and **loss/error curves**(Fig.16,17) are shown in this section.

   4.4. Tune the network from random initializations with the layer structure of Q.4.3, and in the following way:

   - Remove AveragePooling2D, then add a convolution block with 128 filters as a 3rd conv.

   Layer structure(Fig.18) and **loss/error curves**(Fig.19,20) are shown in this section.

   4.5. Tune the network from random initializations with the layer structure of Q.4.4, and in the following way:

   - Using data augmentation.

Loss/error curves(Fig.21,22) are shown in this section.

The summary for Question 4 is listed in Tab.1. All training is deployed in:

```
Keras-1.2.2
Theano-0.8.2
Python-2.7
GPU-TITAN X (Pascal)
```

Table 1: Tunning schema for Q.4

# of Q	tunning schema	time/ep	error	val_error
4.1	3+ReLu $\rightarrow$ Sig and dropout, after both dense(512)	7s	0.262	0.261
4.2	4.1+ 2^{nd} dense(512) $\rightarrow$ dense(256)	7s	0.267	0.266
4.3	4.2+local norm before each activation	52s	0.213	0.243
4.4	4.3+remove AveragePooling, add conv(128)	170s	0.073	0.152
4.5	4.4+data augmentation	170s	0.107	0.125

Model tunning with Keras for CIFAR-10

Q1 Remove the dropout layer after the fully-connected layer

According to the requirement, I remove the dropout layer, then train and save the model. To be more clear and intuitive, the layer structure is shown in Fig.1 with highlighted update from the previous model. Error curves for both training and validation data are shown in Fig.2, loss curves for both training and validation data are shown in Fig.3).

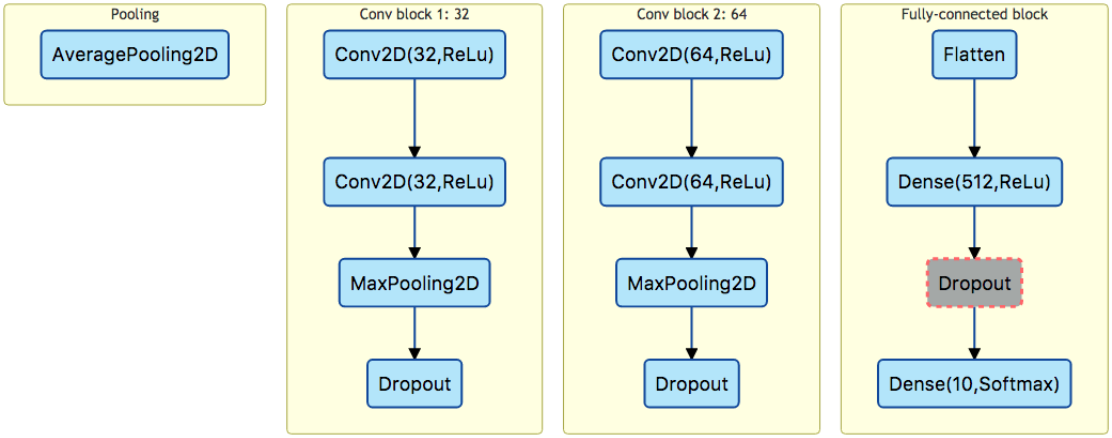

Figure 1: Layer structure for Q.1, new tune highlighted

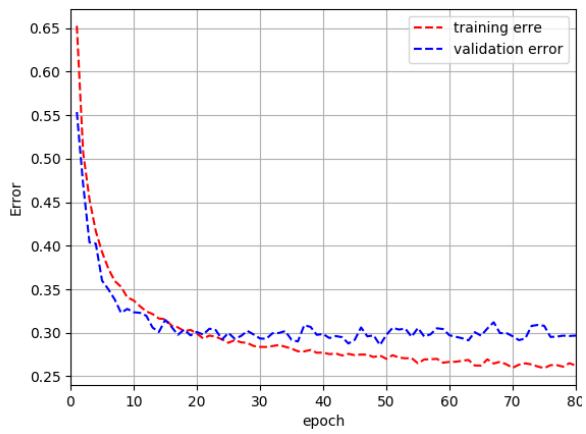

Figure 2: Error against epoch for Q.1

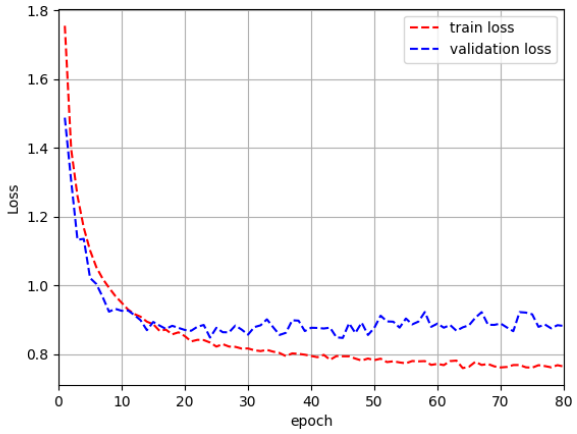

Figure 3: Loss against epoch for Q.1

It's reasonable that both error and loss for training/validation data with respect to epoch are descent and get converge. But dropout is used to prevent over-fitting, removing which will make training data a much more better result than validation data.

Q2 Load moede_1 and add fully connected layer

According to the requirement, I load the model from Q.1 as initialization to the training, add another fully connected layer with 512 filters at the second-to-last layer (before the classification layer), then train and save the model. To be more clear and intuitive, the layer structure is shown in Fig.4 with highlighted update from the previous model. Error curves for both training and validation data are shown in Fig.5, loss curves for both training and validation data are shown in Fig.6).

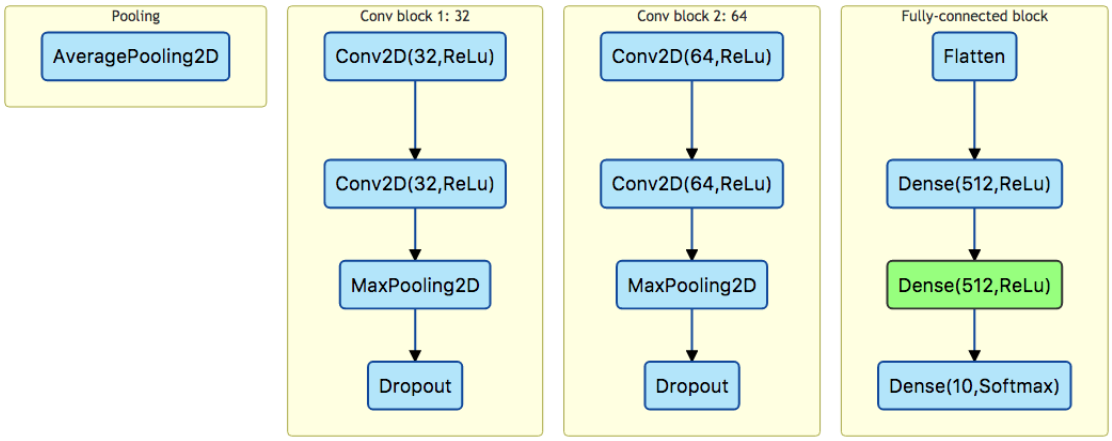

Figure 4: Layer structure for Q.2, new tune to Q.1 highlighted

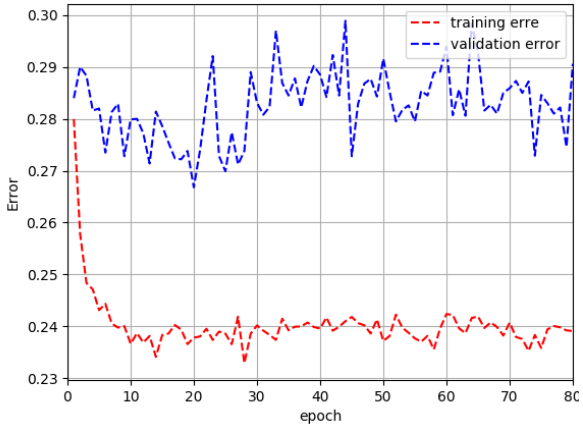

Figure 5: Error against epoch for Q.2

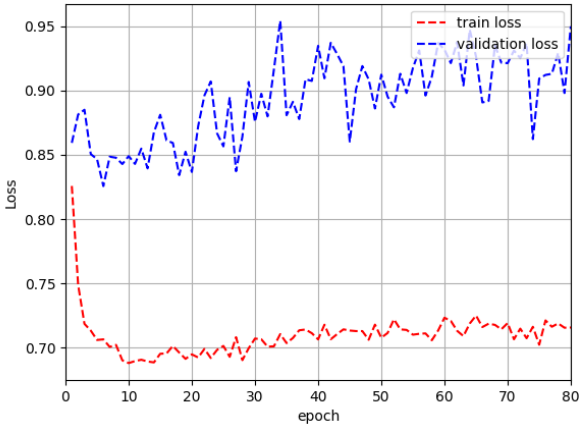

Figure 6: Loss against epoch for Q.2

Following the data of pre-trained model from Q.1, both error and loss for training data still have the trend of dropping at first, then start to turbulent like validation curves. It's easy to say that adding a fully-connected layer will make training date more fitting but do no improvement to validation data.

Q3 Tune the learning rate with adaptive schedule

In this section, I used adam as the optimizer in compiling the model and started from random initialization. The model structure is the same as Q.2.

Listing 1: Adaptive Moment Estimation(Adam) as optimizer

```
adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1
    e-08)
model.compile(loss='categorical_crossentropy', optimizer=
    adam, metrics=["accuracy"])
```

Error curves for both training and validation data are shown in Fig.7, loss curves for both training and validation data are shown in Fig.8).

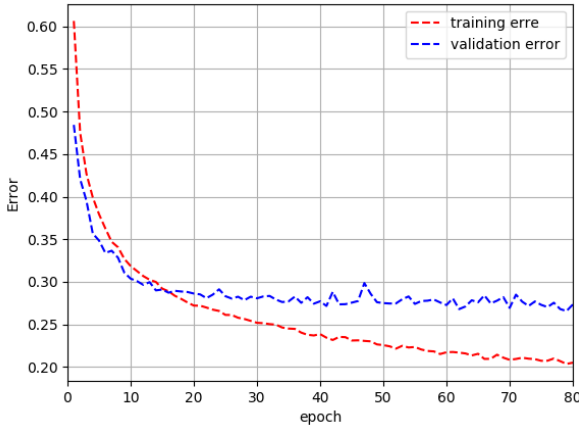

Figure 7: Error against epoch for Q.3

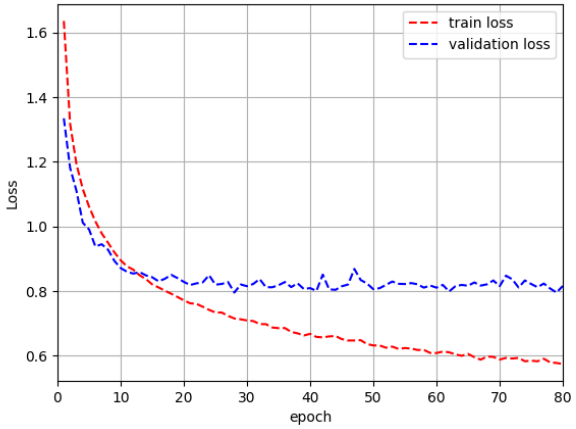

Figure 8: Loss against epoch for Q.3

Using Adam as the optimizer makes both training error and training loss much lower since the learning rate is based on adaptive moment estimation. On the other hand, validation error and validation loss are not improved much. Adam makes the training more fitting than validation, which we can see both figures have the diverse of curve trends.

Q4.1 Add dropout layer and change activation function for both fully-connected layers

In this part, I tune the network from random initializations with the layer structure of Q.3, and in the following way:

- Add dropout layer after both fully-connected layers dense(512).
- Change the activation function for both fully-connected layers dense(512) from ReLu to Sigmoid.

To be more clear and intuitive, the layer structure is shown in Fig.9 with highlighted update from the previous model. Error curves for both training and validation data

are shown in Fig.10, loss curves for both training and validation data are shown in Fig.11).

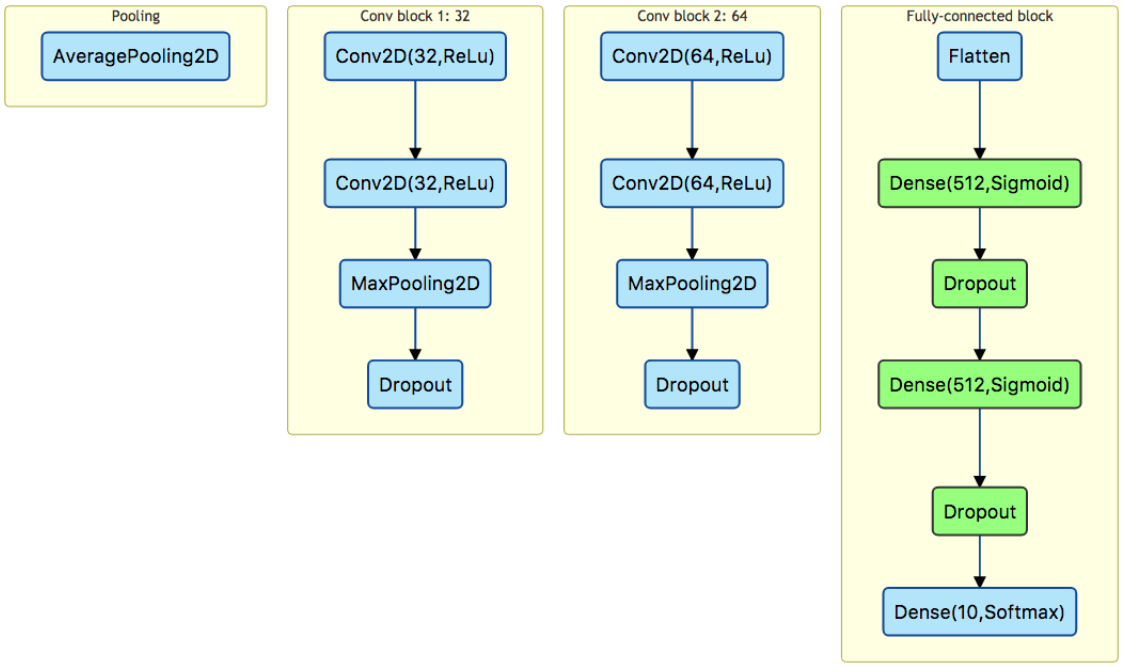

Figure 9: Layer structure for Q.4.1, new tune to Q.2/Q.3 highlighted

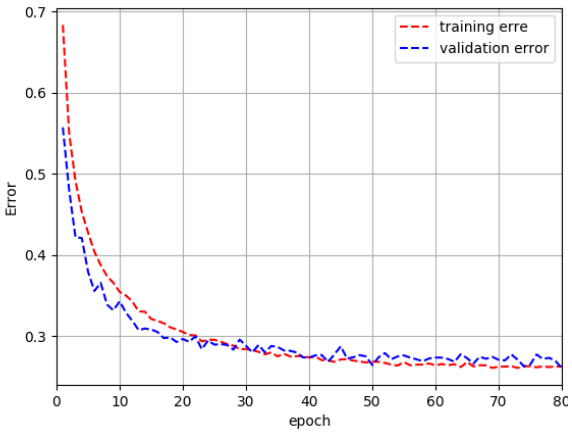

Figure 10: Error against epoch for Q.4.1

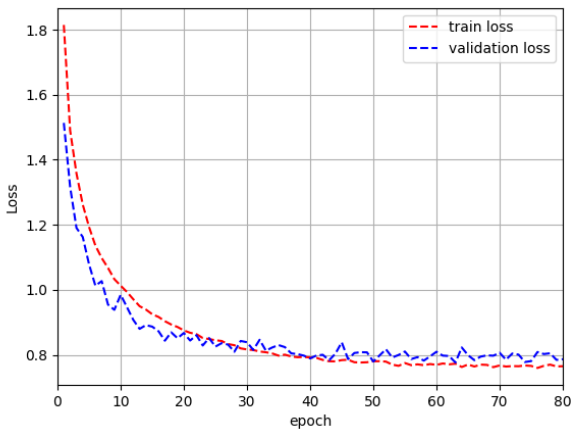

Figure 11: Loss against epoch for Q.4.1

Dropout is used to prevent overfitting in training data. We can see the training error and training loss are more consistent to validation data, making themselves a little higher. But dropout does not help improve validation results.

ReLu activation function lost some information compared to Sigmoid activation function. So changing activation function for from ReLu to Sigmoid for both fully-connected layers dense(512) would bring back higher accuracy and lower loss for both training and validation data.

4.2 Tune hidden units

In this part, I tuned the network from random initializations with the layer structure of Q.4.1, and in the following way:

- Change the number of hidden units of the second fully-connected layer from dense(512) to dense(256), before dense(10).

To be more clear and intuitive, the layer structure is shown in Fig.12 with highlighted update from the previous model. Error curves for both training and validation data are shown in Fig.13, loss curves for both training and validation data are shown in Fig.14).

Compared to Q.4.1, the curve results seems little improved, with only update of the

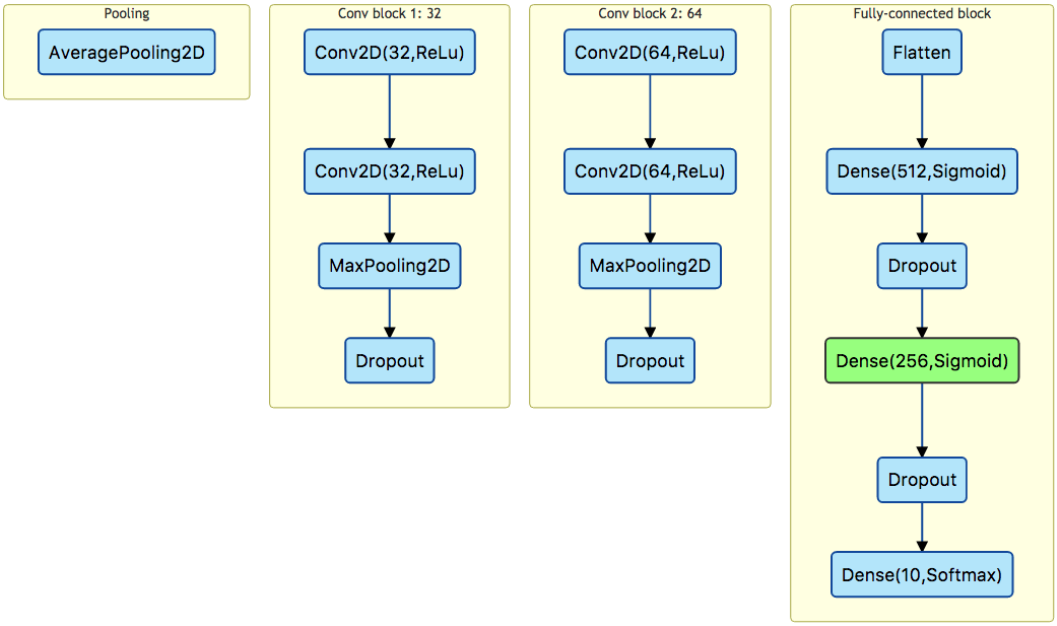

Figure 12: Layer structure for Q.4.2, new tune to Q.4.1 highlighted

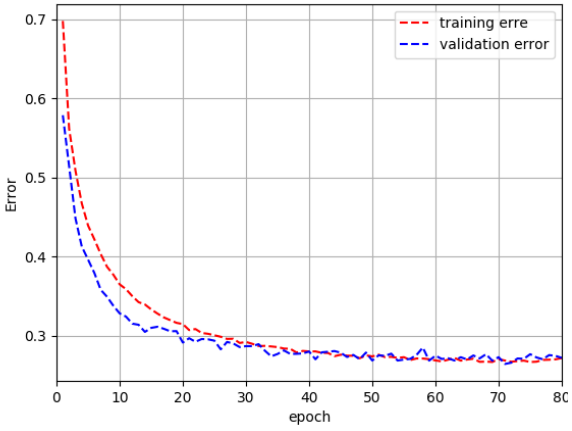

Figure 13: Error against epoch for Q.4.2

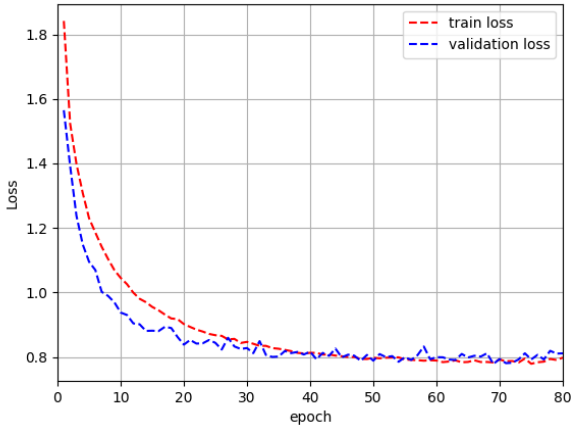

Figure 14: Loss against epoch for Q.4.2

hidden units in second fully-connected layer. Since the second one after first dropout only makes the curves more smooth with a progressive decrement.

4.3 Add local normalization before activation

In this part, I tuned the network from random initializations with the layer structure of Q.4.2, and in the following way:

- Add local normalization before each activation function.

Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

Listing 2: Local normalization before each activation function

```
model.add(BatchNormalization())
model.add(Activation('relu'))
```

To be more clear and intuitive, the layer structure is shown in Fig.15 with highlighted update from the previous model. Error curves for both training and validation data are shown in Fig.16, loss curves for both training and validation data are shown in Fig.17).

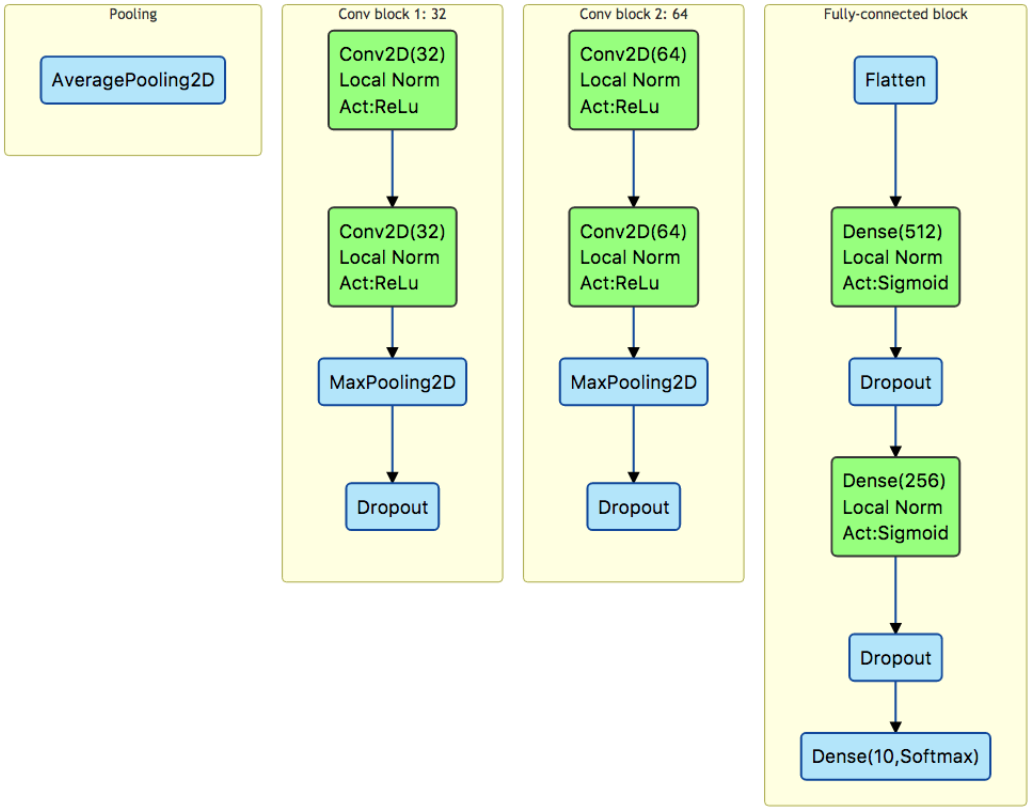

Figure 15: Layer structure for Q.4.3, new tune to Q.4.2 highlighted

From the curves of Fig.16 and Fig.17, we see that both error and loss of training/ validation data drop to a much better result. Because it will make things invariant of the scale of weights.

It's almost a trend now to have a Conv2D followed by a ReLu followed by a BatchNormalization layer.

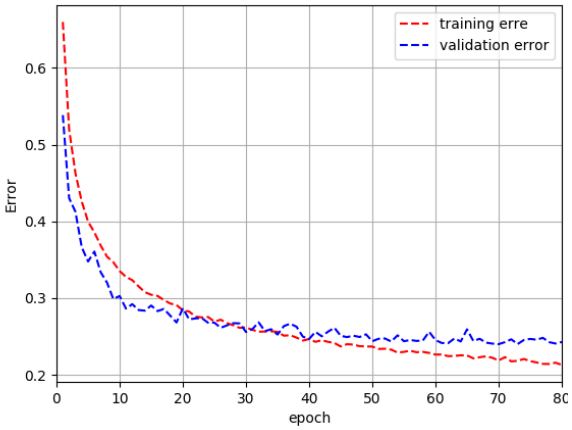

Figure 16: Error against epoch for Q.4.3

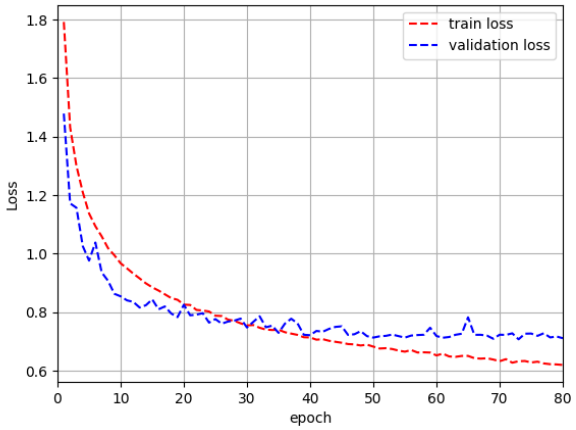

Figure 17: Loss against epoch for Q.4.3

4.4 Add a third convolution block

In this part, I tuned the network from random initializations with the layer structure of Q.4.3, and in the following way:

- Remove AveragePooling2D.
- Add a convolution block with 128 filters as a 3rd conv.

Since the image dimension is 32×32, each pooling(averagePooling2D or MaxPooling2D) will shrink the size into half for both dimension. So after 3 pooling, the image is divided into a 4×4 sub-feature, which cannot do convolution/pooling any more with a 3×3 filter. So I remove the first Average2D and add a third Convolution2D(128) in to the model.

Listing 3: Add a third Convolution2D layer

```
model.add(Convolution2D(128, 3, 3, border_mode='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Convolution2D(128, 3, 3))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

To be more clear and intuitive, the layer structure is shown in Fig.18 with highlighted update from the previous model. Error curves for both training and validation data are shown in Fig.19, loss curves for both training and validation data are shown in Fig.20).

The third convolution2D layer makes the training error break down to 0.1. This is reasonable because more filters decompose the images into more detailed factors to offer more features and more information, which will make training more deep and effective. Of course we cannot forget the contribution from local normalization.

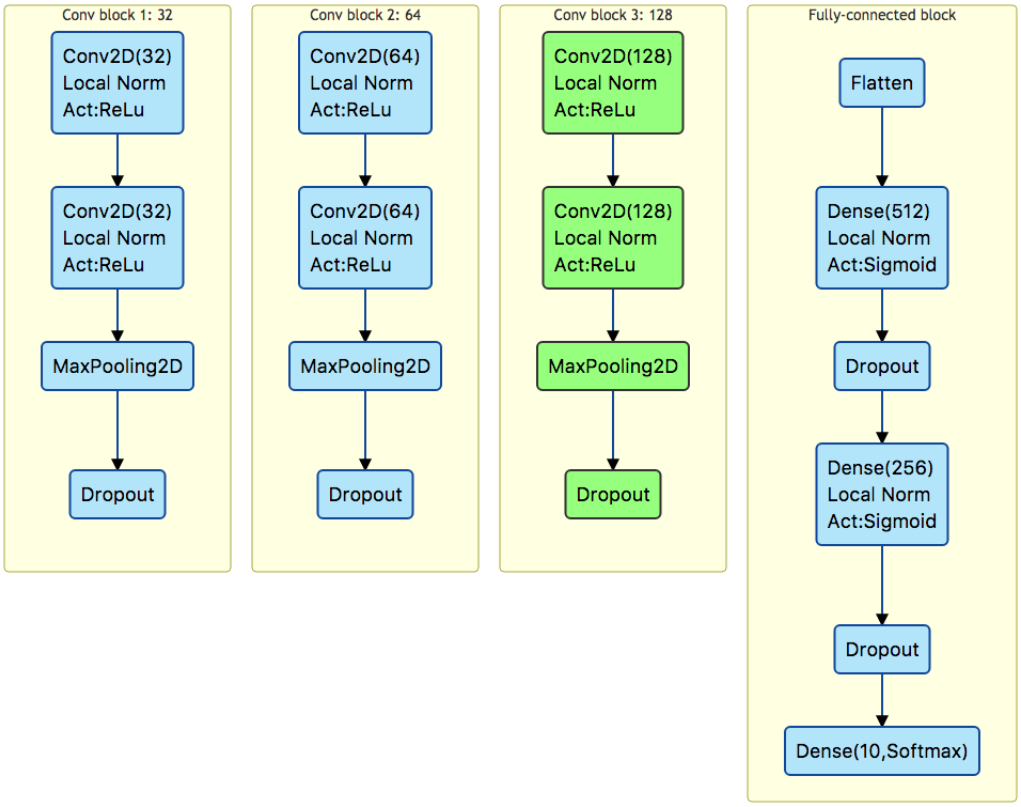

Figure 18: Layer structure for Q.4.4, new tune to Q.4.3 highlighted

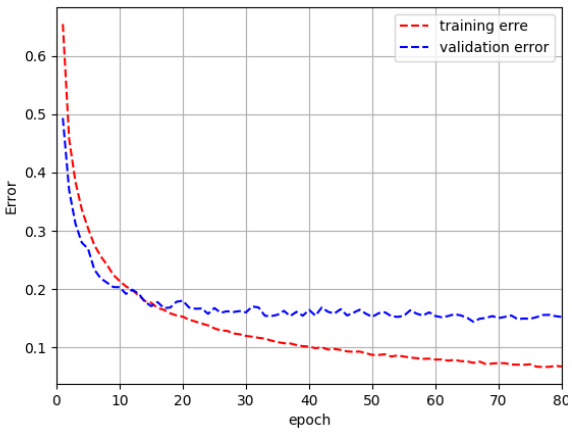

Figure 19: Error against epoch for Q.4.4

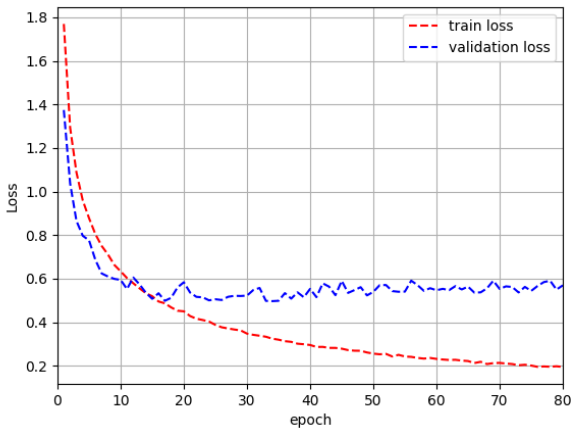

Figure 20: Loss against epoch for Q.4.4

4.5 Use data augmentation

In this part, I tuned the network from random initializations with the same model structure as Q.4.4, and in the following way:

- Using data augmentation.

Error curves for both training and validation data are shown in Fig.21, loss curves for both training and validation data are shown in Fig.22).

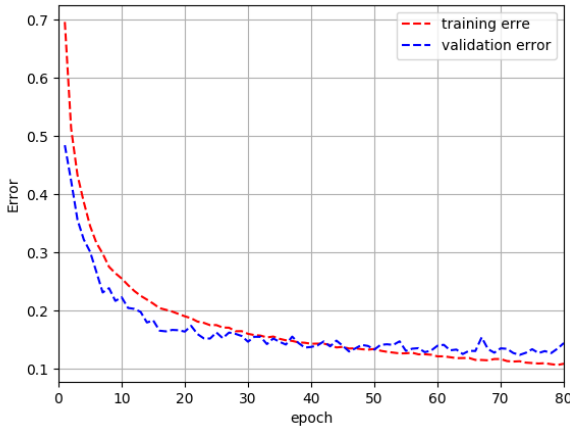

Figure 21: Error against epoch for Q.4.5

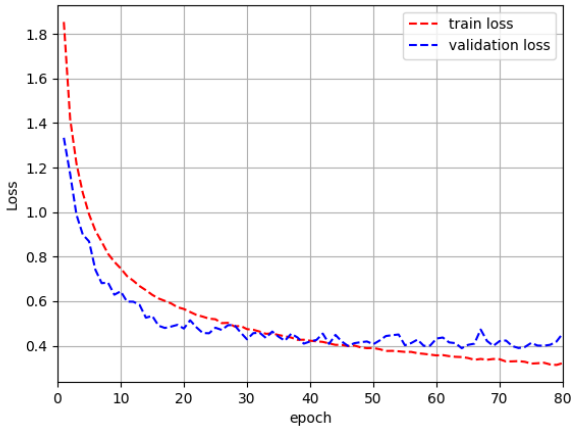

Figure 22: Loss against epoch for Q.4.5

Compared to the result of Q4.4, data augmentation helped extend my lower limit of validation error and validation loss. The result is reasonable that data augmentation is the way used to enlarge the dataset, to prevent overfitting in training data, and to make the training model more robust with more data examples. So it's natural that I got a better validation error, along with a relatively high training error, which are more close to each other.

4.6 Summary for Question 4

Fig23 shows the progressive improvement from Q4.1 to Q4.5 in terms of validation error (FC is short for fully-connected).

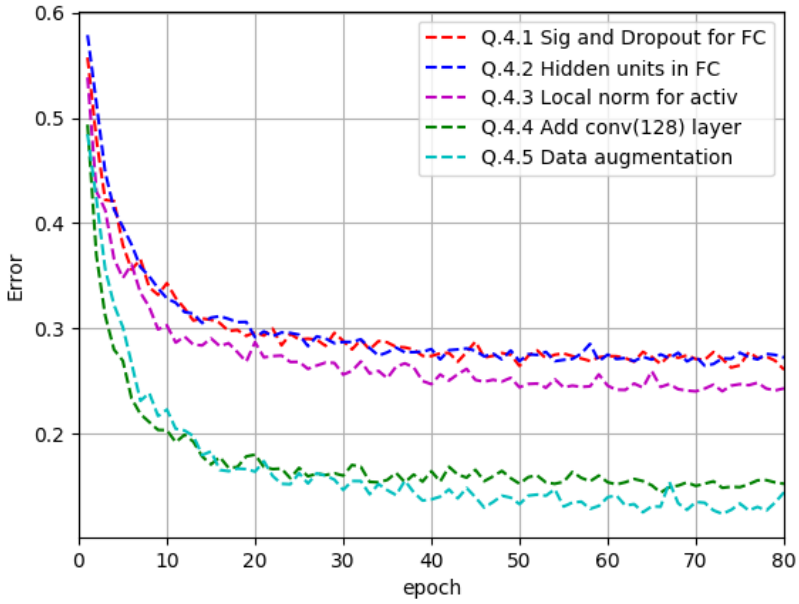

Figure 23: Validation error against epoch for all schedules in Question 4