Deep Learning – Classifier for Cifar-10 Report

First Try:

Network Architecture:

In our first try, we built a neural network based on the 'nin' file from the Cifar Torch blog. In order to get up to 50,000 parameters; we minimized the number of filters in each layer and deleted some of the layers. Finally, the network was consisted of the following layers:

1. Nn.SpatialConvolution(3,64,5,5,1,1,2,2)
2. Nn.SpatialConvolution(64,32,1,1)
3. Nn.SpatialConvolution(32,32,1,1)
4. model:add(nn.SpatialMaxPooling(3,3,2,2):ceil())
5. model:add(nn.Dropout())
6. nn.SpatialConvolution(32,32,5,5,1,1,2,2)
7. nn.SpatialConvolution(32,64,1,1)
8. nn.SpatialConvolution(64,32,1,1)
9. model:add(nn.SpatialAveragePooling(3,3,2,2):ceil())
10. model:add(nn.Dropout())
11. nn.SpatialConvolution(32,32,3,3,1,1,1,1)
12. nn.SpatialConvolution(32,32,1,1)
13. nn.SpatialConvolution(32,10,1,1)
14. model:add(nn.SpatialAveragePooling(8,8,1,1):ceil())
15. model:add(nn.View(10))

After each SpatialConvolution layer, we added a BatchNormalization layer, performing normalization for each training mini-batch and then add a ReLu layer.

Initialization:

We normalized the neurons in each convolutional layer (Ignoring 'ReLu') using normal distribution with a mean of 0 and variance of 0.05.

Normalization:

The final step in our data processing was data normalization with a mean of 0.0 and standard deviation of 1.0 for each image channel.

Data Augmentation:

In order to get better results we tried to do data augmentation, by adding a batch flip layer to model architecture.
The data augmentation method we used in the first try was 'hflip' (flips image horizontally) which we called with probability of o.5.

Optimization:

We used the 'Adam' optimizer with NLL criterion. The learning rate, Momentum and Weight Decay were set with default values.

Tunable Parameters:

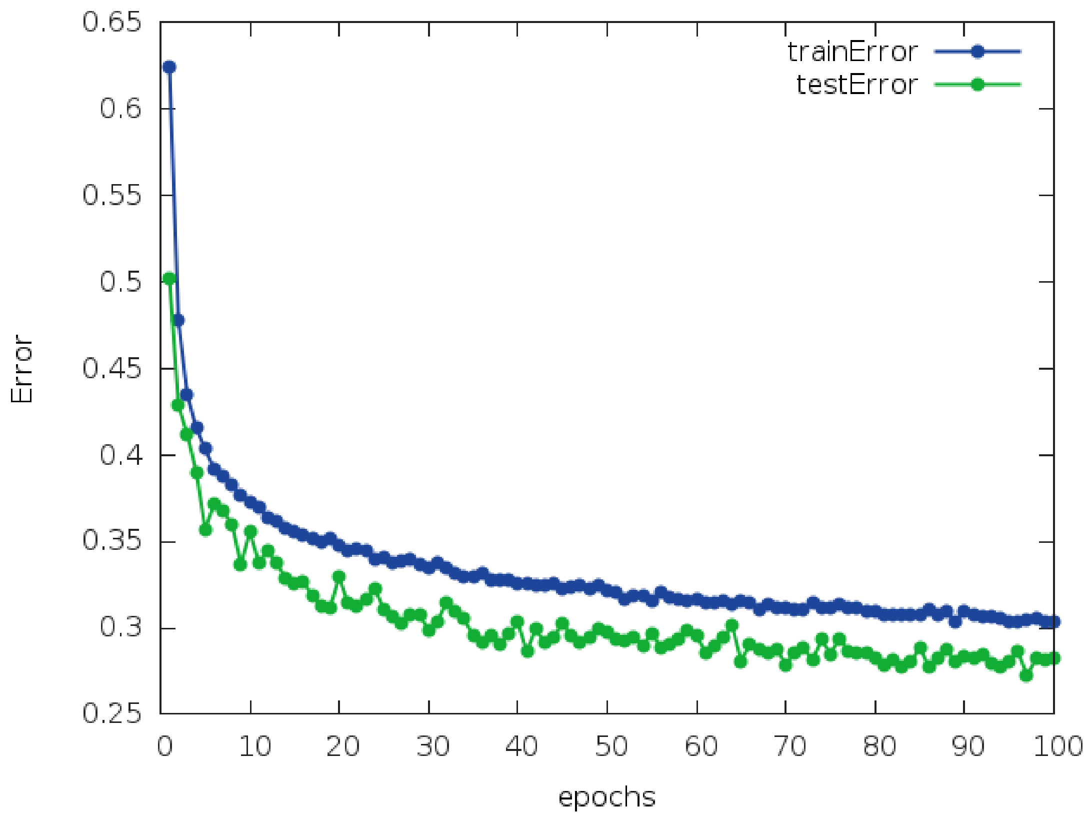
We ran the training method over 100 epochs with batch size of 32.

Best Result:

Accuracy: 74.32%

Best Epoch: 98

Train Error and Test Error VS Epochs Graph:



Second Try - Best Try:

Network Architecture:

No Change.

Initialization:

No Change.

Normalization:

No Change.

Data Augmentation:

We did not used any data augmentation method.

Optimization:

This time we used SGD (stochastic gradient descent) with cross entropy loss with learning rate 1, momentum 0.9 and weight decay 0.0005.
In addition, we add a line in our training method, dropping learning rate every 25 epochs.

Tunable Parameters:

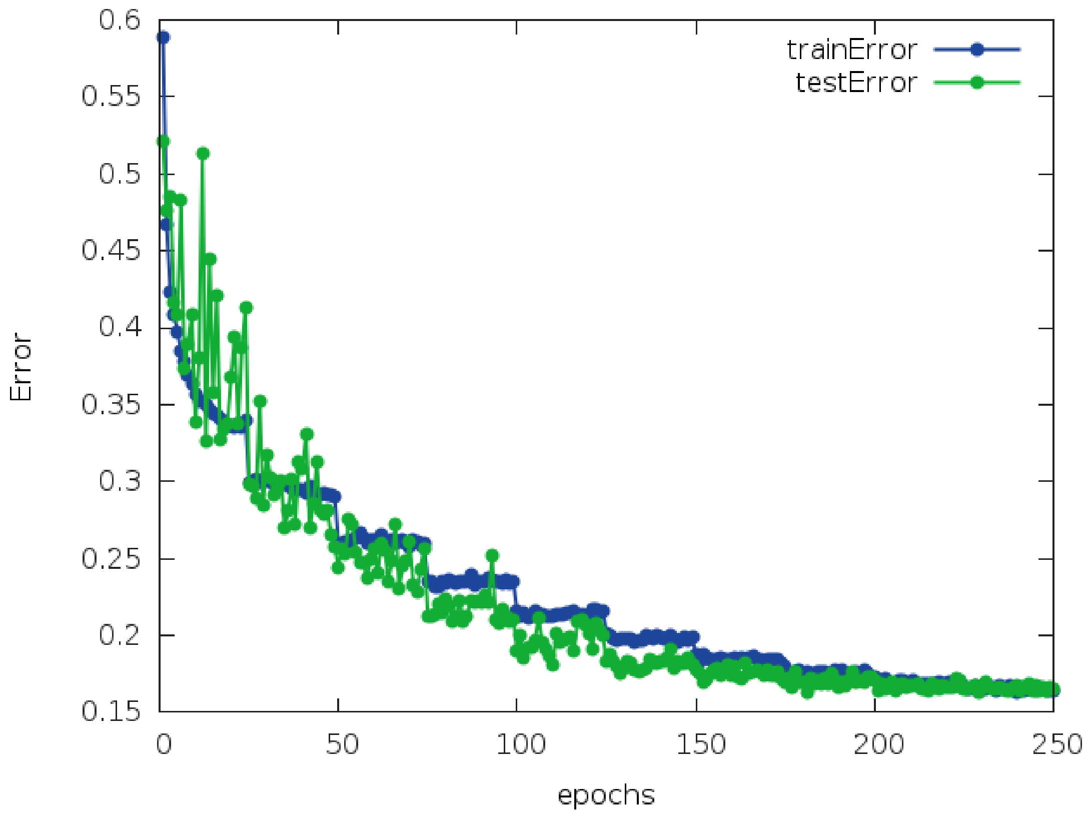
We changed the batch size to 64 and number of epochs to 250.

Best Result

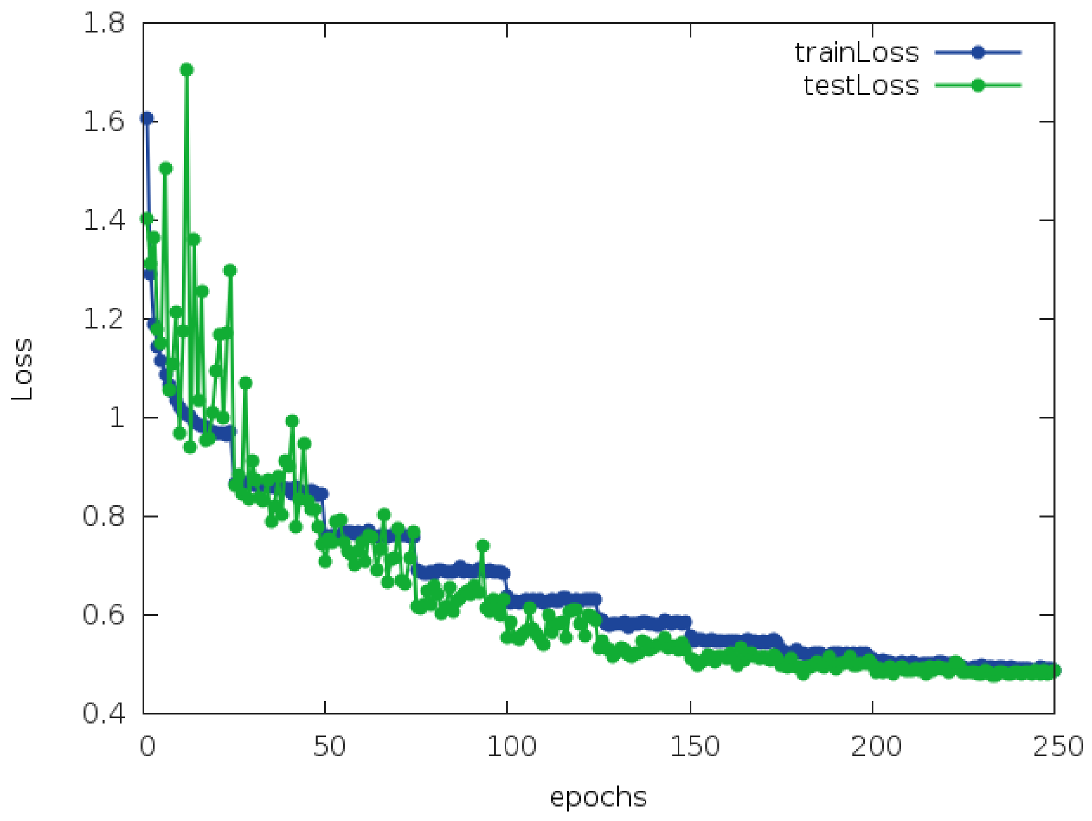
Accuracy: 83.5336%

Best Epoch: 250

Train Error and Test Error VS Epochs Graph:



Train Loss and Test Loss VS Epochs Graph:



Third Try:

Network Architecture:

In this try, we used one fully connected layer in the end of the neural network.

Rest of the neural network included other types of layers, as the following:

```
local model = nn.Sequential()

model:add(nn.BatchFlip())

model:add(nn.SpatialConvolution(3, 64, 5, 5))

model:add(nn.SpatialMaxPooling(2,2,2,2))

model:add(nn.ReLU(true))

model:add(nn.Dropout(0.5))

model:add(nn.SpatialBatchNormalization(64))

model:add(nn.SpatialConvolution(64, 24, 3, 3))

model:add(nn.SpatialMaxPooling(2,2,2,2))

model:add(nn.ReLU(true))

model:add(nn.Dropout(0.5))
```

```
model:add(nn.SpatialBatchNormalization(24))
model:add(nn.SpatialConvolution(24, 32, 3, 3))
model:add(nn.ReLU(true))
model:add(nn.Dropout(0.5))
model:add(nn.SpatialBatchNormalization(32))
model:add(nn.SpatialConvolution(32, 32, 3, 3, 1, 1, 1, 1))
model:add(nn.ReLU(true))
model:add(nn.Dropout(0.5))
model:add(nn.SpatialBatchNormalization(32))
model:add(nn.SpatialConvolution(32, 32, 3, 3, 1, 1, 1, 1))
model:add(nn.View(32*4*4):setNumInputDims(3))
model:add(nn.Linear(32*4*4, #classes))
model:add(nn.LogSoftMax())
```

Initialization:

No Change.

Normalization:

No Change.

Optimization:

No Change.

Data Augmentation:

In our third try, we used the 'Polar' function as our data augmentation method, which converts image source to polar coordinates. We set the mode parameter given as an input to 'full' mode, means that the full image is converted to the polar space (implying empty regions in the polar image).

Tunable Parameters:

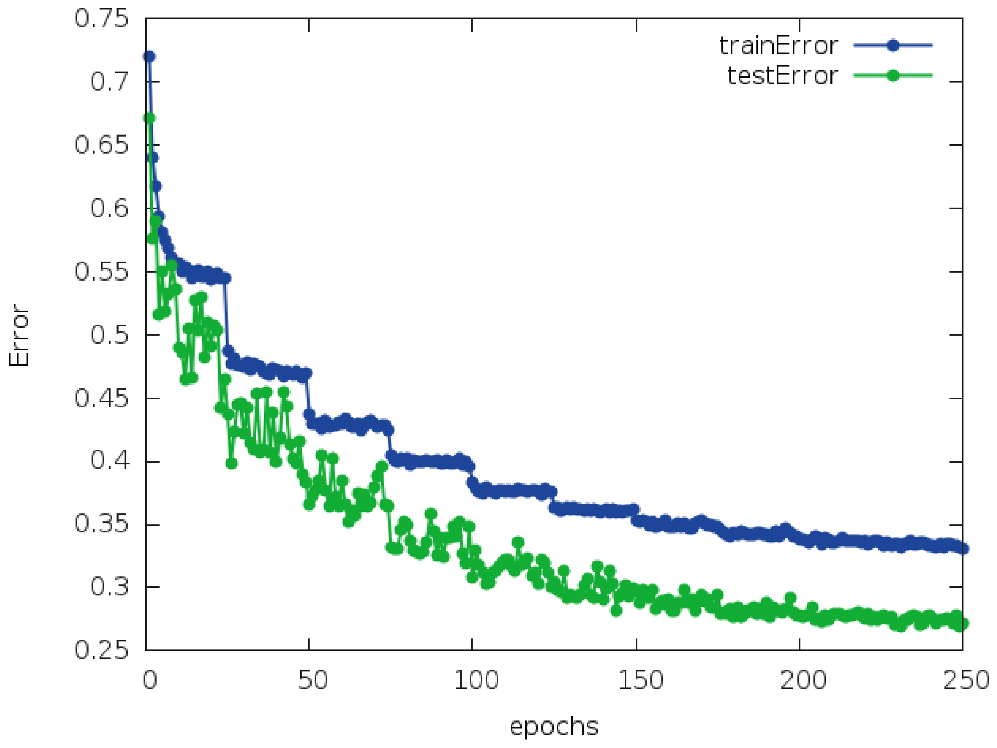
No Change.

Best Result

Accuracy: 72.84655%

Best Epoch: 250

Train Error and Test Error VS Epochs Graph:



Fourth Try:

Network Architecture:

Same as try number two, since it received the best accuracy so far.

Initialization:

No Change.

Normalization:

No Change.

Data Augmentation:

In our fourth try, we used the 'Log Polar' function as our data augmentation method, which converts image source to log polar coordinates. In the log-polar image, angular information is in the vertical direction and log-radius information in the horizontal direction. We set the mode parameter given as an input to 'valid' mode, means that only valid central part of the image is log-polar transformed.

Optimization:

No Change.

Tunable Parameters:

Batch size: 64.

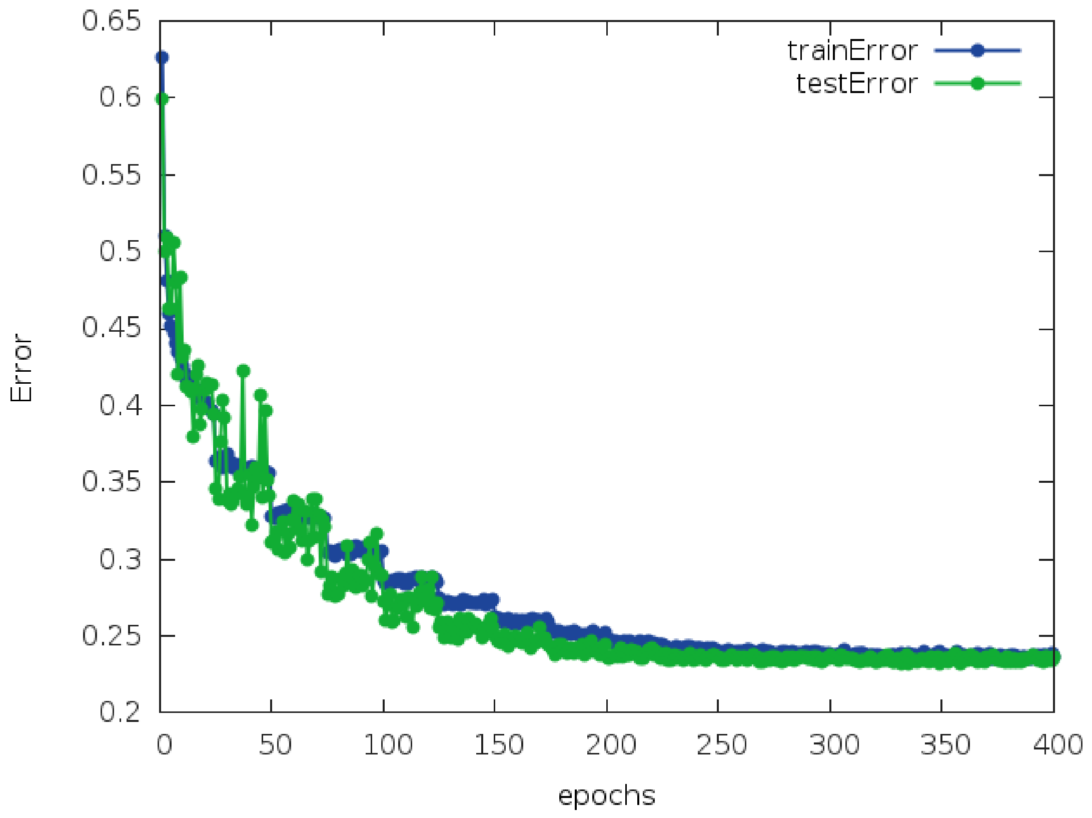
Number of Epochs: 400.

Best Result

Accuracy: 76.6326%.

Best Epoch: 385.

Train Error and Test Error VS Epochs Graph:



Summary:

The best neural network model is as written on try number two, since it gives the best accuracy on test set. It does not include fully connected layers, neither data augmentation. The optimizer is SGD with Cross Entropy Criterion. The batch size is 64 and number of epochs is 250.

GitHub Link:

https://github.com/hanama/Deep_Learning/tree/master/DL%20-%20hw2