Deep Learning - HW2 Report

Nadav Oved - 200689768
Aviv Sugarman - 305652729

Github Link: https://github.com/nadavo/DeepLearningCourse/tree/hw2

Models path: `/home/nadavo@st.technion.ac.il/hw2_models/<optimizer_data-augment>/network.model`

Model Architecture

35 layer convolutional neural network built according to the following structure:

```
  (1): nn.BatchFlip
  (2): nn.SpatialConvolution(3 -> 128, 5x5, 1,1, 2,2)
  (3): nn.SpatialBatchNormalization (4D) (128)
  (4): nn.ReLU
  (5): nn.SpatialConvolution(128 -> 64, 1x1)
  (6): nn.SpatialBatchNormalization (4D) (64)
  (7): nn.ReLU
  (8): nn.SpatialConvolution(64 -> 32, 1x1)
  (9): nn.SpatialBatchNormalization (4D) (32)
  (10): nn.ReLU
  (11): nn.SpatialConvolution(32 -> 16, 1x1)
  (12): nn.SpatialBatchNormalization (4D) (16)
  (13): nn.ReLU
  (14): nn.SpatialMaxPooling(3x3, 2,2)
  (15): nn.Dropout(0.100000)
  (16): nn.SpatialConvolution(16 -> 32, 3x3, 1,1, 1,1)
  (17): nn.SpatialBatchNormalization (4D) (32)
  (18): nn.ReLU
  (19): nn.SpatialConvolution(32 -> 64, 3x3, 1,1, 1,1)
  (20): nn.SpatialBatchNormalization (4D) (64)
  (21): nn.ReLU
  (22): nn.SpatialAveragePooling(3x3, 2,2)
  (23): nn.Dropout(0.100000)
  (24): nn.SpatialConvolution(64 -> 32, 1x1)
  (25): nn.SpatialBatchNormalization (4D) (32)
  (26): nn.ReLU
  (27): nn.SpatialConvolution(32 -> 16, 1x1)
  (28): nn.SpatialBatchNormalization (4D) (16)
  (29): nn.ReLU
  (30): nn.SpatialConvolution(16 -> 10, 1x1)
  (31): nn.SpatialBatchNormalization (4D) (10)
  (32): nn.ReLU
  (33): nn.SpatialAveragePooling(8x8, 1,1)
  (34): nn.View(10)
  (35): nn.LogSoftMax

Number of parameters:      47294
```

We based our convolutional neural network on a “network in network” model.
We chose to build our network with 9 main convolution building blocks (in 3 groups), each block consisting of 3 layers: SpatialConvolution, SpatialBatchNormalization and ReLU (in this order).
The first group of blocks shrink in size from 128 to 16, the second group expands from 16 to 64 and the third group shrinks again from 64 to 10.
After each group of blocks, we added a SpatialMaxPooling or SpatialAveragePooling layer and a Dropout layer.
The first layer of the network (BatchFlip) carries out the different data augmentation methods we defined (per model) and the last layer is a LogSoftMax transfer layer before output.
In order to comply with the exercise’s constraints, we used the following parameters:

- `Batch Size = 64`
- `Total # of Training Epochs = 300`
- `Dropout Layer p-value = 0.1`
- `Loss Criterion = ClassNLLCriterion (Negative Log Likelihood)`
- `Optimization Algorithm = ADAM (3 models) and SGD (1 model)`
- `Data Augmentation Methods = hflip, vflip, randomcrop (reflection)`

We trained a total of 4 different models, each model had a different optimization algorithm and a different combination of data augmentation methods.

Training Procedure

We started off by normalizing the data (training and test sets) according to the mean and standard deviation calculated on each channel (RGB) of the training set data.
We then defined the data augmentation methods used in the first layer of the network, to alter parts of the network’s input data samples.
We then constructed our neural network according to the architecture described above, and defined our loss criterion to be a Negative Log Likelihood criterion.
We defined our batch size to be 64 and proceeded to run the following flow for 300 epochs:

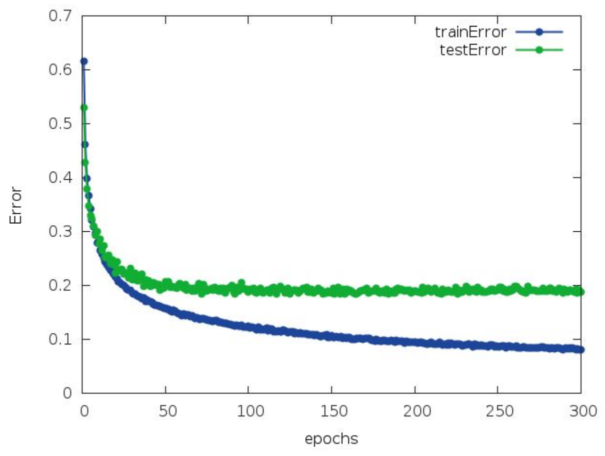
1. Shuffle the data and labels to produce a random input sequence to the network
2. Input the entire training set in batches of 64, were for each batch we did:
    a. Carry out Data Augmentation methods defined in first layer of network
    b. Forward propagation of augmented data samples into network
    c. Loss and Error calculations according to our defined loss criterion
    d. Backward propagation to calculate gradients for network weights using ADAM or SGD as our optimization algorithm.
3. Calculate confusion matrix, average loss and average error for network predictions on training and test sets respectively.

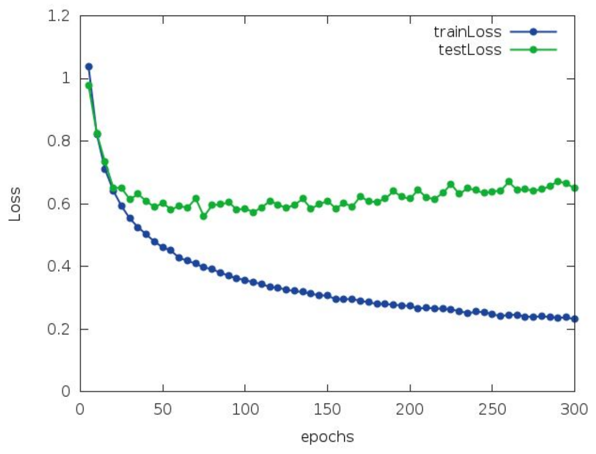
After 300 epochs are over we finished training the network, saved it to a file (network.model) and produced the Loss and Error by epochs graphs for test and training sets.

Model 1 - ADAM + hflip

Best Accuracy: **81.591%** (achieved after 125 epochs)

Error and Loss Graphs:

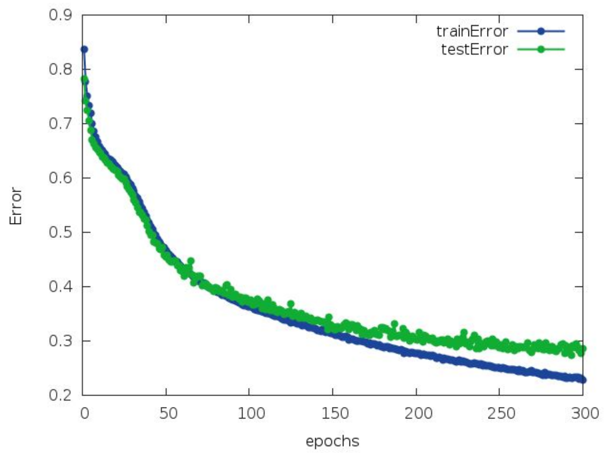


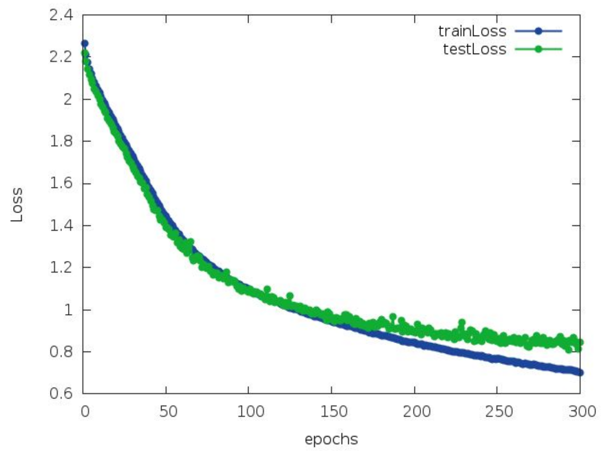


Model 2 - SGD + hflip

Best Accuracy: **71.725%** (achieved after 280 epochs)

Error and Loss Graphs:

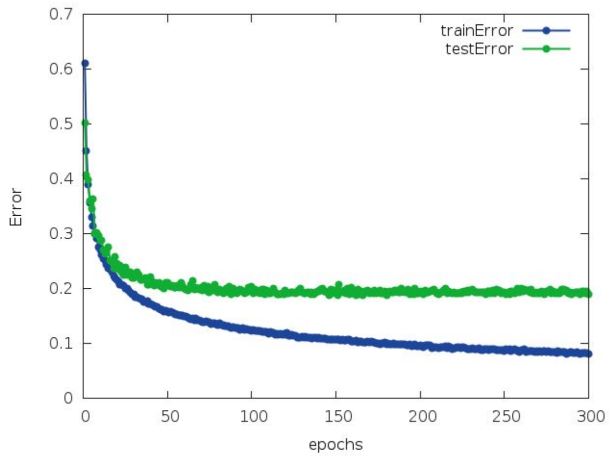


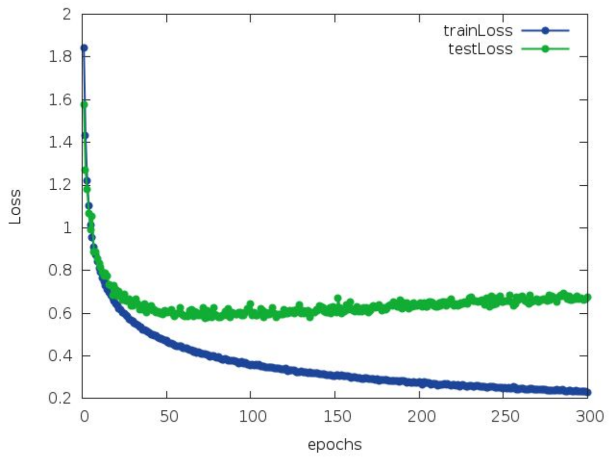


Model 3 - ADAM + hflip + vflip

Best Accuracy: **81.16%** (achieved after 180 epochs)

Error and Loss Graphs:

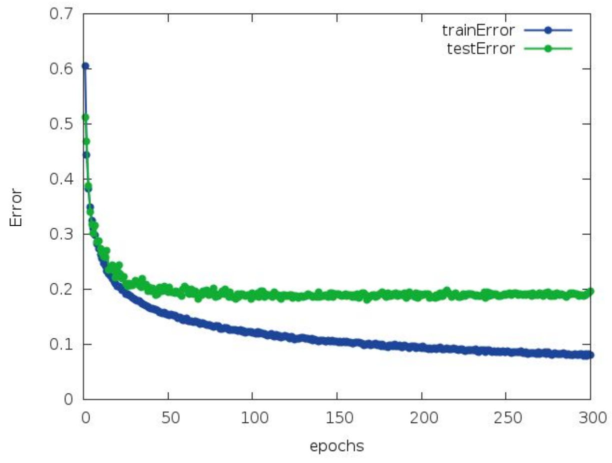


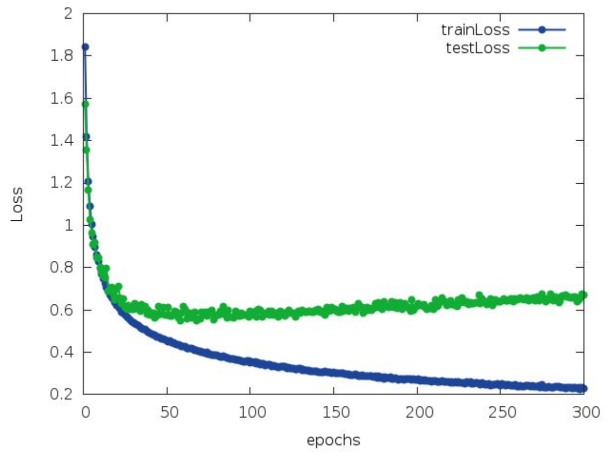


Model 4 - ADAM + hflip + vflip + randomcrop (reflection)

Best Accuracy: **81.751%** (achieved after 90 epochs)

Error and Loss Graphs:





Summary and Conclusion

We got to the network structure defined above, after a lot of trial and error - adding layers, removing layers, tweaking the layer sizes, tweaking Dropout until we got to the final structure which gave us the best results.
We then decreased the batch size to 64 and epochs to 300.
We evaluated 2 optimization algorithms -
ADAM and **SGD**, and the results with ADAM were much better and it got to higher accuracy much faster, so we chose it for our data augmentation evaluations.
We then evaluated 3 different data augmentation methods -
hflip (models 1, 2), **hflip+vflip** (model 3), **hflip+vflip+randomcrop(reflection)** (model 4), and chose to let each individual method augment about 25% of the data set, meaning models 1,2 had a total of 25% augmented data, model 3 had 50% and model 4 75%.
Our best results -
were received for model 4 - **ADAM + hflip + vflip + randomcrop - 81.751% accuracy** (achieved after only 90 epochs) and we think this is due to the relatively extensive use of data augmentation in this model, and an optimization algorithm which converges quicker for this task.

Best Results:

```
Model 4 - ADAM + hflip + vflip + randomcrop(reflection)
Epoch 90:
Training error: 0.12632042253521        Training Loss: 0.36743391231752
Test error: 0.18249198717949    Test Loss: 0.5644412679741
ConfusionMatrix:
[[     781      21      33      26      15       4       7      13      71      27]   78.257%   [class: airplane]
 [      13     893       4       1       2       6       5       1      21      53]   89.389%   [class: automobile]
 [      44       0     709      48      53      63      55      17       8       2]   70.971%   [class: bird]
 [      12       7      38     646      33     169      47      26       6      13]   64.794%   [class: cat]
 [       5       1      46      38     780      34      39      45       8       4]   78.000%   [class: deer]
 [       2       3      24      96      22     812       7      26       2       3]   81.444%   [class: dog]
 [       8       2      31      46      17      15     868       5       5       3]   86.800%   [class: frog]
 [       9       1      26      23      29      49       2     847       2       9]   84.955%   [class: horse]
 [      32       9       1      10       0       4       6       3     917      15]   91.976%   [class: ship]
 [      16      36       5       9       2       3       3       4      13     909]]  90.900%   [class: truck]
 + average row correct: 81.74863755703%
 + average rowUcol correct (VOC measure): 69.697523415089%
 + global correct: 81.750801282051%
```