Computer Vision Assignment-3

Abhinav Gupta (NetID-ag5799)

December 10, 2016

The model and code are present at `https://cs.nyu.edu/~ag5799/index.html`

I did the following preprocessing:

1. Cropping all the training as well as test images according to the coordinates given in the ROI. Then resizing them to an equal dimension (say 32x32).
2. Convert all images from the current RGB scale to YUV scale.
3. Global Normalization was done by calculating the mean and standard deviation per channel (Y,U,V) from all the training images. Then normalize all the training and test images with the calculated mean and standard deviation.
4. Local Contrast Normalization was done on all the training and test images by applying a gaussian kernel (say 5x5 width) to the images per channel.
5. Adding upto 5 jitters to the dataset. Each training image was rotated randomly in the range [-15,15] degrees then translated randomly by [-2,2] positions. Then I normalize the transformed image with its own mean and standard deviation.

Different combinations of the above techniques was used in different models.

Experiments done:

1. Implemented Parallel Dataset Iterator for threading. Achieved a boost in speed of upto 3x. With GPU's the speed goes upto 6x.
2. Also implemented upsampling as suggested in the hints. I extracted 10% of examples from all classes to make the validation set. The rest is used for training set. Then I make the size of all classes equal by randomly copying examples to make them equal to the maximum training instances present in one class. This gives equal weightage to all classes.

Models used:

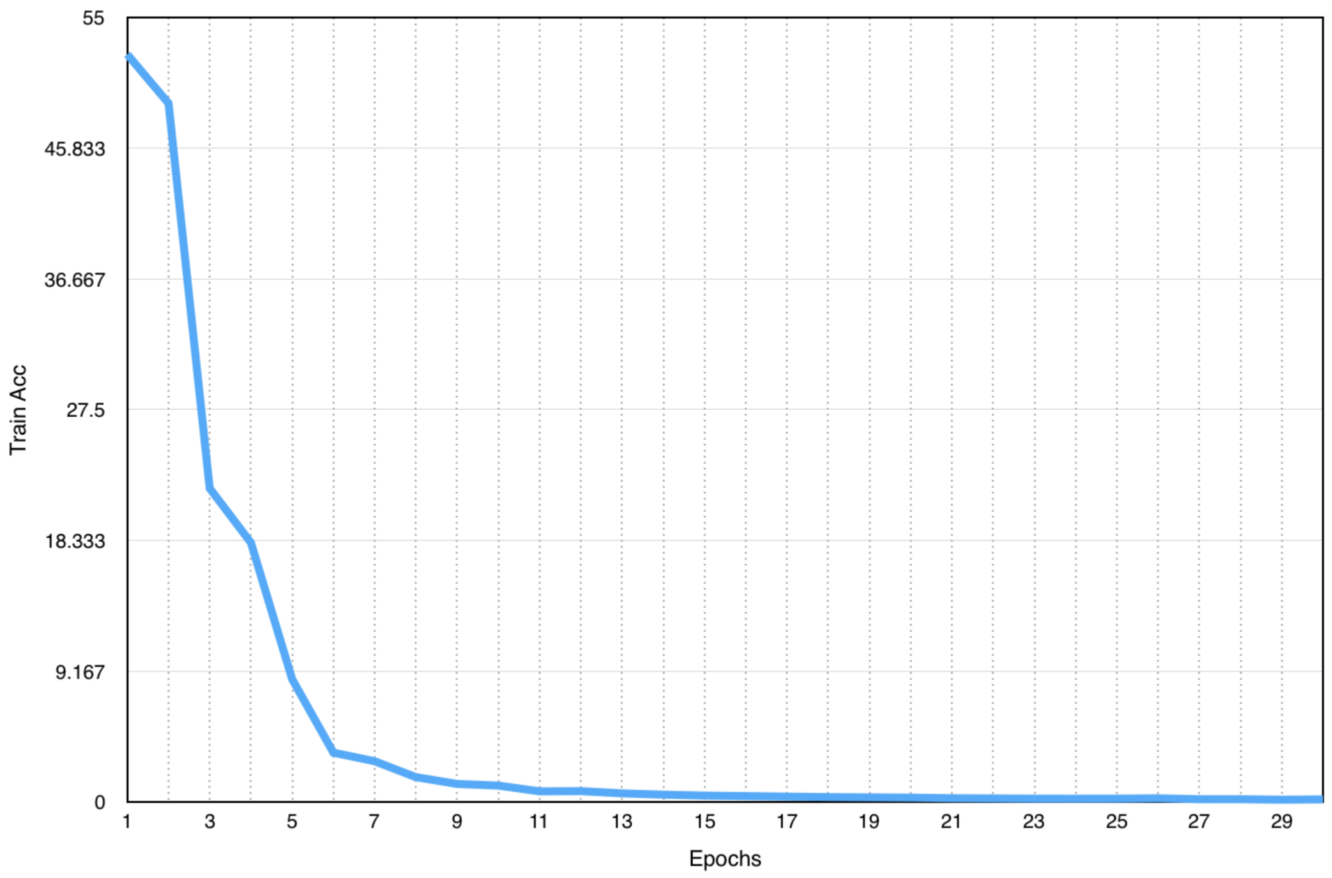

Figure 1

1. I used VGGnet as the training network. This gave my best submission of 98.8% accuracy on the test dataset. I used adagrad with learning rate of 0.1 for the best submission. The training was done for 30 epochs as the model converged after that. The convergence plot for this model is shown in Figure-1.

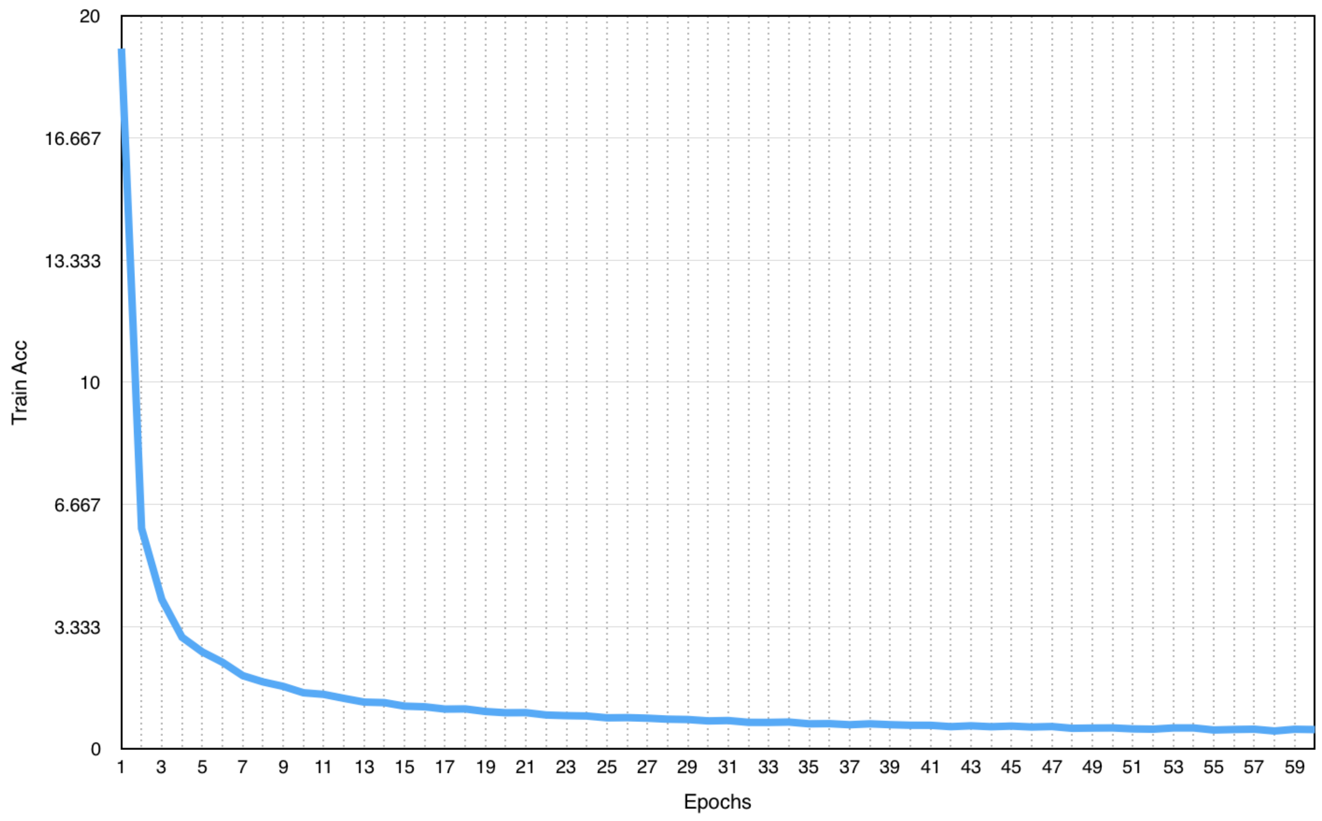

Figure 2

2. Then I created a new model by modifying the given cifar model. I added BatchNorm and Dropout layers, changed the non-linearity to ReLU, etc. This gave a test accuracy of 97.6%. In this case, the training was done for 60 epochs. The convergence plot for this model is shown in Figure-2.

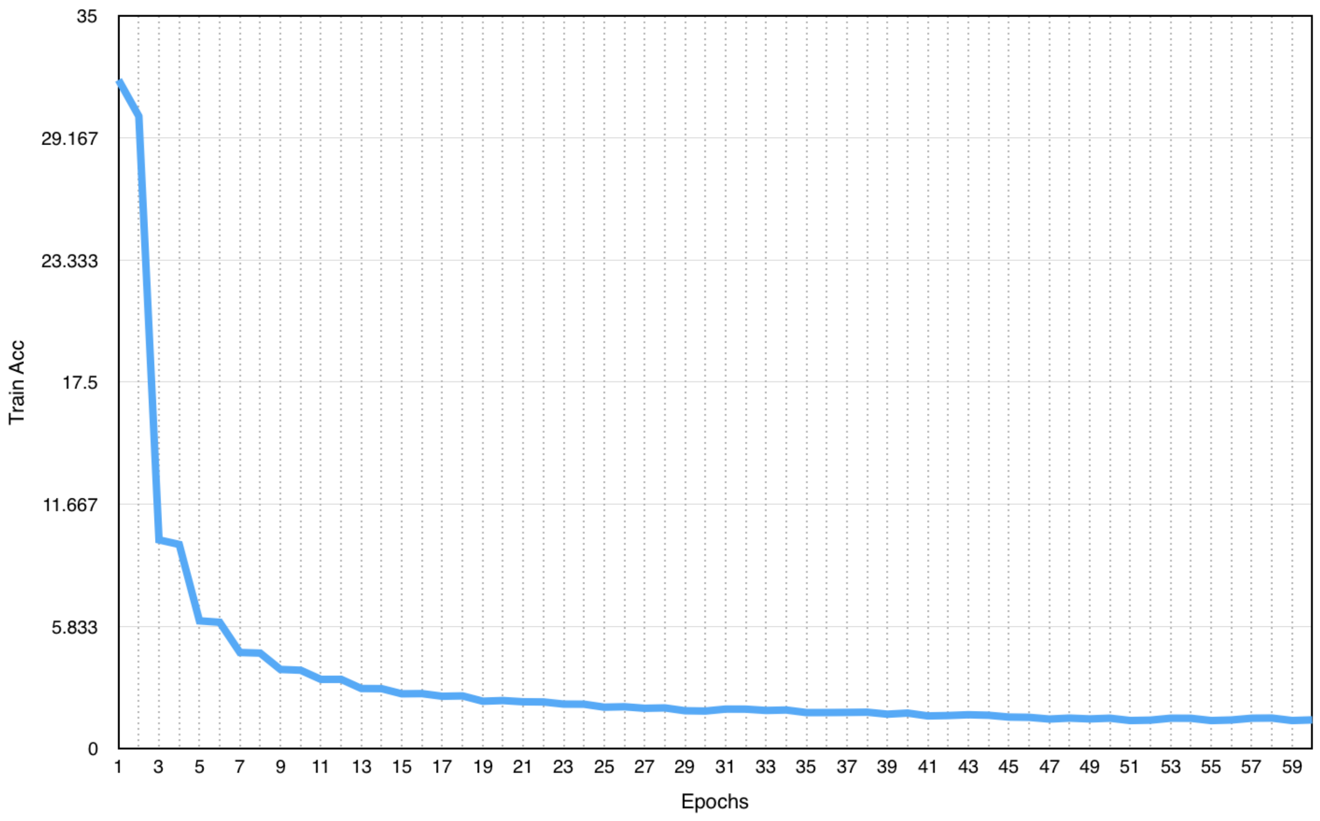

Figure 3

3. I also tested with the given cifar model. The convergence plot for this model is shown in Figure-3.
4. I also implemented the network from the Sermanet's paper. This gave me a test accuracy of 96.4%.

Total number of FLOPS are calculated according to the paper suggested.

For Convolution layers, it is calculated as Featuremapsizewidth * Featuremapsizeheight * Kernelwidth * KernelHeight * NumofInputFeatureMap * NumofOutputFeatureMap.
For MaxPooling layers, it is calculated as Featuremapsizewidth * Featuremapsizeheight * NumofInputFeatureMap
For all other layers, the FLOPS are zero.
vgg:add(nn.SpatialConvolution(3, 64, 3,3, 1,1, 1,1)) - 1769472
vgg:add(nn.SpatialBatchNormalization(64,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(64, 64, 3,3, 1,1, 1,1)) - 37748736
vgg:add(nn.SpatialBatchNormalization(64,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.SpatialMaxPooling(2,2,2,2)) - 65536

```
vgg:add(nn.SpatialConvolution(64, 128, 3,3, 1,1, 1,1)) - 18874368
vgg:add(nn.SpatialBatchNormalization(128,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(128, 128, 3,3, 1,1, 1,1)) - 37748736
vgg:add(nn.SpatialBatchNormalization(128,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.SpatialMaxPooling(2,2,2,2)) - 32768

vgg:add(nn.SpatialConvolution(128, 256, 3,3, 1,1, 1,1)) - 18874368
vgg:add(nn.SpatialBatchNormalization(256,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(256, 256, 3,3, 1,1, 1,1)) - 37748736
vgg:add(nn.SpatialBatchNormalization(256,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(256, 256, 3,3, 1,1, 1,1)) - 37748736
vgg:add(nn.SpatialBatchNormalization(256,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(256, 256, 3,3, 1,1, 1,1)) - 37748736
vgg:add(nn.SpatialBatchNormalization(256,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.SpatialMaxPooling(2,2,2,2)) - 16384

vgg:add(nn.SpatialConvolution(256, 512, 3,3, 1,1, 1,1)) - 18874368
vgg:add(nn.SpatialBatchNormalization(512,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(512, 512, 3,3, 1,1, 1,1)) - 37748736
vgg:add(nn.SpatialBatchNormalization(512,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(512, 512, 3,3, 1,1, 1,1)) - 37748736
vgg:add(nn.SpatialBatchNormalization(512,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0
```

```
vgg:add(nn.SpatialConvolution(512, 512, 3,3, 1,1, 1,1)) - 37748736
vgg:add(nn.SpatialBatchNormalization(512,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.SpatialMaxPooling(2,2,2,2)) - 8192

vgg:add(nn.SpatialConvolution(512, 512, 3,3, 1,1, 1,1)) - 9437184
vgg:add(nn.SpatialBatchNormalization(512,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(512, 512, 3,3, 1,1, 1,1)) - 9437184
vgg:add(nn.SpatialBatchNormalization(512,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(512, 512, 3,3, 1,1, 1,1)) - 9437184
vgg:add(nn.SpatialBatchNormalization(512,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.Dropout(0.4)) - 0

vgg:add(nn.SpatialConvolution(512, 512, 3,3, 1,1, 1,1)) - 9437184
vgg:add(nn.SpatialBatchNormalization(512,1e-3)) - 0
vgg:add(nn.ReLU(true)) - 0
vgg:add(nn.SpatialMaxPooling(2,2,2,2)) - 2048

classifier:add(nn.View(512)) - 0
classifier:add(nn.Linear(512, 512)) - 0
classifier:add(nn.Threshold(0, 1e-6)) - 0
classifier:add(nn.BatchNormalization(512, 1e-3)) - 0
classifier:add(nn.Dropout(0.5)) - 0
classifier:add(nn.Linear(512, 512)) - 0
classifier:add(nn.Threshold(0, 1e-6)) - 0
classifier:add(nn.BatchNormalization(512, 1e-3)) - 0
classifier:add(nn.Dropout(0.5)) - 0
classifier:add(nn.Linear(512, 43)) - 0
```

Total flops - 0398131200