CSE 666 Assignment 1: Tiny Imagenet Classification

Mitul Modi
UB Person No : 50288649
University at Buffalo, The State University of New York
mitulraj@buffalo.edu

February 23, 2019

Problem

Task is to build a deep neural network for image classification. Tiny Imagenet dataset is used for training and testing of model. it has 200 classes and 500 training images and 50 validation and testing images of 64 x 64 dimension.

Model Architecture

I have tried two different models for this task. All models and tasks are implemented using Pytorch framework. Models were trained on Google Colaboratory. Colab Notebook can be accessed by https://goo.gl/9L2FcY

1. Simple Convolutional Neural Network : It has 8 convolution layers followed by two fully connected layers. No pooling layer is used in model, instead it has convolution operation with stride 2 for down-sampling.

2. Resnet : I have implemented model with concept of residual networks with skip connections. There are total 17 convolutional layers with skip connection at every two layers. Last layer is fully connected layer. Here also no pooling layer is used and down-sampling is achieved by convolution operations with stride 2.

In both models, there is a batch normalization layer between each convolution layers. Relu is used as activation function except last layer where softmax is used as activation function for classification. Both models were trained using Stochastic Gradient Descent with nesterov momentum.

Other Parameters and Strategies

I tried following parameter tuning and strategies which gave me performance improvement over raw model performance.

1. Weight Initialization : For both models, weights were initialised by Xavier Uniform method.

2. Weight Decay : To prevent overfitting, I have used 0.001 weight decay for CNN. which gave me $\approx$ 7% boost in test accuracy. As Resnet take long time to train, I couldn't tune Resnet model with Weight Decay.

3. Data Augmentation : This is another method which gave performance boost. I have used Color Jitter, Random Horizontal and Vertical flips, random rotations translation and shear transformations to augment images. Every time pytorch reads images, it will randomly apply some transformations from above mentioned. So at each data load, model will see different set of images which helps it generalize better. Due to this model gives higher test accuracy than validation accuracy.

4. Early Stopping : Early stopping was used with patience value. When validation loss doesn't improve for number of epochs specified by patience value, it will exit training so that it doesn't overfit on training data.

5. Learning Rate Scheduler : I used Pytorch's ReduceLROnPlateau Scheduler to contain learning rate during training, which reduces learning rate by given fraction when validation loss stops decreasing.

6. Check-Point : During training best model with lowest validation loss is saved to disk so that after training best model can be retrieved for inference. Or training can be continued from last saved model in case of any failure.

Performance

Model	Train Loss	Val Loss	Test Loss	Train Accuracy	Val Accuracy	Test Accuracy	Time/e
CNN	0.0295	0.0408	**0.0358**	55.298	41.21	**47.76**	200 s
Resnet	0.0217	0.0491	0.0382	65.1511	38.76	43.96	1800

Table 1: Results

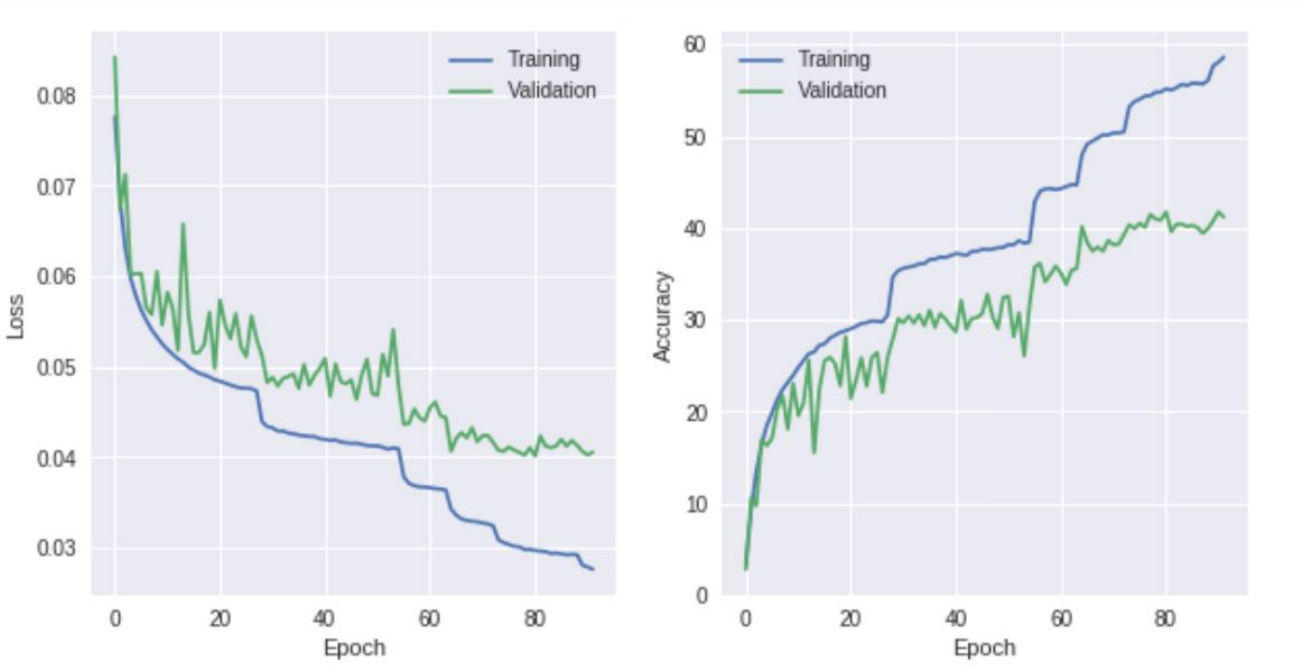

Figure 1: CNN Training Result

References

[1] Pytorch Official Documentation, Tutorial and Source code
https://pytorch.org

[2] Python Progress Bar - Stackoverflow
https://stackoverflow.com/questions/46141302/how-to-make-a-still-progress-in-python/46141777