Mukul Surajiwale
ECSE: 4965 Deep Learning
Final Project
5/9/2017

How to Run/Requirements

- Python 2.7.12
- Tensorflow Version 0.12.1
- Run using: *python gazeEstimate.py*

Model Architecture

The model implements a three-layer Convolutional Neural Network. Due to not having access to a GPU the model is designed with simplicity, without sacrificing accuracy, in mind. In order to reduce the computational load only the left eye pathway data is used. However, the model is still able to achieve rather impressive results given the limited and non-ideal nature of the data.

Model Details

Input image (64 x 64 x 3)

Conv Layer 1

7 x 7 Filter Size, 32 Filters, 1 x 1 Stride, "SAME" Padding

Max Pooling

2 x 2 Kernel Size, 2 x 2 Stride, "SAME" Padding

Conv Layer 2

5 x 5 Filter Size, 64 Filters, 1 x 1 Stride, "SAME" Padding

Max Pooling

2 x 2 Kernel Size, 2 x 2 Stride, "SAME" Padding

Conv Layer 3

3 x 3 Filter Size, 128 Filters, 1 x 1 Stride, "SAME" Padding

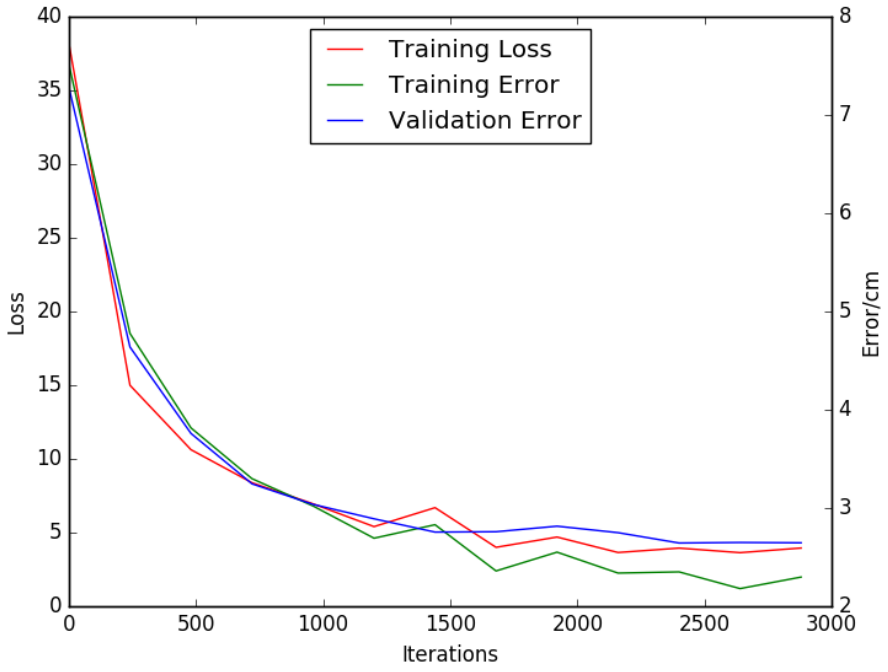
Fully Connected Output (32768 x 2)

The model contains three convolution layers and two max pooling layers. Going from the first convolutional layer to the last one the filter size decreases and the number filters increases. The larger the filter size the less detail from the image is captured. Thus the first layer, having the largest filter, captures higher level features in the image. The subsequent convolutional layers capture more low level and detailed features. Furthermore, the number of filters increases because as the filter size is decreased more filters are needed to capture all the minute features. Pooling layers are used to reduce the dimension of the image to improve computation time. By keeping padding as "SAME" no part of the image is cut off and the dimensionality is retained. Although more computationally expensive it leads to better results in this case. The optimal filter sizes and number of filters was found by experimentation.

Lastly, it should be noted that all the weights are initialized using Tensorflows *xavier_initializer_conv2d(uniform=False).* Xavier initiation helps the error signal generated by the gradients reach deeper into the network by automatically determining the optimal scale of the weights based on the number of input and output neurons. It should be noted that the *uniform* parameter is intentionally set too *False* as this improves performance when using the ReLU activation function.

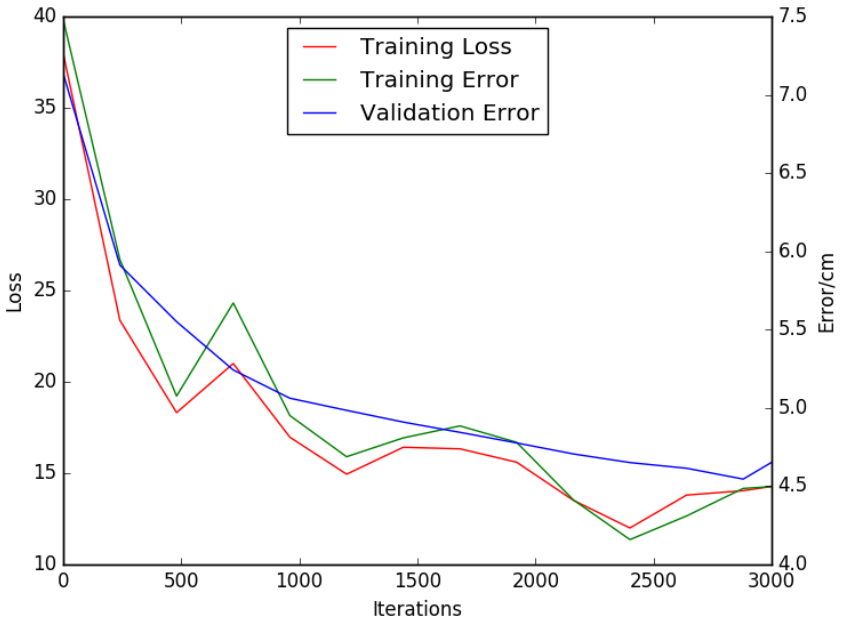
Results

Using batch size of 200 the model is able to achieve a training error or 1.892 cm and a validation error of 2.59 cm. The model is trained for 12.5 epochs and took approximately 9 hours to train.



Architecture Comparison

Since I do not have access to a GPU or a very fast CPU I simply could not run a more complex model like the one provided in the guide. It uses all the data pathways and would take far too long to train. Thus, I benchmarked my model against a simplified version of my model that uses only one convolutional layer and max pooling layer. As anticipated this model performs significantly worse as it is not able to capture enough detailed features. Below is a plot detailing the performance of this single convolution layer model. In the same time, it took it took my more complicated model to achieve a validation error of 2.59 cm, this one reached a validation error of only 4.65 cm. One can also see that the changes in the training and validation error are more erratic compared to my more complex model. This may be a sign that suggests that after each step the network may be adjusting too much which can increase the likelihood of overfitting.



Simplified Model Results

TensorBoard Model Architecture Visualization: (original image is also provided)