**Traffic Sign Classification Project**

**Data Augmentation by Image processing:**

I have analyzed the training data provided by the Udacity for this project and noticed that it is very unbalanced. In order for the trained model to not have any bias towards the labels that have more examples than rest, I have augmented the training data by generating samples using image-processing techniques. This will also enable model to become more robust and be generalized. I have used 4000 samples for each of the traffic sign labels in the training data set.

* Horizontal Flip: flip the image horizontally for some image classes and skip it for others, as a flipped version will fall into other class
* Rotate: rotate an image with a random angle between (-10, 10)
* Translate: translate the image along X & Y axes by an offset in the range (0, 5)
* Shear: sheared an image within a range of (-5, 5)
* CLAHE equalization:

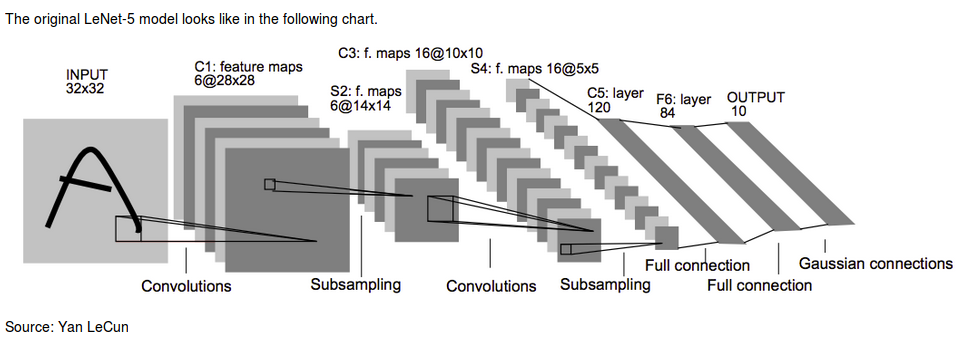
**Pre-processing training data:**

* *Shuffle:* I have shuffled the training data during each epoch in order for model to learn features, independent of the order of training data.
* *Gray-scale:* Training data set has been converted to gray-scale and it will enable model to extract and learn the features in a more robust way. This might also reduce the noise related to varying colors in the image.
* *Normalization:* Training data set has been normalized to avoid high variation in contrast.

**Model Architecture:**

I have used a Convolutional Neural Network (CNN) to classify traffic signs. CNNs perform well for image classification by optimizing the model architecture. CNNs take advantage of translational invariance of images and capture features efficiently with less number of parameters compared to a regular neural network, which in-turn avoids the problem of over-fitting.

*Original LeNet Model:*



My model architecture uses 4 convolutional layers for feature extraction followed by 3 fully connected layers for classification. Features from all 4 convolutional layers have been used for classification purposes.

**Weights & Biases:**

For convolution and fully connected layers I have initialized weights using Gaussian distribution with mean 0 and standard deviation of 0.1.

**ELU:**

For activation, I used Exponential Linear Unit (ELU) as it seemed to perform better than different variants of RELU. ELU avoids the issue of vanishing gradient and also achieves improved accuracy with faster convergence to the solution. One other aspect of ELU is that it bounds the

**Dropout:**

I have employed Dropout with a probability of 0.5 to avoid over-fitting. Dropout mechanism helps in making the model more generalized by dropping the links between layers randomly. In effect, it’s like training over an ensemble of networks, for each iteration and averaging out the results.

**L2 Regularization:**

I have also employed L2 regularization with a lambda of 0.0001, to avoid over-fitting. For applying this regularization, I used convolutional layers’ as well as fully connected layers’ weights.

**Softmax Cross Entropy & Adam Optimizer:**

I have used softmax cross entropy and L2 regularization to compute the cost function. Adam Optimizer is used to minimize the cost. I explored different learning rates and settled with 0.0005, as this seems to be a good trade-off between accuracy and speed of convergence.

**Model Training Parameters:**

Epochs = 200

Batch size = 128

I have chosen the above parameters, as they seem to be reasonable trade-off between speed of training and my MacBook-Pro’s ‘GeForce GT 750 M’ GPU with 2GB memory. It took around ~6 hours to train the model with the above parameters. Increasing the # of epochs might improve the training accuracy, which I will explore in the future and also use ‘Early-Stopping’ mechanism (end the training, if model is no longer able to learn anything new).

**Metrics:**

Training accuracy: ~94%

Test accuracy: ~92%

**Discussion:**

I started off with LeNet architecture, but wasn’t able to train the model to desired training accuracy (unable to achieve more than 87%). I realized that the model was under-fitting (as the training accuracy itself is low), so I increased the number of layers (convolutional and fully connected) and also removed the MaxPooling layer that was present in the original LeNet. I used Dropout and L2 regularization to avoid over-fitting the model to training data. I have employed image pre-processing techniques like horizontal flip, rotation, translation, shearing and CLAHE, to augment the training data set, which in-turn improved the performance of model considerably. I have also converted the images to gray scale and normalized them to avoid high variation in contrast. Applying above techniques helped in improving the training accuracy of Model by ~6 %.

I have also increased the number of filters when adding a new convolutional layer and increased the size of fully connected layers. This change improved the performance slightly ~1%.

**References:**

1. <http://yerevann.com/a-guide-to-deep-learning/>
2. <http://www.pyimagesearch.com/2017/01/02/rotate-images-correctly-with-opencv-and-python/>
3. <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html>
4. <http://stackoverflow.com/questions/24341114/simple-illumination-correction-in-images-opencv-c>
5. <https://en.wikipedia.org/wiki/Adaptive_histogram_equalization>
6. <http://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html>
7. <http://navoshta.com/traffic-signs-classification/>
8. <https://www.quora.com/What-is-the-difference-between-L1-and-L2-regularization>
9. <http://image-net.org/challenges/posters/JKU_EN_RGB_Schwarz_poster.pdf>
10. <http://www.picalike.com/blog/2015/11/28/relu-was-yesterday-tomorrow-comes-elu/>
11. <https://www.quora.com/How-does-ELU-activation-function-help-convergence-and-whats-its-advantages-over-ReLU-or-sigmoid-or-tanh-function>
12. <http://stackoverflow.com/questions/38189119/simple-way-to-visualize-a-tensorflow-graph-in-jupyter>