**LeNet for Traffic Signs**

In the Convolutional Neural Networks lesson we used LeNet with the MNIST data set. This following lesson is a walkthrough on how to use LeNet with MNIST and how to leverage this process and the LeNet architecture to classify traffic signs. This work was originally intended for a local or AWS GPU instance, but should now be conducted using a workspace. Please use the workspace provided in the Traffic Sign Classifier Project lesson.

**Data**

Note that the video directs you to download the data. However, the Traffic Sign Classifier Project workspace comes pre-configured with the required data.

## Strategies to Improve Your Classification Model

Looking at where to go next what could you do to further improve this model? There are several broad categories, to try.

* experiment with different network architectures, or just change the dimensions of the LeNet layers
* add regularization features like drop out or L2 regularization to make sure the network doesn't overfit the training data
* tune the hyperparameters
* improve the data pre-processing with steps like normalization and setting a zero mean
* augment the training data by rotating or shifting images or by changing colors

There are a lot of fun experiments you can run on this network so give it a shot and see how well you can do. Good luck.

While neural networks can be a great learning device they are often referred to as a black box, here we will explore a technique that lets us shine a light into that black box and see closer what it looks like on the inside by observing the kind of shadows that are formed. These shadows will be our feature maps, and after successfully training your neural network you can see what it's feature maps look like by plotting the output of the network's weight layers in response to a test stimuli image, which will be our light. From these plotted feature maps, it's possible to see what characteristics of an image the network finds interesting. For a sign, maybe the inner network feature maps react with high activation to the sign's boundary outline or to the contrast in the sign's painted symbol.

Provided for you below is the function code that allows you to get the visualization output of any Tensorflow weight layer you want. The inputs to the function should be a stimuli image, one used during training or a new one you provided, and then the Tensorflow variable name that represents the layer's state during the training process, for instance if you wanted to see what the LeNet lab's feature maps looked like for it's second convolutional layer you could enter conv2 as the tf\_activation variable.

For an example of what feature map outputs look like, check out NVIDIA's results in their paper End-to-End Deep Learning for Self-Driving Cars in the section Visualization of internal CNN State. NVIDIA was able to show that their network's inner weights had high activation to road boundary lines by comparing feature maps from an image with a clear path to one without. Try experimenting with a similar test to show that your trained network's weights are looking for interesting features, whether it's looking at differences in feature maps from images with or without a sign, or even what feature maps look like in a trained network vs a completely untrained one on the same sign image.

*# image\_input: the test image being fed into the network to produce the feature maps*

*# tf\_activation: should be a tf variable name used during your training procedure that represents the calculated state of a specific weight layer*

*# Note: that to get access to tf\_activation, the session should be interactive which can be achieved with the following commands.*

*# sess = tf.InteractiveSession()*

*# sess.as\_default()*

*# activation\_min/max: can be used to view the activation contrast in more detail, by default matplot sets min and max to the actual min and max values of the output*

*# plt\_num: used to plot out multiple different weight feature map sets on the same block, just extend the plt number for each new feature map entry*

**def** **outputFeatureMap**(image\_input, tf\_activation, activation\_min=-1, activation\_max=-1 ,plt\_num=1):

*# Here make sure to preprocess your image\_input in a way your network expects*

*# with size, normalization, ect if needed*

*# image\_input =*

*# Note: x should be the same name as your network's tensorflow data placeholder variable*

*# If you get an error tf\_activation is not defined it maybe having trouble accessing the variable from inside a function*

activation = tf\_activation.eval(session=sess,feed\_dict={x : image\_input})

featuremaps = activation.shape[3]

plt.figure(plt\_num, figsize=(15,15))

**for** featuremap **in** range(featuremaps):

plt.subplot(6,8, featuremap+1) *# sets the number of feature maps to show on each row and column*

plt.title('FeatureMap ' + str(featuremap)) *# displays the feature map number*

**if** activation\_min != -1 & activation\_max != -1:

plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmin =activation\_min, vmax=activation\_max, cmap="gray")

**elif** activation\_max != -1:

plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmax=activation\_max, cmap="gray")

**elif** activation\_min !=-1:

plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", vmin=activation\_min, cmap="gray")

**else**:

plt.imshow(activation[0,:,:, featuremap], interpolation="nearest", cmap="gray")