Project 3- Using TensorFlow for Road Sign Classification

Udacity Self-driving Car Engineering Nanodegree
Alejandro Terrazas, PhD

Go Small or Go Home

The course instructions recommended using the LeNet architecture as a starting point for the traffic sign project; however, two primary differences exist between LeNet data and the German Traffic Sign Recognition Benchmark Data (GTSRB): First, the GTSRB data has many more classes (43) compared to the digits data, which has 10 (for digits 0-9). Therefore, achieving high accuracy on the GTSRB data is a real challenge. Second, the GTSRB data have three color channels compared to the binary MNIST data. The color data add some features that can be helpful for classification (e.g., red circular border on the speed signs; blue foreground for directional signs). Gray scale versions of the images contain the shape information without color information. .

Guide to the Jupyter Notebooks and HTLM Files. *Most of the work in the project is presented in Jupyter Notebooks and HTML files created after running the models. The key notebooks (.ipynb files) are: 1) Visualize_and_Augment, 2) BuildModel, 3) RunModel, and 4) Visualize_Activation_Maps. Each of these files has an HTML equivalent. All are stored on the github site: alejandroterrazas/SDC-3/. The names are explanatory and cover four key aspects of the assignment.*

Because the images are small (32x32), losing resolution over many convolutional layers is a primary concern. In this project, strides of 1 and padding were used to maintain the image resolution up to a point. Of course, one of the strengths of CNNs is their ability to determine context in local to global levels of abstraction. Therefore, it is important to also down sample the feature maps at some point (toward the last layers) in the network architecture.

As Einstein is often quoted, everything should be as simple as possible but not simpler. Complex models can lead to over fitting; however, using a model with fewer than three convolution layers and one fully connected layer saturated at .80-.85 accuracy on the full 43-class problem.

The final architecture used a fairly simple model with a total of five convolutional layers and two fully connected layers. The fifth convolutional layer was a 1x1 layer to compress the number of feature maps. 3x3 filter sizes were used on each of the first four convolutional layers and strides of 1. Only on the fourth convolutional layer was a 2x2 max pool used, which reduced the size of the filters to 2x2 on the last layer.

So Many Classes, So Little Training Data

Classification to forty-three classes presents a tough challenge for a goal of 95% accuracy. With 43 classes, chance performance is a little over 2%. Super classes of the 43 classes, however, do exist. The existence of these super classes indicate that a hierarchical model might be suitable. The obvious super classes include a) circular, triangular, and other or alternatively, b) speed signs, other circles, triangular signs and other. Using these simple 3- and 4-class divisions, a simple 2-4 layer model was able to classify at over 95% accuracy. Training was quick for these. A hierarchical model was not used in this project but is an interesting avenue for future exploration.

Confusion matrices were used to examine the most common types of errors. On the full 43-class problem, the speed signs showed a clear tendency to be wrongly classified (as other speed signs); therefore, a future version of this model might be effective with a hierarchical design.

Augmenting and Balancing the Training, Validation and Test datasets using a Python Generator

The notebook **Visualize_and_Augment,ipynb** contains the code used to visualize the data and augment it. This same code is used throughout the three notebooks to generate train, validation and test data.

The training, validation and test datasets were substantially unbalanced (that is, contained different numbers of images in each category). Overrepresented categories include 1,2, 4, and37. Underrepresented categories include 0 and 27. The overrepresented classes contained up to 12 times as the underrepresented classes (3000:250). The Visualize_and_Augment notebook and corresponding HTML file shows\ a histogram of the number of images/label for each category of images (train, validation and test). In order to train the network, substantially more images are needed (even for the overrepresented categories).

Because of the small size of the data, data augmentation was used to generate additional examples. Data augmentation refers to ways to copy, and subtly change, the actual images into new training examples. Augmentation tools include scaling, rotating, flipping, and adding noise to the source images. Flipping was not used because many of the road signs have a clear cardinal orientation and, therefore, do not make sense when flips. For example, a stop sign has a clear left-to-right orientation that cannot be violated. For the sign data, minor rotation scaling, translation and adding noise were used.

The number of images (original and augmented) and the number of categories are large; thus, it is not practical to generate the entire train, validation, test data ahead of time. Therefore, a python generator was created that could produce a balanced batch of arbitrary size on demand. The batch was created by generating a random draw from the categories of interest with size equal to the batch size. Then, the batch of labels is used to generate a sample image from the category. For example, if the first item in the batch is 31, then an

image from category 31 is sample. A random number is then generated which determines when an original image is selected or if an augmented sample is to be generated. If an original is selected, then the generator moves on to the next item in the batch. If an augmented sample is to be generated, then a function is called which randomly scales, rotates, and adds noise to the sample image. The augmented image is added to the batch and the processes continues until a full batch is returned. After writing the image augmentation used in this project, I discovered the keras.proprocessing.image package. T

After visualizing the data, it became clear that the data were already augmented. Thus, the augmentation scheme was on top of already augmented data. A primary improvement in the development of this model further would be to increase the number of non-synthetic images in the database.

Xavier Initialization

Deep learning models are particularly sensitive to the initial weights that selected. Xavier initialization creates a principled method on which to base the initial weights. The Xavier weights depend directly on the number of neurons feeding into each neuron (Glorot and Bengio, 2010).

Xavier initialization is the Goldilocks of weight initializations—on the one hand, it helps to ensure the weights are not so large that they degrade to zero in a few iterations and, on the other hand, guarantees the weights are not so large that the network takes excessively long to converge. The basic algorithm is to sample from a unit Gaussian and divide by the "fan in" number to each neuron.

Learning Rate and Exponential Decay

The learning rate is another important hyperparameters that can be tuned. Having a small enough learning rate ensures that the neural network converges smoothly during training and, furthermore, prevents becoming stuck in local minima.

A larger learning rate tends to help networks converge initially but then are a disadvantage later in training. A smaller learning rate causes the network to converge too slowly. A better approach is to use a learning rate that is large initially and then is reduced gradually over the course of training. The exponential decay learning rate was used to improve training performance.

Accuracy Achieved. *The accuracy achieved was 99.5% on the training, 96.4% on validation and 95.3% on the test data. The test accuracy was measured over 128 batches of 256 test images (from the augmented test set.*

Train, Validation and Test Accuracy

Approximately 40-50 iterations were run using different numbers of CNN layers, lateral response inhibition, dropout, and capacity of the fully connected layers. Another variation was the training batch size and the change of the batch size in the growing batch scheme.

Progressive Batch Size

It often makes sense to change the batch size during training, often starting with smaller batch sizes (e.g., 64 or 128) until near saturation and then bumping it to 256 or 512 for further training. In the model used here, a progressive batch size over five epochs was used (32, 96,160, 224, and 288) were used. Using this scheme, the model was trainable in less than 3 hours on an AWS Ubuntu Instance. No GPU was used.

Regularization including Local Response Inhibition (LRN)

Two forms of regularization were used: 1) dropout and 2) lateral response normalization. In addition, a 1x1 convolutional layer was used. The generalization (as measured by performance on the previously unseen test dataset) was excellent with approximately 99% training accuracy, 95% validation and 94% on the test dataset.

Local response inhibition (LRN) is a form of local inhibition inspired by the brain. LRN is a way of suppressing nearby activation to increase the focality of the largest activation.

Top 5 Errors on Test Images

Ten test images were obtained from the Internet and cropped to the 32x32x3 image size of the train, validation, and test data. The model worked well on these images. All ten were in the top 5. Seven-out-of-ten were classified as the top 1. Of the three images not in the top 1, 2 were in the top 2 and the remaining image was in the top 3.

Confusion Matrix

A good way to understand errors on a multiclass classification problem is the confusion matrix. The confusion matrix lays out the labels in an n-label x n-label matrix such that for each label the diagonal elements represent the number of correct classifications and the off diagonal elements are the misclassified labels. An analysis of the worst confusors showed that 20, 21 and 30 were particular prominent as false positives. By focusing on those particular classes (e.g., obtaining more samples, identify particular issues with those classes, etc.), the model accuracy could be improved.

Visualization of Activations

One of the most interesting aspects of CNNs is that it is possible to view the activation maps in image form. An activation maps shows the sensitivity of each convolution filter to the presentation of a single image. This can give insight into which features have been learned. For example, triangular shaped images will “turn on” certain filters that have learned to respond to triangles. Likewise, certain filters will learn to respond to circular and/or blue features present in other images.

The images chosen were not necessarily difficult images. There were several that we darker compared to the others; however, the majority were well lit. This is perhaps one

reason that model generalized well to these examples. Another advantage of the internet images is that they could be cropped nicely to match the training and validation data. In a real-world scenario, there would be more substantial size variations in the input data. Nevertheless, in a real-world scenario there would be the potential to obtain multiple samples of the image (in essence tracking the sign). One of the images contained ice and snow covering, which was not part of the training set (as far as I know). The model was able to classify the ice-covered image. In retrospect, the images obtained from the internet were too easy in many ways. Almost all were derived from straight-on shots and in bright light. The generalization performance might do well in the real-world scenario but that has not been tested.