Srikant Viswanath
Self Driving Car Nanodegree
Apr 23rd 2018

# German Traffic Sign Classifier
## (Convolutional Neural Networks, Deep Learning)

### Dataset Exploration

- (cell#5) Basic summary of the input dataset has been performed by bumpy built in functions
- (cell#30) Visual exploration is performed by picking the first occurrence of each unique class and plotting using matplotlib
- (cell#6, cell#7)Distribution of each class in the dataset has been visualized by plotting a histogram - where some classes were in frequency around 2400 where as others were only around 300

### Design and Test Model Architecture

#### a) Preprocessing

- I started off lazy and used the 3 channeled RGB images (with normalization) to feed into my **LeNet model**. The results were horrendous: **7% validation accuracy** as opposed to the initial prescribed 89%; in addition to the model being very slow because of 3 times the numbers to crunch
- (cell#8)It then hit me that - a traffic sign is color independent, i.e., there is no additional information hidden in the RGB components of the image. Hence turned it into a **grayscale** images using **opencv**
- (cell#10)Followed by **normalization**. This is required to have ~0 mean, so we don't get rounding off/precision errors when dealing high precision floating numbers
- I then proceeded to feed the normalized images as is into LeNet again. The results were yet again **abysmal - 7.5% validation accuracy**

- After almost a week of going back forth between lecture notes and trying out various things, it finally struck me that the problem is in the uneven distribution of input dataset.
- Instead of pruning the classes available in abundance (which would be a sheer waste of useful data), it made more sense to generate fake data for the classes lacking in numbers
- (cell#9)To generate fake data, I used helper functions: *random_scale()*, *random_translate()*, *random_brightness()*
- *(*cell#12, cell#16*)*Data was augmented using one of the helper functions randomly for the classes that were low in numbers - to be boosted until the mean of the classes' distribution (~810 for training, 101 for validation)
- Finally, when I tried feeding the **augmented dataset to an initial LeNet model**, the validation accuracy started to look sane: **~87%**

**b) Model Architecture**

After a ton of playing around with various architectures including the one mentioned in the article from the assignment(http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf), the model seemed to give the best **validation accuracy of ~95%** was rather a very simple one(**cell#16 - tweaked_lenet()**):

- **Input Layer**: 32 x 32 x1
- **Conv Layer 1**: 5 x 5 x 1 x 16, 1 x 1 stride, VALID (Output: 28 x 28 x 16)
- **RELU**
- **dropout**: keep_prob - 0.7
- **max_pool**: 2 x 2 size, 2 x 2 stride, VALID (Output: 14 x 14 x 16)
- **Conv Layer 2**: 5 x 5 x 16 x 32, 1 x 1 stride, VALID (Output: 10 x 10 x 32)
- **RELU**
- **dropout**: keep_prob - 0.7
- **max_pool**: 2 x 2 size, 2 x 2 stride, VALID (Output: 5 x 5 x 32)
- **Conv Layer 3:** 5 x 5 x 430, 1 x 1 stride, VALID (Output: 1 x 1 x 430)
- **RELU**
- **droput:** keep_prob - 0.7
- **flatten layer**
- **dropout:** keep_prob - 0.5

- **Fully connected layer:** 430 x 43
- **RELU**

**c) Training (cell#25, cell#26, cell#27)**

- **AdamOptimizer** was chosen as the optimizer as it comes built in with an adaptive learning rate decay, unlike the vanilla GradientDescentOptimizer which supports only static learning rate. Besides, Adam was encouraged during the class
- **learning_rate** of 0.0007 was used after jumping from 0.001, 0.0001, 0.0005, 0.0009
- **#epochs:** In the initial trials where the validation accuracy was below 90%, I had played around with epochs well above 100 - misleading optimism :)

**d) Arriving at satisfactory validation accuracy(cell#77, cell#16, cell#82, cell#42)**

- I had lost a significant amount of time debugging and figuring out why my **validation accuracy was tanking out at ~85%** while t**raining accuracy was at 98%**, which was a clear case of overfitting. And no amount of dropouts seemed to help. I**t turns out that I made the rookie blunder of feeding in the keep_prob of 0.7 and 0.5 for cone layers and fully connected layers even in *evaluate()* function instead of 1.0!!**
- After fixing, the validation accuracy started to **hover around 91%**
- This was the last part of the struggle where numerous architectures was tried
- Clearly, I was still overfitting, so a lot of tuning of dropouts was performed
- Having more number of fully connected layers seemed to be detrimental to the validation accuracy as it seemed to introduce too much noise/overfitting which could not be cured by dropout. Hence stuck with just 1 layer of fully connected
- In the convolution layers, a lot of juggling was done w.r.t **relative order of max_pool and dropout layers.** Dropout before max_pool seemed to have an edge
- At the convolution layers, my intuition was to have a deepish kernel at each step, so we capture as many low level curves/edges as possible. And then moving on to having an even deeper kernel to capture a wide range of high level features. A sweet spot seemed to around 16 and 32 respectively, without causing too much noise

**d) Test Accuracy(Moment of Truth)**

**A test accuracy of 96.2%** was achieved. Satisfactory, indeed. Finally

### Test Model on New Images

- The new images were downloaded from http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset#Downloads. The **first 6 images**(00000.ppm, 00001.ppm, 00002.ppm, 00003.ppm, 00004.ppm, 00005.ppm) were used to test
- Upon manual inspection, labels were assigned to the images as : **[7, 3, 9, 35, 3, 3]**, i.e., **[Speed limit 100, Speed limit 60, No passing, Straight Ahead, Speed limit 60, Speed limit 60]**
- The second Speed limit 60 sign was dark and pixelated which might be a little concern
- (cell#40, cell#41)The raw RGB images were plotted along with the grayscale preprocessed versions
- **(cell#72)Testing accuracy on the new images set was 100%**

### Top 5 Probabilities(cell#83)

- For the 1st image which is '100 speed limit' following classes had top scores:
    - *100 speed limit:* 99.99%
    - Other classes minuscule probs: *80 speed limit* (0.00006%), *Speed limit 120*(0.00005%), *Roundabout Mandatory*(0.000000003%), *No passing for vehicles over 3.5 metric tons*(0.000000003%)

In fact, the model has an extreme confidence in all of the remaining 5 images' prediction(all 99.99%, some even 100%)

- Actually, many random sets of 5 images were tried, and all of them seem to have such high confidence predictions
- Clearly, my sampling is not good enough