# Traffic Sign Recognition

**Build a Traffic Sign Recognition Project**
The goals / steps of this project are the following:
* Load the data set (see below for links to the project data set)
* Explore, summarize and visualize the data set
* Design, train and test a model architecture
* Use the model to make predictions on new images
* Analyze the softmax probabilities of the new images
* Summarize the results with a written report

**Writeup / README**

## Data Set Summary & Exploration

**1. Provide a basic summary of the data set and identify where in your code the summary was done. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.**

The 2nd code cell of the IPython notebook contains the code for this step.

I used python and numpy library to calculate summary statistics of the **original** traffic signs data set:

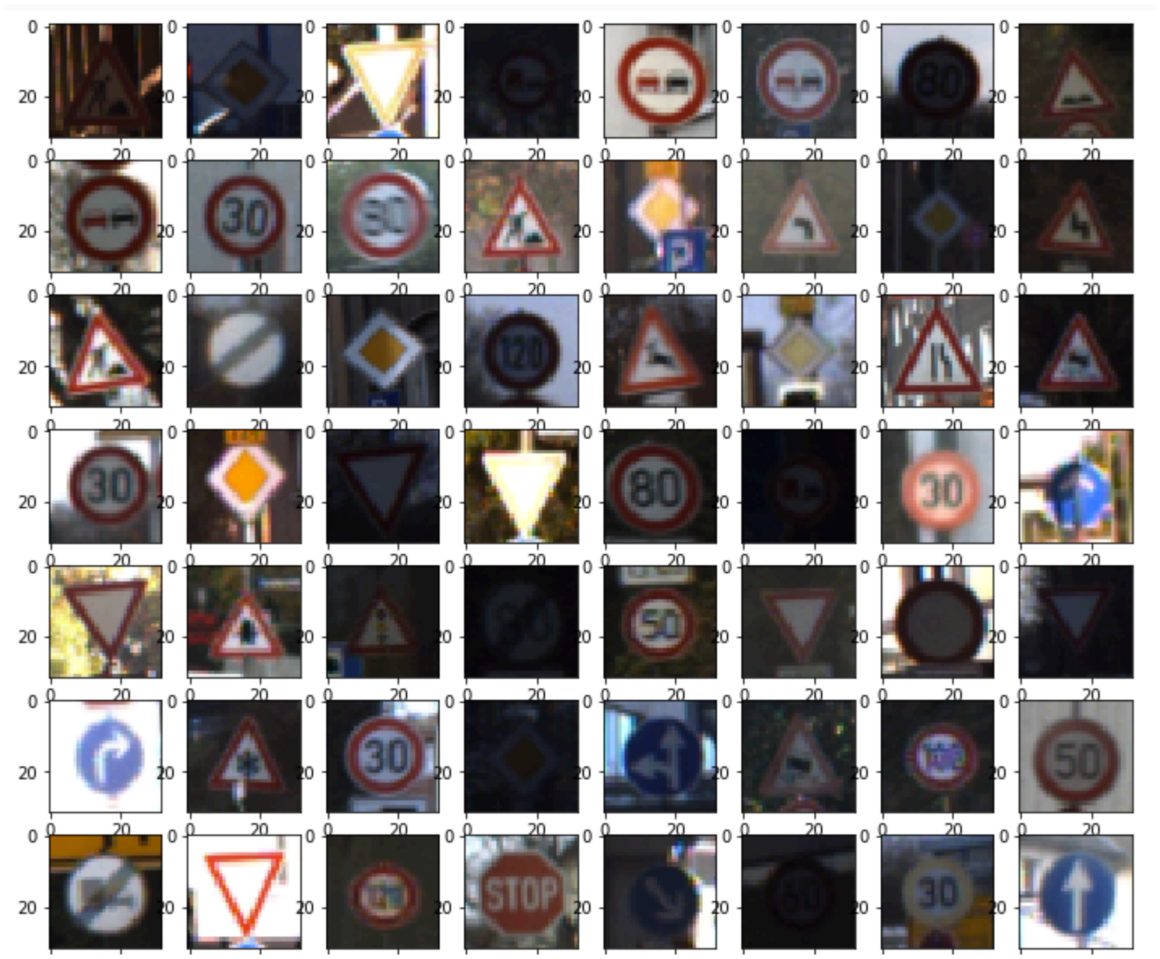
Number of training examples = 34799
Number of valid examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43

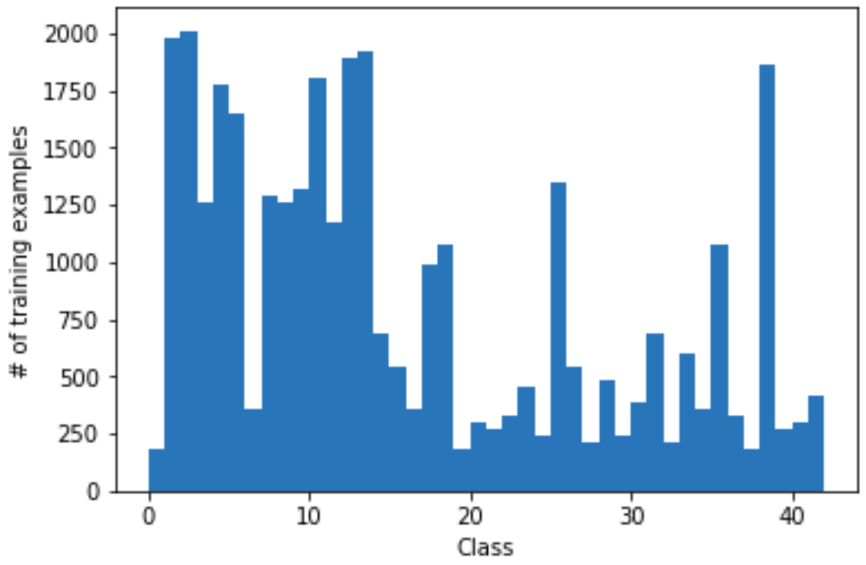
**2. Include an exploratory visualization of the dataset and identify where the code is in your code file.**

The 3rd code cell of the IPython notebook contains the code for this step.

We can observe than:
1º There are many dark images.
2º We have an imbalanced dataset. There are classes with many samples and classes with little samples.

Here is an exploratory visualization of the data set.


STOP


# of training examples
Class
0
250
500
750
1000
1250
1500
1750
2000
0
10
20
30
40

## Design and Test a Model Architecture

**1. Describe how, and identify where in your code, you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.**

I've tested several models with different preprocess techniques and hyperparameters.

<u>1st Model:</u>
Pipeline:

Initialization of weights:
I initialize the weights with truncated_normal function with very low values:
mean=0 and std=0.1

Preprocessing:
Normalization of RGB images between values [0 - 1]

Model:
+ CNN1 layer [32 kernels] >> RELU activation >> MaxPooling >> Dropout
+ CNN2 layer [64 kernels] >> RELU activation >> MaxPooling >> Dropout
+ FC1 layer >> RELU activation
+ FC2 layer >> RELU activation >> Dropout

Hyperparameters:
Keep_prob = 0.5

Results: ~ 92 - 93% accuracy

<u>2nd Model:</u>

Pipeline:

Initialization of weights:
I get some tips of the paper winner in IJCNN 2011 competition:
Multi-Column Deep Neural Network for Traffic Sign Classification
http://people.idsia.ch/~juergen/nn2012traffic.pdf

I initialize the weights with truncated_normal function with very low values:
mean=0 and std=0.05

Preprocessing:
Normalization of images between values [0 - 1]

Model:
+ CNN1 layer [100 kernels] >> RELU activation >> MaxPooling >> Dropout

+ CNN2 layer [150 kernels] >> RELU activation >>  MaxPooling >> Dropout
+ FC1 layer >> RELU activation
+ FC2 layer >> RELU activation >> Dropout

Hyperparameters:
Keep_prob = 0.5

Results: ~ 93 - 95% accuracy after 40 epoch.

<u>3rd model:</u>

Pipeline:

Initialization of weights:
I initialize the weights with truncated_normal function with very low values:
mean=0 and std=0.05

Preprocessing:
I convert images dataset from RGB to HSV.
I use this function from TF: tf.image.rgb_to_hsv(images, name=None).
This function returns values between [0,1].

I follow the same reasoning than I made in Project 1: “Finding Lane Lines on the Road ”.

In Wikipedia article about HSV, we can read: “HSL and HSV models are based more upon how colors are organized and conceptualized in ***human vision*** in terms of other color-making attributes ”. In the same way, if we visualize the hidden layers from a CNN network, we can observe similarities with ***human vision.***

Refs:

http://docs.opencv.org/3.2.0/df/d9d/tutorial_py_colorspaces.html
https://en.wikipedia.org/wiki/HSL_and_HSV
https://en.wikipedia.org/wiki/Convolutional_neural_network

Model:
+ CNN1 layer [100 kernels] >> RELU activation >>  MaxPooling >> Dropout
+ CNN2 layer [150 kernels] >> RELU activation >>  MaxPooling >> Dropout
+ FC1 layer >> RELU activation
+ FC2 layer >> RELU activation >> Dropout

Hyperparameters:
Keep_prob = 0.5

Results: ~ 92 - 93% accuracy

## Final Model:

## Preprocessing Techniques:

I observed than:

1º There are many dark images.
2º We have an imbalanced dataset. There are classes with many samples and classes with little samples.

To get more accuracy I need:

A. Improve the images.
B. Increase the number of images.
C. Balance the dataset

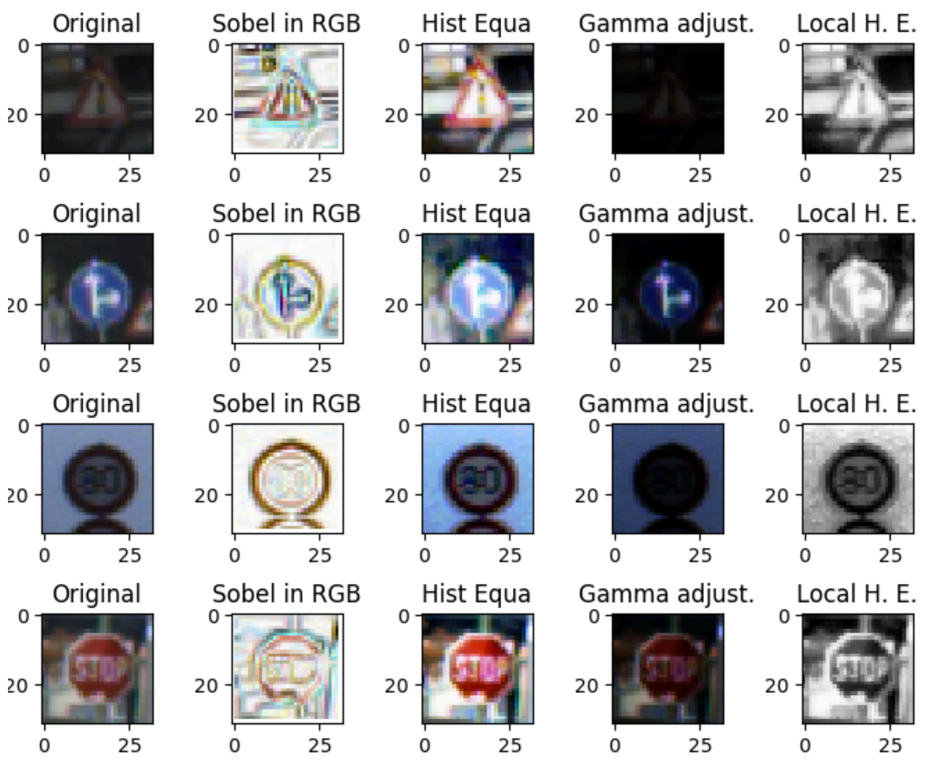
**A. Improve the images.**

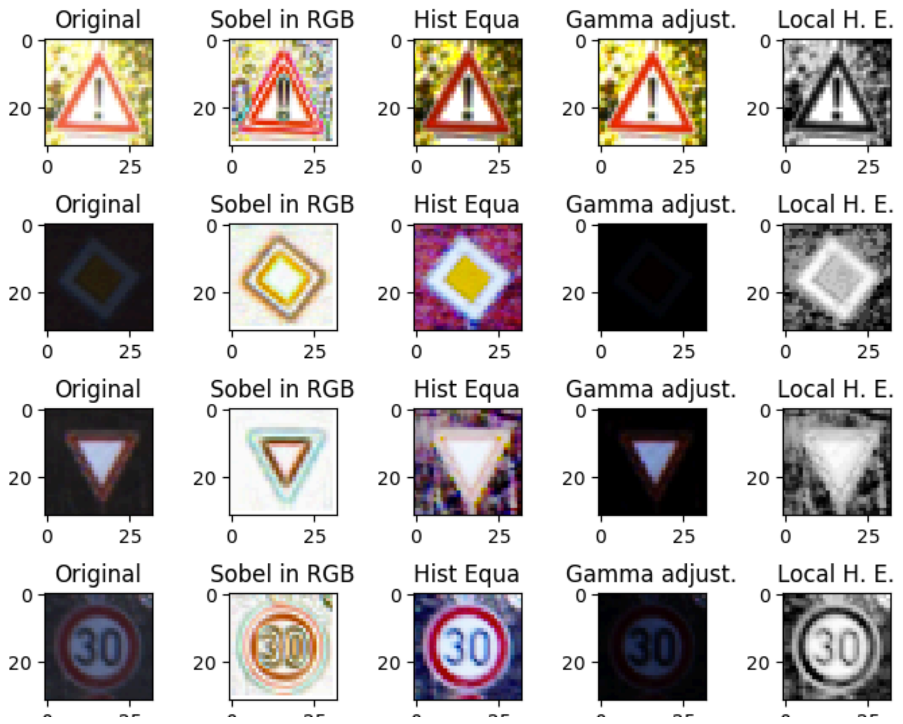
To get better images, I test several methods to preprocessing the images.
I try to follow the methods of the paper “Multi-Column Deep Neural Network for Traffic Sign Classification”.

I use scikit-image library.

The 4th code cell of the IPython notebook contains the code for this step.

Samples of preprocessed Images:





I've applied these filters:

+ Sobel to RGB images.

+ Histogram Equalization. HE¶
+ Gamma and log contrast adjustment.¶
+ Local Histogram Equalization. LHE

All techniques have good and bad results.

In a first approximation, I train the model with preprocessed images with HE.
It's Ok but the training time is very long.

In the next steps I train the model with LHE.
I get a model more little and I can speed the feedback of my experiments.

¶

Refs:
http://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_adapt_rgb.html#sphx-glr-auto-examples-color-exposure-plot-adapt-rgb-py

http://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_equalize.html#sphx-glr-auto-examples-color-exposure-plot-equalize-py

http://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_log_gamma.html#sphx-glr-auto-examples-color-exposure-plot-log-gamma-py

http://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_local_equalize.html#sphx-glr-auto-examples-color-exposure-plot-local-equalize-py

**B. Increase the number of images. Data Augmentation**

I apply data augmentation [DA] before training and during training.

+ For DA before training I reuse code from @navoshta.
+ For DA during training I use ImageDataGenerator from Keras.

Ref:
#Data Augmentation
http://navoshta.com/traffic-signs-classification/
https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
https://keras.io/preprocessing/image/

**C. Balance the dataset**

What can we do when we have imbalanced data:

- Ignoring the problem.
- Undersampling the majority class.
- Oversampling the minority class.
- To Weight the classes in function of the # of the samples in dataset.

I code functions to balance dataset in 8th cell.
Using Keras I can weight the classes during the training.
But I don't apply weights to classes during my training because I get bad results during my tests.

The 8th code cell of the IPython notebook contains the code for this step.

**Ref:**
##Balance of DATASET
#http://www.marcoaltini.com/blog/dealing-with-imbalanced-data-undersampling-oversampling-and-proper-cross-validation
#http://datascience.stackexchange.com/questions/13490/how-to-set-class-weights-for-imbalanced-classes-in-keras
#https://github.com/fchollet/keras/issues/1875
#https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/utils/class_weight.py

**2. Describe how, and identify where in your code, you set up training, validation and testing data. How much data was in each set? Explain what techniques were used to split the data into these sets. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, identify where in your code, and provide example images of the additional data)**

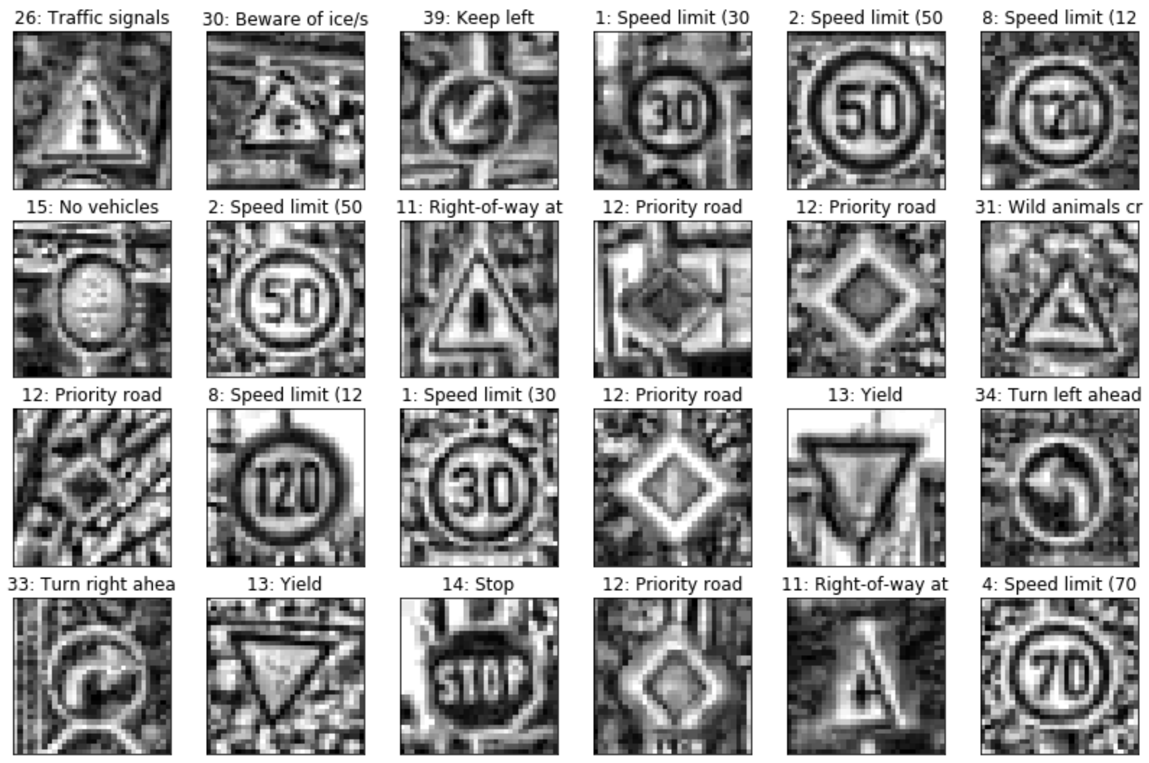
Pipeline for Dataset transformations:

1. First load train, valid and test dataset form original files. Code in the number 1 cell
2. Data Augmentation.
    a. Before training: Code in the number 6 & 7 cell.
    b. During training: Code in the number 17 cell.
3. Preprocessing the data and save to files. Code in the number 9 & 10 cell.

My final dataset:

```
Number of training examples = 59788
Number of valid examples = 7590
Number of testing examples = 21720
```

Here is an example of augmented images:



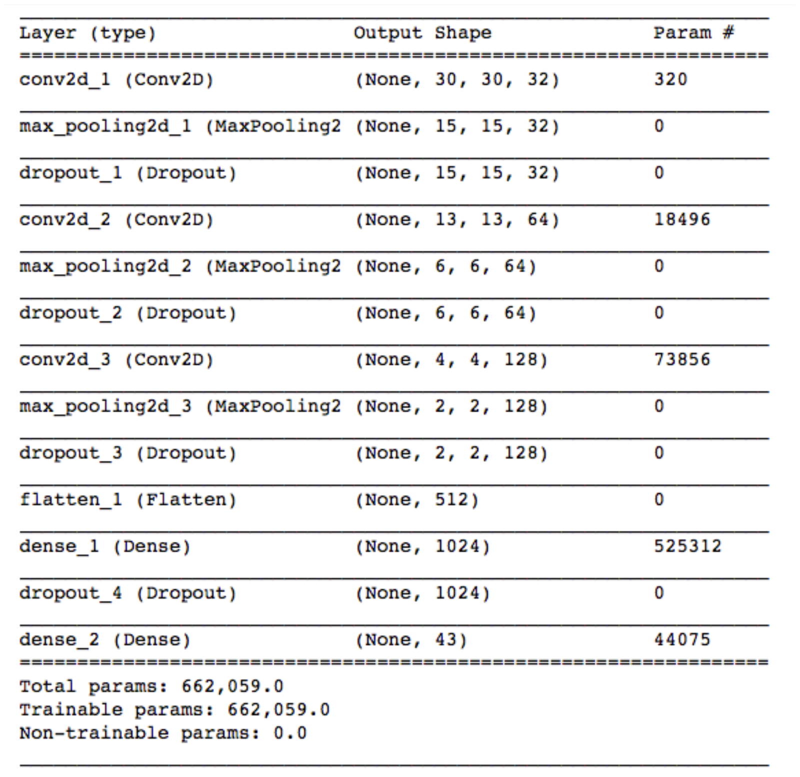
**3. Describe, and identify where in your code, what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.**

The code for my final model is located in the number 19 cell.

**Model:**

I use 3 CNN with 32, 64 and 128 filters.
Kernels size equal to 3x3
Max pooling 2x2

My final model consisted of the following layers:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              525312
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 43)                44075
=================================================================
Total params: 662,059.0
Trainable params: 662,059.0
Non-trainable params: 0.0
_________________________________________________________________
```

Hyperparameters:
#Init Variables
batch_size = 512
num_classes = 43
epochs = 50
dropout = [1.,1.,1.,1.]
# input image dimensions
img_rows, img_cols = 32, 32
input_shape = (img_rows, img_cols, 1)

| **Layer** | **Description** |
|---|---|
| Input | 32x32x1 Gray image |
| Convolution 3x3 | 32 filters, valid padding, RELU activation |
| Max pooling | 2x2 stride |
| Dropout | Value = float between 0 and 1 |
| Convolution 3x3 | 64 filters, valid padding, RELU activation |
| Max pooling | 2x2 stride |
| Dropout | Value = float between 0 and 1 |
| Convolution 3x3 | 128 filters, valid padding, RELU activation |
| Max pooling | 2x2 stride |
| Dropout | Value = float between 0 and 1 |
| Flatten | output= 512 |
| Fully connected | output= 1024 |
| Dropout | Value = float between 0 and 1 |
| Softmax | 43 classes |

| | |
|---|---|
| | |
| | |
| | |

**4. Describe how, and identify where in your code, you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.**

The code for training the model is located in the 20th cell of the ipython notebook.

+ Optimizer: Adam. With values: lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0).
+ Batch Size: 512
+ Epochs: 50
+ Dropuot: [1.,1.,1.,1.]
+ learning_rate=0.0001

Batch size:
Although the paper "Systematic evaluation of CNN advances on the ImageNet" recommends to use 128 batch.

I use a batch size big for several motives:

1° Because Karpathy recommended, in his classes of CS231n(year 2016), to use big batches to a better convergence of the model.
2° Fast training

Optimizer:
I use Adam because is the most recommended in the papers than I've read.
I set Adam parameters to follow the paper https://arxiv.org/abs/1412.6980v8
In the last step of training I down learning_rate to 0.0001

Dropout:
During my experiments, I try several values of dropout but I don't get good results.

Training Steps:
I trained the model in several steps:

Step 1:
I generate random training data during training process.
To do this process, I use Keras class: **ImageDataGenerator.**
I set dropout values to 1. No dropout.
I train the model during 50 epochs and I get an accuracy of 0.96113 in the validation set.
I save the model.

Step 2:

I load the model from step 1.
I generate random training&valid data during the training process.
I set dropout values to 1.  No dropout.
I train the model during 50 epochs and I get an accuracy of 0.97980 in the validation set.
I save the model

Step 3:
Repeat step 2 with learning rate = 0.0001
I train the model during 50 epochs and I get an accuracy of 0.98545 in the validation set.
I save the model

**5. Describe the approach taken for finding a solution. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.**

The code for calculating the accuracy of the model is located in the 21th cell of the Ipython notebook.

My final model results were:
* training set accuracy of 0.9994
* validation set accuracy of 0.98545
* test set accuracy of 0.980156537753

*If an iterative approach was chosen:*
**** What was the first architecture that was tried and why was it chosen?***
My first architecture was the model proposed by the team that won the final phase of the German traffic sign recognition benchmark in the paper: “Multi-column deep neural network for traffic sign classification”.

*** What were some problems with the initial architecture?**
No problem. But it is not simple. It has many filters.
I need many hours to train the model.
I don’t like work with this model.

*** How was the architecture adjusted and why was it adjusted?**

First apply data augmentation over Lenet 5 to get a fast feedback.

Second, when DA is working ok, I change Lenet5 for a more deep architecture.
This will be my final model.

I follow the recommendations of 2 papers and several blogs:

+ “Systematic evaluation of CNN advances on the ImageNet”
+ “Traffic Sign Recognition with Multi-Scale Convolutional Networks ”
+ “Building powerful image classification models using very little data”

I train this model in several incremental steps.

*** Which parameters were tuned?**
I tuned: dropout, learning rate and DA parameters.

*** How were they adjusted and why?**
I down learning rate a decade in the last step of training steps. I need to get a fine optimization of my model.
I use the random generators in the training and validation data to get more “variability” in my data.
I don’t apply dropout. During my experiments, I get very bad results.

*** What are some of the important design choices and why were they chosen?**

The most import design choice is to use DA before and during training.

The second most import choice is to use Keras:
+ Let me to define the model more clearly.
+ Let me to generate random data.
+ Let me to select the best hyperparameters by default.

Tensorflow is an unnecessarily complicate.
Keras like scikit-learn is simple and powerful.
Keras let me to define clearly the pipeline

**For example, why might a convolution layer work well with this problem?**
**How might a dropout layer help with creating a successful model?**

CNN work great with images … like humans.

*In Wikipedia article about HSV, we can read: “HSL and HSV models are based more upon how colors are organized and conceptualized in **human vision** in terms of other color-making attributes ”. In the same way, if we visualize the hidden layers from a **CNN** network, we can observe similarities with **human vision.***

Refs:

http://docs.opencv.org/3.2.0/df/d9d/tutorial_py_colorspaces.html
https://en.wikipedia.org/wiki/HSL_and_HSV
https://en.wikipedia.org/wiki/Convolutional_neural_network

*** How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?**

The values of accuracy show that the model is working well.

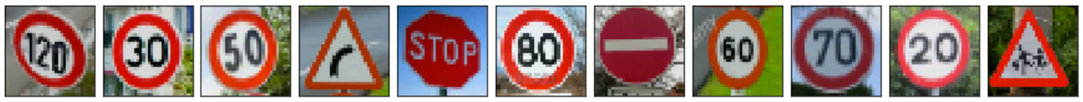
*Train acc > Valid acc > Test acc.*

They follow the correct order with little gap between then.

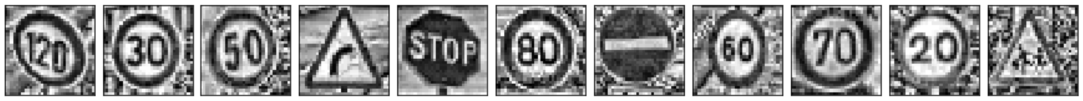
**Test a Model on New Images**
**1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.**

The code for Test a Model on New Images is located in the 24th cell of the Ipython notebook.

Eleven European traffic signs that I found on the web: original and processed.
+ Original: I cut it from image original and reduce to size 32x32.
+ Processed: I processed it with LHE techniques.



```
|████████████████████████████████████████████████████████████| 100%
(11, 32, 32)
```



```
(11, 32, 32, 1)
```

Numbers might be difficult to classify because of the tilt, the angle, etc.

The original imagines here:

| | |
|---|---|
| | |
| | |
| | |
| | |

**2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. Identify where in your code predictions were made. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).**

The code for making predictions on my final model is located in the 26th cell of the Ipython notebook.

Here are the results of the predictions on the new traffic signs:

```
Predicted Labels:      [ 8  1  2 20 14  5 17  5  4  1 28]
True Labels:           [ 8  1  2 20 14  5 17  3  4  0 28]


Test accuracy: 0.818181812763
```

| Image | Prediction |
|---|---|
| Speed limit (120km/h) | Speed limit (120km/h) |
| Speed limit (30km/h) | Speed limit (30km/h) |
| Speed limit (50km/h) | Speed limit (50km/h) |

| | |
|---|---|
| Dangerous curve to the right | Dangerous curve to the right |
| Stop | Stop |
| Speed limit (50km/h) | Speed limit (50km/h) |
| No entry | No entry |
| Speed limit (60km/h) | Speed limit (80km/h) |
| Speed limit (70km/h) | Speed limit (50km/h) |
| Speed limit (20km/h) | Speed limit (30km/h) |
| Children crossing | Children crossing |
| | |
| | |

The model was able to correctly guess 8 of the 11 traffic signs, which gives an accuracy of 81%.

This compares **not** favorably to the accuracy on the test set of 0.98

**ANALYZE NEW IMAGE PERFORMANCE IN MORE DETAIL**
**Calculating the accuracy on these five German traffic sign images found on the web might not give a comprehensive overview of how well the model is performing. Consider ways to do a more detailed analysis of model performance by looking at predictions in more detail. For example, calculate the** precision and recall **for each traffic sign type from the test set and then compare performance on these five new images..**

Metrics:

Precision, Recall, F1 and Confusion Matrix give us more info about our model.

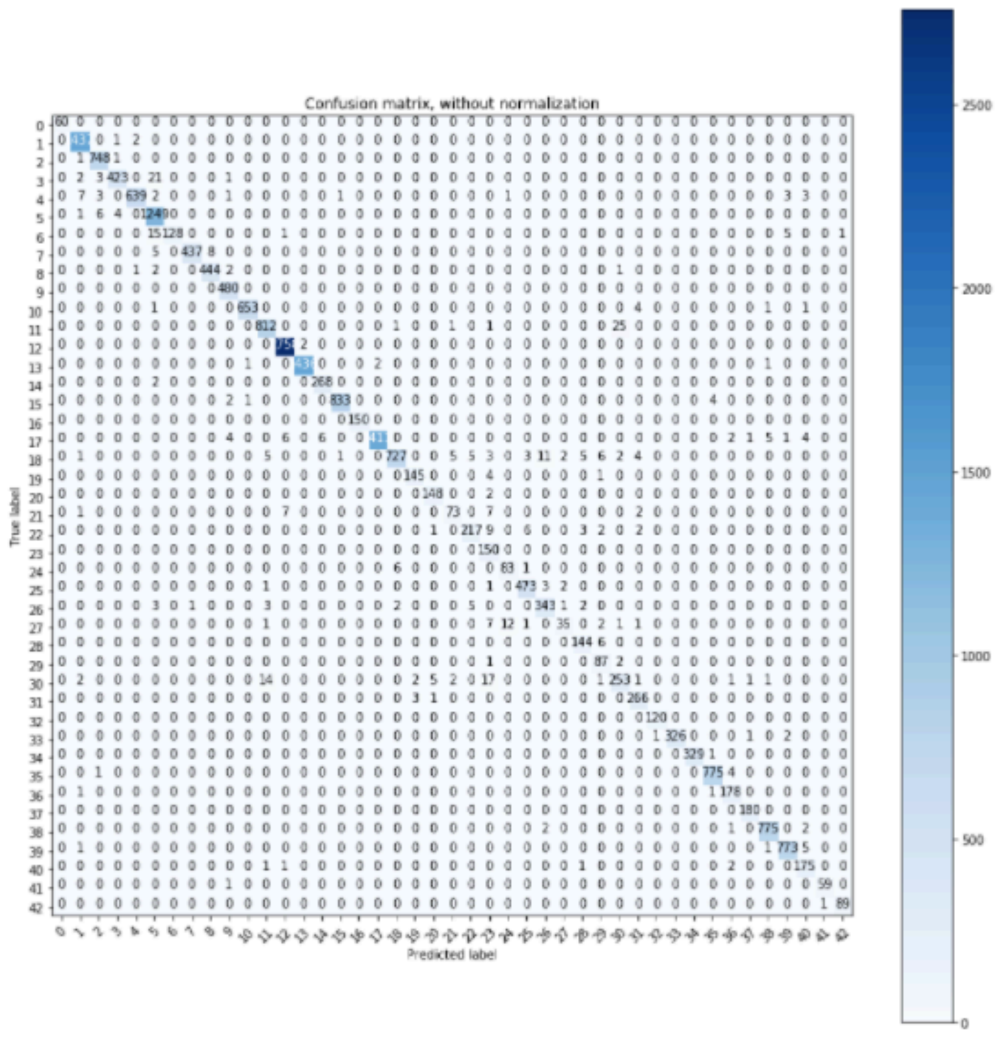
The code for Metrics is located in the 27th-33rd cells of the Ipython notebook.

```
-------------------------------------------------------------------
| Label | Precision | Recall | FScore | Support |
Label_0      1.00    1.00    1.00    60.00
Label_1      0.99    1.00    0.99    1440.00
Label_2      0.98    1.00    0.99    750.00
Label_3      0.99    0.94    0.96    450.00
Label_4      1.00    0.97    0.98    660.00
Label_5      0.96    0.99    0.98    1260.00
Label_6      1.00    0.85    0.92    150.00
Label_7      1.00    0.97    0.98    450.00
Label_8      0.98    0.99    0.98    450.00
Label_9      0.98    1.00    0.99    480.00
Label_10     1.00    0.99    0.99    660.00
Label_11     0.97    0.97    0.97    840.00
Label_12     0.99    1.00    1.00    2760.00
Label_13     1.00    1.00    1.00    1440.00
```

```
Label_14      0.98    0.99    0.99    270.00
Label_15      1.00    0.99    0.99    840.00
Label_16      1.00    1.00    1.00    150.00
Label_17      1.00    0.98    0.99    1440.00
Label_18      0.99    0.93    0.96    780.00
Label_19      0.97    0.97    0.97    150.00
Label_20      0.95    0.99    0.97    150.00
Label_21      0.90    0.81    0.85    90.00
Label_22      0.96    0.90    0.93    240.00
Label_23      0.74    1.00    0.85    150.00
Label_24      0.86    0.92    0.89    90.00
Label_25      0.98    0.99    0.98    480.00
Label_26      0.96    0.95    0.95    360.00
Label_27      0.88    0.58    0.70    60.00
Label_28      0.93    0.96    0.94    150.00
Label_29      0.83    0.97    0.89    90.00
Label_30      0.89    0.84    0.87    300.00
Label_31      0.95    0.99    0.97    270.00
Label_32      0.99    1.00    1.00    120.00
Label_33      1.00    0.99    0.99    330.00
Label_34      1.00    1.00    1.00    330.00
Label_35      0.99    0.99    0.99    780.00
Label_36      0.95    0.99    0.97    180.00
Label_37      0.98    1.00    0.99    180.00
Label_38      0.99    0.99    0.99    780.00
Label_39      0.99    0.99    0.99    780.00
Label_40      0.92    0.97    0.95    180.00
Label_41      0.98    0.98    0.98    60.00
Label_42      0.99    0.99    0.99    90.00
```

Confusion Matrix:

The code for Confusion Matrix is located in the 32nd-33rd cell of the Ipython notebook.



**3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction and identify where in your code softmax probabilities were outputted. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)**

The code for making predictions on my final model is located in the 34th - 36th cells of the Ipython notebook.

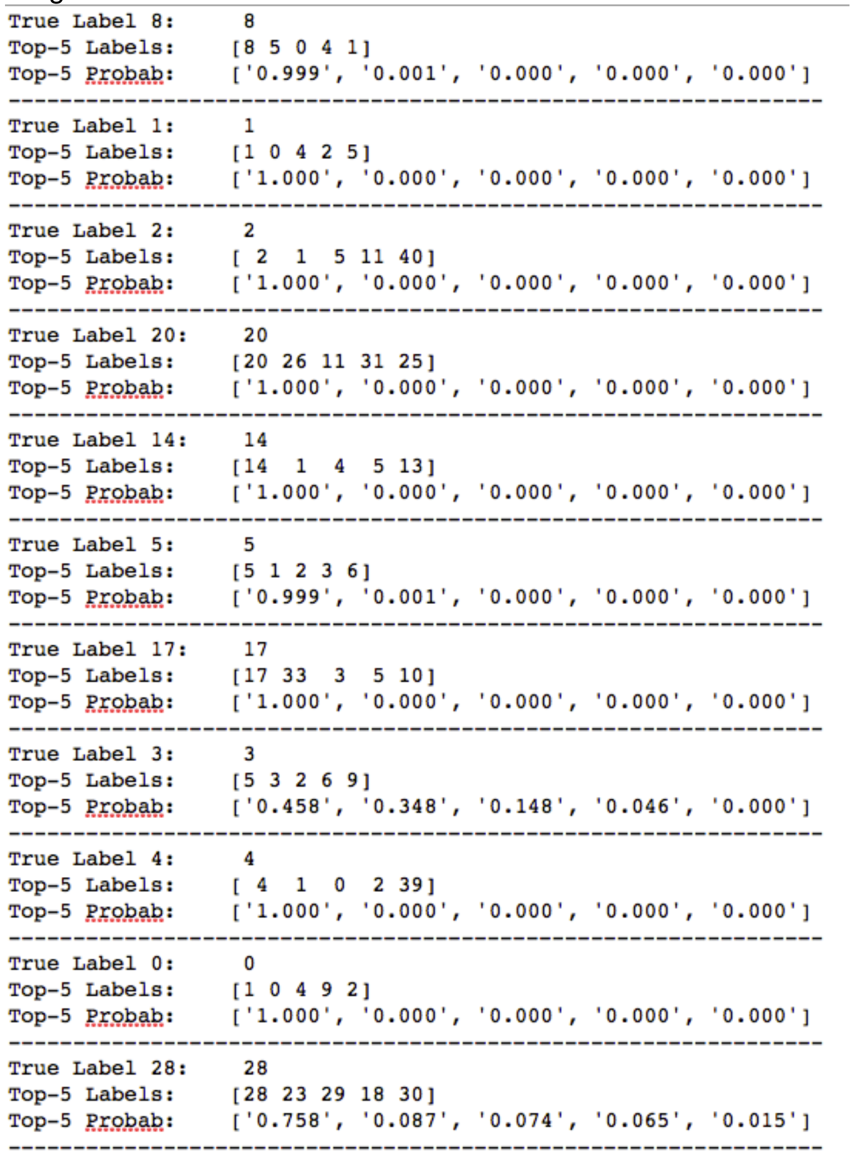
In general, the model is relative sure about its predictions.

**But, but… the model has a very import error in “True Label=0”.**

This image is a clear image but it’s classified like classid=1.  Why?

I think that the problem is this one:
The classid=0 is very imbalanced. We have very little samples in our dataset.

And the solution could be this one:
To retrain the model during more epochs and balance the classes during training with weights.



```
True Label 8:      8
Top-5 Labels:     [8 5 0 4 1]
Top-5 Probab:     ['0.999', '0.001', '0.000', '0.000', '0.000']
---------------------------------------------------------------
True Label 1:      1
Top-5 Labels:     [1 0 4 2 5]
Top-5 Probab:     ['1.000', '0.000', '0.000', '0.000', '0.000']
---------------------------------------------------------------
True Label 2:      2
Top-5 Labels:     [ 2  1  5 11 40]
Top-5 Probab:     ['1.000', '0.000', '0.000', '0.000', '0.000']
---------------------------------------------------------------
True Label 20:     20
Top-5 Labels:     [20 26 11 31 25]
Top-5 Probab:     ['1.000', '0.000', '0.000', '0.000', '0.000']
---------------------------------------------------------------
True Label 14:     14
Top-5 Labels:     [14  1  4  5 13]
Top-5 Probab:     ['1.000', '0.000', '0.000', '0.000', '0.000']
---------------------------------------------------------------
True Label 5:      5
Top-5 Labels:     [5 1 2 3 6]
Top-5 Probab:     ['0.999', '0.001', '0.000', '0.000', '0.000']
---------------------------------------------------------------
True Label 17:     17
Top-5 Labels:     [17 33  3  5 10]
Top-5 Probab:     ['1.000', '0.000', '0.000', '0.000', '0.000']
---------------------------------------------------------------
True Label 3:      3
Top-5 Labels:     [5 3 2 6 9]
Top-5 Probab:     ['0.458', '0.348', '0.148', '0.046', '0.000']
---------------------------------------------------------------
True Label 4:      4
Top-5 Labels:     [ 4  1  0  2 39]
Top-5 Probab:     ['1.000', '0.000', '0.000', '0.000', '0.000']
---------------------------------------------------------------
True Label 0:      0
Top-5 Labels:     [1 0 4 9 2]
Top-5 Probab:     ['1.000', '0.000', '0.000', '0.000', '0.000']
---------------------------------------------------------------
True Label 28:     28
Top-5 Labels:     [28 23 29 18 30]
Top-5 Probab:     ['0.758', '0.087', '0.074', '0.065', '0.015']
---------------------------------------------------------------
```

**VISUALIZE LAYERS OF THE NEURAL NETWORK**
**See Step 4 of the Ipython notebook for details about how to do this.**

The code for Visualize Layers is located in the 37th- cells of the Ipython notebook.

But I have errors when to try to run **outputFeatureMap** function.

I get session and graph tensor from keras model.
I analyze graph looking for “your network's tensorflow data placeholder variable”.
I find it but this value doesn’t work.

Refs:
I have visualized layers of VGG16 following these tutorials:
#https://github.com/fchollet/keras/blob/master/examples/conv_filter_visualization.py
#https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html