Traffic Sign Recognition

BY:

Ayham Zaza

For

SDCND Program

April. 2017

Traffic Sign Recognition

OBJECTIVES

- Load the 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

RUBRIC POINTS

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

WRITEUP / README

1. **Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.**

   You're reading it! and here is a link to my project code

Data Set Summary & Exploration

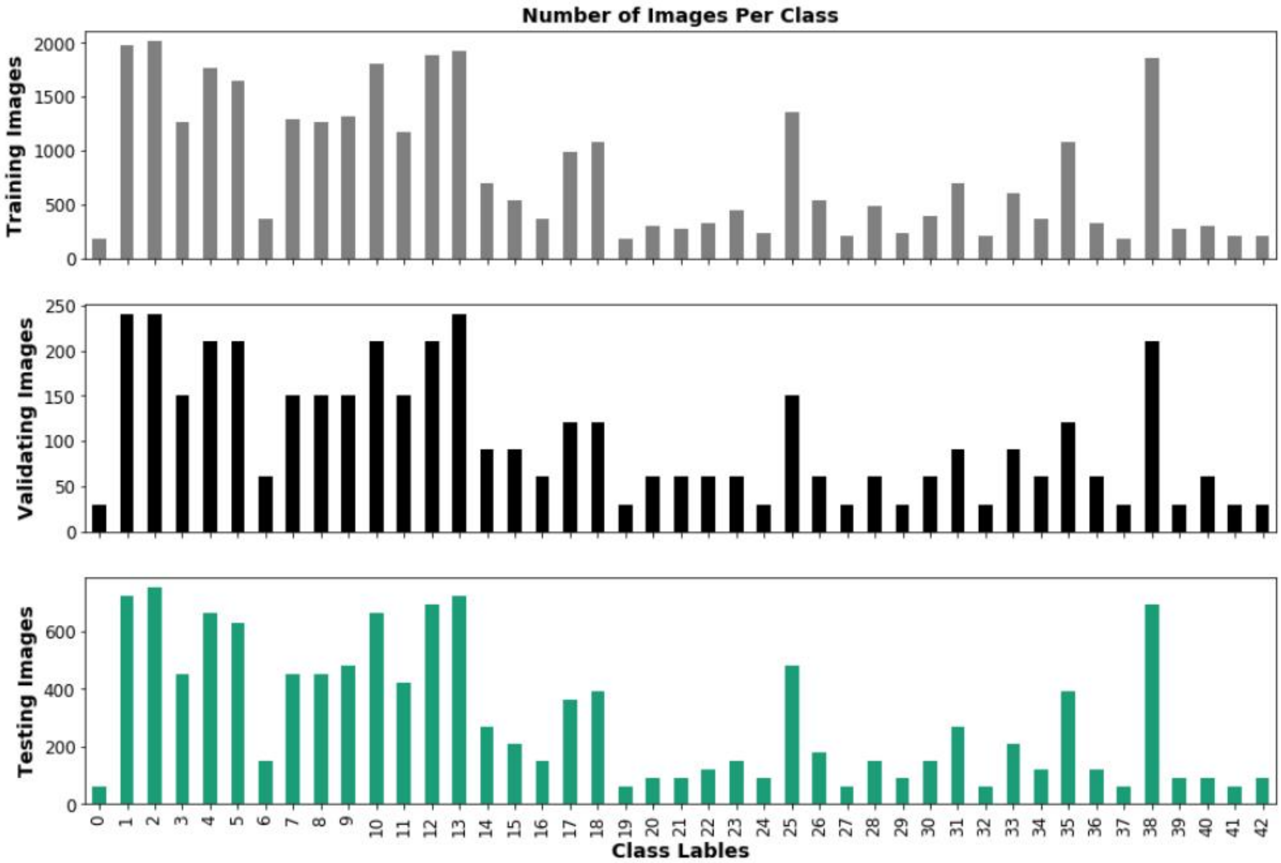
1. **Provide a basic summary of the data set. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.**

   I used the pandas library to calculate summary statistics of the traffic signs data set:

   - The size of training set is **34799**
   - The size of the validation set is **4410**
   - The size of test set is **12630**
   - The shape of a traffic sign image is **32x32x3**
   - The number of unique classes/labels in the data set is **43**

2. **Include an exploratory visualization of the dataset.**

Here is an exploratory visualization of the data set. It is a bar chart showing how the data ...



Design and Test a Model Architecture

1. **Describe how 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. (OPTIONAL: As descry bed 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, and provide example images of the additional data. Then describe the characteristics of the augmented training set like number of images in the set, number of images for each class, etc.)**

As could be seen from the above figure, the distribution of class labels is imbalanced. This will necessarily cause some biasing when training the network by giving more weights to certain images. Thus, additional images were adaptively added to every class as needed to compensate for such discrepancy. The number of additional images was heuristically determined. Two functions served that purpose

- **adjust_images( ):**

defines 13 transformations including image rotation, flipping, smoothing ..etc and returns a number of newly generated images as specified by the pivot.

- **generate_images( )**:

  generate enough images for every class as detected and required using **get_statistics( )**.

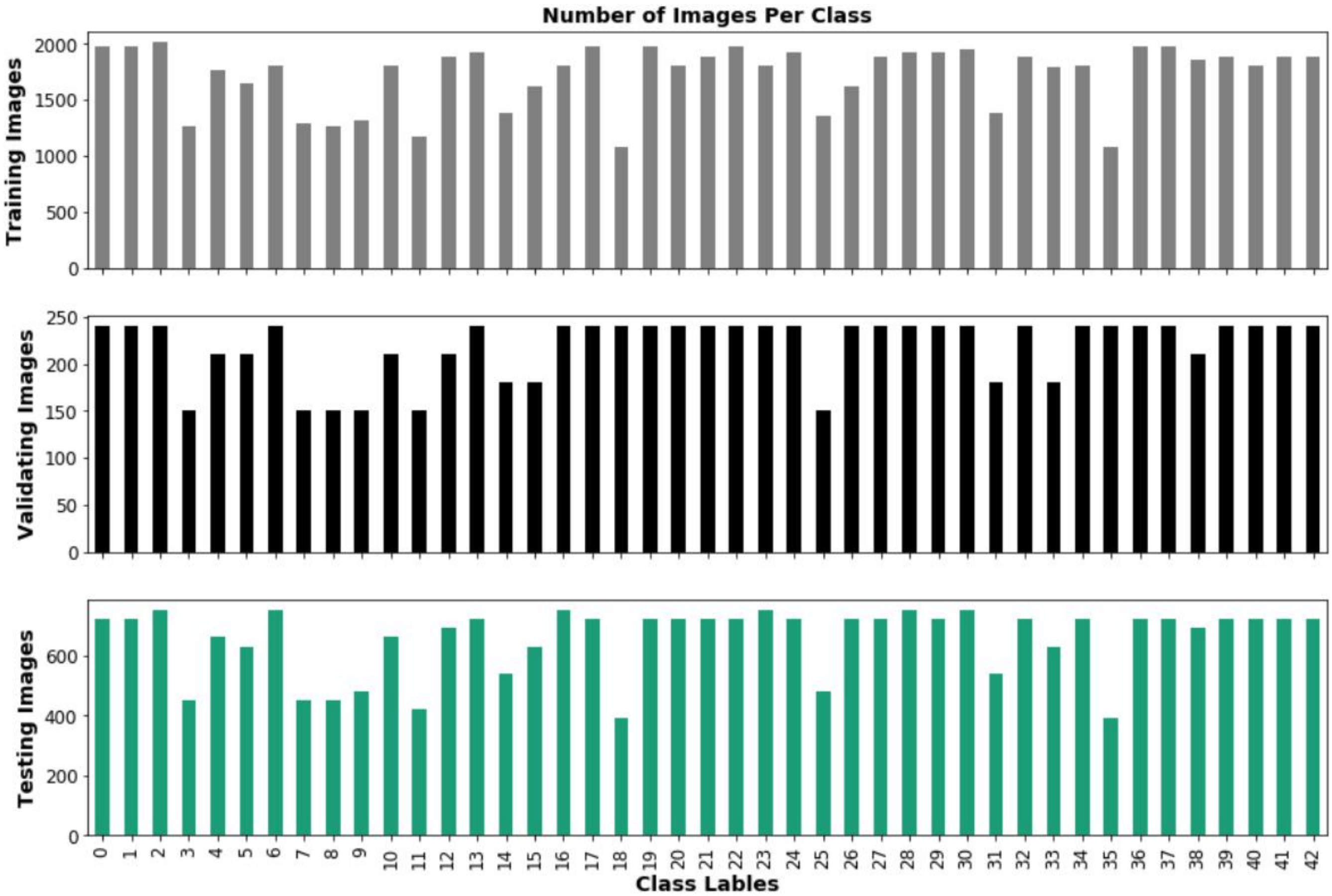
I decided to convert the images to grayscale because to reduce image dimensions as well as any additional unnecessary information. This will speed up the training process.

I also normalized the image data because reduce the gap among various image data and to drive faster training.

Here is an example of a traffic randomly selected traffic sign images

The size of all the sets were verified after that addition operation and summarized below along with the new bar plot that shows the new distribution.

```
For Training Set

The length before extendeing images = 34799 and the length of the assoc. label = 34799
The length after extendeing images = 39418 and the length of the assoc. label = 39418
--------------------------------------------------

For Validation Set

The length before extendeing images = 4410 and the length of the assoc. label = 4410
The length after extendeing images = 4980 and the length of the assoc. label = 4980
--------------------------------------------------

For Testing Set

The length before extendeing images = 12630 and the length of the assoc. label = 12630
The length after extendeing images = 15450 and the length of the assoc. label = 15450
--------------------------------------------------
```



and here is a snapshot of some randomly selected validation images after augmentation.


0: Stop
1: Dangerous curve to the left
2: Road work
3: Speed limit (50km/h)
4: Go straight or right
5: No passing for vehicles over 3.5 metric tons
6: Double curve
7: Ahead only
8: Ahead only
9: Turn left ahead
10: Speed limit (120km/h)
11: Beware of ice/snow
12: Speed limit (70km/h)
13: No passing for vehicles over 3.5 metric tons
14: Double curve
15: Speed limit (60km/h)

3. **Describe 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.**

Training Environment

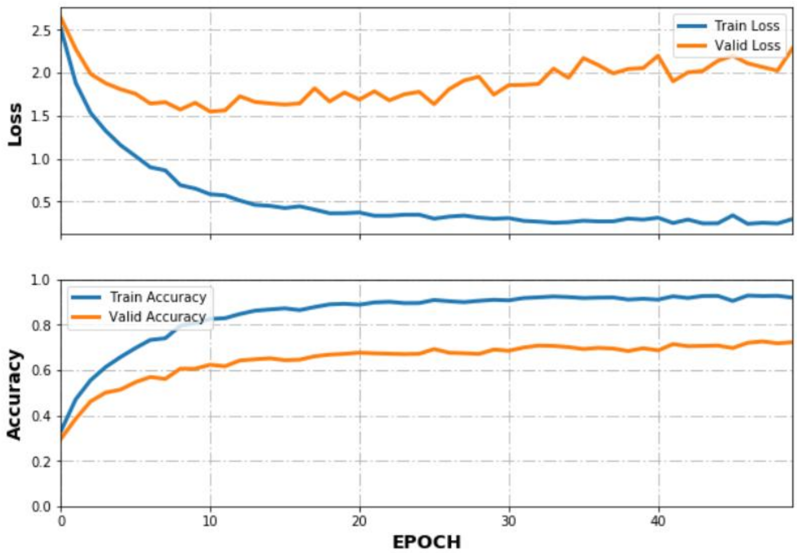
Laptop with windows 10 operating system and x64- based processor, i7 architecture with 2.2 GHz clock. 16 GB memory and NVIDIA GeForce GT 750M GPU.

Various models have been tried and tested, here I list those trials as well as some comments. For all models the input is **32x32x1**

Model 1: Adapted From LeNET

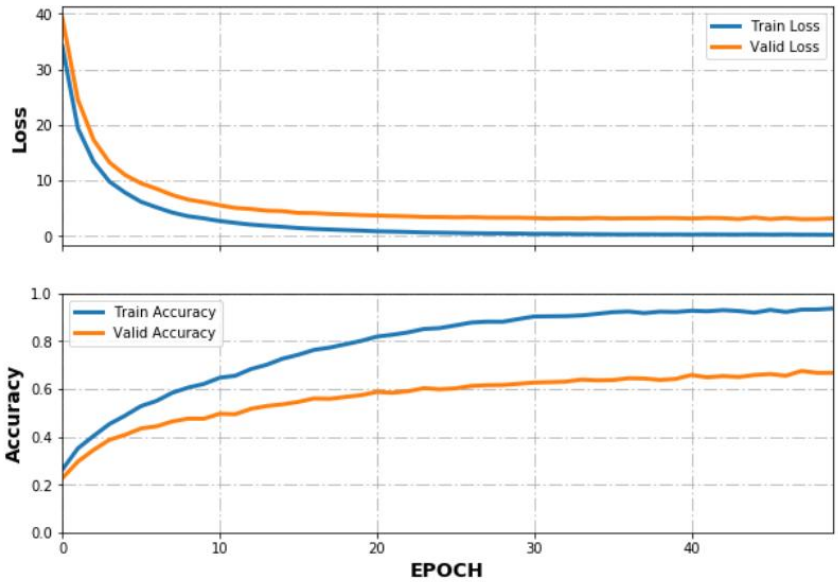
Layer	Output	characteristics
32x32x1 Input Image		
5x5 Convolution	28x28x8	1x1 stride, valid padding
RELU		
Max Pooling	14x14x8	2x2 stride
5x5 Convolution	10x10x16	1x1 stride, valid padding
RELU		
Max Pooling	5x5x16	2x2 stride
Fully Connected	400 x2000	
Fully Connected	2000x256	
Output	256x43	

The following figure shows the accuracy and loss output for both training and validation data after **50 epochs,** using **0.0001** training rate and **AdamOptimizer**



Not only this model training accuracy seems to plateau at 92% but also **overfits** the data. There is no point in examining dropout or any other regularization techniques over this model.

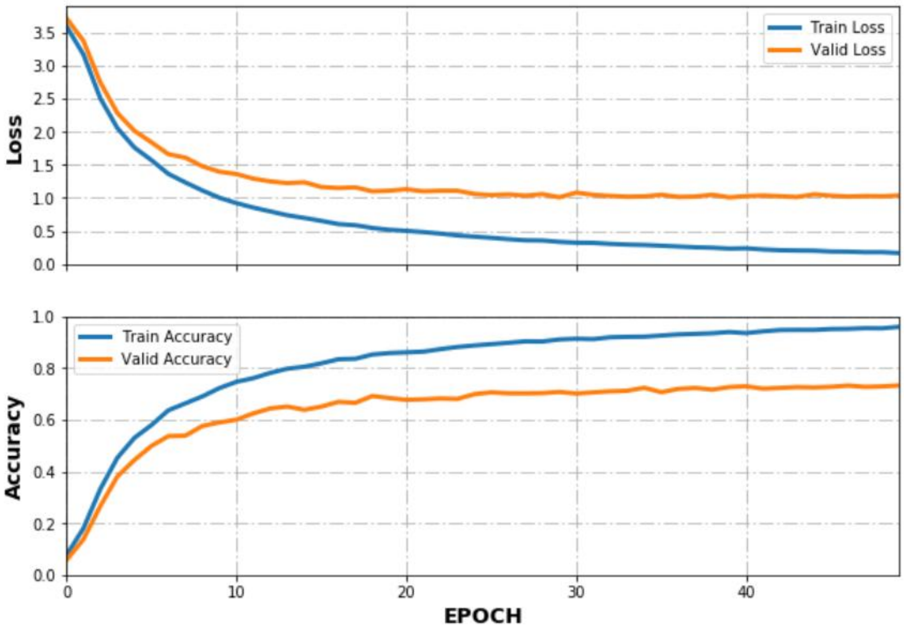
Next and given the same network structure (2 conv-nets followed by 2 fully connected), I increased the depth in both convolution layers to **32** and **64** respectively and adjusted all other parameters. The following plot was obtained for the same testing conditions as above:



This adjustment produced a smother loss curve for the validation data. The learning rate seems very low. The following suggestions will be considered for other models:

1. Experimenting with lower learning rates
2. Having large depth for conv-nets
3. Experimenting with other solvers

Following the previous suggestions and focusing on item 1 & 3, I tried **AdagradOptimizer** with **0.01** Learning rate and keeping everything the same. The following plot was obtained:



The training time took around 25 minutes with the following final output:

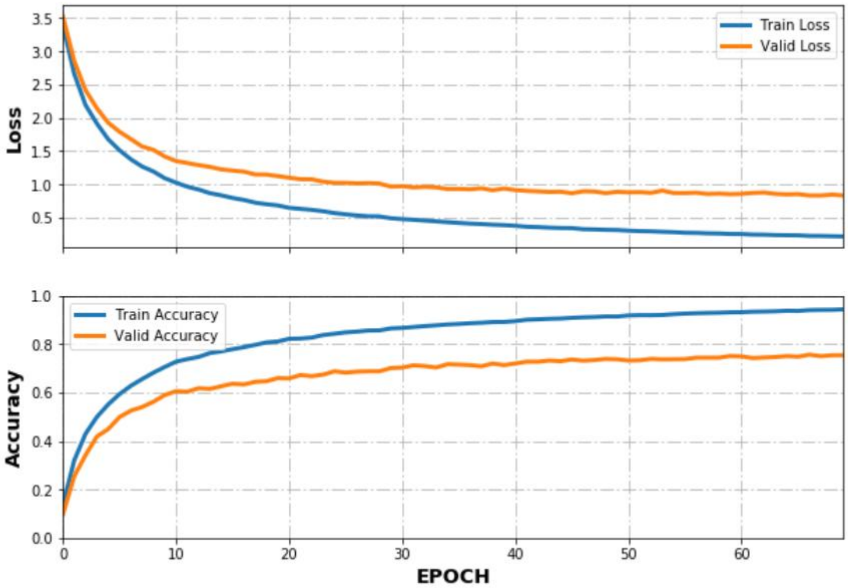
```
EPOCH 50 ...
Train loss = 0.164
Train accuracy = 0.960
Valid loss = 1.035
Valid accuracy = 0.733
Time: 1487.979 seconds
```

The Following could be concluded:

1. The smoothness in both learning and validation curves suggests that higher epochs may indeed reach higher levels of accuracy.
2. Dropout (and/or) regularization could be utilized to enhance the validation accuracy.

By adjusting keep probability in **dropout** to 0.77, and lower the number of hidden units by half the following was obtained:

```
EPOCH 70 ...
Train loss = 0.217
Train accuracy = 0.944
Valid loss = 0.834
Valid accuracy = 0.755
Time: 1460.281 seconds
```



The above results and the slow convergence rate suggest the following:

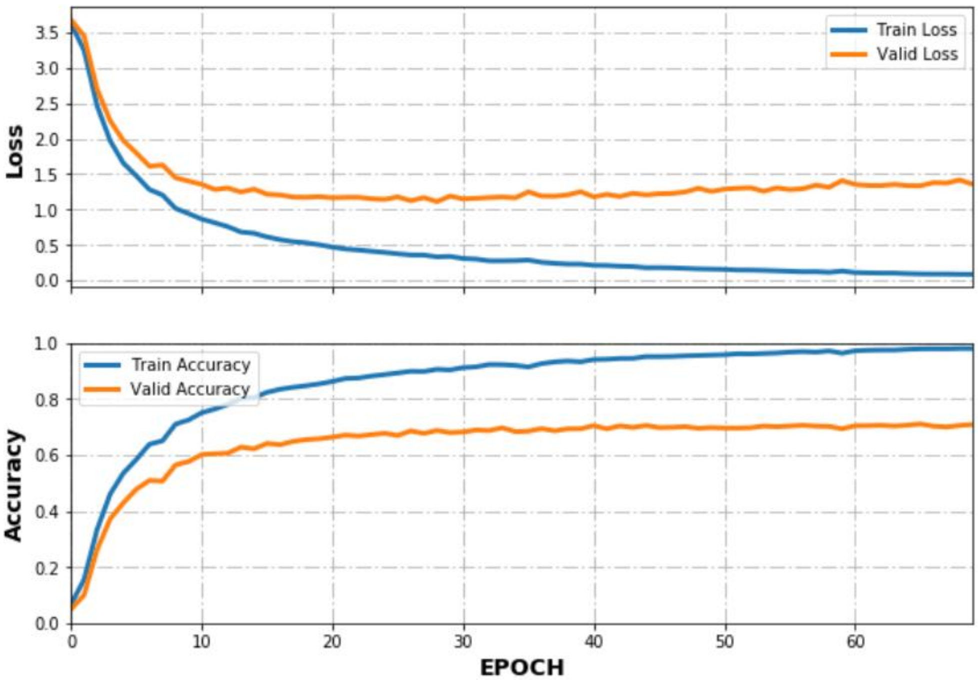
- **Increase the number of epoch to a higher number. 500 for instance, while increasing the learning rate. <u>I cannot afford doing this solution on my laptop. Already with 50 epochs lot of heat is generated</u>.**
- **Changing the model by reducing the number of nodes in the hidden layers and adding more layer**
- **Checking the effect of adding an inception layer.**

Model 2: Increasing the depth of the Network

Layer	Output	characteristics
32x32x1 Input Image		
3x3 Convolution + ReLU	30x30x16	1x1 stride, valid padding
Max Pooling	15x15x16	2x2 stride
4x4 Convolution + ReLU	12x12x24	1x1 stride, valid padding
Max Pooling	6x6x24	2x2 stride
1x1 Convolution + ReLU	6x6x32	1x1 stride, SAME padding
Max Pooling	3x3x3x32=864 ??	2x2 stride
Fully Connected	864 x 600	
Fully Connected	600x 256	
Fully Connected	256 x 128	
Fully Connected	128 x 96	
Output	96 x 43	

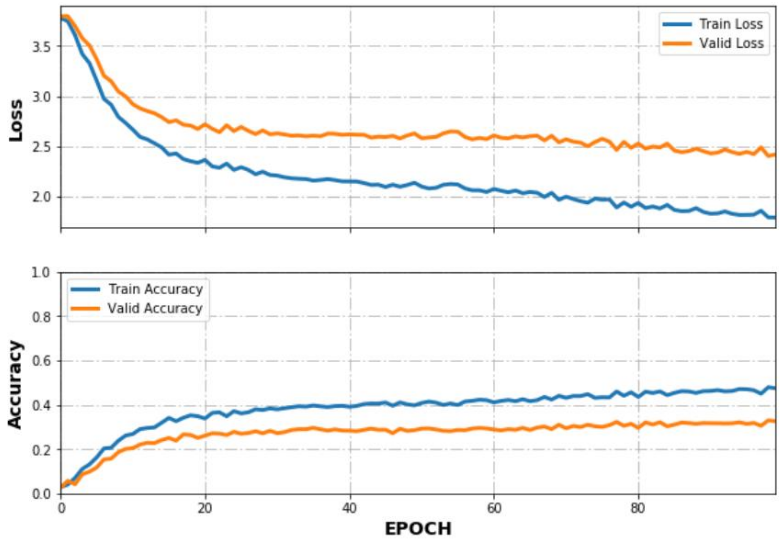
The following output was obtained at epoch 70

```
EPOCH 70 ...
Train loss = 0.082
Train accuracy = 0.981
Valid loss = 1.356
Valid accuracy = 0.709
Time: 1257.198 seconds
```



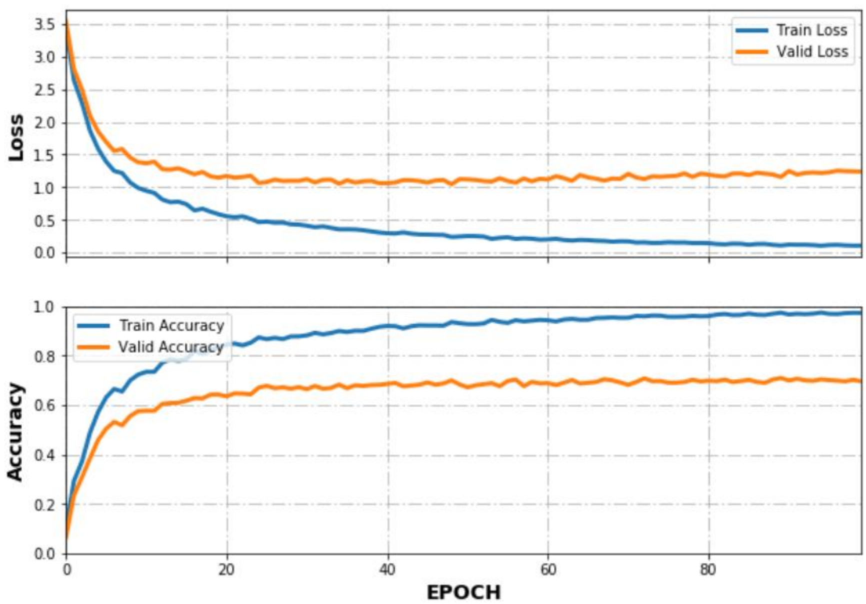
The learning rate of **0.01** seems just fine and **AdagradOptimizer** produced a very high training accuracy that is smooth and monotonically decreasing. To cope with the degradation in validation accuracy, a drop out layer was introduced after the first and second convolution layers. Keep probability was chosen to be 0.5. The following was obtained:

```
EPOCH 100 ...
Train loss = 1.787
Train accuracy = 0.475
Valid loss = 2.417
Valid accuracy = 0.326
Time: 4036.159 seconds
```



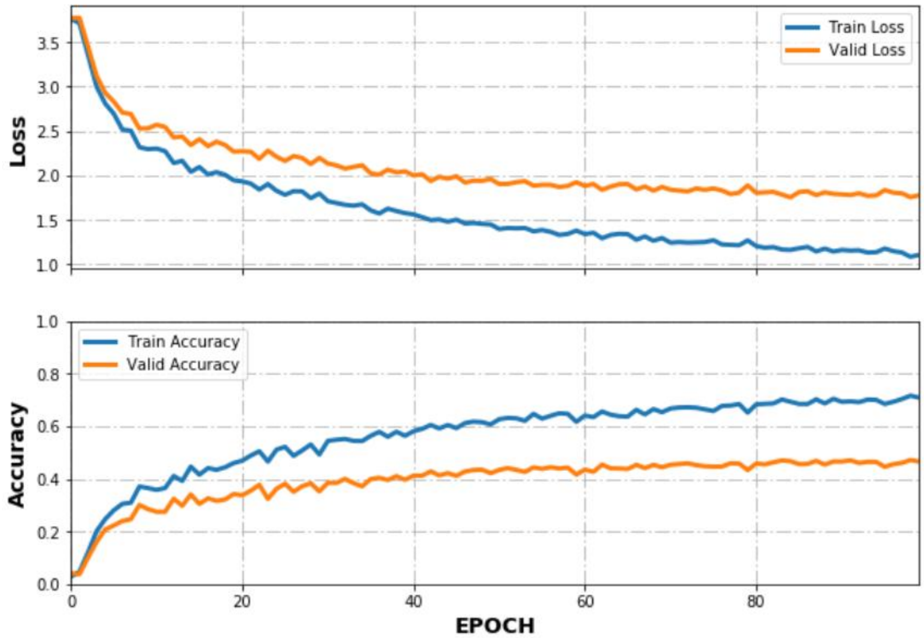
The previous figure suggests that the use of drop out is already exaggerated. In the next experiment a drop out of 0.7 was tried after applying it to the first convolution layer.

```
EPOCH 100 ...
Train loss = 0.102
Train accuracy = 0.974
Valid loss = 1.239
Valid accuracy = 0.696
Time: 2376.808 seconds
```



Next I tried lower the same drop out to as low as 0.3 and the following was obtained:

```
EPOCH 100 ...
Train loss = 1.108
Train accuracy = 0.710
Valid loss = 1.779
Valid accuracy = 0.467
Time: 8535.246 seconds
```



This may lead to good results at the long run, but I cannot try running more than 100 epochs on my laptop as the excessive generated heat may damage the computer. Therefore, another architecture is to examined with the following suggestions:

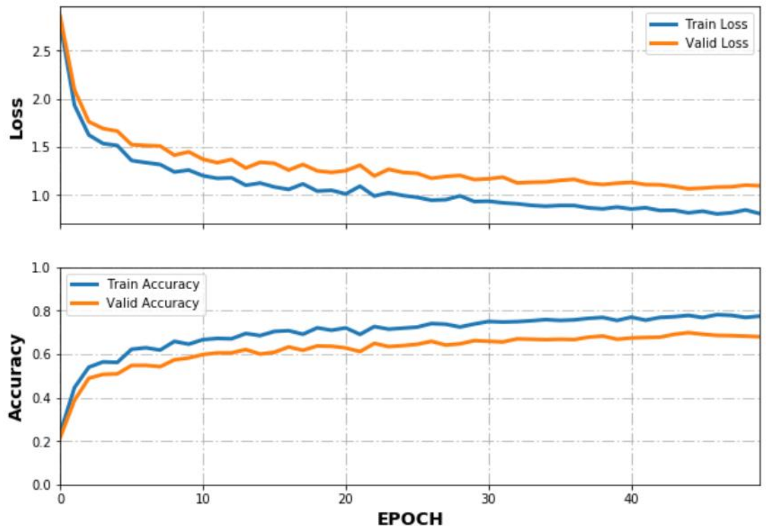
- Lowering the number of nodes at each layer.
- Reducing the depth of the network
- Trying the inception model and average lower pooling

Model 3: Examining with the inception model

Layer	Output	characteristics
32x32x1 Input Image		
3x3 Convolution + RELU	30x30x4	1x1 stride, valid padding
3x3 Convolution + RELU	28x28x8	1x1 stride, valid padding
5x5 Convolution + RELU	24x24x12	1x1 stride, valid padding
Max Pooling	12x12x12	2x2 stride
Inception Layer		
1x1 Convolution + ReLU	12x12x32	1x1 stride, SAME padding
3x3 Convolution + ReLU	12x12x32	1x1 stride, SAME padding
5x5 Convolution + ReLU	12x12x32	1x1 stride, SAME padding
Average Pooling	4x4x32	3x3 stride
Fully Connected	512 x256	
Fully Connected	256 x 94	
Output	94 x 43	

AdagradOptimizer with training rate of 0.01 was utilized here on the extended list of gray images without normalization. The following output was obtained at epoch 50

```
EPOCH 50 ...
Train loss = 0.808
Train accuracy = 0.775
Valid loss = 1.096
Valid accuracy = 0.680
Time: 1733.895 seconds
```



The behavior of the validation rate follows nicely the one for the learning rate. However, they don't provide good accuracy levels at epoch 50. L1 regularization was adapted here.

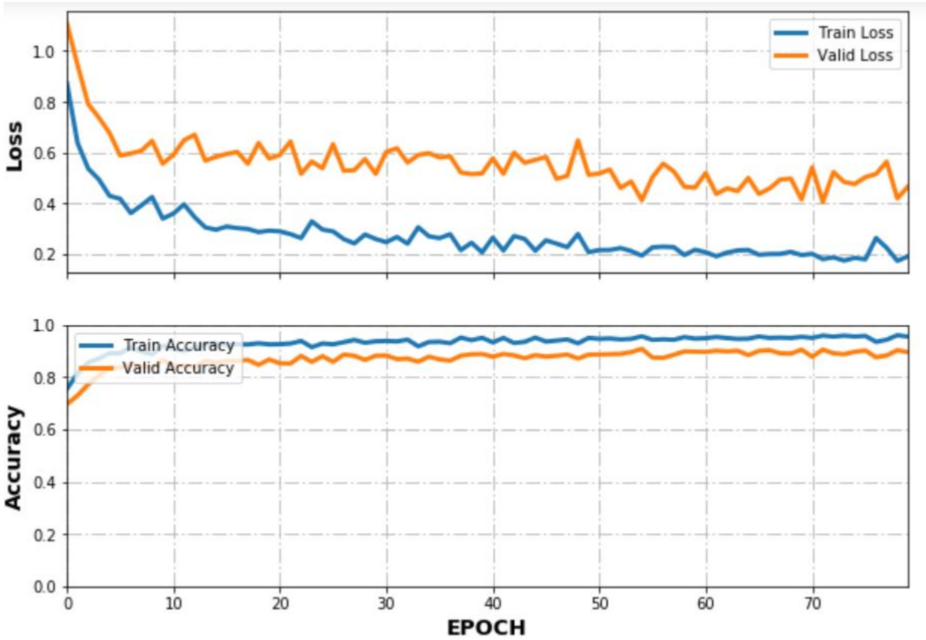
Model 4: Combination of the inception model

Lot and lot of experiments have been tried out using a combination of different models. What was interesting however, was the following experiment where I have not added any extra images and utilized the color images directly. Because of certain stability issues, I decided to go for L1 regularization with AdamOptimizer and 0.001 learning rate.

Layer	Output	characteristics
32x32x1 Input Image		
3x3 Convolution	30x30x4	1x1 stride, valid padding
Inception Layer		
1x1 Convolution + ReLU	30x30x8	1x1 stride, SAME padding
3x3 Convolution + ReLU	30x30x8	1x1 stride, SAME padding
5x5 Convolution + ReLU	30x30x8	1x1 stride, SAME padding
Average Pooling	15x15x8	2x2 stride
Inception Layer		
1x1 Convolution + ReLU	15x15x8	1x1 stride, SAME padding
3x3 Convolution + ReLU	15x15x8	1x1 stride, SAME padding
5x5 Convolution + ReLU	15x15x8	1x1 stride, SAME padding
Average Pooling	7x7x8	2x2 stride
3x3 Convolution	5x5x16	1x1 stride, valid padding
Fully Connected	400 x 400	
Fully Connected	400 x 86	
Output	86 x 43	

Next a combination of regularization and drop out values are examined. The following is the best that could be achieved to the best of my educated guess.

```
EPOCH 80 ...
Train loss = 0.192
Train accuracy = 0.955
Valid loss = 0.466
Valid accuracy = 0.896
Time: 1824.964 seconds
```



Clearly, this model, performs much better than the previous ones. It could be further smoothed by creating balanced colored classes in the training set. Also drop out may be applied to push the learning. In addition, if the learning rate is lowered, better results may be achieved.

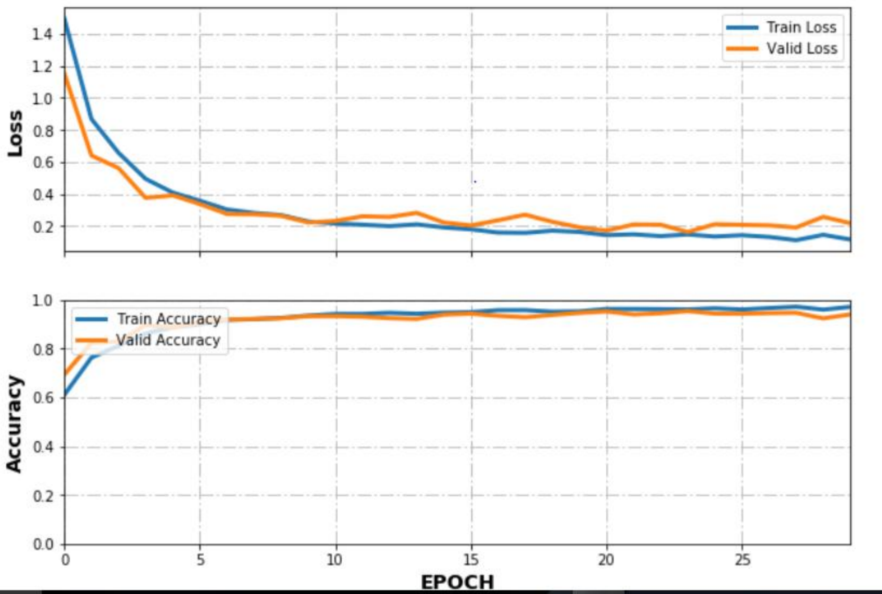
FINAL MODEL

Model 5: Simple Conv Layers and L2 Regularization

Focusing back on gray scale images but this time by applying L2 regularization. The results obtained achieves the required validation accuracy. The learning rate was 0.0004 and the optimization was done using AdamOptimizer.

Layer	Output	characteristics
32x32x1 Input Image		
3x3 Convolution + ReLU	30x30x8	1x1 stride, valid padding
Average Pooling		k =2
4x4 Convolution + ReLU	15x15x16	1x1 stride, valid padding
Average Pooling		k =1
2x2 Convolution + ReLU	12x12x24	1x1 stride, valid padding
Average Pooling		k =2
Fully Connected + ReLU	600x600	
Fully Connected + ReLU	600 x 215	
Output Connected	215 x 43	

```
EPOCH 30 ...
Train loss = 0.117
Train accuracy = 0.972
Valid loss = 0.219
Valid accuracy = 0.941
Time: 442.872 seconds
```



The above model can surely be enhanced further by trying different optimizers and various training rates.

4. **Describe how 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.**

To details were highlighted in the previous questions.

- The learning rate was 0.0004,
- Epochs: 30
- Batch Size: 128
- Optimizer: AdamOptimizer
- Loss Operation: Cross Entropy with L2 regularization

5. **Describe the approach taken for finding a solution and getting the validation set accuracy to be at least 0.93. 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 answer to this question was also detailed when answering Q3. Here I reemphasize some points:

- training set accuracy: **0.972**
- validation set accuracy: **0.941**
- test set accuracy: **0.911**

If an iterative approach was chosen:

- I started with LeNET architecture and built things from there. For this problem, that architecture underfits the data, so I had to add extra layer and adjusted several parameters like the pooling size and others. The details are previously detailed
- The convolution layers summarize values of a group of pixels. They therefore, reduce the complexity of the system and helps in better classifications. Drop out is somehow useful as it forces the network to learn about certain parameters in the absence of others. Nevertheless, placing them at a perfect position is challenging.

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.**

Here are six German traffic signs that I found on the web:

The first image might be difficult to classify because the number may be mistaken with other possible speeds like 20 or 90. The last image could also be difficult to classify because it might look as the caution sign, image 2. Image 5 is challenging as well because the sign is tilted.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. 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).

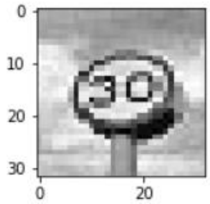
Here are the results of the prediction:

The model could correctly guess 5 of the 6 traffic signs, which gives an accuracy of 83.3%. This is in good accordance with the test accuracy.

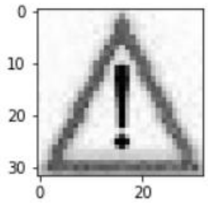
Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part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

The model is certain when predicting the General Caution Sign with probability 1. It is almost sure when predicting dangerous curve, straight or left and the trafic sign with probability > 0.9. it favours 0.6 of the time and mispredict speed limit 30 with 20. The following table summarizes the results.

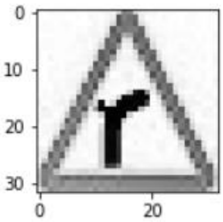
Speed limit (20km/h): 0.949
Keep left: 0.042
Speed limit (30km/h): 0.005
End of speed limit (80km/h): 0.001
Speed limit (50km/h): 0.001



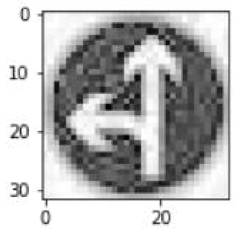
General caution: 1.000
Pedestrians: 0.000
Traffic signals: 0.000
Right-of-way at the next intersection: 0.000
Road narrows on the right: 0.000



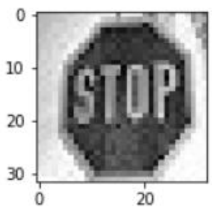
Dangerous curve to the right: 0.935
Dangerous curve to the left: 0.065
Children crossing: 0.000
Slippery road: 0.000
Beware of ice/snow: 0.000



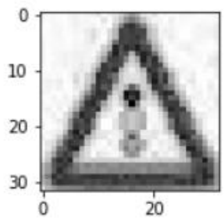
Go straight or left: 0.853
Go straight or right: 0.146
Turn right ahead: 0.000
Keep left: 0.000
Speed limit (20km/h): 0.000



Stop: 0.613
Speed limit (20km/h): 0.116
Go straight or left: 0.084
Turn left ahead: 0.057
Turn right ahead: 0.038



Traffic signals: 0.947
General caution: 0.051
Road narrows on the right: 0.001
Pedestrians: 0.000
Dangerous curve to the left: 0.000



Conclusion

This project was a chance to strengthen my knowledge about deep neural network in general and convolutional variation in particular. I have already appreciated the importance of experience when trying out so many variations to come up with acceptable accuracy levels in the end. Small variations in some occasions caused the network to either fluctuate, or saturate. I also experimented various famous models.

In the end, I would like to thanks for all the exerted efforts by the Udacity team who do not save any effort to make this track enjoyable.

(Optional) Visualizing the Neural Network **(See Step 4 of the Ipython notebook for more details)**

> **Discuss the visual output of your trained network's feature maps. What characteristics did the neural network use to make classifications?**

To be continued in future work