Traffic Sign Recognition

Task Description

In this project, you will use what you've learned about deep neural networks and convolutional neural networks to classify traffic signs. You will train a model, so it can decode traffic signs from natural images by using the German Traffic Sign Dataset.

Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- 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
- Analyse the SoftMax probabilities of the new images
- Summarize the results with a written report

Load the data :

In this step, the provided data is loaded using the pickle library. The images have labels to recognize what they represent. The labels are numbers, but there is a .csv file containing the mapping between the labels and a text name of the image to make it more human-friendlier.

Data Set Summary & Exploration:

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

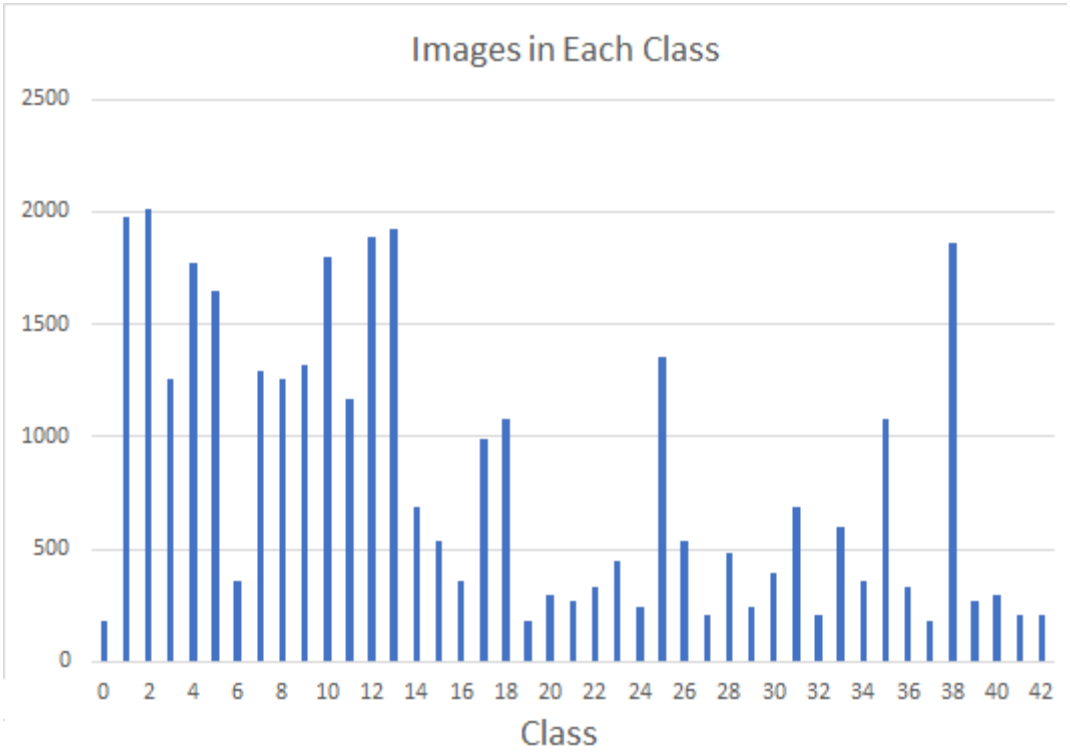
- The size of training set is 34799
- The size of the validation set is 12630
- The size of test set is 4410
- The shape of a traffic sign image is 32x32x3
- The number of unique classes/labels in the data set is 43

Visualization of Training data:

By observing the above images, it can be noted that the images are

- rotated from an ideal traffic sign image.
- transformed from an ideal traffic sign image.
- noisy image.
- blurred images.

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



It could be observed that there is imbalance of data in the training set. Imbalanced data sets are a special case for classification problem where the class distribution is not uniform among the classes.

Since, there is an imbalance of data distribution throughout the classes, the model starts overfitting. This happens because the classes having a greater number of images will be a dominant class and influences the weights of the network drastically. So, any other classes (minority in count) will be misclassified. Due to this reason, we create images by using data augmentation method.

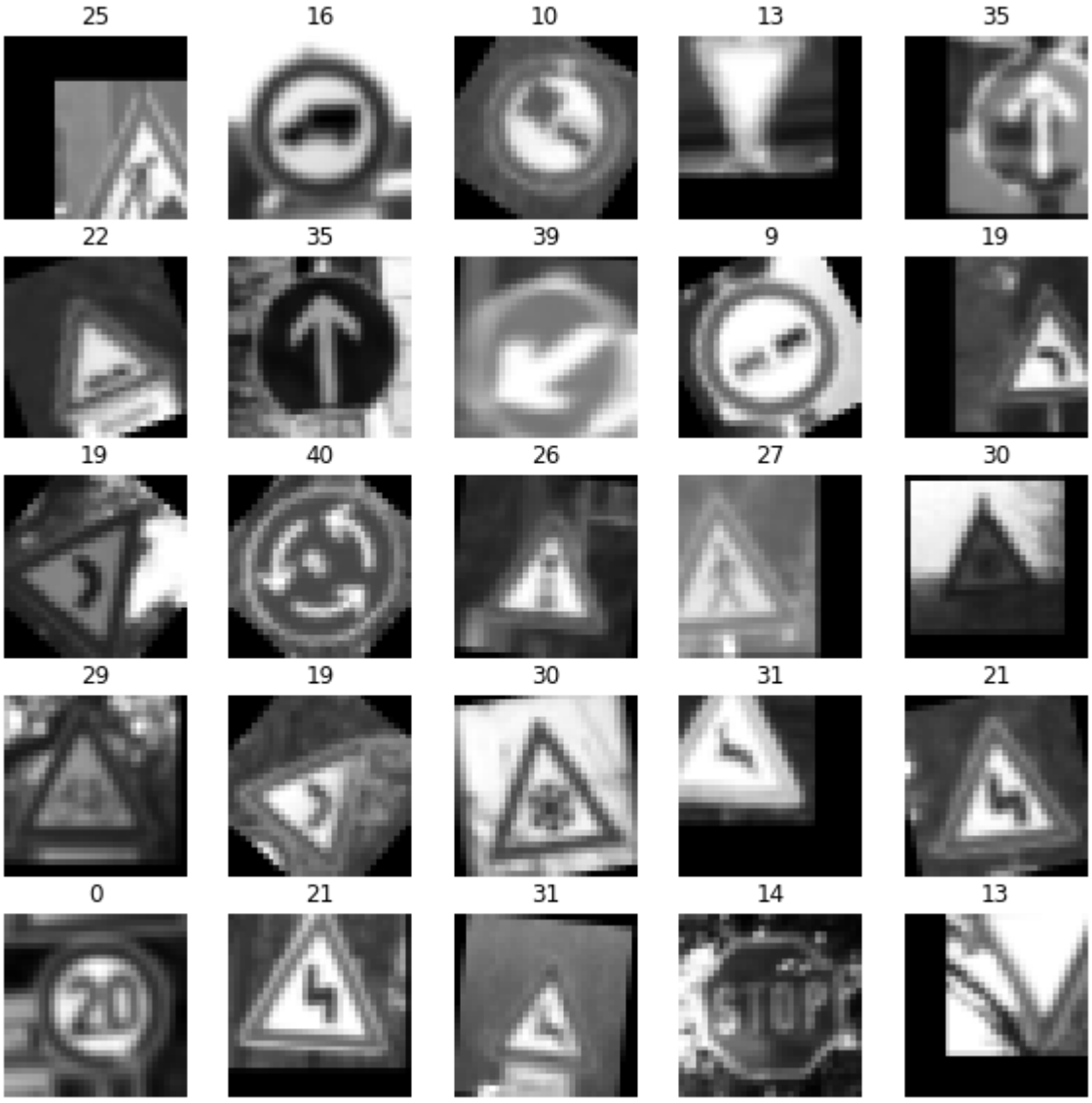
Data augmentation is an automatic way to boost the number of different images you will use to train our Deep learning algorithms. There are a lot of good Python libraries for image transformation like OpenCV or Pillow. We do that by rotating the images, scaling the images and adding random noise in each image. This method generates data by manipulating existing data. Finally, we make sure that all the classes have equally distributed images in it.

The strategies used to generate augmented images are:

- Translate image by tuned threshold
- Rotate image by tuned angle
- Blur image within tuned threshold
- Crop image by tuned threshold
- Sharpen image by tuned threshold

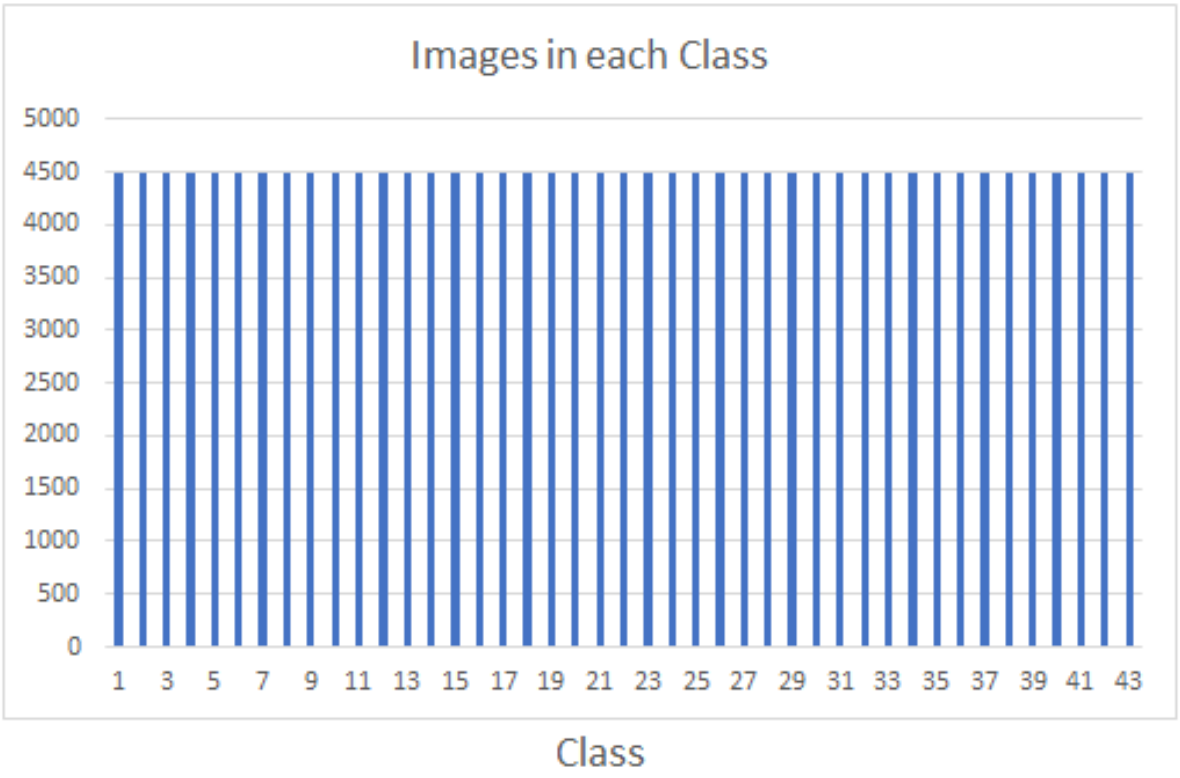
***Gray scaling & Normalization**:*

In their paper "Traffic Sign Recognition with Multi-Scale Convolutional Networks" published in 2011, P. Sermanet and Y. LeCun stated that using grayscale images instead of colour improves the CNNs accuracy. We will use OpenCV to convert the training images into grey scale. Normalization is a process that changes the range of pixel intensity values. We take a strategy of dividing 255 from each pixel value as to normalize the images. After normalizing the range of the image values would be 0 to 1. Below are the random images from training data.



After data augmentation and normalization of images, we make sure that each class has uniform number of images (4500).

Updated class distribution can be seen in the below image.



***Design and Test a Model Architecture*:**

In this step, we will design and implement a deep learning model that learns to recognize traffic signs from our dataset German Traffic Sign Dataset.

Layer	Description
Input	32x32x3 RGB image
Convolution + RELU	5x5 filter with 1x1 stride, valid padding, outputs 28x28x6
Convolution + RELU	5x5 filter with 2x2 stride, valid padding, outputs 14x14x10
Convolution + RELU	5x5 filter with 2x2 stride, valid padding, outputs 8x8x16
Max Pooling	2x2 ksize with 2x2 stride, valid padding, outputs 4x4x16
Flatten	outputs 256
Fully Connected + RELU	Input 256 and outputs 120
Dropout	keep_prob=0.5
Fully Connected + RELU	Inputs 120 and outputs 100
Fully Connected + RELU	Inputs 100 and outputs 84
Fully Connected + RELU	Inputs 84 and outputs 43

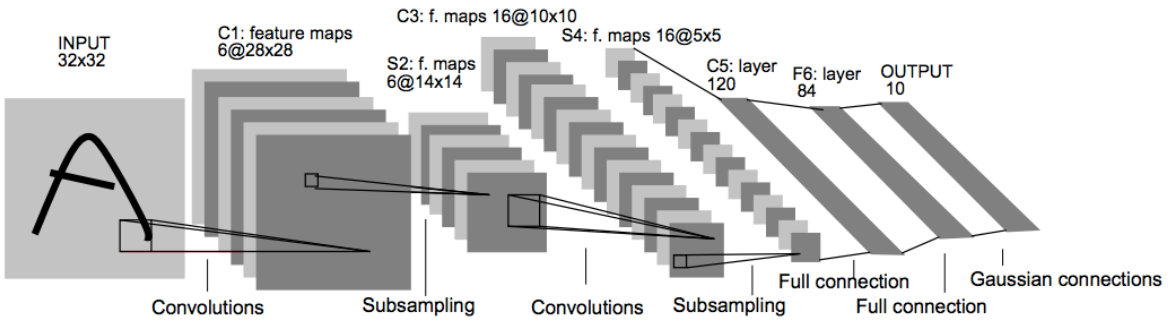
To train the model, the hyper parameters were tuned to get a better accuracy.

Hyper parameters	Value
EPOCH	150
batch Size	128
learning rate	0.001
mu	0
sigma	0.1
dropout	0.5

My final model results were:

- training set accuracy of .99
- validation set accuracy of .971
- test set accuracy of .955

The first architecture was a simple Lenet architecture explained in Yann LeCunns paper. However, his proposal gave a test accuracy around 0.8976. As the number of images across each class is not uniform, overfitting takes place. Below is the architecture used by Yann LeCunn for MNIST classifier.



Imbalance of training data is a major problem with this dataset. Some classes have 2100 images while many of them have lesser than 500. This makes the model to overfit to the dominant classes. A high accuracy on the training set but low accuracy on the validation set indicates over fitting. Thus, prediction is vastly affected when the model evaluates an unseen image.

There is imbalance of training data. So, we need to generate training dataset so that all the classes will have at least a minimum set of data for the model to not to over fit.

Data augmentation is used to generate training data by rotating the image, zooming the image and adding random noise in the image. These images are generated only for the classes which do not have enough training images.

After data augmentation is completed, we manually tune the hyper parameters listed below. This is done by changing the below mentioned parameter and observing the validation & test accuracy for the mode.

Hyper-parameters tuned for Accuracy:

Learning rate:

Learning rate controls how much to update the weight in the optimization algorithm. We can use fixed learning rate, gradually decreasing learning rate, momentum-based methods or adaptive learning rates, depending on our choice of optimizer such as SGD, Adam, Adagrad, AdaDelta or RMSProp.

Number of epochs:

Number of epochs is the number of times the entire training set pass through the neural network. We should increase the number of epochs until we see a small gap between the test error and the training error.

Batch size:

Mini-batch is usually preferable in the learning process of convnet. A range of 16 to 128 is a good choice to test with. We should note that convnet is sensitive to batch size.

Activation function:

Activation function introduces non-linearity to the model. Usually, rectifier works well with convnet. Other alternatives are sigmoid, tanh and other activation functions depending on the task.

Number of hidden layers and units:

It is usually good to add more layers until the test error no longer improves. The trade-off is that it is computationally expensive to train the network. Having a small number of units may lead to underfitting while having more units are usually not harmful with appropriate regularization.

Weight initialization:

We should initialize the weights with small random numbers to prevent dead neurons, but not too small to avoid zero gradient. Uniform distribution usually works well.

Dropout:

Dropout is a preferable regularization technique to avoid overfitting in deep neural networks. The method simply drops out units in neural network according to the desired probability. A default value of 0.5 is a good choice to test with. In this model, adding dropout improved the accuracy of the model as it induces nonlinearity in the model.

Prediction with new images:

11 images from downloaded from internet. These images are not seen by the model during the training phase. We analyse the performance of the model based on these 11 images given below.

Image Name	Actual Result	Predicted Result
1	*No entry*	*No entry*
2	*Speed limit (30 Km/hr)*	*Speed limit (30 Km/hr)*
3	*No vehicles*	*No vehicles*
4	*Stop*	*Stop*
5	*Children zone*	*Children zone*
6	*Ahead only*	*Ahead only*
7	*Yield*	*Yield*
8	*Ahead or left turn*	*Ahead or left turn*
9	*Speed limit (60 Km/hr)*	*Speed limit (80 Km/hr)*
10	*General Caution*	*General Caution*
11	*Left Turn*	*Left Turn*

Analyzing Softmax probabilities:

Image 1:
Original Image Result: No entry
Predicted Image Result: No entry

Sign	Probability (in percentage)
No entry	100.0%
Vehicles over 3.5 metric tons prohibited	8.16541387456e-12%
Go straight or left	4.00861899826e-13%
Keep right	1.71183706926e-13%
Turn left ahead	1.3581114518e-13%

Image 2:
Original Image Result: Speed limit (30km/h)
Predicted Image Result: Speed limit (30km/h)

Sign	Probability (in percentage)
Speed limit (30km/h)	95.9677278996%
Speed limit (50km/h)	3.49726639688%
Speed limit (70km/h)	0.27938215062%
Speed limit (20km/h)	0.179711391684%
Speed limit (80km/h)	0.0434404588304%

Image 3:
Original Image Result: No vehicles
Predicted Image Result: No vehicles

Sign	Probability (in percentage)
No vehicles	99.9654054642%
Speed limit (30km/h	0.0109753447759%
Speed limit (70km/h)	0.00632794763078%
Yield	0.00533499660378%
Speed limit (50km/h)	0.00424968347943%

Image 4:
Original Image Result: Stop
Predicted Image Result: Stop

Sign	Probability (in percentage)
Stop	99.832957983%
Speed limit (50km/h)	0.0761548464652%
Priority road	0.025802970049%
Keep right	0.0215849853703%
Speed limit (30km/h)	0.0179546317668%

Image 5:
Original Image Result: Children crossing
Predicted Image Result: Children crossing

Sign	Probability (in percentage)
Children crossing	99.9964356422%
Road work	0.00124022271848%
Right-of-way at the next intersection	0.000802032718639%
Beware of ice/snow	0.000777662262408%
Pedestrians	0.000555409860681%

Image 6:
Original Image Result: Ahead only
Predicted Image Result: Ahead only

Sign	Probability (in percentage)
Ahead only	99.9994158745%
Go straight or right	0.000364334641745%
Turn left ahead	6.91252182605e-05%
Go straight or left	6.64285892071e-05%
Priority road	4.44358278173e-05%

Image 7:
Original Image Result: Yield
Predicted Image Result: Yield

Sign	Probability (in percentage)
Yield	99.9232053757%
Traffic signals	0.0225453288294%
Priority road	0.0148398627061%
No vehicles	0.0123301622807%
Bicycles crossing	0.00869645082275%

Image 8:
Original Image Result: Go straight or left
Predicted Image Result: Go straight or left

Sign	Probability (in percentage)
Go straight or left	99.9896883965%
Turn left ahead	0.00449770777777%
Ahead only	0.00407699189964%
Dangerous curve to the left	0.00125277010739%
Double curve	0.000278687548416%

Image 9:

Original Image Result: Speed limit (60km/h)
Predicted Image Result: Speed limit (80km/h)

Sign	Probability (in percentage)
Speed limit (80km/h)	84.1474533081%
Speed limit (60km/h)	15.075071156%
Speed limit (100km/h)	0.452945288271%
Speed limit (50km/h)	0.151953124441%
Speed limit (30km/h)	0.117127748672%

Note: The result of this image is not predicted correctly by the model. Yet all the top 5 probabilities go to the speed limit signs. The digits are not rightly identified and so this is not predicted correctly. By tuning the image sharpen / blur parameters, this could also be predicted correctly. However, due to limitation of GPU time, I have proceeded with the obtained model parameters.

Image 10:
Original Image Result: General caution
Predicted Image Result: General caution

Sign	Probability (in percentage)
General caution	100.0%
Bumpy road	2.63030763836e-08%
Traffic signals	1.39174644032e-08%
Pedestrians	5.16172445498e-11%
Bicycles crossing	3.48437167248e-12%

Image 11:
Original Image Result: Turn left ahead
Predicted Image Result: Turn left ahead

Sign	Probability (in percentage)
Turn left ahead	98.021709919%
Ahead only	1.8655044958%
Keep right	0.106045417488%
Go straight or left	0.00189092552318%
Turn right ahead	0.00158489383466%

Out of 11 completely unseen images, the model could predict 10 correctly (accuracy = 91%). This shows that the model is robust enough and with further fine tuning of hyper parameters, the accuracy could be furthermore improved.