Capstone Project 2:Report

Computer Vision for Self Driving Cars to detect traffic sign.

The traffic sign is detected using the convoluted neural network. A model is trained using the German traffic sign dataset. I have used Tensorflow for model development and trained it on GPU.

The model is trained based on previously available dataset containing images of traffic signs and the neural network is used to predict the new traffic signs based on the training set. The model is trained using the Adam Optimizer.

Data Collection:

The data is a dictionary with 4 key/value pairs:

- 'features' is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
- 'labels' is a 1D array containing the label/class id of the traffic sign. The file signnames.csv contains id -> name mappings for each id.
- 'sizes' is a list containing tuples, (width, height) representing the original width and height the image.
- 'coords' is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image. THESE COORDINATES ASSUME THE ORIGINAL IMAGE. THE PICKLED DATA CONTAINS RESIZED VERSIONS (32 by 32) OF THESE IMAGES

The model is trained based on previously available dataset containing images of traffic signs and the neural network is used to predict the new traffic signs based on the training set. The model is trained using the Adam Optimizer.

There are several steps involved:

- Load the data set
- Explore, summarize and visualize the data set
- Design, train and test a model architecture

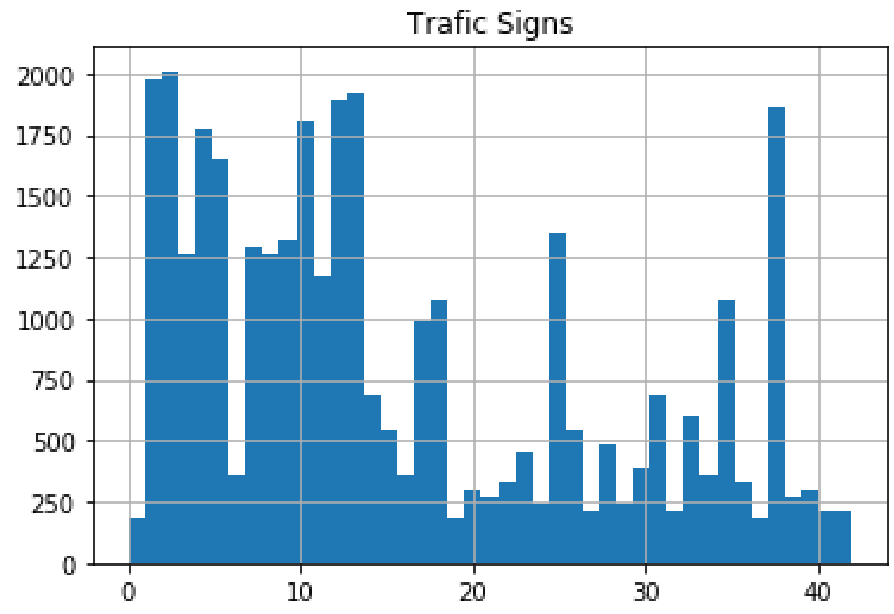
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images

Exploratory Data Analysis:

The German dataset is visualized and random traffic signs are plotted and its count is also determined.

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

Here is an exploratory visualization of the data set. It is a bar chart showing how the data is distributed. We see that the distribution is uneven.



Traffic signs summary statistics: count 43.0 mean 809.0 std 627.0 min 180.0 50% 540.0 max 2010.0

Designing and testing the model architecture:

Design and implementation of a deep learning model to recognize traffic signs.

The LeNet-5 implementation shows a solid starting point. I have changed the number of classes and preprocessing steps.

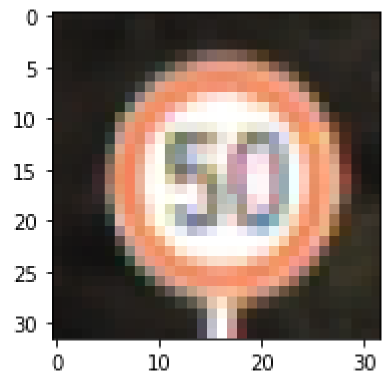
With the LeNet-5 solution I got a validation set accuracy of 0.946 and a test set accuracy of 0.942.

There are various aspects to consider when thinking about this problem:

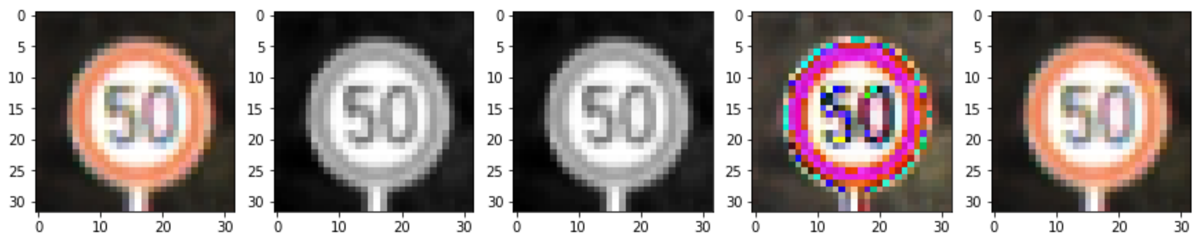
- Neural network architecture (is the network over or underfitting?)
- Preprocessing techniques (normalization, rgb to grayscale, etc)
- Number of examples per label (some have more than others)
- Data augmentation - to generate fake data

1. Pre-processing refers to techniques such as converting to grayscale, normalization

As a first step, I decided to convert the images to grayscale because the image color is not important. Also, I decided to implement normalisation, dividing the image by 255.



Here is an example of a traffic sign image before gray-scaling with normalisation.



Here is an example of a traffic sign image after gray-scaling with normalisation. I tried different normalisation (diving by 255 and dividing by 128 after subtraction 128).

2. Set up training, validation and testing data.

My final training set had 34799 number of images. My validation set and test set had 4410 and 12630 number of images.

3. Final model architecture.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 RGB image
Convolution 3x3	1x1 stride, valid padding, outputs 30x30x10
RELU	
Max pooling	2x2 stride, outputs 15x15x10
Convolution 4x4	1x1 stride, valid padding, outputs 12x12x30
RELU	
Max pooling	2x2 stride, outputs 6x6x30
Dropout	Keep prob 50%
Flatten	Output 1080
Fully connected	Output 270
RELU	

Fully connected	Output 129
RELU	
Fully connected	Output 43
Softmax	

4. Training model.

To train the model, I used an AdamOptimizer.
Here are the following steps:

- Run the training data through the training pipeline to train the model.
- Shuffle the training set before each epoch
- After each epoch, measure the loss and accuracy of the validation set.
- Save the model after training.

5.The approach taken for finding a solution.

The final model results were:

- validation set accuracy of 0.946
- test set accuracy of 0.942

Steps to find final approach:

- The first architecture that I was tried was LeNet Architecture. Sign recognition is similar to MNIST case, so i decided to try it.
- LeNet was pretty good but validation accuracy was less than 90%. So, I decided to abandon it.
- I tried different approaches: to include/exclude colors, to add more layers and etc. But the model was overfitting. So, I decided to stay with LeNet architecture, but to increase number of neurons in layers 3-4 times because

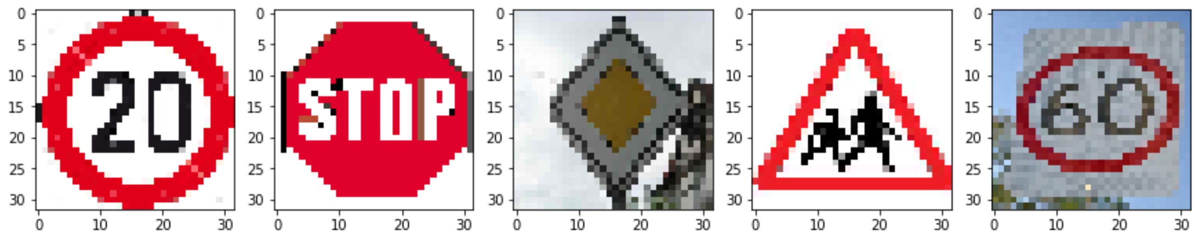
we have more number of labels than in MNIST data (43 vs 10) and to protect model from overfitting I decided to add dropout after 2nd convolution layer with 50% keep ratio. It helped to increase accuracy to 94%.

- Also, in the future I want to increase number of observation by adding to the training sample modified images (by different angles, noise and etc.)

Test the Model on New Images

1. New five German traffic signs found on the web.

Here are five German traffic signs that I found on the web. They were in a much bigger resolution so, I used TF `tf.image.resize_nearest_neighbor` function to resize them to (32, 32) format.



2. Model's predictions on the new traffic signs.

Here are the results of the prediction:

Image	Prediction
Speed limit (100km/h)	Speed limit (100km/h)
Stop Sign	Stop Sign
Priority road	Priority road

Children crossing	Children crossing
Speed limit (60km/h)	Speed limit (60km/h)

The model was able to correctly guess 5 of the 5 traffic signs, which gives an accuracy of 100%. This compares favorably to the accuracy on the test set of 94.4%

3. Softmax probabilities for each prediction.

For each of the new images I printed out the model's softmax probabilities to show the certainty of the model's predictions (the output is limited up to top 5 probabilities for each image). tf.nn.top_k could prove helpful here.

tf.nn.top_k will return the values and indices (class ids) of the top k predictions. So if k=3, for each sign, it'll return the 3 largest probabilities (out of a possible 43) and the corresponding class ids.

Top 5 Probabilities for the prediction:

Image	**Top-5 Probability**
1 image	1. 0. 0. 0. 0.
2 image	1. 0. 0. 0. 0.
3 image	1. 0. 0. 0. 0.
4 image	1. 0. 0. 0. 0.
5 image	0.82 0.07 0.06 0.02 0.01

Top 5 labels for the prediction:

Image	**Top-5 Pred. Lab.**
1 image	0 1 29 38 4
2 image	14 13 1 38 12
3 image	12 40 35 13 15
4 image	28 30 3 11 29
5 image	3 2 40 16 5

Correct labels: 0, 14, 12, 28, 3