

 writeup_template.md

Traffic Sign Recognition

Writeup

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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

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 Numpy library to calculate a summary of statistics fo the traffic signs data set:

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

This is the code I've used:

```
#### Replace each question mark with the appropriate value.
#### Use python, pandas or numpy methods rather than hard coding the results
import numpy as np
# TODO: Number of training examples
n_train = X_train.shape[0]

# TODO: Number of validation examples
n_validation = X_valid.shape[0]

# TODO: Number of testing examples.
n_test = X_test.shape[0]
```

```
# TODO: What's the shape of an traffic sign image?
image_shape = X_train.shape[1:]

# TODO: How many unique classes/labels there are in the dataset.
n_classes = len(np.unique(y_train))

print("Number of training examples =", n_train)
print("Number of validation examples =", n_validation)
print("Number of testing examples =", n_test)
print("Image data shape =", image_shape)
print("Number of classes =", n_classes)
```

2. Include an exploratory visualization of the dataset.

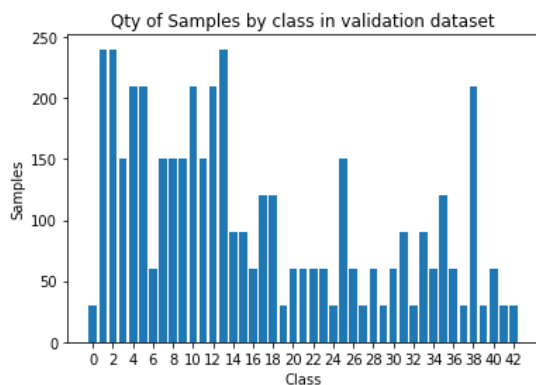
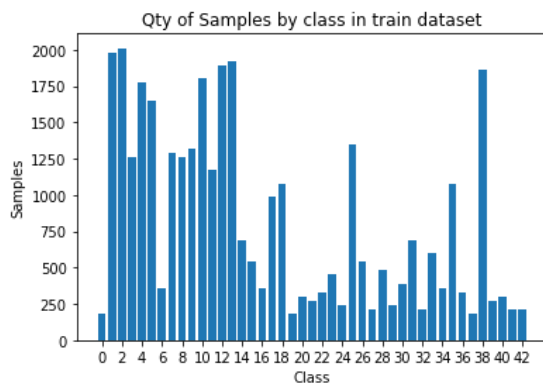
I've decided to calculate the number of samples of each class in each data set. I've used this code:

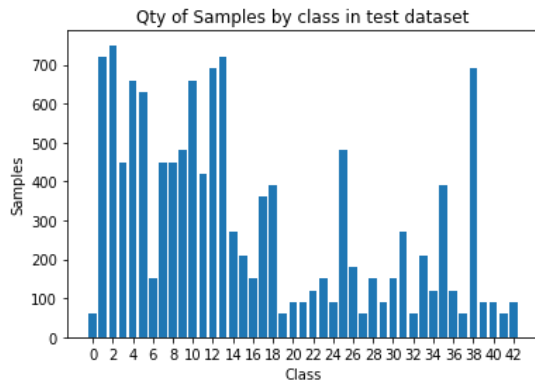
```
def show_samples_classes(title, y_data):
    samples_of_each_class = [0]*n_classes
    for y in y_data:
        samples_of_each_class[y] = samples_of_each_class[y]+1

    plt.ylabel('Samples')
    plt.xlabel('Class')
    plt.title("Qty of Samples by class in "+title)
    plt.xticks(range(0,n_classes,2))
    plt.bar(range(n_classes), samples_of_each_class)
    plt.show()

show_samples_classes('train dataset', y_train)
show_samples_classes('validation dataset', y_valid)
show_samples_classes('test dataset', y_test)
```

To show these results:





Design and Test a Model Architecture

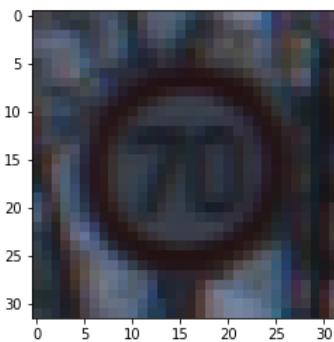
1. Describe how you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc. (OPTIONAL: As described 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, 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.)

I've converted the image to grayscale. I'm interesting in the shape of the sign and not the color for the classification. Also, it will take less time to train if I reduce the shape of the training data.

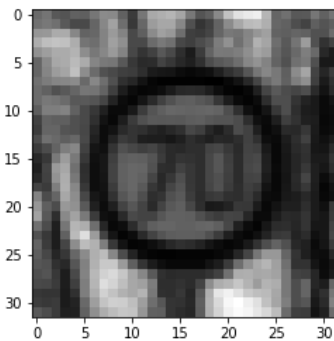
I used OpenCV library to convert to grayscale using this code:

```
res = np.zeros(shape=data.shape[:-1])
for i, frame in enumerate(data):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    res[i] = gray
```

This is an example of an image before grayscale:



And this is the result after applying my method:



Also, the image is normalized with the following script:

```
res = ((res-128)/128)
```

I haven't used data augmentation.

2. 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.

I used an improved version of LeNet seen in the Lab. I called the network MyNet. I added Dropout to prevent overfitting and BatchNormalization layers that will normalize the input between the layers increasing accuracy and speeding up the training.

Also, I used Tensorflow tf.layers classes to an easy writing.

My final model consisted of the following layers:

Layer	Description
Input	32x32x1 Gray scale image
Convolution 5x5	1x1 stride, valid padding, outputs 28x28x6 filter=6
RELU	Activation
Max pooling	2x2 stride, pool size=2x2 outputs 14x14x6
Convolution 5x5	1x1 stride, valid padding, outputs 10x10x16 filter=16
RELU	Activation
Max pooling	2x2 stride, pool size=2x2 outputs 5x5x16
Flatten	5x5x16=400 output
Batch Normalization	
Fully Connected	Input = 400. Output = 120
RELU	Activation
Fully Connected	Input = 120. Output = 84
RELU	Activation
Dropout	keep_prob = 0.5
Fully Connected	Input = 84. Output = 43
Linear	

I used Linear at the end because I want to get logits. Then, I will use a softmax with logits optimizer.

3. 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 train the model I used a 0.5 dropout, 0.002 learning rate, 64 batch_size, and 20 epochs.

I used Adam optimizer, that performs quite well, more than RMSProp, and softmax cross entropy with logits loss function.

I've tried different values in learning rate 0.01, 0.1 but 0.002 was the best.

The training was very dizzy. But I found that it start a platau over epoch 11. So, I gave only 9 epochs more to increase a bit the accuracy.

4. 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.

My final model results were: EPOCH 20 ... Train Accuracy = 0.991 Validation Accuracy = 0.946 Test Accuracy = 0.928

For selecting the model, first I took the LeNet network from the Lab. The accuracy of this model was less then the 93% required.

I remembered to use Relu as activation function in the inner layers because it works better than softmax.

Then, I see that the accuracy was very low, so I've tried adding a BatchNormalization layer. I put it in the first layer, but I doesn't work. So, finally, I added it after the flatten.

Then, I've noticed that the training accuracy was higher than the validation accuracy. So, that indicates the model was overfitting. I've tried to reduce it. I could only a bit using a dropout layer. I've tried using 3 dropouts, less keep probabilities, but the best result was using just 1 dropout with 0.5 keep probability.

Besides the dropout, I've needed to adjust the learning rate. If it was too high, the model will underfit and train too fast. But, if I set it too low, it will take a lot of epochs to train and running in a risk of overfitting. Finally, I've chosen 0.002 learning rate.


The batch size is the number of mini-batches the training is going to make. The number of sample per iteration that will try to train. If I set it too high, it will take longer to train but I will be more detailed. It usually selected in power of 2. I've tried 32, and 64. Finally 64 was the best value.


The model is overfitting a bit, I couldn't reduce it without decreasing the validation accuracy required.


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 five German traffic signs that I found on the web:

Bumpy road: This first image has a white board under the sign. That could make the prediction hard: 

General caution: The second image, it has not background, it's in white. Also, the prediction could be tricky: 

Speed limit (30km/h): The third image, seems an standard image. The circle is not perfect but I think the algorithm will not have issues: 

No entry: The forth image, same as before. The circle is not perfect but I also think it wouldn't be an use. Maybe if I had use dataaugmentation, I could deform a bit the training images . 

Stop Sign: The fifth image, has some white background, but I'm sure that the prediction will be ok. 

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:

Image	Prediction
Bumpy road (22)	Bumpy road
General caution (18)	General caution
Speed limit (30km/h) (1)	Speed limit (30km/h)
No entry(17)	No entry
Stop Sign (14)	Stop Sign

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 92.8%

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. 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)

Notice that the algorithm is very certain in many guesses so the probabilities are very low in the rest of the classes.

These are the top 5 softmax probabilities for each prediction:

1. Bumpy road (22)

Probability	Prediction
1.0	22 Bumpy road
3.3151048e-13	14 Stop
2.4081888e-24	26 Traffic signals
9.1490214e-28	25 Road work
3.8303990e-28	28 Children crossing

2. General caution (18)

Probability	Prediction
1.0	18 General caution
0.0	0 Speed limit (20km/h)
0.0	1 Speed limit (30km/h)
0.0	2 Speed limit (50km/h)
0.0	3 Speed limit (60km/h)

3. Speed limit (30km/h) (1)

Probability	Prediction
0.995	1 Speed limit (30km/h)
4.4318428e-03	0 Speed limit (20km/h)
3.9381231e-04	40 Roundabout mandatory
1.5541940e-05	4 Speed limit (70km/h)
1.5048345e-05	18 General caution

4. No entry(17)

Probability	Prediction
1.0	17 No entry
0.0	0 Speed limit (20km/h)
0.0	1 Speed limit (30km/h)
0.0	2 Speed limit (50km/h)
0.0	3 Speed limit (60km/h)

5. Stop Sign (14)

Probability	Prediction
1.0	14 Stop
4.9372356e-15	17 No entry
6.6622492e-16	3 Speed limit (60km/h)
1.7873340e-18	12 Priority road
1.4789622e-18	34 Turn left ahead

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

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

