Self-Driving Car Engineer Nanodegree

Deep Learning

Project: Build a Traffic Sign Recognition Classifier

Jussi Wright / 29.6.2017

Summary of the task.

Step 0: Parse and Load the Training Data

Splitting the dataset

- Data splitting
- Test set 20% of given dataset

Step 1: Dataset Summary & Exploration

The basic summary of the Data

Visualization of the Dataset

- Random examples of classes pictures
- First image of each class
- Distribution of the train dataset (chart)
    - Min number of images per class = 141
    - Max number of images per class = 1607

Preprocessing the Data Set

- Shuffle the training dataset and Split validation dataset off from training dataset
- Normalized data makes Gradient Descent faster. Done by the line of code X_train_normalized = (X_train - 128)/128.
- Grayscaling, reduce training time.
- Preprocessing is made with Min-Max Scaling normalization / preprocessing techniques
- One-Hot Encoding was used to convert label numbers to vectors.
- Visualizing rgb vs grayscale

Data augementation

Data analyze

In original data has too much differences between the classes.

Solution: Create more data to balance the number of inputs.

Made by making copies randomly translating, scaling, twisting, and adjusting brightness of the images.

Result: The data is more balanced, and each class has at least 500 images.

Step 2: Design and Test of the Model Architecture

Design, train and test the model that learns to recognize traffic signs. Use the German Traffic Sign Dataset.

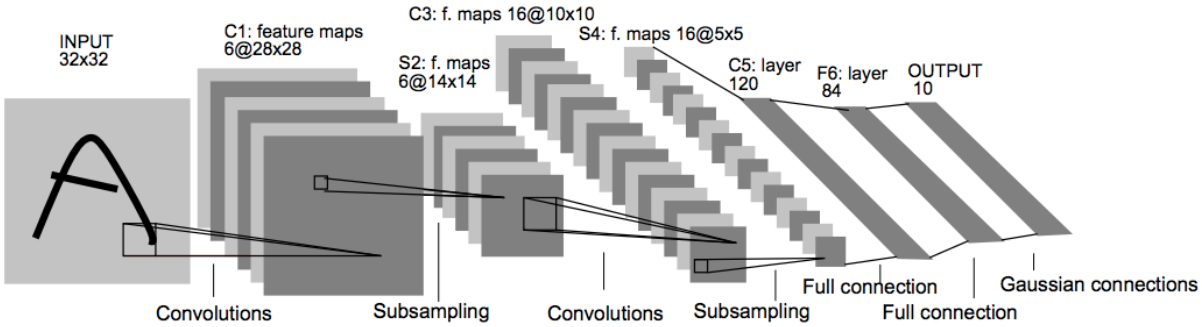
Validation accuracy should be at least 0.93.

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

- Neural network architecture (is the network over or underfitting?)
- Play around preprocessing techniques (normalization, rgb to grayscale, etc)
- Number of examples per label (some have more than others).
- Generate fake data.

CNN Arcitecture

Model based on the LeNet model. Modifications: 2 Dropouts



- **Layer 1: Convolutional.** Output 28x28x6
    - Activation (ReLU)
    - Pooling (MAxPool) Output 14x14x6
- **Layer 2: Convolutional.** Output10x10x16
    - Activation (ReLU)
    - Pooling (MAxPool) Output 5x5x16. (
- Flatten 3D->1D(done: `tf.contrib.layers.flatten`) Output 400
- **Layer 3: Fully Connected.** Output 120
    - Activation (ReLU)
    - Dropout
- **Layer 4: Fully Connected.** Output 84
    - Activation (ReLU)
    - Dropout
- **Layer 5: Fully Connected.** Output 10 (84)
    - Logits

      Output
      Return the result of the 2nd fully connected layer.

EPOCHS = 50 BATCH_SIZE = 100 rate = 0.001 #for Adams optimazer

Hyperparameters

mu = 0 sigma = 0.1

Features and Labels

CNN Training

CNNs was trained with the Adam optimizer and batch size was 100 images. The model was trained for 50 epochs (55678 images in each epoch) with one dataset. Trainin parametry was (0.001), Hyperparameters was mu (0) and sigma (0.1).

The loss was 0,987 and accuracy 0,958.

Train the Model

Step 3: Test a Model on New Images

Output Top 5 Softmax Probabilities For Each Image Found on the Web

For each of the new images, print out the model's softmax probabilities to show the certainty of the model's predictions (limit the output to the top 5 probabilities for each image). tf.nn.top_k could prove helpful here. The example below demonstrates how tf.nn.top_k can be used to find the top k predictions for each image. 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 correspoding class ids. Take this numpy array as an example. The values in the array represent predictions. The array contains softmax probabilities for five candidate images with six possible classes. tk.nn.top_k is used to choose the three classes with the highest probability:

Load and Output the Images

- 8 Finnish traffig signs

Predict the Sign Type for Each Image

Analyze Performance

- Input, Top Guess, 2nd guess, 3rd guess fo every image with gray scaling

Evaluate the Model

Once you are completely satisfied with your model, evaluate the performance of the model on the test set.

Be sure to only do this once!

If you were to measure the performance of your trained model on the test set, then improve your model, and then measure the performance of your model on the test set again, that would invalidate your test results. You wouldn't get a true measure of how well your model would perform against real data.

```
Test Set Accuracy = 0.955
```

How to improve this model?

- Different network architectures
- Chance dimensions of the LeNet layers
- Add recularisation features (Dropout or L2) to make sure that Network doest overfit the training data
- Tune the hyperparameters
- Improve the data preprosessing (normalization)
- more augmenting data by rotating, fliping, sifting or chancing colours