Machine Learning Engineer Nanodegree

Capstone Proposal: German Traffic Sign Classifier

Ali Parsaei,

December 12, 2017

Proposal

Note: This project is heavily inspired by the Udacity CarND Deep Learning Project. The same dataset is also used to carry out the project. I selected this project after consulting my Udacity mentor on my capstone project. The project can be found on Udaciy CarND Traffic Sign Classifier Project GitHub Link.

Software Requirements

This project is done using Jupyter Ipython Notebook with Python. **Python 3.5.** and the following Python dependencies need to be installed to successfully compile the code:

- Ipython
- Jupyter
- Numpy
- Pandas
- SciPy
- Scikit-Learn
- OpenCV
- Matplotlib
- TensorFlow

Domain Background

One of the classical challenges in self-driving cars has been correctly recognizing road traffic signs using vision. This is crucial to car's functionality since the computer needs to make a decision based on the signs installed on the road. Before the current era of computers, this task was considered infeasible due to low speed of processors, absence of proper methods to classify images, and possibly lack of data. Within the past two decades, thanks to the amount of data available on the web, the computational capabilities of the current computers, and various architectures introduced in the domain of convolutional neural networks, solving the problem of classifying traffic signs is feasible now.

Plenty of research has been done in the field of Convolutional Neural Networks (CNN). One of the most relevant publications on applying CNN's to classifying traffic signs is Traffic Sign Recognition with Multi-

Scale Convolutional Networks by Pierre Sermanet and Yann LeCun, which is the baseline for this project as well.

I have had an interest on mobile robots since grad school. My research in grad school was on deploying a fleet of autonomous robots. After graduation, I continued to work on the same domain, in a larger scale, focusing on the industrial application of fleets. After reading about autonomous cars during my MLND, I became interested in applying Machine Learning, CNN in particular, to self-driving cars; therefore, I feel this project is a very good project to do as my capstone project.

Problem Statement

One of the important tasks that need to be done on a self-driving car is to identify the traffic signs that the camera on the cart captures while moving. One way to solve this problem is to train a model on images of different traffic signs that have been recorded already, then use the compiled version of the model as a pipeline, which gets an image of a traffic sign as input, and classifies the image as one of the traffic sings that it has been already trained on. This result can be fed into the decision making part of the system to decide what to do based on the sign.

German Traffic Signs is widely used throughout the world. It mainly includes 43 classes of signs. The goal of this project is to build a program that can classify an image of a German Traffic Sign into one of the 43 different signs available in the dataset provided to the program.

Datasets and Inputs

The dataset used for this project is the German Traffic Sign Dataset. Udacity has provided a pickled version of the images that are converted to (32, 32) color images for ease of use and smaller size. I used the same version of the dataset that can be downloaded from here.

Similar to a lot of Deep Learning Classification projects, the dataset needed for training a model should include the images and their correct label (class). This dataset also is composed of the images as inputs and labels as outputs.

Moreover, the data is divided into training and testing datasets. The Training dataset and the Test dataset have 29406 and 12630 entries respectively. Note that every single input entry (X) is a (32, 32, 3) (width, height, channels) image and every output entry (y) is an integer label between 0 and 42 (inclusive).

Another file that is included in the dataset is a CSV file that represents what each class ID/label represents. For example, Class ID = 14 represents "Stop" sign. This is for a better understanding of the data and sanity check of the results.

Data Sources:

- German Traffic Sign Dataset.
- J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In Proceedings of the IEEE International Joint Conference on Neural Networks, pages 1453–1460. 2011.
- Udaciy CarND Traffic Sign Classifier Project

Solution Statement

As described in the previous sections, every input entry of the dataset is a (32, 32, 3) image and every output entry in the dataset is a label between 0 and 42. Based on the dataset, a proper way to accomplish the task of predicting the label of an image is to create a pipeline that takes in images as inputs, preprocesses the images, feeds the processed data into a CNN with multiple layers (which needs to be trained and tuned), and finally outputs the class ID of the image.

Benchmark Model

The benchmark model for this project is the LeNet model, which defined in Traffic Sign Recognition with Multi-Scale Convolutional Networks by Pierre Sermanet and Yann LeCun. The variation of LeNet architecture used in this model has 2 convolutional layers as well as 2 fully connected (dense) layers. The input to the model is the 32x32 colored images of the German Traffic Sign and the output is the label of the sign predicted by the model.

The evaluation metric is the accuracy of the prediction on test dataset. The accuracy of the LeNet model is 98.97%, Note that a notable portion of these images are not very easy to interpret (bad angle of view, darkness, etc.) such that Human's accuracy in predicting the traffic signs is 98.81% using these 32x32 color images. Therefore, the accuracy attained by the benchmark model is truly incredible.

Evaluation Metrics

Different evaluation metrics can be used in classification problems. The common performance metrics used for classification problems are accuracy, precision, and recall. There is always a tradeoff between recall and precision. In some of the problems such as detecting cancerous tumor cells, a high recall model is desired, while for some other problems such as the spam detection, a high precision model is desired. Note that both of the mentioned problems fall into binary classification, and the reason that recall and precision have different priorities roots in consequences of a correct or incorrect classification for a certain class. One the other hand, in the general form of our problem, which is a multi-class classification, the accuracy of the model seems to be the best criterion to evaluate the performance of the model. The reason behind this statement is the fact that in general, it is equally important to detect all the signs correctly; therefore, neither recall nor precision has higher importance over the other. Note that in reality, correctly detecting some of the signs may have higher importance compared to others; however, in this project, we treat all the signs equally in a general situation.

Based on the introduction provided, the evaluation metric used for this problem is the accuracy of the prediction both in the benchmark model and my capstone project. My goal is to attain an accuracy of 90% or better over the test set images to indicate a successful result.

Project Design

Solving a Machine Learning problem in general requires the following steps to have a successful pipeline:

1. Loading the data and saving the data into proper data structures.
2. Then dividing the data to training and testing datasets.
3. If required, treating and preprocessing the data to prepare the data to feed into the algorithm

4. Train and validate the algorithm using the training data (i.e. training and validation datasets). This step usually takes the most time.
5. Testing the algorithm with the Test dataset to measure the performance of the algorithm.

In order to accomplish the goal of this project, which is correctly classifying the Class ID of an input image to a pipeline, we perform a more detailed version of the steps mentioned above.

1. First, we need to load the data from the files to the script. The inputs (X) are going to be sets of images and the outputs (y) are going to be sets of labels.
2. Then, we need to define the data into training (training , validation) and testing datasets. The data used for this project have been already divided to the training and testing datasets.
3. After defining the datasets, we need to take a look at the data to have a better intuition of the nature of the data points.
4. The data set does not have missing entries or other issues that require treating; therefore, we only need to preprocess the data in order to be appropriate to feed into the algorithm.
5. After preprocessing the images, we need to create our model architecture, which takes in the preprocessed images as inputs and provides the most likely label for each image. The type of dataset and goal of the problem are well-suited for a Convolutional Neural Net (CNN). We need to define an architecture that takes images into the neural net and outputs Softmax probabilities (set of probabilities of which each entry corresponds to the likelihood of the image belonging to that class ID).

   The CNN architecture used in this project is a variation of the LeNet architecture with two convolutional and two fully connected layers. The inputs are normalized grayscale images and outputs are Softmax probabilities. The following figure depicts the architecture that will be used in this project.

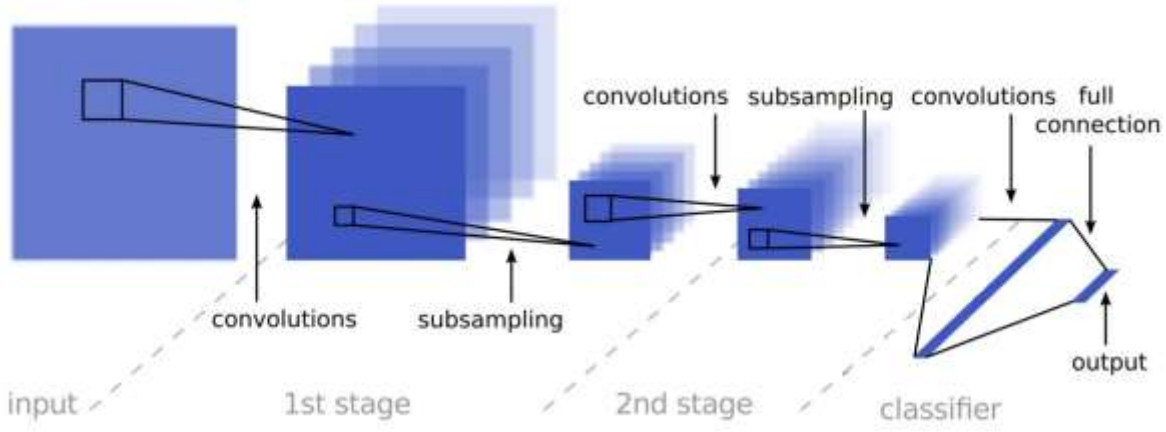

Figure 1: LeNet Architecture used in this Poblem

6. After defining the Architecture, evaluation metrics should be defined. In this case the percentage of the correct prediction (accuracy) is the proper metric to use base on the previous discussion in the proposal.
7. In this step, the model should be trained using the training dataset and validated against the validation set. This step probably is the longest section of the problem, where the architecture

and its weights need to be tuned. After the algorithm works well on both training and validation datasets, the final model is stored in a local file.

8. After the model is successfully trained, validated, and stored in a local file, the model is restored to test the test dataset. In this step, we need to make sure we can attain the evaluation metric defined in the proposal and if not, tune the model to reach our goal.

Following figure summarizes the data flow in this project:

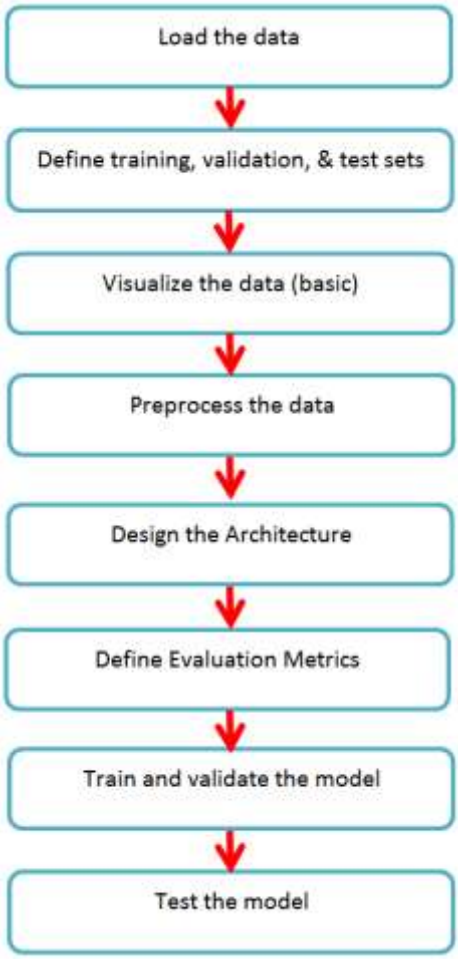

Figure 2: Data Flow