Machine Learning Engineer Nanodegree

Using Convolutional Neural Networks to Identify Traffic Signs

Danny Glover
January 4, 2017

Proposal

Domain Background

Recognizing, and properly classifying, traffic signs is a problem of computer vision. According to The British Machine Vision Association Computer Vision is,

> *"The science that aims to give a similar, if not better, capability to a machine or computer. Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images."*[1]

Computer vision is one of the major pillars of modern machine learning research in the 21st century. Advancements in deep learning neural networks have resulted in major breakthroughs, such as Apple's newly released Face ID.[2] Convolutional Neural Networks now provide machine learning engineers the ability to create powerful models that generalize well. This ability to generalize is a key requirement in street sign recognition. In the real world street signs will often be partially blocked by obstacles like hanging branches, or obfuscated by inclement weather.

Problem Statement

One of the most fundamental challenges of developing fully autonomous vehicles, also known as level 5 autonomy, is teaching cars to understand and follow traffic laws required to drive on public roads.[3] This includes speed limits, right of way, traffic lights, and a large array of street signs. In this capstone project I will focus on teaching an autonomous vehicle to recognize street signs using a convolutional neural network.

Datasets and Inputs

For this project I will be using a dataset of street sign images hosted by Ruhr-Universität Bochum's INSTITUT FÜR NEUROINFORMATIK. The dataset is hosted to allow developers to attempt to solve the problem of identifying street signs, specifically solving a multi class classification problem using single images. The dataset can be found here.[4]

Some information on the dataset according to the RUB website:
- 43 types of street signs
- 50,000+ images total
- Reliable ground-truth data due to semi-automatic annotation
- Physical traffic sign instances are unique within the dataset
- (i.e., each real-world traffic sign only occurs once)

Training vs Testing data: The dataset has been split into a training set of 39,209 images for training and 12,630 images for testing. The dataset is unbalanced, before using the data I will balance it in a preprocessing step.

Each image in the dataset contains a single street sign and ranges in size from 15x15 to 250x250 pixels. Not all images are perfectly squared, which is something I will address in the data preprocessing step. Adjusting the images to be perfectly square will make it easier to construct an effective convolutional neural network with square filters. Below is a small sample of the dataset images.[4]

Solution Statement

In this project I will create a meaningful, and necessary, part of a level 5 autonomous vehicles image recognition software - the ability to classify street signs. I will accomplish this by creating a deep learning convolutional neural network. The CNN will be trained using the previously described dataset containing over 50,000 images. Once trained, the model should be able to recognize a new image and classify it correctly with 95% or greater accuracy. The solution architecture can be found in the project design section below.

Benchmark Model

In a paper titled "Traffic Sign Recognition – How far are we from the solution?"[5] the authors, Markus Mathias, Radu Timofte, Rodrigo Benenson, and Luc Van Gool, demonstrate that using modern image recognition algorithms can result in accuracy scores of 95-99%. The researchers used two datasets, one the German dataset I will use in this project and the other a belgium dataset that I will consider further training my model with after initial completion.

The goal for this project will be to create a model that performs above 95%. I will also create a simple ‘vanilla’ CNN that does not use as complex of an architecture or regularization techniques to show how these improve the model.

Evaluation Metrics

This is a multi class classification problem with each image having a single label provided. The accuracy of the model will be evaluated by calculating the percentage of images properly classified on the training data, and eventually the testing data. The data will be balanced so accuracy will be a sufficient metric.

Project Design/Solution

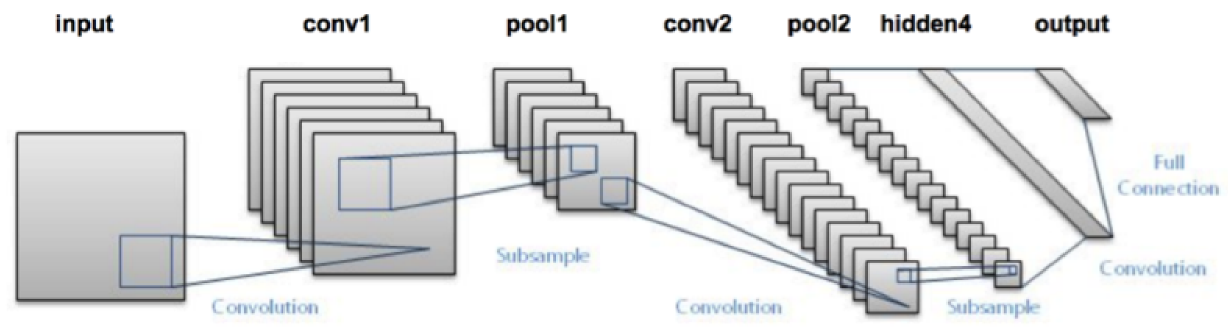
Image Preprocessing

The first step of the project will be to prepare the images to be most useful for training the network. First I will convert the images from 3 channel color to grayscale. Next I will convert any images that are not perfect squares to be perfect squares to better suit the square filters used in the convolutional layers.

I have seen some discussion of creating augmented images by rotating the images and adjusting the brightness. Using additional augmented images would help provide more data for the model to train on. I would like to see how my model performs on the original dataset before performing this action, but will consider it once I feel I have a good architecture accomplished.

Model Architecture

My model architecture will resemble that of the famous LeNet 5 Architecture. The LeNet architecture consists of two sets of convolutional, activation, and pooling layers, followed by a fully-connected layer, activation, another fully-connected, and finally a softmax classifier.[6]



Visualization of Model Architecture (image source)

The key to this architecture are the two convolutional layers. The convolutional layers focus on small parts of the image, learning the 'local' features. These small pieces are then passed to the pooling layers which combine them. This dividing of the image into smaller pieces results in a very large amount of weights for the model to learn, which can make it prone to overfitting. To combat overfitting I will randomly zero a certain percentage of neurons, this is a strategy called dropout.[8]

Below are the parameters I will use for each layer. I expect I will adjust these as I train the model and begin to see results.

Input Layer - 32x32

Convolutional Layer 1 - 16 feature maps
5x5 kernel 1 stride
Pooling: 2x2 with 2 stride
Activation function: ReLu

Convolutional Layer 2 - 32 feature maps
5x5 kernel 1 stride
Pooling: 2x2 with 2 stride
Activation function: ReLu

Hidden Layer - fully connected - 120 feature maps
Activation function: ReLu
Dropout: 20%

Hidden Layer - fully connected - 80 feature maps
Activation function: ReLu
Dropout: 20%

Output Layer - Fully connected with 43 outputs (43 types of classification)

Activation function: softmax

As stated previously I expect I will be adjusting this architecture as I begin to train the model and analyze the results. I’m very much looking forward to testing a fully trained model on new real world examples.

Regularization Techniques

To prevent overfitting of the training data I will use the following regularization techniques:

- Dropout - randomly turning off a percentage of nuerons. This will be most effective in the fully connected layers.
- L2 Regularization - penalizes the square value of the weight, driving all the weights to smaller values.

References

1. http://www.bmva.org/visionoverview
2. https://machinelearning.apple.com/2017/11/16/face-detection.html
3.https://www.techrepublic.com/article/autonomous-driving-levels-0-to-5-understanding-the-differences/
4. http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset
5. https://rodrigob.github.io/documents/2013_ijcnn_traffic_signs.pdf
6. https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/
7. https://blog.dataiku.com/deep-learning-with-dss
8. https://en.wikipedia.org/wiki/Dropout_(neural_networks)