Machine Learning Engineer Nanodegree

Using Convolutional Neural Networks to Identify Traffic Signs

Danny Glover
January 4, 2017

Report

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] Additionally, recent advancements in graphics card processors are accelerating these breakthroughs by allowing state of the art models to be trained in hours or minutes instead of days.

Semi and fully autonomous vehicles will rely heavily on computer vision. Driverless cars will need to use machine learning models to observe and react to the world around them in real time. This is an exciting new frontier that will bring machine learning directly into billions of people's lives. These computer vision models will need to be extremely accurate since human lives will depend on them.

In this exciting new world of computer vision, Convolutional Neural Networks 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, but I believe the dataset is large enough to create a model that will generalize well.

Each image in the dataset contains a single street that is 32x32 pixels. The images are in 3 channel RGB color, but I will be converting them to grayscale before training.

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.

I created a benchmark CNN based on the famous LeNet-5 architecture. Using no dropout and running for 10 epochs with batch sizes of 128 and a learning rate of .001 this straightforward CNN was able to perform with 90.8% accuracy on the test set. This performance was my base to improve upon in my more advanced 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.

Below is the definition of the evaluation function I am using to calculate the various models’ accuracy.

```
def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0
    sess = tf.get_default_session()
    for offset in range(0, num_examples, BATCH_SIZE):
        batch_x, batch_y = X_data[offset:offset+BATCH_SIZE], y_data[offset:offset+BATCH_SIZE]
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: batch_y, keep_prob: 1.0})
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

Project Design/Solution

Image Preprocessing

The first step of the project is to prepare the images to be most useful for training the network. First I will convert the images from 3 channel color to grayscale. The images are all perfect squares so there will not be any size adjustment needed.

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 decided against augmenting the data due to time constraints, but I will explore this in the future. The dataset seemed to be large enough to allow the model to generalize fairly well without augmentation.

Below are the outputs from gray scaling and then normalizing the dataset.

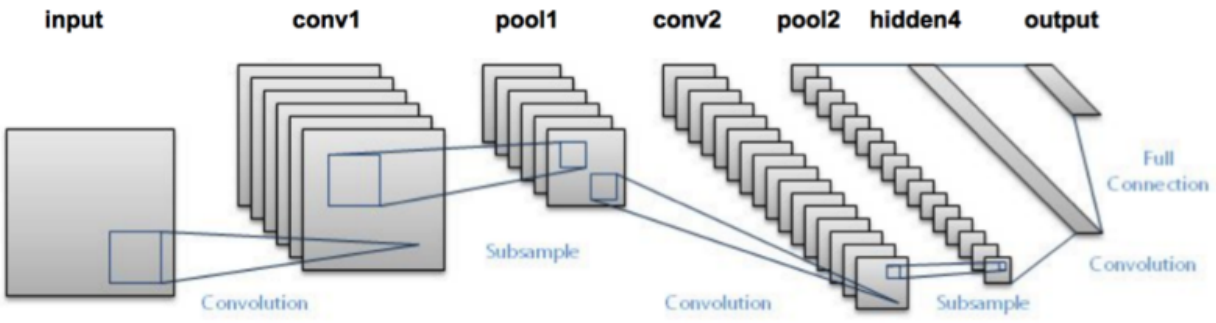
```
RGB shape: (34799, 32, 32, 3)
Grayscale shape: (34799, 32, 32, 1)
```

```
X_train mean before normalization:  82.677589037
X_test mean before normalization:  82.1484603612

X_train mean after normalization:  -0.354081335648
X_test mean after normalization:  -0.358215153428
```

Model Architecture - LeNet

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

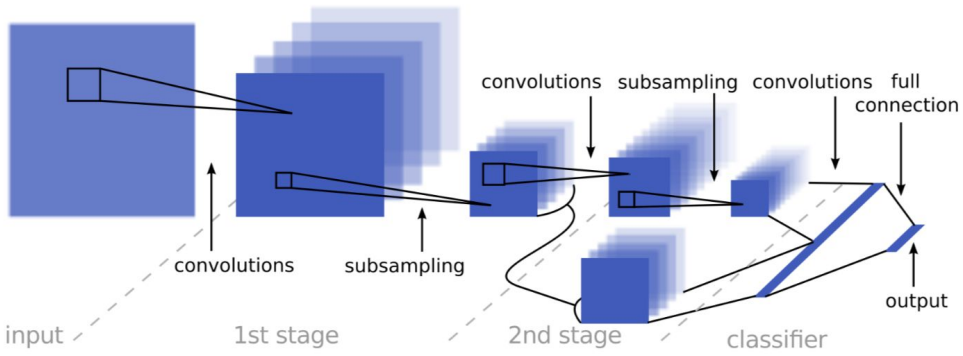
Hidden Layer - fully connected - 120 feature maps
Activation function: ReLu
Dropout: 0%

Hidden Layer - fully connected - 80 feature maps
Activation function: ReLu
Dropout: 0%

Output Layer - Fully connected with 43 outputs (43 types of classification)
Activation function: Relu

Model Architecture - LeNet Advanced

After reading the suggest paper Traffic Sign Recognition with Multi-Scale Convolutional Networks I changed the architecture to more closely recognize the figure shown below.



While this network still uses convolutional layers and pooling it creates parallel tracks that have some of the data skip convolutions in a nonlinear fashion. The data is eventually all flattened and added to together to create one large fully connected layer. The final fully connected layer has 800 inputs that reduce down to the 43 traffic sign shapes.

In this model I used 50% dropout to prevent overfitting. As with the LeNet basic setup I used batches sizes of 128. I increased the epochs of training to 60 for this architecture. I adjusted the learning rate from .0008 to .001 and achieved the best results at .001. This architecture achieved 99.4% validation accuracy and 94.4% test accuracy.

Regularization Techniques

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

- Dropout - randomly turning off a percentage of nuerons. This will be most effective in the fully connected layers. I used 50% dropout throughout training.
- L2 Regularization - penalizes the square value of the weight, driving all the weights to smaller values.

Training Results

The following results were achieved after three separate training sessions:

Lenet Basic Results

Params:

- Epochs: 10
- Batch size: 128
- Learning rate: .001
- Keep prob: 1.0

Results:

- Validation accuracy: 97.4%
- Test set accuracy: 90.8%

Lenet Advanced Results

Params:

- Epochs: 60
- Batch size: 128
- Learning rate: .0008
- Keep prob: 0.5

Results:

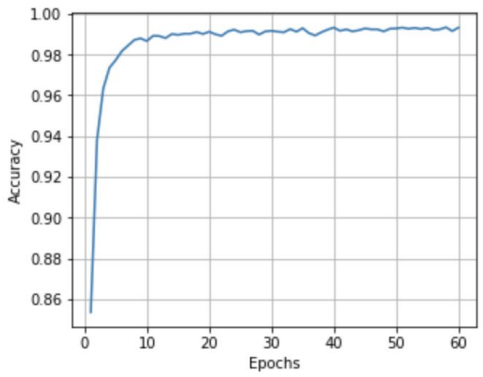
- Validation accuracy: 99.4%
- Test set accuracy: 93.4%

Params:

- Epochs: 60
- Batch size: 128
- Learning rate: .001
- Keep prob: 0.5

Results:

- Validation accuracy: 99.3%
- Test set accuracy: 94.4%



Conclusion

Using the modified “LeNet Advanced” architecture I was able to produce a model that can be trained in under an hour and recognize new test data with 94.4% accuracy. In a real world level 5 autonomous vehicle this would not be a high enough accuracy for safety reasons, but I was pleased with this result for my educational CNN. Image recognition and the Convolutional Neural Networks that power it are sure to continue to change the way computers and humans

interact with the world and I am excited to continue to learn and contribute to this cutting edge field of machine learning.

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)
9. http://yann.lecun.com/exdb/lenet/
10. http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf