Machine Learning Engineer Nanodegree

Capstone Project

Traffic signs classifier

Abdalla Elshikh

March 5th, 2019

Machine Learning Engineer Nanodegree

Capstone Project

Abdalla Elshikh
March 5th, 2019

I. Definition

Project Overview

Self-driving cars has always picked my attention, which is one of the main reasons I decided to take this program. Even though there's has been a lot of experiments conducted on self-driving cars since the 1950s, but recently this field has seen drastic improvements in the past couple of years.

One of the most common tasks that we do while driving is classifying traffic signs to decide whether we should stop, decrease our speed or watch for slippery roads. We are able to do this daily without ever noticing how hard it might be for a computer to mimic our actions.

This is a hard task for a computer to do because it only understands the pictures as some pixel values and can't determine the content present in these pictures. Teaching a computer to classify traffic signs with traditional image processing approaches would be impossible as we would have to detect very complex patterns within the images to differentiate between the different traffic signs. On the other hand, training a deep learning model to do this task is the more appropriate approach because we don't need to provide those patterns for the network manually as it is able to pick up on those complex patterns during the training process.

*** check these papers published about classifying traffic signs*:

http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf

https://www.scitepress.org/papers/2018/67188/67188.pdf

Problem Statement

Computers are much better than humans at paying attention as we don't have the computers' advantage of always being attentive. Classifying traffic signs has always been an integral task for developing self-driving cars as the idea of autonomous vehicles relies heavily on the vehicle being able to interpret traffic signs and accordingly decide what is the appropriate action to take. Training a computer to classify traffic signs just as good as we humans is one of the milestones of building self-driving cars that will lead to a safer and cleaner future. Autonomous driving promises more than just saved time, it's enough to make society reassess its concepts of time and space. Self-driving cars could also improve safety as the Transport and Road Research Laboratory estimated autonomy could prevent around 40 percent of accidents. Unlike humans, self-driving cars benefit from the experiences of all the self-driving cars out on the road whereas humans learn to drive from scratch.

This can be approached by utilizing deep learning techniques to train a model to identify traffic signs. The good thing about this approach is that we don't need to provide the computer with features to look at in the images as it learns to identify those features on its own.

***Watch 3blue1brown's playlist on neural networks for a better intuition:*
https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

Metrics

The evaluation metric that is going to be used for this project are the model's accuracy on the testing data of the BTSD dataset. Accuracy is defined as the model being able to correctly predict the label of the traffic sign present in an image. This is clearly an appropriate metric to use when evaluating our model as we're concerned with its ability to correctly classify traffic signs. Furthermore, a subset of our training data will be taken as a validation set, which is used to assess our model's performance during the training process. The model's accuracy on the validation set is calculated at each epoch and is then used to save the weights that got us the best validation accuracy.

$$accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

II. Analysis

Data Exploration

Belgium traffic signs dataset (BTSD) is used for this project. This dataset contains 4,591 training images, 2,534 testing images and is composed of 62 classes where each class corresponds to a different traffic sign.

Set	Number of images	Size
Training set	4,591	171.3 Mega bytes
Testing set	2,534	76.5 Mega bytes

Each of the training and testing directories contain 62 subfolders, named sequentially from 00000 to 0006. The name of these subdirectories represent the class ID such that the photos lying in the same subdirectory belong to the same class.

The images in this dataset are in ".ppm" format which is not supported by most tools, but luckily, I was able to load these images using Scikit Image library. Displayed below is a sample image of each label in our dataset with the number of images in each class being displayed between brackets.

It's obvious that this is a very good dataset as the quality of the images is great, and there are variety of angles and lighting conditions. Moreover, the traffic signs occupy most of the area of the image so we don't have to determine the exact location of the traffic sign in the image, it also guarantees that our network will focus on the patterns in these traffic signs and not dwell on finding patterns within other background objects.

One thing I noticed when visualizing the dataset is that all traffic speed limit signs fall under the same class as shown below (Class 32). This won't be a problem as we now

know what output we expect when a speed limit traffic sign is provided to our network.

**You can download the BTSD dataset from here: http://btsd.ethz.ch/shareddata/*

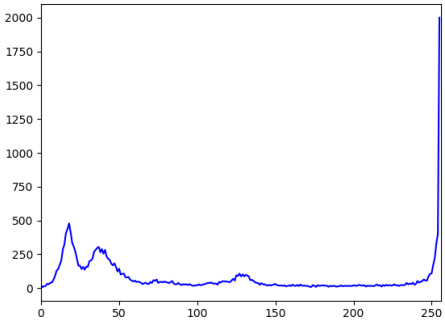
Exploratory Visualization

After analyzing the data, I found that there are significant variations in the color intensities of the training images. Shown below is the histogram distributions for the 3 Red, Green and blue channels of two different images of the same class.

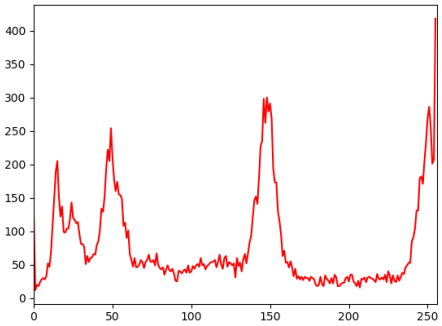
This can make the task difficult for our model because, in addition to that, the training data is not large as there are few thousands training images which are also not uniformly distributed. More about that is discussed in the Free-Form Visualization section of this report.

- ***Histogram distributions of Image 1 of Class 0***

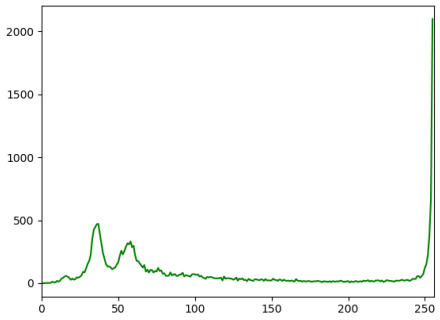
Image 1



Blue channel histogram



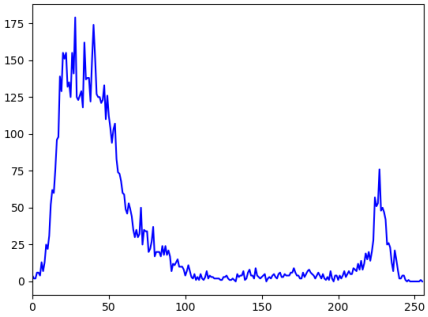
Red channel histogram



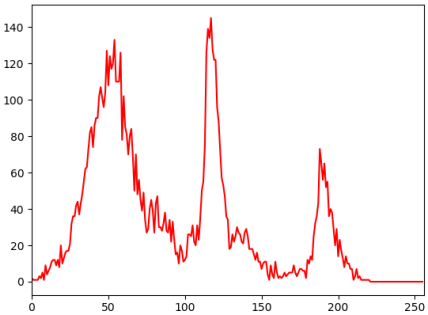
Green channel histogram

- ***Histogram distributions of Image 2 of Class 0***

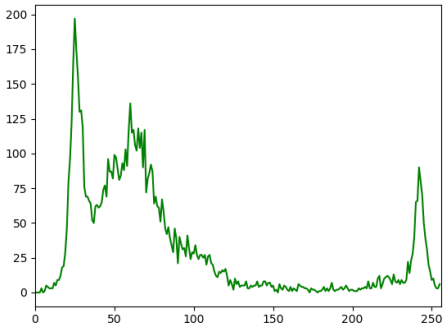
Image 2



Blue channel histogram



Red channel histogram



Green channel histogram

Algorithms and Techniques

A Convolutional Neural Network (CNN) is used as our classifier. It's considered state-of-the-art algorithm for most of the image processing and classification tasks. The CNN proves to be faster and have higher accuracy than traditional fully connected neural networks, namely MLPs. CNNs need a great deal of data for training it which is satisfied by our dataset. The CNN will be trained on the training set after doing several preprocessing steps on it, more on that to follow.

The output of the CNN will be used as an input to fully-connected neural network which outputs an array that contains the probability that the traffic sign belongs to each of the 62 classes. Moreover, a subset of our training data will be split to form the validation set, which is used to evaluate our model's performance during the training process.

There are several parameters that can be tuned to optimize our model:

- Training parameters:
    - Number of epochs: which is the number of steps we run feedforward and backpropagation algorithms to train our model.
    - Batch size: which is the number of images we look at during each training step, namely each epoch.
    - Loss function: which is the function used to compute the error of our model's prediction.
- CNN parameters:
    - Number of layers present in our CNN.
    - Number of filters: which constitutes the depth of the convolutional layers.
    - Layer types (convolutional, pooling, fully-connected…etc.).
    - Activation function used for each layer, more on the activation functions to follow.
    - Other layer parameters (kernel size, padding, strides…etc.).

***See links below for a better intuition about CNNs:*

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://en.wikipedia.org/wiki/Convolutional_neural_network

Benchmark

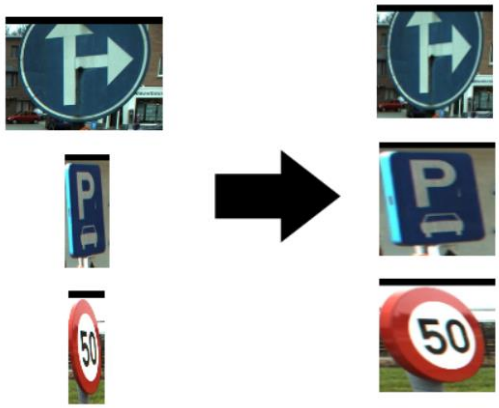
We are going to use the CNN model trained by Pierre Sermanet and Yann LeCun as a benchmark for this project. This benchmark model was able to reach an accuracy of 98.97%. I will be trying to train my model to reach a high accuracy score as that of the benchmark model.

*** More information about the benchmark model can be found here:*
http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf

III. Methodology

Data Preprocessing

We can notice from the ***Data Exploration*** *section* that the images are square-ish but have different aspect ratios, but the network expects a fixed size input, so we need to do some preprocessing to our data.



First, we will need to resize all the images to be of the same size. However, doing so will cause some of the images to be stretched vertically or horizontally as shown. Hopefully, this won't be a big problem in our case because the differences in the aspect ratios is not that large and after resizing the images, we can clearly still classify them, so the network should be able to do so too.

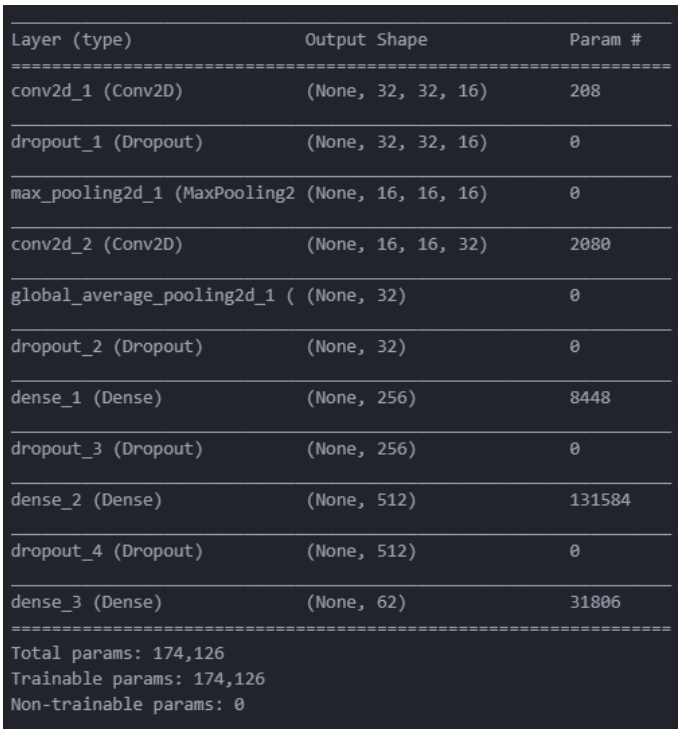
After printing the sizes of several images, I noticed that they hover around 128x128, however, we will resize the images to be 32x32 pixels to train our model faster as it reduces the size of the model and the training data by almost a factor of 16.

Moreover, the CNN expects an input of 4 dimensions, so we need to expand the training data into 4d tensors. The "*path_to_tensor*" function, which can be found at *"utils.py" file,* is used to do this step.

I also noticed that the dataset contained images of traffic signs with different orientations and lighting conditions so there was no need to augment the data before training our model.

Implementation

I started with a model that consists of 2 convolutional layers. Each convolutional layer consists of several 2x2 convolution filters and uses ReLU activation function. The depths of the filters are 16 and 32. The first convolutional layer is followed by a dropout layer with 20% dropout rate which is then followed by a max pooling layer with the same padding and a stride of 1x1. The output from the second convolutional layer is connected to a global average pooling layer to flatten the output so that it can be fed to a fully connected neural network. This global average pooling layer is then followed by a dropout layer with a 10% dropout rate.



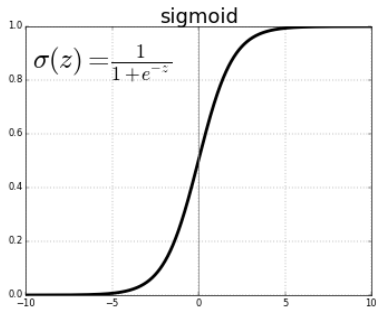
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 16)	208
dropout_1 (Dropout)	(None, 32, 32, 16)	0
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 16)	0
conv2d_2 (Conv2D)	(None, 16, 16, 32)	2080
global_average_pooling2d_1 (	(None, 32)	0
dropout_2 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 256)	8448
dropout_3 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 512)	131584
dropout_4 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 62)	31806

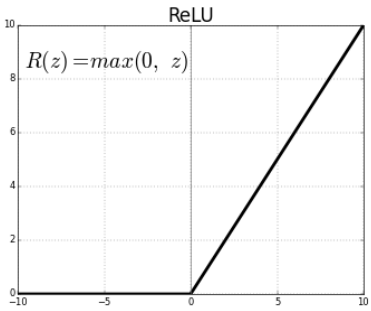
Total params: 174,126
Trainable params: 174,126
Non-trainable params: 0

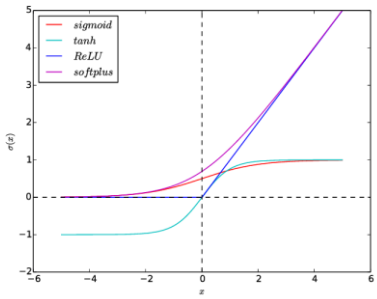
The output is then followed by 2 fully connected layers. The first fully connected layer consists of 256 nodes, while the second one consists of 512 nodes. The activation function used for these two layers is the ReLU activation function. Each of these fully connected layers is followed by a dropout layer with a dropout rate of 0.2. The final output layer consists of 62 nodes, one for each class. The activation function used for output layer is the Softmax activation function. The total number of model's trainable parameters is 174,126 parameters as shown in the figure.

The reason behind choosing the ReLU activation function over the Sigmoid is that it converts all the negative values to zero. It's been shown to work well in classification tasks and trains faster than sigmoid or tanh. The figure below illustrates both the ReLU and sigmoid activation functions formulas. The reason behind choosing the Softmax activation function for the output layers is that the softmax activation function outputs the probability that the input image belongs to each of the 62 classes.

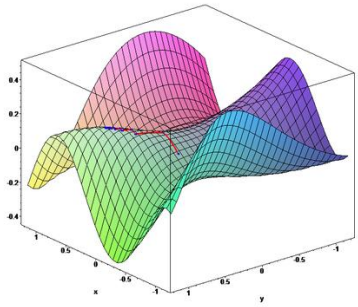
***more details about these activation functions can be found here http://cs231n.github.io/neural-networks-1/*







The loss function used is the categorical cross entropy which is the most common loss function in classification tasks.



** Check this link for a better explanation of the categorical cross entropy loss function: *https://gombru.github.io/2018/05/23/cross_entropy_loss/*

The optimizer used is the Adam optimizer which combines the advantages of two other extensions of stochastic gradient descent. Specifically:

- **Adaptive Gradient Algorithm** (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).
- **Root Mean Square Propagation** (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

Adam realizes the benefits of both AdaGrad and RMSProp.

*** More information about the Adam optimizer can be found here:*

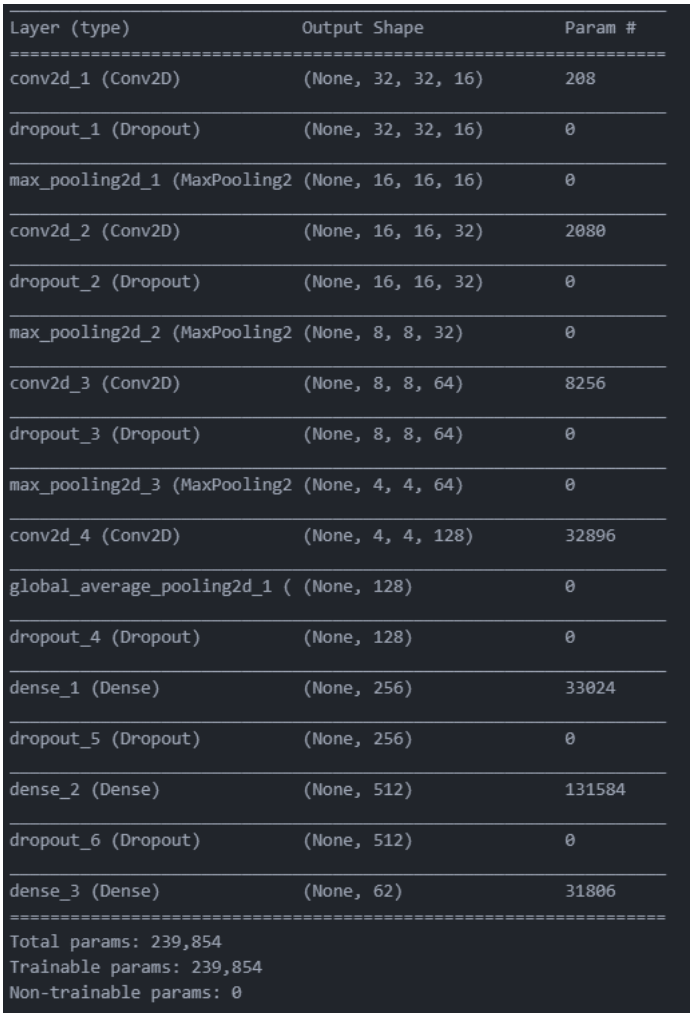
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

I trained this model with a batch size of 100 and 50 epochs, however, I was not satisfied by the results obtained from this model as it got me a training accuracy of 37.5%, which led me to increase the number of epochs to 100 epochs and was able to get a training accuracy of 47.3%.

I divided my code into modules to be easier to trace and debug. The “train.py” file contains the code for the model’s training. The “utils.py” file contains the utility functions that are used by the different modules, while the “predict.py” contains the code for making a prediction, this is implemented in the “predict_class” function which returns the index of the class that has highest probability.

Refinement

After tweaking many parameters and changing the number of layers in the network, I settled on a model that consists of 4 convolutional layers. Each convolutional layer consists of several 2x2 convolution filters and uses ReLU activation function. The depths of the filters are 16, 32, 64 and 128. Each of the convolutional layers is followed by a dropout layer with 20% dropout rate. The dropout layers are then followed by max pooling layer with the same padding and a stride of 1x1. The output from the final convolutional layer is connected to a global average pooling layer to flatten the output so that it can be fed to a fully connected neural network. This global average pooling layer is then followed by a dropout layer with a 10% dropout rate



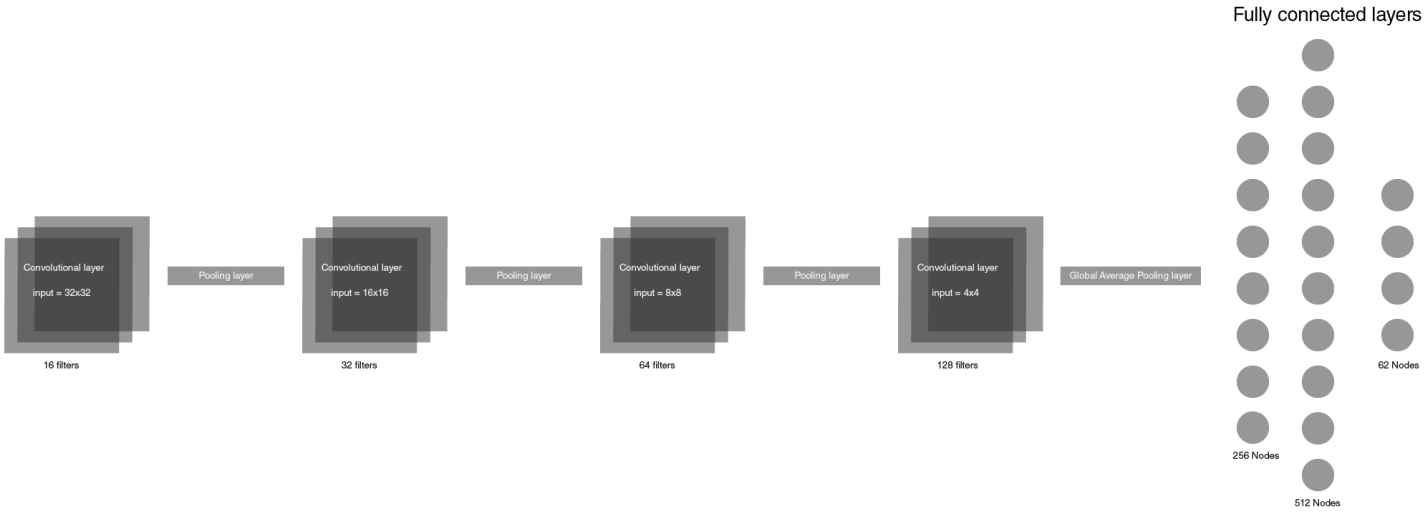
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 16)	208
dropout_1 (Dropout)	(None, 32, 32, 16)	0
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 16)	0
conv2d_2 (Conv2D)	(None, 16, 16, 32)	2080
dropout_2 (Dropout)	(None, 16, 16, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 32)	0
conv2d_3 (Conv2D)	(None, 8, 8, 64)	8256
dropout_3 (Dropout)	(None, 8, 8, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 64)	0
conv2d_4 (Conv2D)	(None, 4, 4, 128)	32896
global_average_pooling2d_1 (	(None, 128)	0
dropout_4 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 256)	33024
dropout_5 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 512)	131584
dropout_6 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 62)	31806

Total params: 239,854
Trainable params: 239,854
Non-trainable params: 0

The output is then followed by 2 fully connected layers. The first fully connected layer consists of 256 nodes, while the second one consists of 512 nodes. The activation function used for these two layers is the ReLU activation function. Each of these fully connected layers is followed by a dropout layer with a dropout rate of 0.2. The final output layer consists of 62 nodes, one for each class. The activation function used for output layer is the Softmax activation function. The total number of model’s trainable parameters is 239,854 parameters as shown in the figure.

I also used the categorical cross entropy loss function and the Adam optimizer for this model and trained it on a batch size of 100 for 50 epochs. I was able to get a testing accuracy of 89.08% which is a huge improvement from the first 2 initial models.

After training the model we save its weights into an “h5” file to load it each time we need to make a prediction so that we don’t have to train it from the beginning again. The code for the training process can be found in the “train.py” file.



Architecture of the final model

IV. Results

Model Evaluation and Validation

After only 30 epochs, it was noticed that the final model outperformed the two initial models. My final model finished with a validation loss of 0.15110, a validation accuracy of 96.17% and a testing accuracy of 89.08%, which indicates how good that model is compared to the other ones.

Since the dimensions and quality of images vary significantly, even after resizing the images, the model is robust enough to solve this problem and its output can be trusted as it achieved a great testing accuracy as stated before.

After testing the model on different images obtained from the web, I was surprised to see how good the model performs given that it didn't take much resources and time to train. Below is an example of the model's output tested on different images obtained from the web

Stop sign being recognized correctly

Speed limit sign being recognized correctly

Justification

The model doesn't expect any image of a fixed size or taken from a particular angle. It has been designed to automatically resize the images and extract the features from it to make predictions. The model got a decent score when tested on the provided testing dataset, given that the testing dataset contains a large number of images of different sizes and angles.

Even though the model's accuracy is less than that of the benchmark model, which has a whopping accuracy of 98.97%, it's still has a great high accuracy score and can be trusted to classify traffic signs correctly. It should also be stated that the benchmark model is trained on a much larger set that contains more training data and is also trained on a GPU with a very high processing power, whereas my model was trained on my local machine.

I believe the model could have been improved more by making it more complex by adding more hidden layers to make it pick on more complex patterns within the images. However, I'd probably need more processing power such as a GPU to increase the speed of the training time which is not available to me at the time.

The model could have also been improved by using transfer learning to use the weights of a pre-trained model to train a new model that learns to classify traffic signs. This approach has proven to give great results and also save time in the training process.

However, for this project, I wanted to build my own CNN from scratch without using a pre-built model.

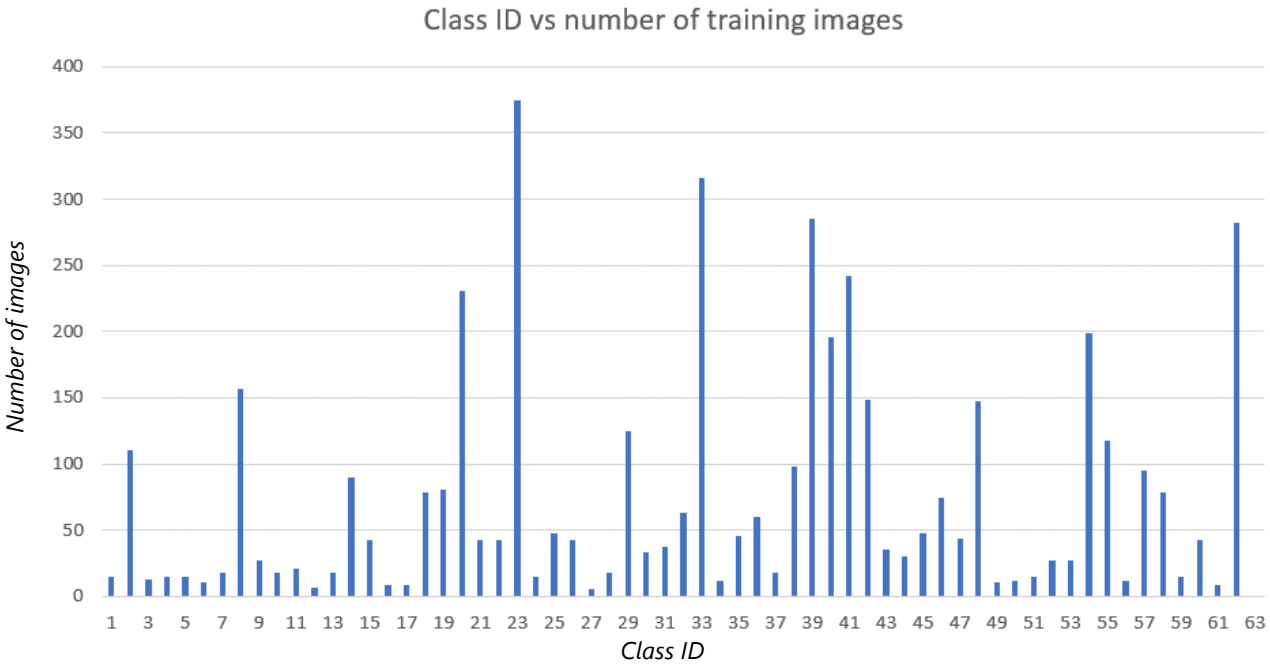
***More information about transfer learning can be found here:*
https://machinelearningmastery.com/transfer-learning-for-deep-learning/

V. Conclusion

Free-Form Visualization

The dataset used for this project contains images captured at different times of the day, with different lighting and at different angles. The images also differ in their quality and resolution. As shown on the graph below, the training images are not distributed uniformly among different classes.

It's clear that some classes contain far more images than other classes which can cause a problem for the model as some classes don't have enough images to make our model be able to predict new unseen data.



Reflection

My project began with collecting the dataset used for training my model, which is the Belgium Traffic signs (BTSD) dataset. After collecting the data, I had to do some preprocessing to be able to fit these data into my model so I had to resize the images to be of a fixed size.

After collecting the data and doing the required preprocessing steps, I experimented with building multiple initial models to see which one will give me the best results. I started with building two initial models that got me a testing accuracy of 37.5% and 47.3% respectively, however, I was able to build a more powerful model by increasing the number of hidden layers, tweaking some parameters and by training my model on more epochs. This final model got a testing accuracy of 89.08%.

Two very interesting and challenging issues arose during this project. The first one is that I didn't know from where to start with building my model. I didn't know how many convolutional layers I should use, what depth should I choose for these layers and how to choose the suitable number of hidden fully connected layers. Fortunately, after

experimenting with these parameters a lot, I was able to build a robust model that gave a high testing accuracy and predicts the traffic signs very well. The second issue is that this was my first project to do from scratch in the machine learning field as sometimes I felt I needed some guidance, so I had to do a lot of research to get the job done. Thankfully, this paid off as I now have a better understanding of some concepts about CNNs and deep learning, and for the first time, I build my own fully convolutional neural network from scratch. I will definitely be looking forward to doing more projects to learn more about this interesting field.

Improvement

Since CNNs require a lot of computing power, I sometimes had to stop training the model because I didn't have neither the time nor the computing power to train complex models as they would take too long to train. I believe if I had a more powerful machine or a GPU, I would have been able to develop a better model, one that has a higher accuracy score.

I believe transfer learning would have got me a better result and would have enabled me to develop a more complex model that has a higher accuracy in a shorter time. However, since I didn't use this method a lot, I decided not to use it for this project, but this is going to be the field of my research in the following weeks because after doing some research, I came across some amazing projects that was done using transfer learning without needing powerful machines.