UDACITY

MACHINE LEARNING ENGINEER NANODEGREE

Capstone Project Report

Traffic Signs Classification

Adnan Karim

October 10 2020

I. Definition

Project Overview

In this project, I will performed the traffic signs classification using the data provided by International Joint Conference on Neural Networks (IJCNN) in 2011 for a competition (available in public domain) and deep learning techniques. Traffic signs classification is an important task in self-driving cars. Thus, I have built a deep learning model to classify the German traffic signs to one of the 43 classes.

Problem Statement

The problem statement for this project is **"How to classify the German Traffic Signs to their respective classes"**. I will use a supervised learning approach to extract the features from data and trained the model to get desired results.

Traffic sign classification is the process of automatically recognizing traffic signs along the road, including speed limit signs, yield signs, merge signs and all kionds of signs you can imagine. Being able to automatically recognize traffic signs enables us to build "**smarter cars**".

Self-driving cars need traffic sign recognition in order to properly parse and understand the roadway. Similarly, "**driver alert**" systems inside cars need to understand the roadway around them to help aid and protect drivers.

Traffic sign recognition is just one of the problems that computer vision and deep learning can solve.

Metrics

As it is supervised problem. Accuracy is the best metric for evaluating this model which is automatically done by using tensorflow 2.0. Probably the confusion matrix to visualize the predictions of model.

- ⇨ As it is multiclass classification problem then we are bound to use categorical cross-entropy loss function
- ⇨ Moreover, I used the Adam Optimizer as it generally the first choice recommended by various experts.

II. Analysis

Data Exploration

Image source @steemit

The dataset I'll be using to train our own custom traffic sign classifier is the German Traffic Sign Recognition Benchmark (GTSRB).

The GTSRB dataset consists of 43 traffic sign classes and nearly 50,000 images.

A sample of the dataset can be seen in Figure above — notice how the traffic signs have been pre-cropped for us, implying that the dataset annotators/creators have manually labeled the signs in the images and extracted the traffic sign Region of Interest (ROI) for us, thereby simplifying the project.

In the real-world, traffic sign recognition is a two-stage process:

Localization: Detect and localize where in an input image/frame a traffic sign is.

Recognition: Take the localized ROI and actually recognize and classify the traffic sign.

Deep learning object detectors can perform localization and recognition in a single forward-pass of the network.

I download the pickled data provided in link of almost 50k images

traffic-signs-data.zip

train.p

valid.p

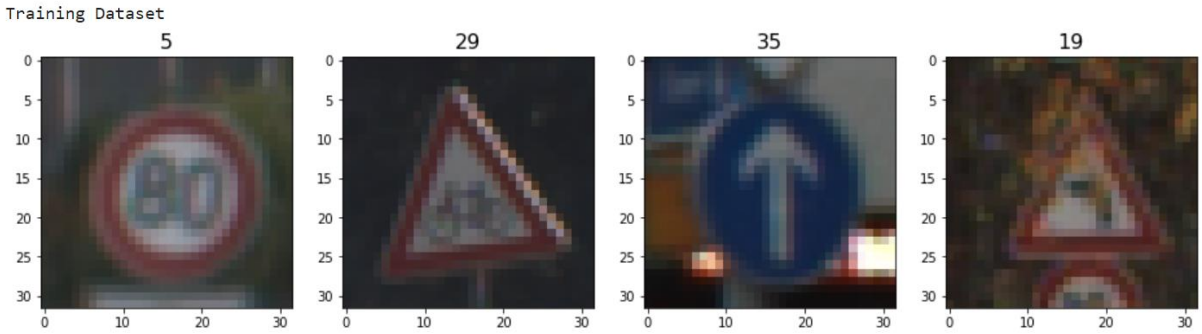
test.p

The pickled data is a dictionary with 4 key/value pairs:

- `'features'` is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
- `'labels'` is a 1D array containing the label/class id of the traffic sign. The file `signnames.csv` contains id -> name mappings for each id.
- `'sizes'` is a list containing tuples, (width, height) representing the original width and height the image.
- `'coords'` is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image. **THESE COORDINATES ASSUME THE ORIGINAL IMAGE. THE PICKLED DATA CONTAINS RESIZED VERSIONS (32 by 32) OF THESE IMAGES**
-

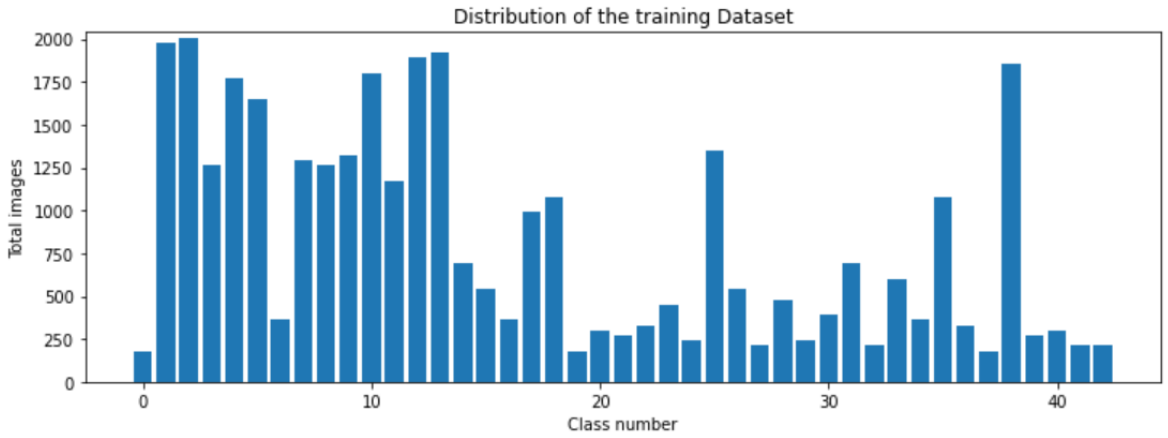
Exploratory Visualization

I used the **matplotlib** to visualize the data as shown below, an example from training dataset.



There are a number of challenges in the GTSRB dataset, the first being that images are low resolution, and worse, have poor contrast as shown below. These images

are pixelated, and in some cases, it's extremely challenging, if not impossible, for the human eye and brain to recognize the sign.



One class has over 2,000 examples while the least represented class has under 250 examples that's an order of magnitude difference!

In order to successfully train an accurate traffic sign classifier we'll need to devise an experiment that can:

- ⇨ Preprocess our input images to improve contrast.

Algorithms and Techniques

This is an image classification problem mostly CNN are used to solve these kinds of problems. I will preferably use a convolutional neural network to classify the images to their respective class as they are best suitable for image classification. After analyzing many architectures like ResNet. I thought **LeNet-5** architecture will be best suitable for this job.

Benchmark

For this problem it is suggested to use **LeNet-5** based on consulted data sets of historical relevance on Kaggle and different blogs performing nearly +80%.

III. Methodology

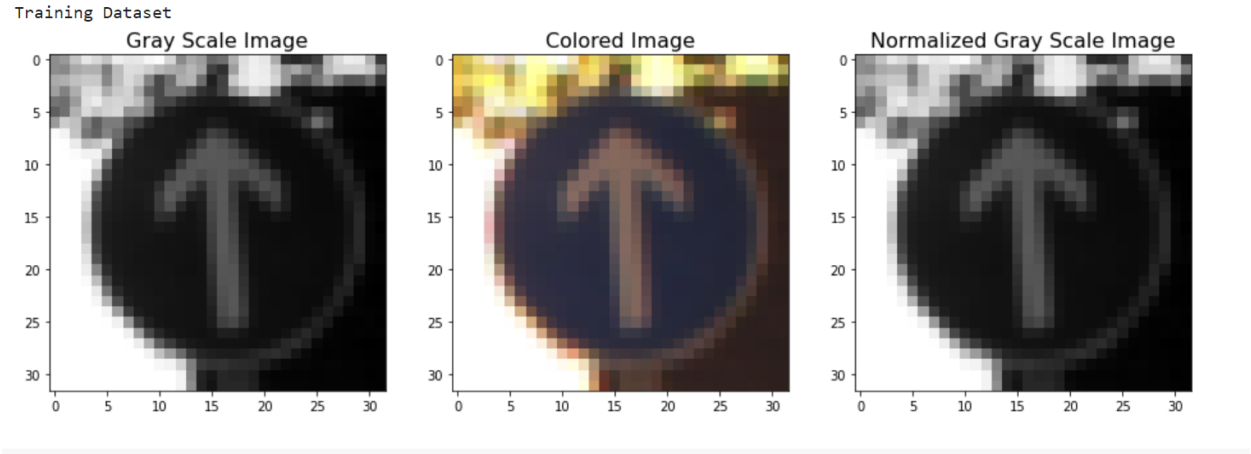
Data Preprocessing

Data preprocessing is an important methodology to identify and learn data. It also helps in identification of important features and removal of many unnecessary features.

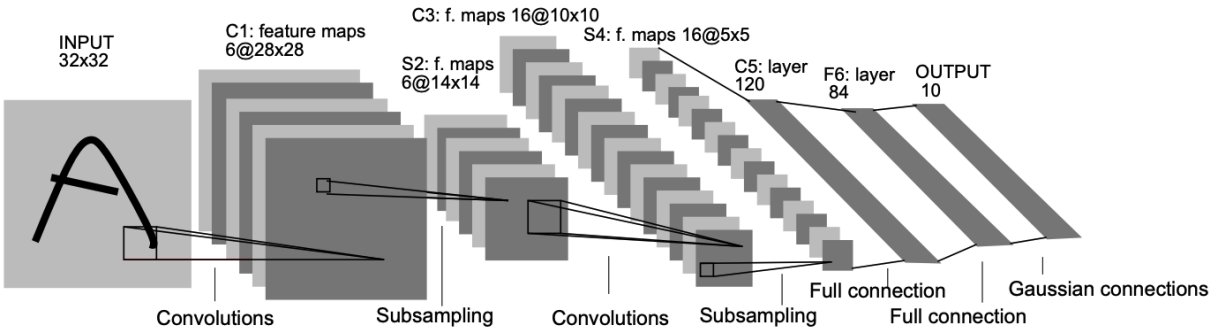
I applied shuffling of datasets, normalization and converting the images to grayscale for faster processing.

```
from sklearn.utils import shuffle
# Shuffling
X_train, y_train = shuffle(X_train, y_train)

# Change to grey scales
X_train_gray= np.sum(X_train/3,axis=3,keepdims=True)
X_valid_gray= np.sum(X_valid/3,axis=3,keepdims=True)
X_test_gray= np.sum(X_test/3,axis=3,keepdims=True)

# Normalizing the images
X_train_gray_normalize = (X_train_gray-128)/128
X_valid_gray_normalize = (X_valid_gray-128)/128
X_test_gray_normalize = (X_test_gray-128)/128
```

And I visualize the training dataset using the matplotlib as shown below similar is the case for validation and test datasets.



Implementation



Source: [Gradient-based learning applied to document recognition]

I implemented the above LeNet's which is for MINUST hand written digits classification. In the above figure it has 10 outputs labels to predict but I have 43 classes to predict. I use the Tensorflow to design above LeNet's Architecture as shown below.

```
import tensorflow as tf
from tensorflow.keras import datasets,layers,models
```

```
[53] CNN = models.Sequential()

     CNN.add(layers.Conv2D(6,(5,5),activation='relu',input_shape=(32,32,1)))
     CNN.add(layers.AveragePooling2D())
     CNN.add(layers.Dropout(0.2))
     CNN.add(layers.Conv2D(16,(5,5),activation='relu'))
     CNN.add(layers.AveragePooling2D())
     CNN.add(layers.Flatten())
     CNN.add(layers.Dense(120,activation='relu'))
     CNN.add(layers.Dense(84,activation='relu'))
     CNN.add(layers.Dense(43,activation='softmax'))
     CNN.summary()
```

The Summary of this model is.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 43)                3655
=================================================================
Total params: 64,511
Trainable params: 64,511
Non-trainable params: 0
```

I used 256 batch size for faster training, and used Adam optimizer, and Loss function of categorical cross-entropy. My metric was “Accuracy”.

```
CNN.compile(loss='sparse_categorical_crossentropy',
              optimizer='Adam',
              metrics=['accuracy'])
```

```
batch_size=256
n_epochs=60
training_data=CNN.fit(X_train_gray_normalize, y_train,
          batch_size=batch_size,
          epochs=n_epochs,
          verbose=1,
          callbacks=[LearningRateReducerCb()],
          validation_data=(X_valid_gray_normalize, y_valid),
          )
```

I trained the model for 60 epochs and for 60th epoch training and validation’s loss, and accuracy is shown below.

```
136/136 [==============================] - 17s 123ms/step - loss: 0.0278 - accuracy: 0.9916 - val_loss: 0.4558 - val_accuracy: 0.9052
Epoch 59/60
136/136 [==============================] - ETA: 0s - loss: 0.0290 - accuracy: 0.9913
Epoch: 59. Reducing Learning Rate from 0.0005582665326073766 to 0.0005526838940568268

136/136 [==============================] - 18s 136ms/step - loss: 0.0290 - accuracy: 0.9913 - val_loss: 0.4255 - val_accuracy: 0.9098
Epoch 60/60
136/136 [==============================] - ETA: 0s - loss: 0.0269 - accuracy: 0.9919
Epoch: 60. Reducing Learning Rate from 0.0005526838940568268 to 0.0005471570766530931

136/136 [==============================] - 17s 124ms/step - loss: 0.0269 - accuracy: 0.9919 - val_loss: 0.4477 - val_accuracy: 0.8993
```

Refinement

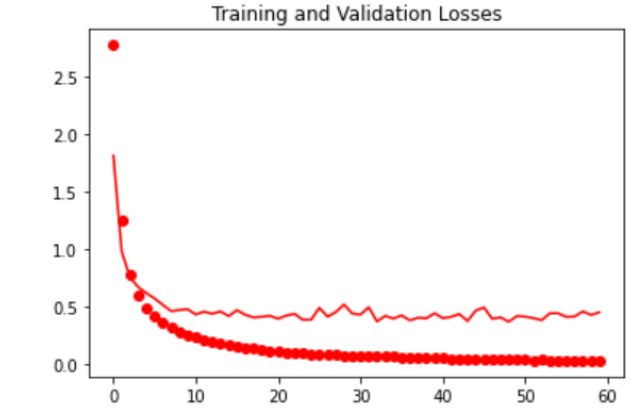
Initially I was getting 80 % validation accuracy for batch size 32 and then I increase the batch size to 256 for faster convergence and epcohs were 30. So I increased the epochs to 40, 50, and finally 60 to get 90%+ validation accuracy.

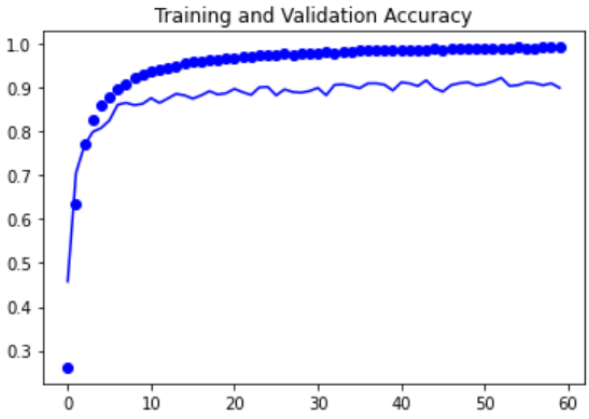
IV. Results

Model Evaluation and Validation

I trained the model by keeping an eye on validation and training accuracy. They were found almost very close to each almost more than 90%.

```
Text(0.5, 1.0, 'Training and Validation Losses')
```



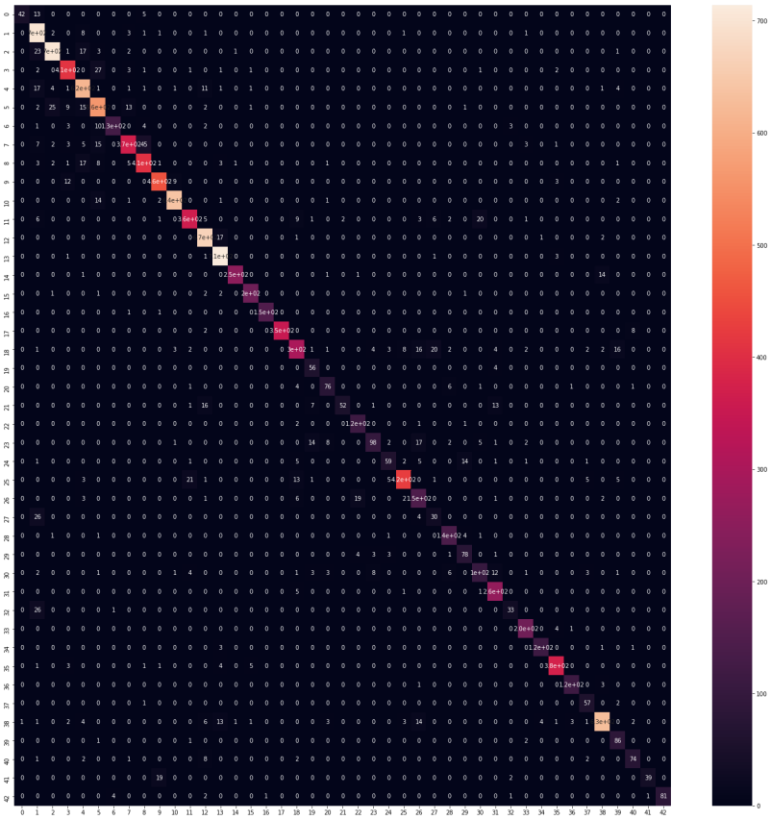


The test accuracy of model is: **90.9%**

```
model_score = CNN.evaluate(X_test_gray_normalize, y_test)
print('Accuracy (Test): {}'.format(model_score[1]))
```

```
395/395 [==============================] - 4s 10ms/step - loss: 0.6250 - accuracy: 0.9100
Accuracy (Test): 0.9099762439727783
```

Further I use the confusion matrix to check the model's performance.



Justification

As shown in link LeNet-5 almost showed more than 90% accuracy. Also I was able to achieve more than 90 % test accuracy for the classification of traffic signs. Thus LetNet-5 was a good choice for traffic signs classification.

```
model_score = CNN.evaluate(X_test_gray_normalize, y_test)
print('Accuracy (Test): {}'.format(model_score[1]))
```

```
395/395 [==============================] - 4s 10ms/step - loss: 0.6250 - accuracy: 0.9100
Accuracy (Test): 0.9099762439727783
```

IV. Conclusions

I was able to classify the German traffic signs with almost more than 90% accuracy.

I check the some of the images from test dataset to check the accuracy of my model. It predicted the label of almost all of the images correctly.

The process used for this project can be summarized using the following steps:

1. Get data from link.

2. Separate all datasets and visualize.

3. Preprocess Dataset.

4. Define LeNet's Model..

6. Train and keep an eye on validation losses and accuracy.

7. Measure Test Accuracy.

8. Finally predict for unseen test data.

Works Cited

Citation 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, 2011.

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; Jacker, L. D. (June 1990). "Handwritten digit recognition with a back-propagation network" (PDF). *Advances in Neural Information Processing Systems 2*: 396–404.

ICVIP 2019: Proceedings of the 3rd International Conference on Video and Image Processing December 2019 Pages 13–18 https://doi.org/10.1145/3376067.3376102

G. Hong, B. Kim, D. Dogra, P. Roy, 2018, A Survey of Real-time Road Detection Techniques Using Visual Color Sensor, Journal of Multimedia Information System (KMMS), 5 (1), (May, 2018), 9--14, DOI= 10.9717/JMIS.2018.5.1.9