Traffic Sign Recognition with a CNN

Dominic Morin (40032939)

Abstract

Convolutional Neural Networks to recognize Traffic Signs are becoming more and more popular with the rise of self-driving cars but their inner working still remain a black box kept from the public. This paper is an effort to create a CNN capable of classifying Traffic Signs and attempts to understand what makes or breaks a Traffic Sign Recognition CNN.

1. Introduction

The main goal of this project is to understand failure modes of CNNs when attempting to classify road signs and come up with an educated solution. The point of all of this is to get insight (or gain confidence) in the real world machine learning applications such as self driving for example.

The primary data source for this project is the Mappilary Traffic Sign Dataset. It contains over 52'000 annotated examples broken down into (36'000 images for training, 5'000 images for validation, and 11'000 images for testing)

Since Traffic Sign Recognition is a an example of a real word problem that puts the lives of people at stake (in self driving cars for example) I am hoping to achieve a reasonable level of accuracy. Especially on the most important classes of signs such as stops, yields and speed limits.

The code for this project is available at github:dominicmathieumorin

2. Methodology

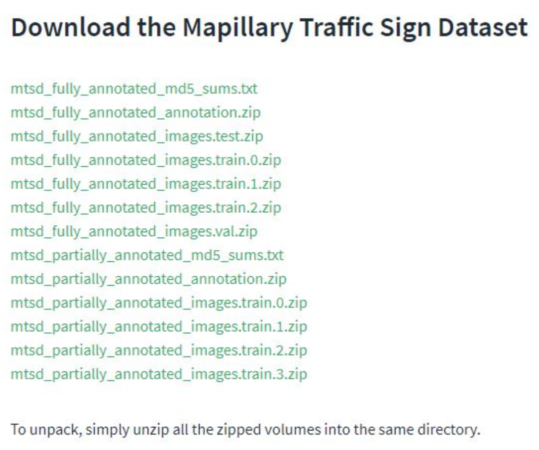
The methodology is quite straightforward:

1. Download all the data from the MTSD website
2. Extract the bounding boxes from the original images and downscale them to 32x32
3. Create train/test/val datasets and save them as .pkl files
4. Build the model in PyTorch
5. Train the model and plot train/test/val accuracy
6. Optimize hyperparameters based on validation set loss
7. Write some visualization tools to make sense of results

2.1. Data Collection and Preprocessing

The code for this section can be found in **preprocessing.py**.

The original data Figure 1 can be found on the MTDS website. The total size of this download is 100G. Once the data is completely downloaded and unpacked it needs to be processed. Figure 2 shows the unprocessed data. Every image consists of a (image, JSON annotation file) pair. The JSON annotation file contains the bounding boxes for all the traffic sign in the larger image. Using the CV2 python library I extracted all the bounding boxes from the individual images and resized them to 32x32 pixels. I then saved the train/test/val into .pkl files so I can load them in memory later for training. Figure 3 shows the processing steps taken for each image.


Download the Mapillary Traffic Sign Dataset

mtsd_fully_annotated_md5_sums.txt
mtsd_fully_annotated_annotation.zip
mtsd_fully_annotated_images.test.zip
mtsd_fully_annotated_images.train.0.zip
mtsd_fully_annotated_images.train.1.zip
mtsd_fully_annotated_images.train.2.zip
mtsd_fully_annotated_images.val.zip
mtsd_partially_annotated_md5_sums.txt
mtsd_partially_annotated_annotation.zip
mtsd_partially_annotated_images.train.0.zip
mtsd_partially_annotated_images.train.1.zip
mtsd_partially_annotated_images.train.2.zip
mtsd_partially_annotated_images.train.3.zip

To unpack, simply unzip all the zipped volumes into the same directory.

Figure 1. Contents of the MTDS website data

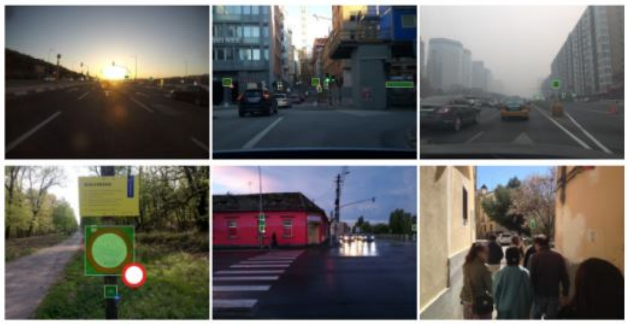


Figure 2. Example images from the train dataset

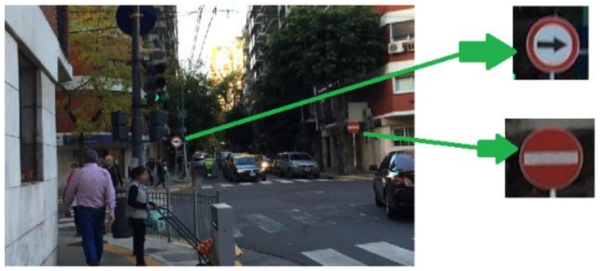
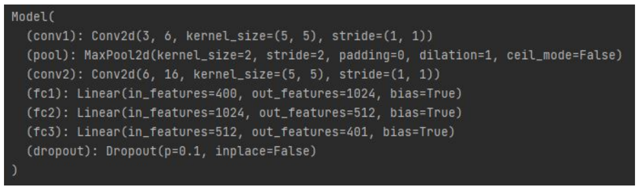


Figure 3. Data preprocessing example

2.2. Building the Model in PyTorch

The code for this section can be found in **model.py**.

The CNN model used in this project was modeled after an existing paper called A Lightweight Model for Traffic Sign Classification Based on Enhanced LeNet-5 Network. The model has diverged from the original paper and adds an extra Fully Connected Linear Layer and also a Dropout Layer to the model as-well-as changing the size of the Conv2D layers and accepting 3 in-channels (RGB) instead of 1 in-channel (Grayscale).



```
Model(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=400, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=401, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
```

Figure 4. CNN Model in PyTorch

2.3. Training the Model and Plotting

The code for this section can be found in **train.py**.

Since Traffic Sign Recognition is Multiclass Classification problem I trained the initial model with a CrossEntropy loss function and the optimizer is Stochastic Gradient Descent. The initial hyperparameters are:

- SGD(learning rate = 0.01, momentum = 0.9)
- Number of Epochs: 100

During the training of the model, I used a Tensorboard Writer to log train/test/loss accuracy Figure 5. I also plotted some random images from the Train Set 6 Test Set 7 and Validation Set 8 to vizualize later.

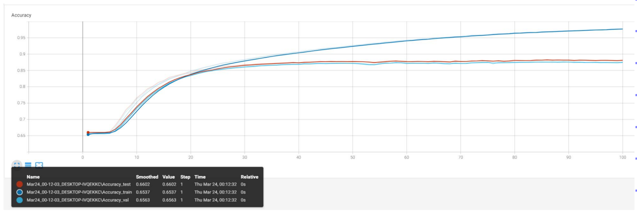
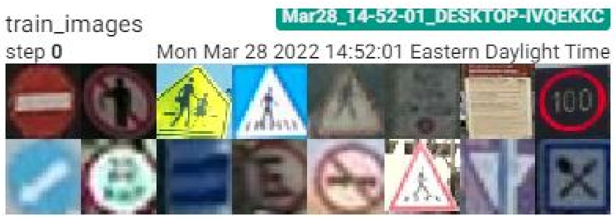
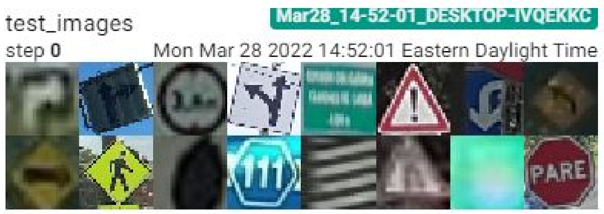
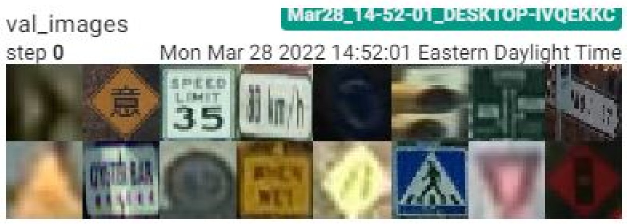


Figure 5. Plotting the train/test/val accuracy during training for 100 epochs


train_images
Mar28_14-52-01_DESKTOP-IVQEKKC
step 0
Mon Mar 28 2022 14:52:01 Eastern Daylight Time
Figure 6. Train Set images


test_images
Mar28_14-52-01_DESKTOP-IVQEKKC
step 0
Mon Mar 28 2022 14:52:01 Eastern Daylight Time
Figure 7. Test Set images


val_images
Mar28_14-52-01_DESKTOP-IVQEKKC
step 0
Mon Mar 28 2022 14:52:01 Eastern Daylight Time
Figure 8. Validation Set images

When the model was done training, I saved the Model class to be loaded up later for inference.

2.4. Result Visualization

The code for this section can be found in **visualization.py**.

Result visualization uses a saved model in eval mode to look at test data and explore best and worst performing classes.

Figure 9 shows a 4-axis diagram displaying correctly-predicted/high-confidence, correctly-predicted/low-confidence, wrongly-predicted/high-confidence, wrongly-predicted/low-confidence. The results of this visualization will be further analyzed in experimental results.

Furthermore the visualisation tools let you plot the learned weights of the first Conv2D layer. Figure 10 shows the weights for the 6 out-channels of self.conv1. Note: the CNN is clearly learning some "proper" features from the images during training.

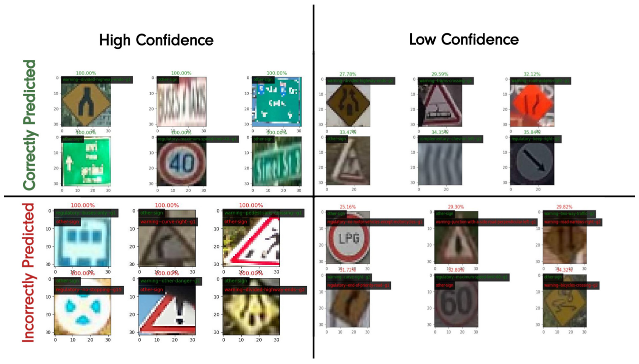


Figure 9. Prediction Comparisons

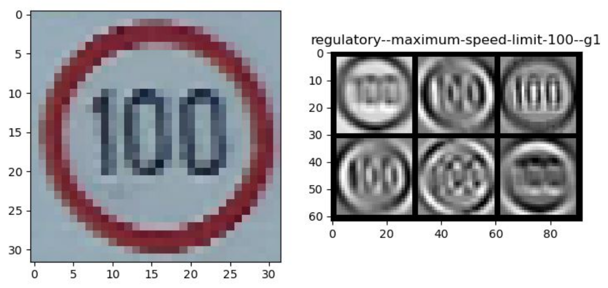


Figure 10. Plotting the Conv2D weights for label: "regulatory–maximum-speed-limit-100–g1"

2.5. Optimization

The code for this section can be found in **optimization.py**.

The only hyperparameter optimization that was done with the model is EarlyStopping. When looking at Figure 5 you can clearly see that while the Validation Set accuracy starts converging towards 87% at epoch 20, the training accuracy keeps rising. This is a sign of over fitting, our model has stopped generalizing on new data but keeps becoming better on the training set. So it better to stop training early. And from the graph epoch 20 is the perfect epoch to stop at.

3. Experimental Results

3.1. Model Accuracy

In the end, according to our training accuracy tracker 5, the model achieved 87% accuracy on the test set in 20 epochs.

3.2. Correctly Predicted Classes

Amongst of the correctly predicted classes 11 (best) and 12 (worst) are:

- maximum speed limit 40 (100%)
- warning divided highways (100%)
- other signs (100%)
- regulatory keep right (35.84%)
- construction sign (32.12%)

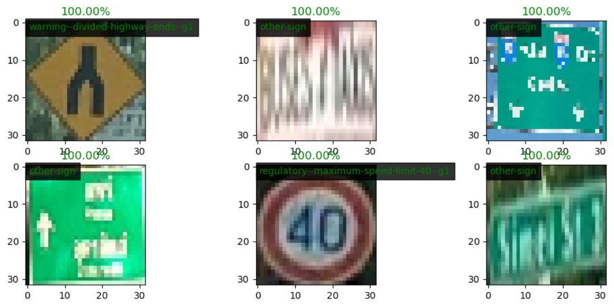


Figure 11. Most confident correctly-predicted classes

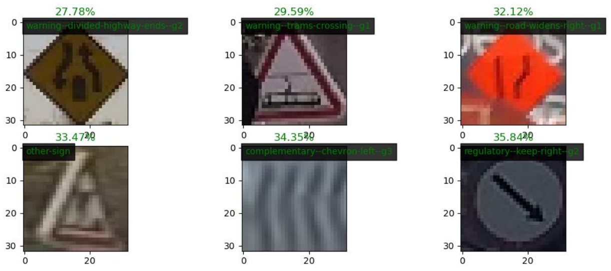

27.78%
warning--divided-highway-ends--g1
29.59%
warning--trams-crossing--g1
32.12%
warning--road-widens-right--g1
33.47%
other-sign
34.35%
complementary--chevron-left--g1
35.84%
regulatory--keep-right--g1

Figure 12. Least confident correctly-predicted classes

3.3. Incorrectly Predicted Classes

Amongst of the incorrectly predicted classes 13 (best) and 14 (worst) are:

- warning danger (100%)

- warning pedestrian (100%)
- bus sign (100%)
- maximum speed limit 60 (32.8%)
- curve right (31.72%)

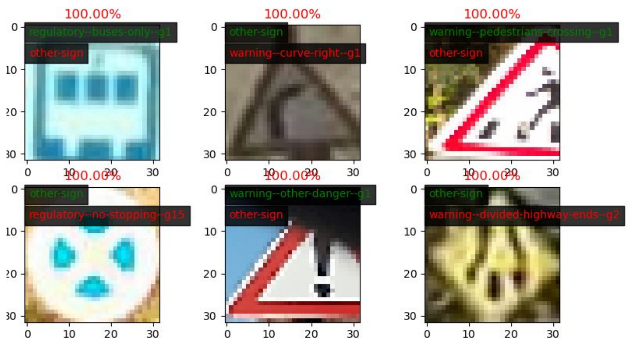

100.00%
regulatory--buses-only--g1
other-sign
100.00%
other-sign
warning--curve-right--g1
100.00%
warning--pedestrians-crossing--g1
other-sign
100.00%
other-sign
regulatory--no-stopping--g15
100.00%
warning--other-danger--g1
other-sign
100.00%
other-sign
warning--divided-highway-ends--g2

Figure 13. Most confident wrongly-predicted classes

It is worth noting here that 2 of the incorrectly predicted classes are either occluded or have partially missing data.

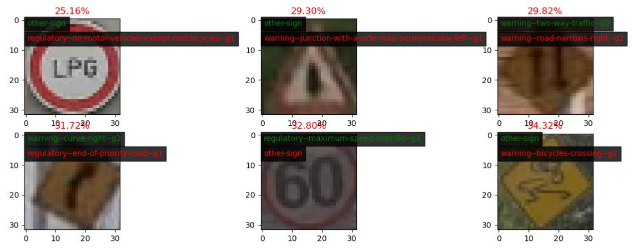
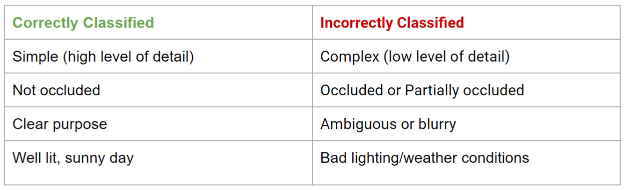

25.16%
other-sign
regulatory--no-motor-vehicles-except-motorcycles--g1
29.30%
other-sign
warning--junction-with-a-side-road-perpendicular-left--g1
29.82%
warning--two-way-traffic--g2
warning--road-narrows-right--g2
31.72%
warning--curve-right--g2
regulatory--end-of-priority-road--g1
32.80%
regulatory--maximum-speed-limit-60--g1
other-sign
34.32%
other-sign
warning--bicycles-crossing--g2

Figure 14. Least confident wrongly-predicted classes

3.4. Quality of low-level Conv2D Filters

According to the plotted Conv2D weights for the label: "regulatory–maximum-speed-limit-100–g1" 10, the low-abstraction filters learned by the first Conv2D layer of the model seem to have learned some meaningful weights and are accurately able to distinguish a "maximum-speed-limit" traffic sign.

4. Conclusions



Correctly Classified	Incorrectly Classified
Simple (high level of detail)	Complex (low level of detail)
Not occluded	Occluded or Partially occluded
Clear purpose	Ambiguous or blurry
Well lit, sunny day	Bad lighting/weather conditions

Figure 15. Where CNN's fail

From Figure 15 it's clear that a big component of failure is the prepprossessing of images. Low resolution and occluded images are very hard to interpret and result in missclassifications. Sometimes downsampling an image to 32x32 pixels makes the image lose too much meaningfull data. Maybe it would be better to use larger images such as 64x64 or even 128x128 pixels Also, there is a "other-sign" catch-all class in the dataset and when the network isn't sure what an image is it seems to correcly classify it as "other-sign". This contributes to overestimating the accuracy of the model and doesn't contribute to learning the more important classes such as speed limits and stops.

From the Experiments and Results section I can confidently say that the model presented in this paper, at 87% accuracy, is NOT good enough to be put in the real world and drive a self driving car. Moreover the study of Incorrectly Predicted Classes show that the model sometimes incorrectly predicts with a 100% confidence traffic signs that could endanger the lives of either the driver or pedestrians. For example: in 13 it predicted with 100% certainty that a "warning-pedestrian-crossing-g1" was a "other-sign" which is a big enough mistake to kill somebody.