Using Faster RCNN for Traffic Sign Detection

Clement Yan
cleyan@connect.hku.hk

Leo Lu
u3507111@connect.hku.hk

1 Introduction

In our project, we attempt to compare and constrast different image processing and computer vision techniques for traffic sign detection. On one hand, we used the OpenCV library to implement shape recognition and traditional machine learning methods. On the other, we used Convolutional Neural Networks, and in particular, Faster Region-based Convolutional Neural Networks (FRCNN) to perform bounding box regression and classification.

The document serves as a readme file and a mini-report for the CNN side of the project.

2 Background

2.1 Datasets

The datasets we used are the German Traffic Sign Recognition Benchmark and the German Traffic Sign Dectection Benchmark[3], each of which was used in a competition.

The German Traffic Sign Recognition Benchmark (GTSRB) consists of 50,000+ images, each with a single traffic sign covering most of the image. The competition task was classification of the images into 43 classes.

The German Traffic Sign Detection Benchmark (GTSDB) consists of 900 images, each with possibly multiple traffic signs in a wide scene. The competition task was detection of 3 classes of traffic signs (Prohibitive, Mandatory, or Danger).

2.2 Faster RCNN

Faster Region-based Convolutional Neutral Networks (FRCNN) is an object detection technique published in 2016[4] which provides a number of advantages over existing techniques at the time. It introduces the use of a region proposal network (RPN). In FRCNN, a base network, usually the convolutional part of a pretrained network like vgg, outputs a feature map that is fed into a small,

2-layer RPN to produce region proposals – rough regions with high probabilty of being regions of interest (RoIs). Then the feature map and the region proposals are fed into a fully-connected classifier network, via a RoI pooling layer, to refine the bounding box and produce class probabilities. In our project, we attempted to use FRCNN for the detection task of GTSDB.

3 Codebase

3.1 Libraries

For this part of the project, we used the following libraries and existing code:

- Keras[1], a high-level library that provides API abstraction over other neural network frameworks, including TensorFlow.
- TensorFlow, which is needed as a backend for Keras.
- keras-frcnn[2], an implementation of FRCNN in Keras

3.2 Datasets

The `dataset` directory contains preprocessed versions of the GTSRB and GTSDB datasets. In particular, all images were converted from ppm format to png format using ImageMagick (version 8:6.8.9.9-7ubuntu5.9) that comes pre-installed in our Ubuntu 16.04 installation. The original dataset is available online.

The `PNG_train` and `PNG_test` directories contains the GTSDB dataset training data and testing data respectively. The annotation file `gt.txt` in the training data was changed to accomodate the change of the image file extension and convert into comma-separated fields instead of semicolon-separated fields. The readme files are also copied from the original dataset.

The `GTSRB` directory contains the GTSRB dataset. The 12630-image test set is in the same structure as the original dataset. The 36000-image train set and 3209-image validation set are both taken from the original train set. 3209 images from that set was randomly choosen to be the validation set and the remaining 36000 images are used as the test set. The ground-truth annotations are not used as Keras can infer the class based on the directory sturcture.

3.3 keras-frcnn

The `keras-frcnn` directory contains a clone of the `keras-frcnn` Github repository. We have made some modifications in order to fit our purpose. Major ones include reducing epoch size to 600 to match the dataset and implementing a simpler base (feature map generation) neural network instead of using a pre-trained vgg16 network.

Also included are:

- an hdf5 file containing pretrained weights for the vgg16 network
- config and trained weights for our trained model
- a directory containing result images

3.4 GTSRB and GTSDB

These directories contains our own code using the Keras library. Details of our implementation is discussed below. The `GTSRB` directory contains code for the GTSRB classification task, and the `GTSDB` directory contains code for the GTSDB detection task. Each of them contains subdirectories `models` for saving the model and training checkpoints, and `tblogs` for Tensorboard visualization. See below for more details on using Tensorboard.

Pretrained model files are included for in `GTSRB/models` the GTSRB code. Each file is labelled with the epoch number. For convenience, the fully trained weights file is also labelled `last`.

4 Installation

Since our codebase is actually 2 codebases combined. Running each of them requires a different environment setup. Therefore, we used Anaconda as our package and environment manager, and we encourage anyone using the codebase to do the same.

4.1 keras-frcnn

To run the `keras-frcnn` code, the following environment and packages (and their dependencies) are needed:

- python 2.7
- tensorflow
- keras
- opencv
- h5py

In our setup, we first created a new Anaconda environment with python 2.7

```
$ conda create -n keras-frcnn python=2.7
$ source activate keras-frcnn
```

Then installed TensorFlow and Keras via pip as recommended

```
(keras-frcnn)$ pip install tensorflow
(keras-frcnn)$ pip install keras
```

Finally, installed other dependencies through Anaconda

```
(keras-frcnn)$ conda install opencv
(keras-frcnn)$ conda install h5py
```

4.2 GTSRB and GTSDB

To run the `GTSRB` and `GTSDB` code, the following environment and packages (and their dependencies) are needed:

- python 3.6
- tensorflow
- keras
- h5py
- scikit-image
- matplotlib
- jupyter (optional)

Anaconda comes with Jupyter notebook so it does not need to be installed separately. Installing the other packages are similar to the setup procedure above

```
$ conda create -n gts
$ source activate gts
(gts)$ pip install tensorflow
(gts)$ pip install keras
(gts)$ conda install h5py
(gts)$ conda install scikit-image
(gts)$ conda install matplotlib
```

5 Running the code

5.1 keras-frcnn

To train a model with `keras-frcnn`, run

```
$ python train_frcnn.py --network=simple \
> --parser=simple --path=../dataset/PNG_train/gt.txt
```

If the working directory is not `keras-frcnn`, change the script and `gt.txt` accordingly.

For starting the training using pretrained vgg network, make sure the included hdf5 file for vgg weights is in the working directory, and use `--network=vgg` instead of simple.

To test our trained simple network, run

```
$ python test_frcnn.py --network=simple \
> --path=../dataset/PNG_test \
> --config-filename=simple_config.pickle
```

Result images will be generated in the `results_imgs` directory. Some of the code in the python script is commented out to prevent text labels from being generated in the image and to avoid generated image poping up from slowing progress. Uncomment lines as appropriate for your use case.
To test other trained networks or datasets, change the command line options accordingly. Note that a `results_imgs` directory needs to be present in the working directory for the result images to be saved.

5.2 GTSRB

To train a model with the GTSRB code, run

```
$ python GTSRBTrain.py
```

This script does not accept command line arguments. Instead most values are hardcoded. Change as appropriate. Note that the hardcoded directory paths are relative to the working directory.

The training lasts at most 100 epochs but may stop early if the validation accuracy stops improving (sign of overfitting). Trained weights are saved in the `models/gtsrb3` directory and Tensorboard logs are saved in the `tblogs/gtsrb3` directory, see below for instructions on using Tensorboard.

To continue training from a previous run, change the `CONTINUE` value to the last saved checkpoint number.

To test a model, run

```
$ python GTSRBTest.py
```

By default, the script loads the fully trained `gtsrb1` model and tests it on 1000 random images in the test set of GTSRB. The collection of images predicted for each class are displayed using matplotlib.

Alternatively, Jupyter notebook is provided for training. Jupyter is a web-based interactive environment for running snippets in Python and various languages. See the documentation for more details on how to run the notebook.

5.3 GTSDB

To train an RPN with the GTSDB code, run

```
$ python GTSDB_RPN_Train.py
```

Trained weights are saved in the `models/gtsdb_rpn` directory and Tensorboard logs are saved in the `tblogs/gtsdb_rpn` directory, see below for instructions on using Tensorboard.

An annotated Jupyter notebook is also provided.

5.4 Tensorboard

Tensorboard is a visualization utility provided by TensorFlow. The GTSRB and GTSDB training scripts both have hooks to writing training events into their respective `tblogs` directory. See the official documentation for more details.

To launch Tensorboard, run

```
tensorboard --logdir=<path>
```

where `<path>` is the appropriate tblogs directory. Note that Tensorboard has a known issue that warnings are generated if more than one event file is in the same directory. Change the log directory for each training session to avoid this.

6 Methods

6.1 GTSRB

For the GTSRB classification task, we use a simple CNN with 2 convolutional layers, 2 pooling layers and 2 fc layers. Implementing this in Keras is surprizingly easy, achievable within 50 lines of code. See the training script.

Because the network is a simple one, training finished very quickly when the validation accuracy plateaued at 99%. However, testing the model on the test set shows that the accuracy may not be as high as advertized. This may be due to high homogeneity in the training data, reducing the efficacy of the validation set to guard against overfitting. We will investigate more in the results section.

6.2 GTSDB

For the GTSDB detection task, we attempted to use a Faster RCNN to train a detector. At first, we used the existing codebase `keras-frcnn` on Github. However, because the training is slow for a large base network like vgg, and preliminary results show that a large base network maybe unsuitable for this task. This is because the RPN use anchor boxes centered at each pixel at the input feature map, which means adjacent anchor boxes are actaully separated by the

stride of the base network. By stride, we mean the factor by which the base network downsamples, so called because that is the amount moved in the original image space corresponding to moving a single pixel in the feature map space.

A large base network stride cause problems because then the anchor boxes essentially "misses" the ground-truth box by taking large strides, producing low intersect over union (IoU) values between them and thus few positive anchors for each image. Training the RPN involves sampling 256 anchors per images with at most 128 positive anchors to prevent biasing towards negative anchors. However, with less than 128 postive anchors, more negative anchors are chosen to pad to 256, defeating the propose of such. sampling.

The vgg16 network used by `keras-frcnn` has a stride of 16, which is very close to the average size of the traffic signs we are trying to detect (32×32).

To combat the problems with `keras-frcnn`, and also in order to better our understanding of FRCNN, we decided to both create our own implementation, and to modify the `keras-frcnn` to use a simpler base network. Due to limited time and processing power we possess, we have only implemented the RPN part of the FRCNN and was unable to do any significant training using that code.

Some of our code is adapted from `keras-frcnn`. However, as can be seen, our code is much more compact and readable than `keras-frcnn`. We also believe it is much faster because we perform bulk numpy array manipulation during data load and use a single call to `model.fit_generator` in training. By delegating the looping to library code instead of our own loops in Python code, we can take advantage of highly efficient low-level code in numpy and Keras.

Our implementation differs from the one described in the FRCNN paper at a few places. First, the paper uses 1:1, 1:2 and 2:1 anchor boxes with sizes 128, 256, and 512. We only use 1:1 anchor boxes with sizes 16, 32 and 64. The box sizes and ratio are chosen to match the size of the objects we are trying to detect. Second, we do not perform balancing between nor normalization in classification loss and regression loss in the RPN. Implementing these would not be too difficult, but the authors of the FRCNN paper notes that they have determined emprically that the RPN training result is not sensitive to these changes.

7 Results

7.1 GTSRB

As noted about, we have trained a classification network for GTSRB that has achieved 99% training and validation accuracy. Because some of the images in the dataset are of very bad quality, we believe 99% is better than human accuracy.


02577, an example of low-quality images from the GTSRB test set

However, as noted above, the testing data shows that the accuracy may not be as high as advertized. (Since no ground-truth label is provided for testing data, this is just gut instincts while looking at testing script output) This may be due to the fact that the training data were cropped stills of a video image sequence. Thus, many of the images are very similar. This reduces the effectiveness of the randomly sampled validation set.

7.2 GTSDB

While the original FRCNN paper has shown very good results when tested agains the Pascal VOC dataset, we were unable to perform any GPU training with out equipment. Constrainted in both time and resource, we are only able to train a FRCNN using a simple base network for 5 epoches, which are insufficient to produce significant results on GTSDB. To run the trained network, consult the "Running the code" section above.

20 images from the train and test sets are tested on the trained network, and the results can be found in the `keras_frcnn/results_imgs` directory. Almost all of the identified regions are true background regions, where there are no traffic sign. Luckily, the RPN has already learned something about interesting areas, as seen by the fact that it does not propose regions of constant intensity (e.g. the sky). The classifier also correctly classified the region proposals as background. While some regions overlap with traffic sign, most of them are too big, too small or too displaced for the classifier to identify as traffic signs. The few good proposals are also classified as background because the classifier has not learned enough.

However, in `Test_20.png`, one of the regions is good enough that the classifier correctly classified as class 38 with score 0.4869. This shows that both the RPN and the classifier are starting learn. We believe that, with more training, these positive samples will back propagate into the classifier and RPN so that the network can eventaully perform fairly well in this task.

8 Conclusion

In the other part of the project, we have shown that, using OpenCV to perform shape identification and SVM classification is a very effective strategy at detection. Comparatively, the disadvantages of using CNN and FRCNN for traffic sign detection becomes obvious – that training a good network both require a large dataset and a lot of time. FRCNN is also conceptually much more complex.

Very early on into the project, we have already implemented a very effective traffic sign locator in OpenCV using shape identification, and the SVM classifier has comparable or higher accuracy as the one produced by CNNs. In constrast, coding an FRCNN takes much more effort and training one takes considerable amount of time. Also, the GTSDB is not a very large dataset

All in all, we believe that for traffic sign detection is much better solved using traditional computer vision techniques using OpenCV. Keeping in mind that Faster RCNN and its predecessors are designed for general object detection tasks – with a large set of diverse object classes and large datasets (e.g. ImageNet, Pascal VOC), using it for traffic sign detection is both overkill and not its intended purpose.

References

[1] Keras. URL: `https://keras.io/`.

[2] Yann Henon. yhenon/keras-frcnn, August 2017. original-date: 2017-01-04T21:44:21Z. URL: `https://github.com/yhenon/keras-frcnn`.

[3] Institut fr Neuroinformatik. German Traffic Sign Benchmarks, July 2015. URL: `http://benchmark.ini.rub.de/index.php?section=home&subsection=news`.

[4] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL: `http://arxiv.org/abs/1506.01497`.