Traffic Sign Recognition System

M.Eng. in Robotics and Autonomous Vehicles, Capstone Project Report

John Titus

Integrative Systems + Design,
College of Engineering,
University of Michigan, Ann Arbor

Supervised by,

Dr. Jason J. Corso

Electrical Engineering and Computer Science,
University of Michigan, Ann Arbor

Abstract

This report presents a Traffic Sign Recognition (TSR) System built with pre-trained Convolutional Neural Network descriptors. The Network used in the project is similar to the one used by team Supervision (Krizhevsky et al.) to win the in the ImageNet 2012 challenge with impressive results. The project had three phases; annotation of video, generation of data set and implementation of the system. In the annotation phase, video clips containing traffic signs were annotated using VATIC. The annotations were used to generate 256x256 images containing the signs at different positions and scales using matlab scripts. The data set consists of 10,000 samples for each signs and the training, validation and test sets were divided such that it was stochastically independent. These images were passed through the pretrained Convolutional Neural Network, and the first fully connected layers output was used to learn a linear SVM classifier. Finally the system was implemented on the Motorola Moto x powered by Qualcomm Snapdragon 801, running the android OS.

I. Introduction

Traffic Sign Recognition is a real-world computer vision problem of high practical relevance and has been a research topic for several decades. There is no question about the importance of Traffic Sign Recognition Systems in vehicles; this is evident as they has found their way into a few cars in Europe. It will be quite a while before all cars get these systems as a standard feature. To bridge the gap between now and then, smart phone application developers have tried to implement a TSR system on mobile devices. Though there is some success in the space, there is a lot of room for improvement.

There are more than 120 types of signs belonging to 8 major categories used in the U.S. It is not within the scope of this project to be able to build a system that can recognize all the signs. An article in the 'Journal of Safety Research' claims that "Stop sign violations accounted for about 70% of all crashes in cities" [10], another book on road safety claims that human beings have very poor memory of seeing pedestrian signs and game signs as compared to speed limit signs [12]. This shows that that not all signs

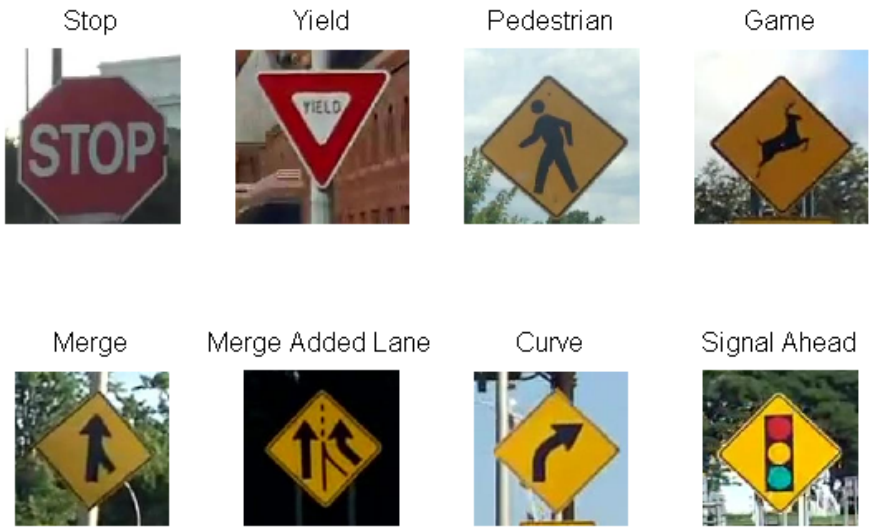

Fig. 1: Signs targeted to be recognized

needs to be weighed equally, hence for the project a set of eight signs (see Fig 1) were chosen depending on their importance and also availability of videos.

The report is organized as follows: Section II provides an overview of the Convolutional Neural Networks used. Section III provides an overview of the annotations. Section IV explains how the data set was generated. Section V discuses the classifier and its performance. The implementation specifics are presented in section VI. The report is finally concluded in section VII with plans for future work in section VIII.

II. Convolutional neural networks

The impressive performance of the Deep Convolutional Neural Network in the ImageNet 2012, as described by Krizhevsky et al. [8] attracted a lot of attention in the vision community. A study evaluating the performance of the CNN's descriptors on other datasets can be seen in 'Neural Codes for Image Retrieval', Babenko et al [9].

A. Network Architecture

The network architecture (Fig 2) contains eight layers, five convolutional and three fully-connected layers. A detailed explanation of the network can be found in [8].

B. DeepBeliefSDK

DeepBeliefSDK [1] is a pre-trained network library, that implements the Deep Convolutional Neural Network as described by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton [8], and trained on the ImageNet 2012 dataset. It provides libraries for a variety of platforms including android and embedded

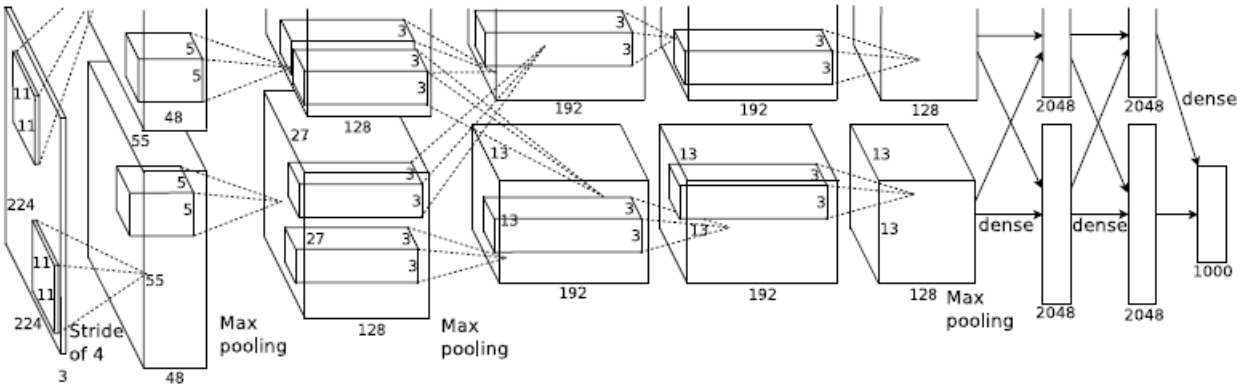

Fig. 2: CNN architecture
(source: 'ImageNet Classification with Deep Convolutional Neural Networks', Krizhevsky et al. [8])

linux distributions running on arm. According to the implementation, the CNN uses 'Rectified Linear units' (ReLU) with 'local response Normalization'. All the convolutional layer and the first two fully connected layers implement these neurons. DeepBeliefSDK allows the user to specify the layer and the type of output (raw, ReLU or Local response Normalized) required.

III. Annotation

The first phase of the project was to make up a collection of dashboard videos, and annotate them. The sources from which the videos were collected, a brief description of the annotation tools available and the annotation process is explained in this section.

A. Video

A youtube channel [2] containing a large collection of dashboard footage of drives through interstate highways was most helpful for the project. A few of these videos containing the targeted signs were selected to be annotated. The videos were selected such that the signs had visibly different backgrounds and were in different lighting conditions. Additionally few videos were recorded with a phone mounted on the dashboard. The videos from youtube were primarily used as the training set, while the videos from the phone serves as the test set.

B. Annotation tools

There are a lot of annotation tools available, a few were considers for the project, but finally VATIC was chosen and reasons are briefly explained in this section. LabelMeVideo [3], is an extension for annotating videos in the popular image annotation tool LableMe [4]. The tool is still in the early stages of development and a general opinion is that it is very buggy. A popular software for video annotation is Anvil. The tool has a lot of options and settings especially for annotating videos of humans. The project did not need all the functionally offered by the tools, and in fact it would have made annotations that much more complex. Finally a tool that is slowly gaining popularity in the vision research community, VATIC [5] was chosen. VATIC is a free, interactive Video Annotation Tool from Irvine, California. It can

be used either in offline or online mode. In the online mode it can crowd-source work to AmazonâĂŹs Mechanical Turk. Thought the tool is not what is considered production level it was good enough for the project.

C. Annotation process

VATIC was installed on ubuntu and used in the offline mode for the project. The VATIC installation guide [5] provided by the author is limited, as it does not document the installation procedure for the supporting systems. A more comprehensive installation guide for ubuntu, documenting procedures for a few of the supporting systems, can be found here [6]. A complete installation guide for ubuntu, taking elements from both these documents, was compiled and is given in Appendix A.

The videos to be annotated were first split into frames using ffmpeg. These frames were then formates into the desired folder structure required by VATIC and then loaded onto the server for annotation. All the warning signs, not just the targeted signs were annotated in the videos. A few pointers that were observed, and found to be useful, when annotating signs are given in Appendix B.

D. Results of Annotation

TABLE I: Annotated Signs Count - Targeted

S.No	Sign	32-63	64-95	96-127	128-159	>=160	Total
1	Stop	428	181	65	49	19	742
2	Yield	513	129	35	7	0	684
3	Pedestrian	494	143	43	28	22	730
4	Game	358	110	34	5	0	507
5	Merge	566	195	76	16	5	858
6	Merge Added Lane	500	317	82	39	20	958
7	Curve	377	113	20	5	1	516
8	Signal Ahead	525	143	54	17	8	747
	Total Targeted Signs	3761	1331	409	166	75	5742
	% of Total Targeted Signs	65%	23%	7%	3%	1%	100%

TABLE II: Annotated Signs Count - Other

	32-63	64-95	96-127	128-159	>=160	Total
Other Signs	2847	1321	256	64	38	4526
% of Other Signs	63%	29%	6%	1%	1%	100%

The final tally after annotating the videos are given in tables I,II. Each row of the Table I shows the number of signs belonging to each targeted class. The column show the number of signs according to the size; for example the third column in the table shows the number of signs in each class that had signs of size 32x32 to 63x63 pixels. The last two rows show the total number of targeted signs in each size category, and the percentage of targeted signs belonging to each size category respectively. Table II show the total number of other signs in each size category, and the percentage of other signs belonging to each size category.

IV. Data Set

In the "The German Traffic Sign Recognition Benchmark" challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011, teams achieved super-human performance on using CNN. This gives confidence that CNN is the way to go to achieve good results, but one has to be cautious since the competition was primarily geared toward classification, rather than detection,

The goal of the project is to implement a real world system capable of both detecting and classifying signs, where as the competition was primarily geared toward classification, rather than detection, since each image contains exactly one sign without much background.

The input image size of CNN is a 256x256 image. As seen in the table I the annotations are of a variety of sizes, to be able to process the signs, images of sizes 256x256 were sampled of the signs by scaling and cropping as necessary, even negative samples were generated by sampling the background.

A. Positive Samples

The frames were scaled and cropped such that the size of the sign was between 64x64pixels and 256x256 pixels. The signs were sampled while it was placed at the center and also placing them in a random position inside the image. All the positive sampled images were put under the respective signs in the 'Pos' folder and all the negative samples were put in 'Neg' folder of the data set.

TABLE III: Initial Classification Performances

	Min Sign Size - 32x32		**Min Sign Size - 64x64**	
Size	**True Positive**	**False Positives**	**True Positive**	**False Positives**
32-63	41%	37%	-	3%
64-95	62%		50%	
96-127	79%		76%	
128-160	91%		87%	
>=160	100%		100%	

A subset (373 training and 74 testing) of the 'yield' signs were scaled to different sizes and cropped to 256x256 images, this made up the initial positive samples. The negative samples were sampled around the annotations of the sign. This initial training set had about 8000 samples and the initial test set had about 1000 samples. This set was passed to the CNN and the output of the penultimate layer was stored. Two Neural network classifiers were learned on the penultimate layer of the CNN. One trained on the entire training set and the other only on signs greater than 64x64 pixels, the results are shown in the table III. It can be seen that though the True Positive rate of the network trained on the entire set is greater than the other, its false positive (37%) is high . For this reason it was decided that the samples generated would have a minimum size of 64x64 pixels.

B. Data Augmentation

The data was augmented by scaling and translations alone. The scaling and positioning strategies used to sample the signs are described here. One sample was generated by cropping to the annotation and then the sign was scaled so that it occupy almost the entire 256x256 image. Each frame was scaled to 0.5, 0.75, 1, 1.5, 2, 2.5, 3, 4, 5 depending on the size of each annotation. For example: if the annotated size is 50x50 pixels the frame was scaled to 1.5, 2, 2.5, 3, 4 hence the sampled images have signs

Fig. 3: An Example of all the samples generated from a single frame

of sizes 75x75, 100x100, 125x125, 150x150, and 200x200 respectively. After scaling the frames to all possible scalings, a sample was generated by placing the sign at the center of the image. Samples are also generated by positioning them randomly in the image. The number of such random samples generated varies depending on the size of the sign; larger sign are sampled less and smaller sign are sampled more. As the size of the sign increases the variation in position in the image decreases hence lesser number of samples are required to represent them

TABLE IV: Augmented Data

S.No	Sign	Total
1	Stop	11089
2	Yield	11185
3	Pedestrian	11391
4	Game	8200
5	Merge	13479
6	Merge Added Lane	14502
7	Curve	8104
8	Light Ahead	11867
	Total	95240

C. Negative Samples

A portion of negative samples were generated by sampling around a sign. When a sign is present in the frame, samples are generated on all sides of the sign. Sampling successive annotated frame produced similar samples which isn't really necessary and may in fact prove counter productive. So only one frame in every four frames is sampled. Additionally frames were also sampled at random positions to get a diverse set of samples. To do this, only those frames are selected that don't have any annotations in them, and a 256x256 section was sampled at random.

D. Compiling the Data Set

All images of one traffic sign was assigned to the same set, as otherwise the datasets could not be considered stochastically independent. Splitting the data set was done by matlab script after the whole dataset was passed to the cnn. This allowed for handcrafting the training data set by removing bad samples that skew the validation performances.

V. Training The Classifier

Initially Artificial Neural Networks were trained to be the classifier. The idea was that it would maintain the flow if future implementations would use hardware accelerated libraries for neural networks. But the results of the ANN were slightly worse off than SVM so finally a linear-SVM classifier was trained for the final implementation. While training the SVM 3000 samples belonging to the class and 3000 samples from every other class were taken at random. The SVM was trained in the one vs rest model, the model was also trained to give out the probability estimates so that the class with the highest probability was chosen as the classification.

TABLE V: Confusion Matrix SVM

		Predicted							
		Stop	**Yield**	**Pedestrian**	**Game**	**Merge**	**Merge Added Lane**	**Curve**	**Signal Ahead**
Actual	**Stop**	92%	1%	0%	0%	0%	0%	0%	7%
	Yield	4%	94%	0%	0%	1%	0%	0%	0%
	Pedestrian	0%	0%	27%	6%	5%	10%	2%	50%
	Game	0%	0%	0%	79%	14%	0%	7%	0%
	Merge	0%	0%	10%	1%	79%	0%	6%	3%
	Merge Added Lane	0%	0%	20%	4%	0%	17%	59%	1%
	Curve	0%	0%	22%	0%	2%	0%	75%	1%
	Signal Ahead	1%	1%	5%	0%	27%	0%	1%	65%

The performance of the classification is shown in table V confusion matrix. The Merge Added Lane and the Pedestrian sign performed the worst among the group with only 17% and 27% true positives respectively. The highest miss-classification was that of Merge Added Lane being classified as a curve sign, and pedestrian signs being classified Signal Ahead.

VI. Implementation

The strategy followed for detecting the sign is to keep sampling in the area where the sign is most probable to occur. To find the most probable location a heat map (fig 4) was created from the annotations. The maximum temperature was found to be around (880,300) from the map. Input images of 256x256 were sampled around this area. This image was fed into the CNN and its sixth layer (the first fully connected layer) output array was passed to a svm_predict module. svm_predict loads the stored models of the classifier and predicted the class.

The system was installed on Motorola Moto x powered by Qualcomm Snapdragon 801 and 2GB of RAM. The phone runs on a Pure android 5.0 OS. The results are by observation and are not yet quantify. The application was a bit unstable but it was able to detect at least 80% of the signs and was able to classify the signs as well. The phone was only able to process 1.18 frame per second (71 frames in 60 S), this could be one of the reasons for not having a higher detection rate seen with the data set.

Fig. 4: Heat Map

VII. Conclusion

"Finding the optimal architecture of a ConvNet for a given task remains mainly empirical" [11]. That being said, implementing an Off-the-shelf Convolutional Neural Network descriptors to perform a task it wasn't specifically designed to do, and still performing a good enough job, saves a lot of time in designing domain specific network and training it.

The project successfully implemented an off-the-shelf pre-trained Convolutional Neural Network for recognizing traffic signs. The project successfully used the annotations from VATIC to generated input images for the DeepBeliefSDK whose output was used by libsvm to train a classifier.

The project can be of value to those trying to implement other recognition tasks using pre-trained Convolutional Neural Network. The project's code can be considered a framework to seamlessly handled the flow of data between the tools (VATIC, DeepBeliefSDK and libsvm). The matlab scripts used to handle the annotations and extract the signs can be considered as a tool by itself.

The code can be found in https://github.com/titusjj/TSR

VIII. Future Work

The Jetson TK1 dev kit is a powerful embedded platform and is gaining popularity among roboticist interested in vision. The default OS, L4T a flavor of embedded linux which runs on arm is not able to support the DeepBeliefSDK at this time. A work around this would be to flash the board with jetdroid (android for Jetson), as described here [7]. And then run the android apk on the board. The main draw back of this approach would be that it would limits the access to the CUDA platform, which can be used to accelerate the CNN.

Signs mostly appear on the right edge of the road, and only very rarely does a sign appear on the left but doesn't have the same sign on the right. Taking this into account a better positioning strategy would be, to detect the right edges of the road, and use this information to crop the frame.

The project treats the TSR as a discrete problem, but in reality it is a continuous problem and using the temporal information may provide valuable information for improving the performance of the system. Some promising research use Recurrent Neural Networks to deal with Temporal information, these avenues can to be explored more for these types of problems.

APPENDIX

A. *VATIC installation guide*

Install Apache 2 web server

```
sudo apt-get install apache2
```

Make sure it works by typing in "localhost" in the address bar of the browser. You should see the default apache server homepage (or) a page saying "It works!".

Next install and activate mysql and the module that ties Apache 2 web server to MySQL database. During the installation it will prompt for a root password, it is recommended to fill in a $< password >$

```
sudo apt-get install mysql-server libapache2-mod-auth-mysql
sudo mysql_install_db
sudo /usr/bin/mysql_secure_installation

sudo /etc/init.d/mysql restart
# If running the above command throws this error:
# ERROR 2002 (HY000): Can't connect to local MySQL server through socket
   '/var/run/mysqld/mysqld.sock' (2)
# A system restart may be required.
```

Log into the mysql command window using the root as the username and the password provided during installation.

```
sudo mysql -u root -p
Enter password:<password>
```

Using the mysql command window create a databases named "vatic" with the following command.

```
mysql> create database vatic;
mysql> exit;
```

Download and install python and supporting tools.

```
sudo apt-get install python-setuptools
sudo apt-get install build-essential
sudo apt-get install autoconf libtool pkg-config python-opengl python-imaging
   python-pyrex python-pyside.qtopengl idle-python2.7 qt4-dev-tools
   qt4-designer libqtgui4 libqtcore4 libqt4-xml libqt4-test libqt4-script
   libqt4-network libqt4-dbus python-qt4 python-qt4-gl libgle3 python-dev
sudo apt-get install libapache2-mod-wsgi

sudo a2enmod mod-wsgi
# if the above command does not work, try the command given below.
# $ sudo a2enmod wsgi

sudo apt-get install python-mysqldb
sudo easy_install munkres
sudo apt-get install python-pip
```

```
sudo pip2 install parsedatetime
```

Install libjpeg-dev and python imaging Library Pillow.

```
sudo apt-get install libjpeg-dev
pip install -I pillow
```

Create symbolic links in /usr/lib

```
# if on a 64 bit system:
sudo ln -s /usr/lib/x86_64-linux-gnu/libjpeg.so /usr/lib
sudo ln -s /usr/lib/x86_64-linux-gnu/libfreetype.so /usr/lib
sudo ln -s /usr/lib/x86_64-linux-gnu/libz.so /usr/lib

# if on a 32 bit system:
sudo ln -s /usr/lib/i386-linux-gnu/libjpeg.so /usr/lib/
sudo ln -s /usr/lib/i386-linux-gnu/libfreetype.so.6 /usr/lib/
sudo ln -s /usr/lib/i386-linux-gnu/libz.so /usr/lib/
```

Re-install pillow

```
pip install -I pillow
```

Download FFmpeg, extract the tar ball and from the extracted directory run these commands.

```
sudo apt-get install -y yasm frei0r-plugins-dev gnutls-bin libgnutls-dev
   libass-dev libgsm1-dev libmp3lame-dev libopencv-dev libopenjpeg-dev
   libopus-dev libpulse-dev libschroedinger-dev libsoxr-dev libspeex-dev
   libtheora-dev libv4l-dev libvorbis-dev libvpx-dev libx264-dev
   libxvidcore-dev libopenal-dev libcdio-paranoia-dev

./configure --prefix=/usr --bindir=/usr/bin --datadir=/usr/share/ffmpeg
   --incdir=/usr/include/ffmpeg --libdir=/usr/lib/x86_64-linux-gnu/
   --mandir=/usr/share/man --arch=x86_64 --optflags='-O2 -g -pipe -Wall
   -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong
   --param=ssp-buffer-size=4 -grecord-gcc-switches -m64 -mtune=generic'
   --enable-bzlib --disable-crystalhd --enable-frei0r --enable-gnutls
   --enable-libass --enable-libcdio --enable-libdc1394 --disable-indev=jack
   --enable-libfreetype --enable-libgsm --enable-libmp3lame --enable-openal
   --enable-libopencv --enable-libopenjpeg --enable-libopus
   --enable-libpulse --enable-libschroedinger --enable-libsoxr
   --enable-libspeex --enable-libtheora --enable-libvorbis --enable-libv4l2
   --enable-libvpx --enable-libx264 --enable-libxvid --enable-x11grab
   --enable-avfilter --enable-avresample --enable-postproc --enable-pthreads
   --disable-static --enable-shared --enable-gpl --disable-debug
   --disable-stripping --shlibdir=/usr/lib/x86_64-linux-gnu/
   --enable-runtime-cpudetect

make
sudo make install
```

Now download and install VATIC.

```
wget http://mit.edu/vondrick/vatic/vatic-install.sh
chmod +x vatic-install.sh
./vatic-install.sh
```

After installing VATIC, edit the apache configuration to give permission to the folder containing VATIC. The file can be accessed by the following command

```
sudo gedit /etc/apache2/apache2.conf
```

Add the following to the permissions section in the file, to grant access to the folder ‘vatic’.

```
<Directory /path/to/vatic/>
  AllowOverride None
  Require all granted
</Directory>
```

Now follow the instructions given in the README file of VATIC to complete the installation.

B. Tips for using VATIC for sign annotations

Set the –blow-radius to 0 when loading a video, this essentially allows one to annotate each frame. In the default mode annotating one frame would erase all the annotation present within 5 frames of the latest annotation, this is a hazel when consecutive frames have to be annotated.

VATIC's interpolation algorithm is linear and was primarily designed for surveillance camera where the objects are almost always in the same distance relative to the camera. But here the change in the size and position of the signs is large when the signs are closer. To tackle this problem, it is recommended that annotation be done on the frame where the sign is just outside the bounding box of the previous annotation; this ensures that the interpolated signs fit in the bounding boxes correctly.

It is recommended that the video be decomposed into highest quality images, to clearly distinguish the signs that are far. This can be done by replacing the following commands:

```
$ turkic extract /path/to/video.mp4 /path/to/output/directory
```

with

```
$ ffmpeg -i /path/to/video.mp4 -qscale:v 1 /tmp/all/frames/%d.jpg
$ turkic formatframes /tmp/all/frames /path/to/output/directory
```

REFERENCES

[1] https://github.com/jetpacapp/DeepBeliefSDK/.

[2] https://www.youtube.com/user/roadwaywiz/.

[3] https://code.google.com/p/labelme-video/.

[4] https://github.com/CSAILVision/LabelMeToolbox/.

[5] https://github.com/cvondrick/vatic/.

[6] http://cs-people.bu.edu/sameki/blog2.pdf.

[7] http://jetson.co/wiki/main_page/development.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[10] Richard A Retting, Helen B Weinstein, and Mark G Solomon. Analysis of motor-vehicle crashes at stop signs in four us cities. *Journal of Safety Research*, 34(5):485–489, 2003.

[11] Pierre Sermanet and Yann LeCun. Traffic sign recognition with multi-scale convolutional networks. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2809–2813. IEEE, 2011.

[12] David Shinar. *Traffic safety and human behavior*, volume 5620. Elsevier, 2007.