# Investigating Deep Learning Methods to Improve Traffic Sign Recognition Under Challenging Conditions

Oguzhan Yilmaz
Georgia Institute of Technology
School of Electrical and Computer Engineering
oguzhan@gatech.edu

## Abstract

*In this work, I was able to reproduce a lot of the benchmark results of CURE-TSR with comparable accuracies. The particular challenge that needs work from the previous attempts at improvements on results is the codec-error, an edge case yet still important in ensuring correct recognition despite transmission errors in remote driving and cloud computation scenarios with autonomous vehicles. Focusing on the results of codec-error challenge, I propose two methods to obtain further improvement, used together with data augmentation: SIFT feature based matching and Autoencoder-CNN networks. I achieved accuracy improvements using the latter method. Although the experiment scope was on codec-error, the method is expected to be robust to other adversarial perturbations.*
**Note: I changed my project topic from the proposal. I was going to work on Bag of Tricks paper but switched to this.*

### 1. Introduction

Self-driving cars and all kinds of autonomous vehicles are entering our lives at a rapid speed. The functionality and safety of such systems plays an important role in ensuring that lives are not lost using this new technology. Therefore, the inference and recognition tasks performed on these machines need to be robust and accurate. There are not many already existing traffic datasets. CURE-TSR[1] addresses this with the most comprehensive publicly-available traffic sign recognition dataset to date. The work at top self-driving companies doesn't yield visibility into this kind of exploration even though they perform countless hours of on-road driving. Still, collecting enough information to train these systems physically is a costly and overwhelming process. CURE-TSR is a dataset of ~1.7 million 28x28 images of traffic signs, with types including speed limit, goods vehicles, no overtaking, no stopping, no parking, stop, bicycle, hump, no left, no right, priority to, no entry, yield, and parking. Induced challenge types are rain, snow, haze, shadow, darkness, brightness, blurriness, dirtiness, colorlessness, sensor and codec errors. They show up in dataset in 5 different degradation levels, with level 5 yielding the most visually corrupt.

As shown by the work of others [11][1], the challenging conditions cause degradation in performance of sign recognition algorithms. Ideally, we would like to find a robust method and architecture that can eliminate such adversarial effects. Although CURE-TSR dataset is a great contribution, the analyzed techniques struggle a lot with increasing challenge levels.

This work aims to implement and reproduce the said benchmark algorithms: RGB-Softmax, RGB-SVM, Intensity-Softmax, Intensity-SVM, CNN, HoG-Softmax and HoG-SVM (opted instead for SIFT). The codec-error is selected to be experimented with in this work, since it proved to be the most challenging type stated in the paper[1]. I also investigate ways to improve upon the reproduced benchmark results using SIFTNet and Autoencoders.

Full source code is available at
https://github.com/oguzhan-yilmaz1/CURE-TSR-ECE6258-Project

### 2. Related Work

There is a lack of metadata corresponding to exact challenge types and levels. CURE-TSR provides a benchmark dataset and additional insight into what models work with this recognition task on the dataset and what do not. The paper "CURE-TSR: Challenging Unreal and Real Environments for Traffic Sign Recognition" forms the basis of this work presented. It is concluded in t paper that the flipping as an augmentation technique does not improve the test accuracies due to the fact that shape nature of the traffic signs can lead to similarities between different class samples when they are flipped, which ultimately increases the confusion of the model. Still, augmentation can be achieved by including unreal synthetic images in the training set. They use diverse data augmentation methods and show that utilization of limited simulator data along with real-world data can enhance the recognition performance. This is significant in that simulator data can be used for enhancing performance even in the face of challenging conditions. The paper includes 2 experiments. First experiment establishes the baseline, which I reproduced to most degree. The second experiment show that augmentation using unreal challenge images helps with

accuracy for most of the challenges. However, codec error still struggles for example.

### 3. Approach

Starting with the available codebase, I tried to understand the training flow and different models implemented. There are only two models already given in the codebase and nothing else. Those are the basic CNN and the RGB with a softmax, implemented though the selection of loss function as cross entropy loss. I implemented everything else myself, which include the SVM loss and intensity model and the proposed SIFT based neural net model. I used PyTorch on a Google Cloud VM with a NVIDIA K80 GPU. I modified the code from repository, which was written only with being able to run the CNN model on mind, heavily to incorporate training of different models. I extended the user line commands and changed the training code. I was able to reproduce all the accuracies for all models except SIFT, which I couldn't get working. Another thing I struggled with is the gpu data parallelization not properly handling model state dictionary when you have multiple models running. The workaround was to call load_state_dict with a strict=False option.

Codec Error

Instead of trying to implement a brand new pytorch HoG-SVM, I opted to use SIFTNet feature extraction and running a fully-connected neural net that learns parameter to map the SIFT histogram features to the traffic sign classes. This especially made sense because the codec-error as seen maintains most of the image gradient features except for the corrupt parts. The provided baseline results in paper on HoG based methods show that it is the best performing one among all the baseline methods. The question I asked was 'What if the SIFTNet can be used as a preprocessing step for the downstream classification task'. As will be seen in the results, the downstream model didn't benefit from the SIFT features in my implementation, as I saw a steady accuracy across multiple epochs. The model reached an optimum point and got stuck.

A better alternative would have been to create a bag of SIFT features and using k-means clustering to match the testing image features against. Unfortunately, this was not implemented.

I visualized the HoG features to see if my hypothesis was right. As Figure 1 shows, within the same traffic sign class, the increasing challenge level mostly maintains the HoG features same with intensities changing slightly. However, a different traffic sign class has a significantly different HoG representation. This is because the edges and patterns that are intensified in HoG domain is very characteristic to the sign type. HoG can mitigate the degradation effects for classification purposes. This insight can lead to a possibility that we can explore HoG based methods further, and my SIFT implementation could actually be improved to obtain even better results.

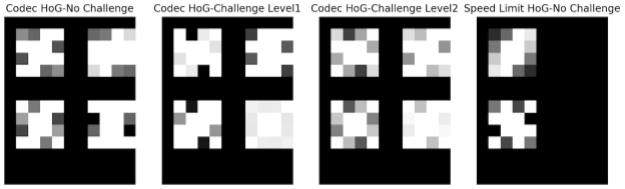


Figure 1. Comparison of different HoG representations of traffic sign images.

The failure from the SIFT could be mitigated with a different selection of the optimizer instead of stochastic gradient descent. Adam could be a more rigorous option.

The approach I took to solving the accuracy problems differs from the paper slightly in that I found the idea of introducing challenge images in the training phase to be somewhat 'cheating'. If the goal is to classify the challenge images accurately, in a real scenario, we should be testing the robustness of networks trained on the data we actually have against such conditions. The naturally collected data and the unreal synthetic data in training is not going to include a, for instance, codec-error; this is introduced in data transfer. Therefore, I decided to use only the challenge free real and unreal data for the next experiments I ran.

After the baseline implementations, secondly, in order to observe if a lower level representation of the images would help with the accuracy, I implemented a model with convolutional Autoencoders, using convolutional, maxpool and ReLU layers. The core idea is to reduce the 28x28x3 to a lower dimensional encoded representation where following the same steps in reverse during the decoding stage would yield a MSE-loss-minimized reconstruction of the original image. The trained autoencoder would then be used as a preprocessing step for baseline CNN. The main assumption here is challenged images can have a similar representation in a lower dimension (output of encoding stage) and using the same decoding trained with unchallenged images, we can reconstruct the challenge image to one that is 'closer' to its original unchallenged version.

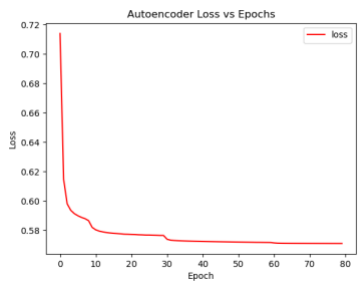


Figure 2. MSE-loss of the reconstructed image batches against the original image batches during autoencoder training.

| Method | Level 0 | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 |
|---|---|---|---|---|---|---|
| CNN | 82.2% | 49.85% | 49.0% | 37.0% | 33.2% | 30.3% |
| RGB-Softmax | 84.2% | 44.6% | 43.2% | 33.6% | 28.8% | 26.0% |
| RGB-SVM | 85.2% | 44.0% | 42.1% | 33.0% | 28.6% | 25.9% |
| Intensity-Softmax | 72.8% | 39.4% | 40.4% | 29.8% | 26.5% | 23.7% |
| Intensity-SVM | 72.0% | 36.7% | 38.8% | 28.0% | 24.8% | 23.0% |
| SIFT-Softmax | 32.3 (error)% | NA% | NA% | NA% | NA% | NA% |
| SIFT-SVM | 35.8 (error) | NA | NA% | NA% | NA% | NA% |

Table 1: Codec Error Results

Third, I used the same autoencoder as dimension reduction method. The idea is to only keep the weights of encoder and use them to help tune the CNN. The complete CNN trains on the dataset and finetunes its own layers. The main idea is to learn features at a reduced dimension and directly use those features in inference. The resulting CNN will be used in testing.

## 4. Experiments

### 4.1 Baseline Method Implementations

I implemented the aforementioned baseline methods. The accuracy results are in Table 1. The challenge that I had was that there were no baseline numbers as figure, so all I have is the plot from the experiment 1 section of the CURE-TSR paper. The numbers I obtained are very close to the numbers in the figure. Unfortunately, SIFT training got stuck with the given accuracies. Earlier, I provided the discussion on the potential reasons for it. Looking at the below Figure 2, CNN is the best generalizing algorithm over the challenge free validation set. This fact makes the table results more explainable. Notice that CNN is a clear winner as challenge level increases.

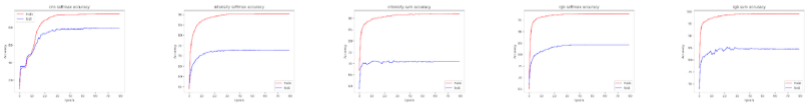

Figure 2. Accuracy over epochs

### 4.2 AutoEncoder as Preprocessing Method

As explained, Autoencoder was trained only with challenge free data. Then it was used on images from all 6 challenge levels in testing set, hoping that the recovery of images using encoder would provide a clearer representation for the downstream CNN to perform classification. Remember that also CNN was trained only on real challenge free data and is identical to the baseline discussed above. The percent changes from the baseline results are in Table 2.

| **Level** | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| **+ %** | -10.7 | -5.35 | -5.4 | -1.6 | -0.537 | -0.246 |

Table 2. Marginal changes of accuracies on testing with Codec-Error images at level 0-5 compared to the baseline numbers in Table 1.

Looking at these results, it is exactly what I expected. The challenge free results will of course be less because we are using the same CNN on not the original but the reconstructed images, which will have artifacts not seen before in the original testing. Since the CNN is not touched to account for the new reconstructed data, we expect the accuracy to be lower. That was a sacrifice I was willing to make in order to amp up the challenge images' accuracies and see how they will behave. The trend confirms my earlier hypothesis that the challenge images are more 'similar' to challenge free images when reduced to a lower dimensional representation using a method like encoding as done here. Notice that as the challenge level increased, the marginal change became less negative.

### 4.3 AutoEncoder Fine-tuned CNN

The previous results gave us a hope that training with real and unreal images can give a marginal benefit as challenge level increases. Therefore I finetuned the parameters of the original CNN using the Autoencoder-reconstructed images from both real and unreal challenge free images Table 3 gives the incremental accuracies over the benchmark results presented in Table for Codec-Error class of challenge. This is my main contribution and improvement over the original paper. See appendix for a clarification on the two methods I introduce here.

| **Level** | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| **+ %** | +0.92 | +3.7 | +1.84 | +2.71 | +2.88 | +3.26 |

Table 3. Marginal changes of accuracies on testing with Codec-Error images at level 0-5 compared to the baseline numbers in Table 1 using Autoencoder-finetuned CNN.

The important thing to highlight here is that at no point in the training, I used challenge images. This is critical because we would ideally want our recognition algorithms to be robust even in the face of unexpected image corruptions like Codec-Error. The changes are slight in number but I believe shows a path in the right direction.

References


[1] D. Temel, G. Kwon, M. Prabhushankar, & G. AlRegib (2017). CURE-TSR: Challenging unreal and real environments for traffic sign recognition. In Neural Information Processing Systems (NeurIPS) Workshop on Machine Learning for Intelligent Transportation Systems.

[2] D. Temel, M. Chen, & G. AlRegib (2019). Traffic Sign Detection Under Challenging Conditions: A Deeper Look into Performance Variations and Spectral Characteristics*IEEE Transactions on Intelligent Transportation Systems*, 1-11.

[3] D. Temel, & G. AlRegib (2018). Traffic Signs in the Wild: Highlights from the IEEE Video and Image Processing Cup 2017 Student Competition [SP Competitions]*IEEE Sig. Proc. Mag.*, *35*(2), 154-161.

[4] J. Lu, H. Sibai, E. Fabry, and D. Forsyth. No need to worry about adversarial examples in object detection in autonomous vehicles. In arXiv:1707.03501, 2017.

## A. Appendix

### A.1 Model Details

The following are the architectures of each model.

**1.** RGB-Softmax

Image(28x28x3) → flatten → fc(14) → CrossEntropyLoss

**2.** Intensity-Softmax

Image(28x28x3) → mean on channels → flatten → fc(14) → CrossEntropyLoss

**3.** RGB-SVM

Image(28x28x3) → flatten → fc(14) → multiclass SVM loss

**4.** Intensity-SVM

Image(28x28x3) → mean on channels → flatten → fc(14) → svm_loss

**5.** Autoencoder as Preprocessing Step

• Encoder:

16 conv kernels of 3x3, padding=1 (ReLU)

Maxpooling with stride = 2

8 conv kernels of 3x3, padding=1 (ReLU)

Maxpooling with stride = 2

• Decoder:

16 transposed conv kernels of 2x2, stride=2 (ReLU)

1 transposed conv kernels of 2x2, stride=2 (Sigmoid)

• CNN:

Image(28x28x3) → Encoder → Decoder → Conv(3,6,5) → MaxPool(2, 2) → Conv(6, 16, 5) → fc(16 * 4 * 4, 120) → fc(120, 84) → fc(84, 14)

**6.** AutoEncoder Finetuned CNN

Same Architecture as 5, but trained on real and unreal challenge free dataset.

While the paper refers to SVM to be using a radial basis function, I implemented a multiclass, margin minimizing svm loss function and still achieved comparable results.

### A.2 Dataset

The details about the dataset can be found at https://github.com/olivesgatech/CURE-TSR and the CURE-TSR paper.

### A.3 Code

https://github.com/oguzhan-yilmaz1/CURE-TSR-ECE6258-Project

Dependencies:

- Python3 or higher
- CUDA, CuDNN
- PyTorch (www.pytorch.org)
- Skimage, numpy
- Optionally, tensorflow-cpu for tensorboard

Training and testing file along with README is provided. The repository contains pre-trained models under checkpoints to reproduce the baseline results for CNN, RGB-SVM, RGB-Softmax, Intensity-SVM, Intensity-Softmax, and the faulty (explained) SIFT-Softmax. Many of the modular layers used are from PyTorch. Majority of the SIFTNet implementation is mine and follows Szeliski's publication https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf .

The models are trained with 80 epochs. The range of training time is between 140s-150s. The logs provide that information for every run. The model was trained and tested on Google Cloud VM's n1-standard-8 cpu with an NVIDIA K80 GPU. The model accuracy metric is the testing accuracy on each of the challenge levels for all the challenge types. It is expected that the accuracy will decrease with increasing challenge level. No hyperparameter tuning was performed. The learning rate is adjusted by division with 10 every 30 epochs. I used batch size of 256. The default training parameters are not changed. This was to produce a comparable baseline that would allow me to see the marginal improvement.