Traffic-Sign Detection and Recognition Using Deep Learning

Zexin Chen
Georgia Institute of Technology
Atlanta, GA 30332
zchen849@gatech.edu

Saran Gopavaram
Georgia Institute of Technology
Atlanta, GA 30332
sgopavaram6@gatech.edu

Devarsh Pandya
Georgia Institute of Technology
Atlanta, GA 30332
dpandya30@gatech.edu

Yinzhu Quan
Georgia Institute of Technology
Atlanta, GA 30332
yquan9@gatech.edu

Abstract

As the autonomous vehicle space continues to grow, there is a great opportunity for improving upon current Deep Learning frameworks for detecting and recognizing traffic signs. Previous research has shown that increased data augmentation as well as varying the backbone of a chosen architecture can help improve the accuracy of detecting traffic signs. However, there does not exist a comprehensive study that compares the effects of both in the same experiment. In this paper, we propose four different experiments to determine the effectiveness of the aforementioned approaches. In the first experiment, we utilize Faster R-CNN with a ResNet50 backbone (this will be our baseline model) and use simple augmentation (random horizontal/vertical flips and resizing). In the second experiment, we utilize the baseline model with more rigorous augmentation (padding and appearance transformation in addition to random horizontal/vertical flips and resizing). In the third experiment, we utilize Faster R-CNN with a custom SqueezeNet1.1 backbone, which features a linear classifier) and use simple augmentation. Finally, in the fourth experiment, we utilize Faster R-CNN with SqueezeNet and the advanced augmentation. Interestingly and perhaps unexpectedly, the experiments show that while SqueezeNet is faster during training and typically leads to lower training loss, the Average Precision/Recall and F-1 scores are worse when compared to ResNet. Additionally, more rigorous augmentation leads to worse accuracy when compared to the simpler augmentation for both ResNet and SqueezeNet.

1. Introduction

The ability to detect and classify traffic signs through Deep Learning is paramount in the autonomous driving industry. Most commonly, different flavors of the CNN (Convolutional Neural Networks) algorithm have been used to achieve high accuracy in detecting traffic signs. The problem with current traffic sign detection is that tiny changes in pixels can lead to opposite results, and this instability is a large drawback that prevents it from being applied to real industrial products. The error in sensors can lead to interference with other components of the whole process, potentially threatening user safety and impairing brand reputation. Even a relatively low error rate could lead to disastrous consequences, especially if the general trend toward creating fully autonomous vehicles continues. We hope that through our investigations based on existing methods, we will be able to find feasible improvements or exclude wrong research directions in traffic sign detection.

In terms of inputs, we will utilize the German Traffic Sign Detection Benchmark dataset (GTSDB), from which we will utilize 600 images. The size of the images will always be between 16x16 and 128x128. Finally, the traffic sign photos have been taken with varied lighting and perspective. In terms of outputs, our model will try to accurately detect traffic signs utilizing the Faster R-CNN detector (one of the most common methods for detecting traffic signs) as a baseline, and compare it with a customized Faster R-CNN detector extended with several improvements. This customized detector will feature more rigorous data augmentation techniques (by combining resizing and randomly flipping resized images with padding and appearance distortions). The customized detector will also alter the baseline Faster R-CNN by switching the traditional Faster R-CNN convolutional layers with a custom

backbone.

We plan to address the problem of detecting and recognizing a large number of traffic-sign categories. The proposed Faster R-CNN system provides an efficient deep network for learning a large number of categories with an efficient and fast detection. We propose several adaptations to the model that improve the learning capability on the domain of traffic signs as well by using different versions of Faster R-CNN. Further, we propose a data augmentation technique based on the distribution of padding and appearance distortions combined with random flipping of resized images. We will also evaluate our proposed improvements and compare them against the baseline Faster R-CNN. We hope to achieve improved performance of the proposed models over the original Faster R-CNN in several metrics, including Average Recall, Average Precision, and F1-score.

2. Related Work

There is a ton of literature on the subject of Traffic Sign Detection, and a number of review articles are accessible [7]. Different techniques were utilized in Traffic Sign Detection. There are a great many typically hand-crafted characteristics usually utilized by researchers, including scale invariant feature transform (SIFT) [4], histogram of oriented gradients (HOG) [3], [2]. Additionally, a variety of machine learning techniques have been utilized, consisting of support vector machine (SVM) [12], random forests [2], and logistic regression [8].

Torres et al. [11] provides a unique approach to creating databases that only need templates for the traffic signs and arbitrary natural photos. It is demonstrated that the easily created training database works well for training a deep detector (such Faster R-CNN) on German traffic signs, averaging 95.66% of mAP (Average Precision). This paper can be considered as the "nearest neighbor paper" to our project because, in addition to the use of Faster R-CNN for traffic sign detection on the GTSDB dataset, the paper also used the same data augmentation method as the one in our baseline model (random vertical and horizontal flipping on the resized images). The primary difference is that our model is implemented in PyTorch (as opposed to Tensorflow) and does not utilize the special database generation approach.

Tabernik et al. [10] proposes and analyzes a novel approach to detecting and recognizing traffic signs using CNN. The paper utilizes the Mask R-CNN (an extension of Faster R-CNN) approach and finds that the proposed model appears to have better performance than Faster R-CNN, though there are some ways to improve both the baseline and proposed approaches (namely by reducing the number of missed detections through better data augmentation techniques). The paper utilizes two classes of data distortion methods: (i) geometric/shape distortions , and (ii) appearance distortions. To extend upon our baseline model, we plan to generate our own brand new data augmentation approach by combining data augmentation methods in this paper and [11] to reduce the aforementioned missed detections.

Du et al. [1] provides some strategies for improving and customizing existing Deep Learning models to improve object detection. The paper compares the performance of the vanilla ResNet-FPN backbone with that of RetinaNet and R-CNN detectors. The authors show that minor architectural changes to the same models can at times be even more beneficial than using a completely different architecture (as in the case of ResNet with just a few architectural changes beating EfficientNet). This paper introduces SqueezeNet and ResNet architectures as potential backbones for Faster R-CNN. Utilizing these architectures instead of the traditional Faster R-CNN convolutional layers in the proposed model could allow us to further improve upon the results from the baseline model.

3. Method/Approach

3.1. Faster R-CNN

Faster R-CNN will be used for the detection of the GTSDB dataset with resized images. Faster R-CNN is composed of two modules. The first module is the deep fully convolutional network, called Region Proposal Network (RPN), which takes an image as an input and outputs a set of rectangular object proposals. The second module is a region-based CNN, called Fast R-CNN, that classifies the proposed regions into a set of predefined categories. This module is highly efficient since it shares convolutions across individual proposals and performs bounding box regression to further refine the quality of the proposed regions. The entire system is a single unified network, in which both modules are merged by sharing their convolutional features.

Faster R-CNN is trained for the region proposal task and the classification task. This is performed with stochastic gradient descent. This network is initialized with the ImageNet pre-trained model before it is trained on a specific domain. This method enables fast detection and recognition in the test phase. For each input image, the trained model outputs a set of objects bounding boxes, where each box is associated with a category label and a SoftMax score.

The baseline model will utilize Faster R-CNN with simple data augmentation (random flips of resized images). To improve upon the baseline model, we propose an additional data augmentation. The nature of the traffic-sign domain allows us to construct many new samples using artificial distortions of existing traffic-sign instances. Additional synthetic traffic-sign instances are created by modifying segmented, real-world training samples. The traffic signs in the proposed dataset will be annotated with tight bounding

boxes, allowing them to be segmented from the training images. Two classes of distortions will be performed (in addition to the random flips of resized images as done in the baseline): (i) padding, and (ii) appearance distortions (variations in brightness and contrast).

Before applying padding and appearance distortions, we will first normalize each traffic-sign instance. For the appearance change, we will estimate the distribution of averaged intensity values. We will additionally estimate the distribution of scales using the size of rectified instances.

When generating synthetic distortions, we will sample random values from the corresponding distributions. In the appearance distortion, the distributions are not generic for all classes, instead, we will use different distributions for each class. We will use class-specific mean instead of mean of overall categories, but we will still apply common variance calculated from all the categories. With the whole augmentation process, we will generate enough new instances.

Furthermore, we will extend the architecture of Faster R-CNN by changing the convolution layers of different choices. This means that we can use any of the pre-trained classification models from Torchvision and then extract their feature layers along with the pre-trained weights (and use them as backbones with the first module of Faster R-CNN). We will use SqueezeNet architecture as a backbone. Further, we will apply data augmentation techniques to the modified Faster R-CNN models.

3.2. Reason for Improvement

Although Faster R-CNN did achieve higher accuracy in the deeper neural networks by short-cutting certain layers, it still took a long time to run in our project. So, in order to further improve our models, after having a thorough literature review, we chose to use SqueezeNet1.1 as the backbone of our optimized model since SqueezeNet is a more compact replacement to Faster R-CNN. SqueezeNet1.1 decreases the number of input channels to 3 × 3 filters and down samples late in the network so that convolution layers have large activation maps. Thus, SqueezeNet makes the deployment process easier due to its small size. Moreover, we also customize the SqueezeNet by changing the classifier from sequential classifier with dropout, conv2d, relu, and AdaptiveAvgPool2d activations to linear. By optimizing the original model with SqueezeNet1.1, the running time of our new model improved (which is the main reason why we chose it since image datasets are always huge and computationally-heavy for training).

3.3. Alternatives

We also considered alternative approaches, such as the capsule network. Since CNN has a drawback with its pooling layer, it makes CNN easily lose information of input images. Also, CNN requires massive amounts of data to learn. We thought of using capsule networks to mitigate the negatives of CNN. Capsule nets are networks that are able to fetch spatial information and more important features so as to overcome the loss of information that is seen in pooling operations. However, the result we got from capsule nets was not ideal in terms of optimization. We did a literature review about capsule nets and tried to figure out the reason why it didn't perform as we expected. Unfortunately, debugging was difficult due to limited information available for implementing capsule networks.

3.4. Loss Function

Faster R-CNN architecture contains two task layers, each layer defines a loss function that is a combination of losses from the classification operation and the bounding box regression. The loss function was used in combination with stochastic gradient descent (SGD) optimizer.

For the RPN layer, the loss function is as mentioned in figure 1. It combines the classification loss and the bounding box regression loss. In the region proposal layer, the regions are classified into only two classes: the background, or the object. Here, the variable p_i^* is the confidence target. It equals 1 when it is an object. Otherwise, it equals zero.

For the Fast R-CNN layer, the equation for the loss function is very similar, with two exceptions. First, the samples do not come from the anchors. Whether the sample is positive or not is based on the value of the IoU between the proposed bounding boxes and the ground truth. The second difference is that the classification loss for the Fast R-CNN layer is defined as the multi-class loss function. The variable p_i^c is the predicted probability for the proposed region belonging to class c, that is, the ground truth class.

Finally, the total loss for the Faster R-CNN is a linear combination of loss for the Fast R-CNN layer and the Region Proposal Layer. $\lambda_{FasterRCNN}$ and $\lambda_{RegionProposal}$ are the weights for these two layers, which are set to 1.

3.5. Evaluation

Our Faster R-CNN models will utilize COCO's Detection Evaluation metrics. Within the COCO Detection Evaluation, Average Recall (AR), Average Precision (AP), and F1-score will all be used to evaluate and compare our baseline and proposed model. Average Recall, in mathematical terms, is the average of: (number of true positives) divided by (number of true positives + number of false negatives). This measure, in our example, tests how many relevant traffic signs a model is able to find (in a specified class within the dataset). Average Precision, in mathematical terms, is the average of: (number of true positives) divided by (number of true positives + number of false positives). This measure, in our example, tests how well a model is able to cor-

Loss Function for the Region Proposal Layer

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}}$$

$$\mathcal{L}(\{p_i\},\{t_i\}) = \frac{1}{N_{\text{cls}}}\sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{box}}}\sum_i p_i^* \cdot L_1^{\text{smooth}}(t_i - t_i^*)$$

$$L_{cls}\left(p_i, p_i^*\right) = -p_i^* \log p_i - \left(1 - p_i^*\right)\log\left(1 - p_i\right)$$

Loss Function for the Fast R-CNN Layer

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}}$$

$$\mathcal{L}(\{p_i\},\{t_i\}) = \frac{1}{N_{\text{cls}}}\sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{box}}}\sum_i p_i^* \cdot L_1^{\text{smooth}}(t_i - t_i^*)$$

$$L_{cls}\left(p_i, p_i^*\right) = -\log\left(p_i^c\right)$$

Total Loss for Faster R-CNN

$$Loss_{FasterRCNN} = \left(\lambda_{FastRCNN} * Loss_{FastRCNN}\right) + \left(\lambda_{\text{Re gion Pr oposal}} * Loss_{\text{Re gion Pr oposal}}\right)$$

Figure 1. Loss Function

rectly classify a traffic sign as "relevant." Finally,

$$F1\_score = \frac{precision * recall}{precision + recall}$$

This measure combines the precision and recall metrics to tell us which classifier (between the baseline and one of the proposed models) has better performance [9].

4. Data

4.1. Dataset

The dataset we choose to use is the German Traffic Sign Detection Benchmark (GTSDB) dataset[1], which consists of a sizable collection of actual photos and a methodical evaluation process, both of which are backed by a public online interface. The images for our dataset have been selected from sequences recorded near Bochum, Germany, on several tours in spring and autumn 2010 [5]. Edge-adaptive, constant-hue demosaicking approach was used to transfer all of the images in the dataset to RGB color space, and they were all saved in raw PPM file type. The size of the traffic sign varies from 16 to 128 pixels in relation to the longer edge. The dataset consists of 600 complete pictures with 846 traffic signs. However, there are only 506 images annotations (labeled), so we narrowed it down to 506 images. Among 506 whole images, a training dataset (including 406 images) and a test dataset (including 100 images) were created by randomly dividing the whole dataset. From Figure 2, it is obvious that the distribution of GTSDB dataset is not balanced. Some classes have many samples, while others do not.

[1] https://sid.erda.dk/public/archives/ff17dc924eba88d5d01a807357d6614c/published-archive.html

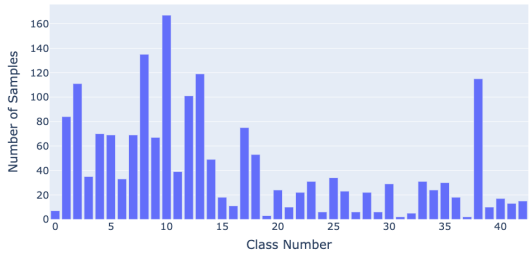

Figure 2. Data distribution of GTSDB.

The performance metrics include F1-score, Average Recall (AR), and Average Precision (AP). AP and AR are averaged over multiple Intersection over Union (IoU) values [6], which is defined as the approximate area under the precision/recall curve obtained for a fixed IoU threshold (0.7 in [11]). The AP, AR, and F1-score for the proposed method in the Torres paper were 90.76%, 91.99%, and 0.9135 [11]. The paper separated the test into lower bound, upper bound, and proposed template-based: the upper-bound baseline makes use of the entire training set, and the lower-bound baseline has one training image of each traffic-sign category [11]. Their metrics indicate that AP is better in upper-bound than lower-bound as well as their proposed experiment, which is valid since testing on more data means more robust performance. However, there was a large gap between lower-bound and the proposed method (9.13% lower) [11], which we believe is due to the limited data for lower-bound. Their metrics seem to imply that one image is not enough for the neural network to learn. Thus, we will improve upon this in our testing stage (by having more than one image). Also, the close values in AP of the proposed method and higher-bound cannot validate the efficiency of their method since the baseline's training and test sets have many shared images.

4.2. Data Preprocessing

Since in the real-world, the traffic situation is so dynamic, the extracted image from the sensor is not always complete or well-placed as in the original datasets. In data preprocessing, we import padding and appearance distortion to convert the images in order to simulate the reality as much as we can. The baseline method only has random flipping and resizing to larger scale, which we believe is not enough for testing the performance of detection. As a result, we added padding with distortion_scale= 0.5 for the distortion to get more randomized data. For appearance distortion, we have modified the original datasets with brightness, hue, contrast, and saturation in ColorJitter to fool the model. During trial and error, we figured out the balanced hyper parameter for our expected dataset, which achieve both blurred and noise images. The training dataset has 406

Figure 3. Data augmentation method 1: random vertical and horizontal flipping on the resized images.

Figure 4. Data augmentation method 2: combining random vertical and horizontal flipping on the resized images and appearance distortions.

images. Thus, our original training dataset has 406 images. After three basic data augmentation, vertical flipping, horizontal flipping, and resize, the training dataset has 406 * 3 + 406 = 1624 images. After advanced data augmentation, ColorJitter and padding, the size of training dataset added 406 * 2 = 812 images. Finally, the training dataset is 1624 + 812 = 2436 images in advanced augmentation.

5. Experiments and Results

5.1. Experiments

In the data augmentation part, we generate a brand new transforming method by combining two kinds of data augmentation techniques: (i) random vertical and horizontal flipping on the resized images, (ii) appearance distortions and padding. Figure 3 shows the result of flipping and resizing. With respect to the first data augmentation, we randomly flip images vertically and horizontally with the probability of 0.5 among four images and resize the input image to 1500×882 pixels. Figure 4 indicates the result of the combination of the two data augmentation methods, random vertical and horizontal flipping on the resized images, appearance distortions, and padding. In the appearance transformations part, there are four steps we have implemented: (1) jitter brightness to 0.5 for making images darker, (2) jitter contrast to 0.5 for letting pictures have low contrast, (3) jitter saturation to 0.5, and (4) jitter hue to 0.3. Finally, after augmentation, we compare two different types of Faster R-CNN. The first uses ResNet50 as the backbone. The second uses modified SqueezeNet1.1 as the backbone. In total, we run 4 different experiments to test the effectiveness of different augmentation techniques as well as the effectiveness of varying the backbone of Faster R-CNN.

	Precision	Recall	F1-score	loss
	ResNet			
basic	0.3819	0.4006	0.3910	0.0769
basic+color+pad	0.2697	0.3703	0.3121	0.1712
	SqueezeNet			
basic	0.2710	0.2905	0.2804	0.0481
basic+color+pad	0.1590	0.2373	0.1904	0.0936

Table 1. Evaluation for two models

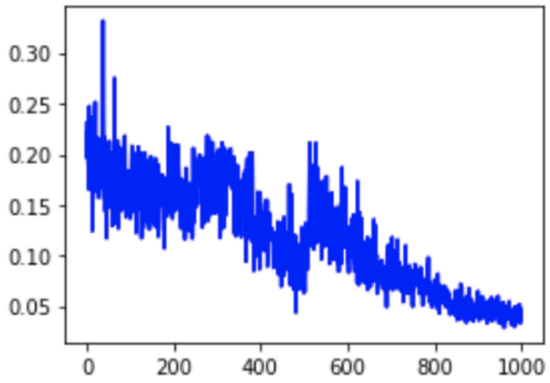

Figure 5. Loss for ResNet based on basic data augmentation.

5.2. Results

We chose 775 epochs to train because that was the maximum that Colab Pro Plus would allow without automatically quitting for some of the models. We investigated four models for traffic sign detection as part of this project by combining different augmentation and pre-trained models. The two models are the Faster RCNN model with ResNet50 FPN as the backbone and the Faster RCNN model with a SqueezeNet1.1 backbone (custom modification to have a linear classifier).

From Figure 5 and Figure 6 , we can see that the loss of SqueezeNet is smaller than the loss of ResNet, which means the SqueezeNet performs better than ResNet in the training dataset. Note that these figures show values up to 1000 epochs because we were able to run 1000 epochs for two of the experiments (and get the associated graphs). However, we report Precision, Recall, and F1-score values for 775 epochs in all experiments.

From Figure 7 and Figure 8 (Average Precision), Figure 9 and Figure 10 (Average Recall), we can also see that the modified augmentation didn't improve results for either the ResNet or the SqueezeNet models. Better loss (which is a training metric) but worse AR and AP (evaluation metrics) means that we have overfitting. In the future, we could make a few changes to improve our results. For example, a much larger data set for training can reduce overfitting.

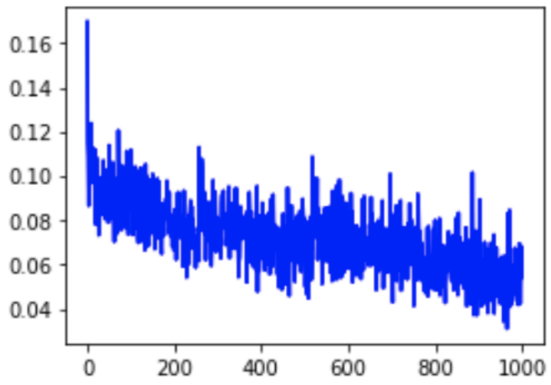

Figure 6. Loss for SqueezeNet based on basic data augmentation.

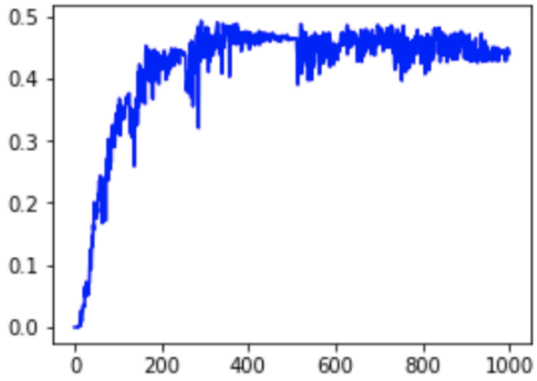

Figure 7. Average Precision for ResNet based on basic data augmentation.

Also, it is possible that padding and flipping are not very good representations of images. Instead of color change and padding in augmentation, we could explore perspective changes (changing the 3D orientation of image). We could also try running for 1,000 epochs and then compare the results. Also, we can see from the tables as well as the figures that SqueezeNet is less accurate than ResNet. SqueezeNet is good at reducing model size (which is why it was so much faster), but that can sometimes come at the expense of accuracy, as we saw in our experiments. Perhaps further modifications of SqueezeNet or even using something like RetinaNet would be better in future experiments. RetinaNet is built on ResNet with the aim of being more accurate (instead of faster) so it may perform better.

6. Conclusion

For this project, we explored four models for traffic sign detection by varying different augmentation types and different backbones for Faster R-CNN. From the metrics, we found that the loss decreased with our new Squeezenet1.1 model; however, Average Precision and Average Recall

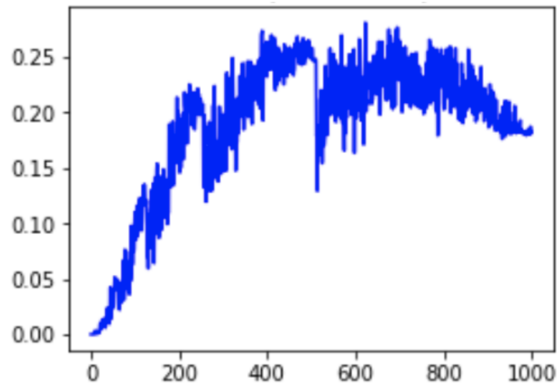

Figure 8. Average Precision for SqueezeNet based on basic data augmentation.

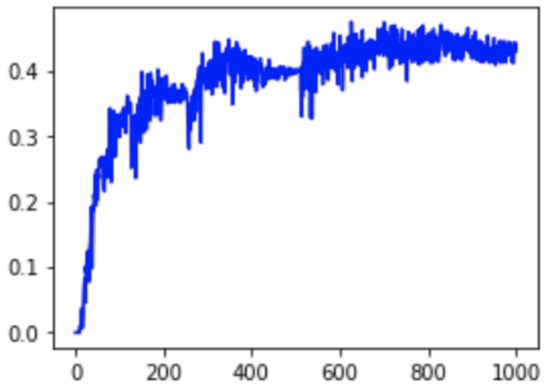

Figure 9. Average Recall for ResNet based on basic data augmentation.

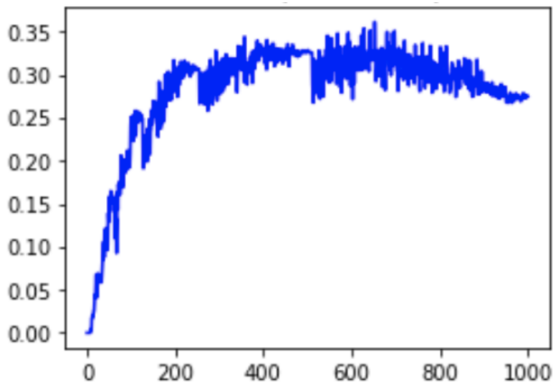

Figure 10. Average Recall for SqueezeNet based on basic data augmentation.

aren't optimized as expected. In hindsight, there are several reasons why we got these types of results. First, there is overfitting in our training stage because of the unbalanced dataset (as we saw in Figure 2). Second, our data

augmentation methods may not be the best for this traffic sign detection task. We used padding in data augmentation, but it is possible that padding is not representative of traffic signs in the real-world based on these results. Finally, we were only able to run 775 epochs instead of 1000 epochs due to the limitation of Google Colab Pro Plus. In the future, it may be prudent to ensure that the data is fully representative of real-world traffic signs by ensuring the dataset is large and balanced before performing any relevant data augmentations. Additionally, we were able to learn from our experiments that SqueezeNet is very good at reducing model size (as seen by the low loss values and the fact that it had fewer parameters during training compared to ResNet). Future studies could look into modifying the backbone of SqueezeNet in other ways, especially since its speed makes it a very attractive choice for Faster R-CNN.

References

[1] Xianzhi Du, Barret Zoph, Wei-Chih Hung, and Tsung-Yi Lin. Simple training strategies and model scaling for object detection. *arXiv preprint arXiv:2107.00057*, 2021. 2

[2] Ayoub Ellahyani, ME Ansari, IE Jaafari, and Said Charfi. Traffic sign detection and recognition using features combination and random forests. *International Journal of Advanced Computer Science and Applications*, 7(1):686–693, 2016. 2

[3] Jack Greenhalgh and Majid Mirmehdi. Real-time detection and recognition of road traffic signs. *IEEE transactions on intelligent transportation systems*, 13(4):1498–1506, 2012. 2

[4] Mrinal Haloi. A novel plsa based traffic signs classification system. *arXiv preprint arXiv:1503.06643*, 2015. 2

[5] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2013. 4

[6] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 4

[7] Andreas Mogelmose, Mohan Manubhai Trivedi, and Thomas B. Moeslund. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1484–1497, 2012. 2

[8] Deli Pei, Fuchun Sun, and Huaping Liu. Supervised low-rank matrix recovery for traffic sign recognition in image sequences. *IEEE Signal Processing Letters*, 20(3):241–244, 2013. 2

[9] David MW Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *arXiv preprint arXiv:2010.16061*, 2020. 4

[10] Domen Tabernik and Danijel Skočaj. Deep learning for large-scale traffic-sign detection and recognition. *IEEE transactions on intelligent transportation systems*, 21(4):1427–1440, 2019. 2

[11] Lucas Tabelini Torres, Thiago M. Paixao, Rodrigo F. Berriel, Alberto F. De Souza, Claudine Badue, Nicu Sebe, and Thiago Oliveira-Santos. Effortless deep training for traffic sign detection using templates and arbitrary natural images. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2019. 2, 4

[12] Fatin Zaklouta and Bogdan Stanciulescu. Real-time traffic sign recognition in three stages. *Robotics and autonomous systems*, 62(1):16–24, 2014. 2

	Contribution part
Zexin Chen	Coding, Research
Devarsh Pandya	Coding, Research
Saran Gopavaram	Coding, Research
Yinzhu Quan	Coding, Research

Table 2. Group member contributions

Everyone contributed equally for Programming and Research. For programming, we have Yinzhu and Zexin for data augmentation, and Saran and Devarsh for optimization in architecture. For research and writing, Saran and Zexin did the method/approach part and had related research on loss functions and Capsule Networks. Yinzhu and Devarsh focused on experiment design, results, related works, and conclusion. All members helped with writing the Introduction.