Traffic Sign Classification

This is my writeup for the 2nd project. **TL:DR—see the red & bolded parts**.

Dataset Exploration

The three datasets for **training, validation and testing contain 34799, 4410 and 12630 images** respectively of traffic signs from German roads , with each image labeled into **43 classes**. Each image is **32x32 pixels in size in 8-bit color**.

I naively assumed 43 sign types should cover all the possible types, but then realized that is only a small subset of all possible sign types. Figure 1. shows the 43 signs included in the GTSRB in the order of the labeling in this dataset, along with sample images. Each of the **data-points is a 4d tensor** in this order: (example#, x-pixel, y-pixel, color-channel); thus the range of the **Test Set is (12630, 32, 32, 3).**

Figure 1. The 43 sign-types used in this dataset, with real-world examples on the right.

I have displayed 9 samples in the Notebook randomly chosen from the training dataset, along with their provided labels.

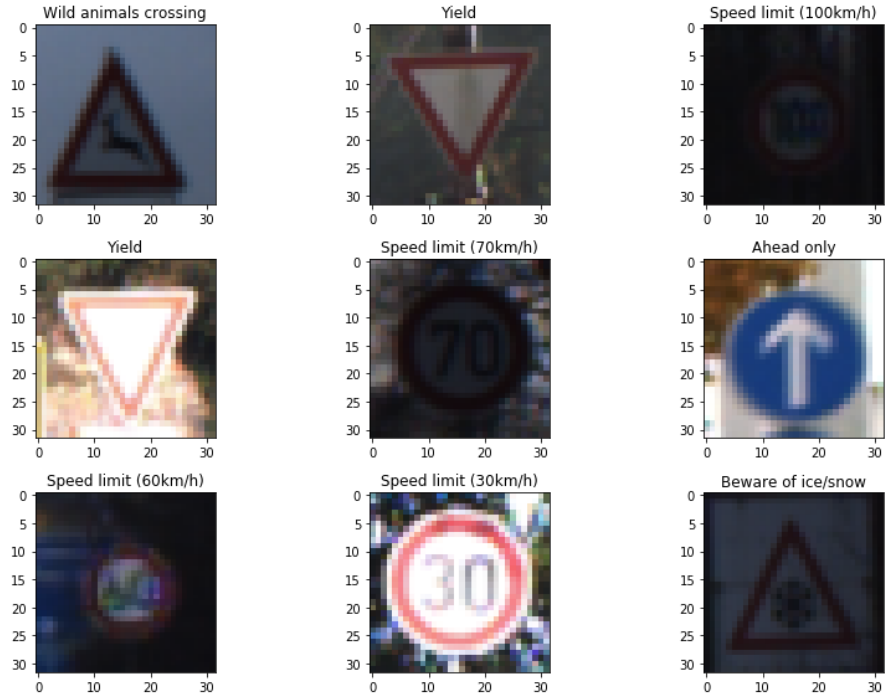

Figure 2. Nine randomly chosen signs in validation set with their respective labels.

The **distribution of sign types in the datasets is non-uniform** and perhaps reflects the real-world frequency of the signs. Thus for example, the 20km/hr speed-limit sign shows up in 0.7% of the examples in the training set, while 5.7% of set are 50km/hr speed-limit signs. Note, that if the signs were evenly distributed in all 43 classes, each class would have about 2.3% of the examples.

Examining Figure 3. carefully suggests that the Test set variations mimic that of the training set quite closely, while the variations in the validation set are just slightly less so (more apparent at a higher resolution scale for validation frequency).

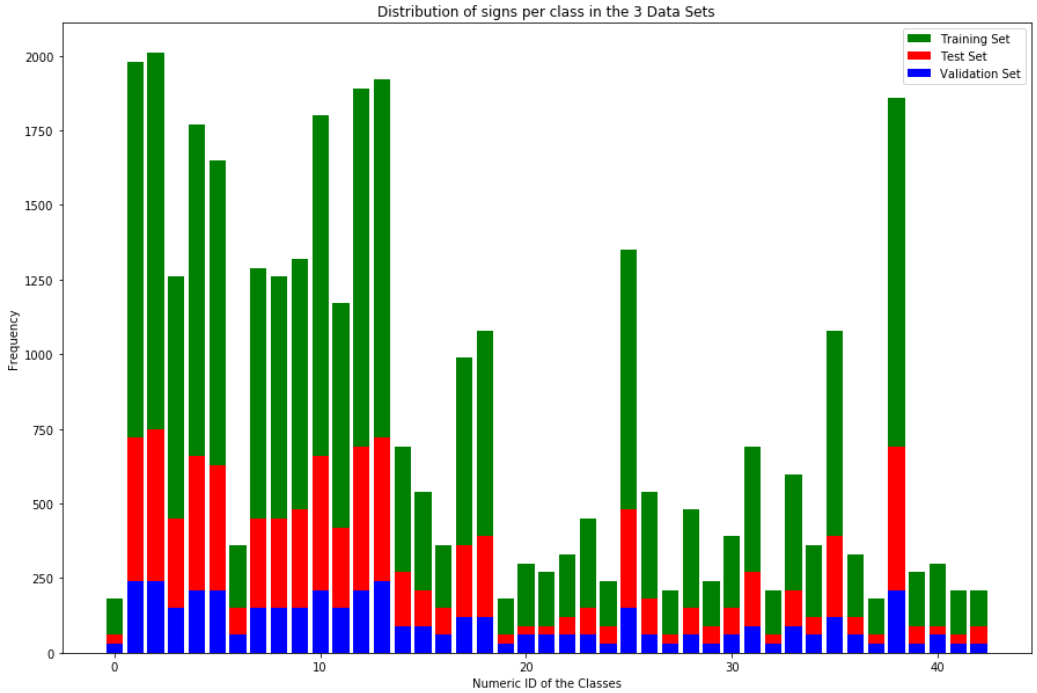

Figure 3. Frequency distribution of sign-types in the 3 data-sets. Approximately 8 signs account for 40% of all examples.

Data Pre-processing

Image conversion to Grayscale and Normalization were used to help improve classification accuracy. All things being equal, **Normalization increased validation accuracy from 83% to 93%** after 10 epochs in a non-optimized network. Conversion to **Grayscale** added only a few percentage points improvement after 10 epochs, however **added ~6 percentage points (68% to 74%) on epoch 1**; thus grayscale conversion led to faster convergence (all other things being equal).

Visually, the grayscale conversion ***appears*** to make a huge difference, especially on examples where lighting was poor.

I **did not augment the data**, as I was able to hit the benchmarks with tweaks to the neural network, **but I have ideas**, which I will discuss in the "Classifying New Data" section.

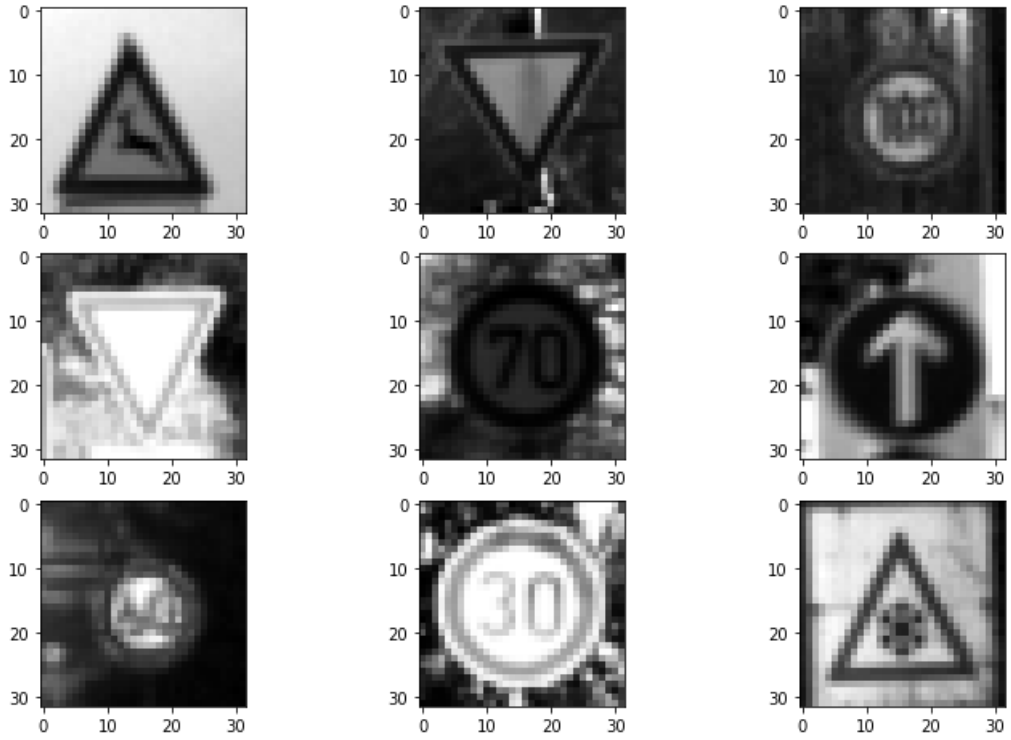

Figure 4. The 9 randomly chosen signs in validation set from Figure 2. after pre-processing.

Neural Network Architecture

I used the **standard LeNet** architecture described in the lectures, with the 32x32x1 images fed into two back-to-back Convolution layers, each with ReLU activation and 2x2 maxpooling. This was followed by three back-to-back fully-connected layers, outputting the 43 logits.

The cross-entropy of the resulting logits (after softmax) and the one-hot encoded data labels was calculated and minimized using AdamOptimizer. Prediction accuracy was evaluated by comparing the logits to the on-hot encoded data labels.

I had considered exploring the architecture in this paper, where the first convolution layer is also connected to the first fully connected layer (along with the 2nd convolution layer. I also considered using ELU activation instead of ReLU to speed up convergence; however, running on a GTX1070 GPU was not particularly time consuming, so I stuck with ReLU. I chose to increase the depth in the two convolution layers first. The paper described using a depth of 108 in the Conv Layer 1; to limit learning time, I experimented with a **depth size up to 24 in layer 1, and up to 32 in layer 2**. These modifications increased Validation Accuracy well above the target of 93%, and so I did not experiment with a modified architecture.

The table below summarizes the slightly modified LeNet architecture used for my training.

Layer	Description
Input	32x32x1 grayscale image; normalized
Convolution1	5x5 filter, depth: 24 layers; 1x1 stride, valid padding

Activation	ReLU
MaxPooling	2x2; Output = 14x14x24
Convolution2	5x5 filter, depth: 32 layers; 1x1 stride, valid padding
Activation	ReLU
MaxPooling	2x2; Output = 5x5x32
Flatten	Output = 800x1
Fully Connected1	Output = 120x1
Activation	ReLU
Dropout	keep_prob = 0.7
Fully Connected2	Output = 84x1
Activation	ReLU
Dropout	keep_prob = 0.7
Fully Connected3	Output : logits (43x1)

Network Optimization

I evaluated numerous modifications within the network including hyper-parameters, learning rate, dropout and depth of convolution layers, and below I have listed some of the key ones. I used the Validation accuracy as the performance metric. Note that in each case, all other factors are left unchanged (i.e. accuracy within each table below may be compared, but not between the tables).

Validation Accuracy vs. Batch size

	Batch size = 128	**Batch size = 64**
Epoch 1	0.51	0.85
Epoch 10	0.74	0.90

Validation Accuracy vs. Learning rate

	α = 0.01	**α = 0.0001**	**α = 0.0005**	**α = 0.001**	**α = 0.002**
Epoch 1	0.048	0.26	0.67	0.74	0.71
Epoch 10	0.054	0.80	0.87	0.89	0.88

Validation Accuracy vs. Convolution Lyr1 Depth

	Depth = 6	**Depth = 12**	**Depth = 24**
Epoch 1	0.669	0.693	0.731
Epoch 10	0.913	0.913	0.921
Epoch 20	0.924	0.932	0.942

Validation Accuracy vs. Conv. Lyr2 Depth

	Depth = 16	**Depth = 32**
Epoch 1	0.731	0.793
Epoch 10	0.921	0.940
Epoch 20	0.942	0.945

Before introducing dropout I confirmed that model was in an over-fit mode: training accuracy near 99%, but validation accuracy not improving much over subsequent epochs.

Validation Accuracy vs. FC1,2 Dropout

	Keep_prob = 0.5	**Keep_prob = 0.7**
Epoch 1	0.596	0.793
Epoch 10	0.900	0.940
Epoch 20	0.915	0.945

Note that Accuracy dropped as Dropout increased, which is what I would have expected. However, since dropout should improve "resilience" of the network, I went with a compromised keep_prob value of 0.7. Perhaps if the test set was significantly different, a higher dropout rate may improvement accuracy. In either case, I have not pursued that line of experimentation yet.

Accuracy vs. number of epochs

	Epochs = 10	**Epochs = 20**	**Epochs = 40**	**Epochs = 50**
Validation Accuracy	0.960	0.968	0.978	0.975
Test Accuracy	0.946			0.956

Network Performance

Following the optimizations in the prior section, I chose a batch size of 64, learning rate () of 0.00125, dropout of 0.3 (keep_prob=0.7), convolution layer1 depth of 24, convolution layer2 depth of 32, and number of epochs to train for = 10, based upon loss and accuracy vs. epochs. Allowing the network run for more epochs does indeed further improve performance, but after **10 epochs training accuracy of 99.7% and validation accuracy of 0.96** was already above the target of 0.93.

Testing the model on the test set resulted in a **Test Accuracy of 0.949.**

Classifying new Images

From an end-users perspective, this is probably the most important part of this entire project: how well does the code classify the images (in the earlier part of the lectures where there were discussions of getting test accuracy in the 90% range, I would have liked to see cases of running the model on samples).

While the network test accuracy of 94.9% was still below the ~98% of human accuracy, I chose images of signs that I thought would be difficult for the network to classify. Some signs were distorted (taken off-axis), others were excessively cropped, one sign was either vandalized or made inactive, and I also included one sign that was not in the 43 classes the neural net trained on.

Figure 5. Images of 9 signs found on the web. The last sign in second row does not exist in the training data. The last picture in the bottom row is an older style sign.

The turn left or turn right sign (middle row, 3rd column in Figure 5.) does not exist in the 43 sign classes the network was trained on. However, I had hoped to see that that the prediction confidence for this sign would be low, or the network would classify it as something that looked similar. For the purpose of the accuracy calculation, I labeled this image in class 36 (go straight or right). On the first iteration of classifying these images, **accuracy was an abysmal 33%,** or 3 out of 9 pictures were correctly classified.

I decided to run the network again for a greater number of epochs (50). Both, the validation and test accuracy rose by about 1 percentage point, but accuracy on the new images did not improve.

Re-training the network, and passing through these new images to the neural net again, resulted in an **accuracy of 55.6%,** or 2 more pictures were correctly classified. Either the random initialization of the biases is responsible for this improvement (or perhaps the checkpoint saved file used was incorrect). Note that the 2 additional correctly classified signs previously had the correct choice within the top-5 choices.

The HTML and Jupyter Notebook submitted have the 55.6% accuracy result.

Discussion of New Image classification

The chart below summarizes the results of the classification of the 9 new images. I would posit that the classification was qualitatively better than just looking at the accuracy of 55.6% suggests.

☑☺ 100% confidence in top choice	☑☺ 100% confidence in top choice	☑😐 97% confidence in top choice
☑☺ 100% confidence in top choice	☒☹ 100% confidence in top wrong choice; Correct class not in top 5	☒☺ Sign not in training set, but top choice is quite similar
☑☺ 100% confidence in top	☒☹ 99% confidence in top wrong choice	☒😐 Older style sign; 94% confidence & correct class not in top 5

Figure 6. Results of the classification of the 9 new images found on the web

Four of the 5 correctly classified signs had a prediction confidence of nearly 100%, which is great. One of these 5 correctly classified signs (top row, 3rd column) had a 96.6% confidence. Note that the lighting was poor, and perhaps more importantly, I may have cropped the image too aggressively.

Image #1: Correctly classified with 99.6% confidence; all top 4 choices are round signs.

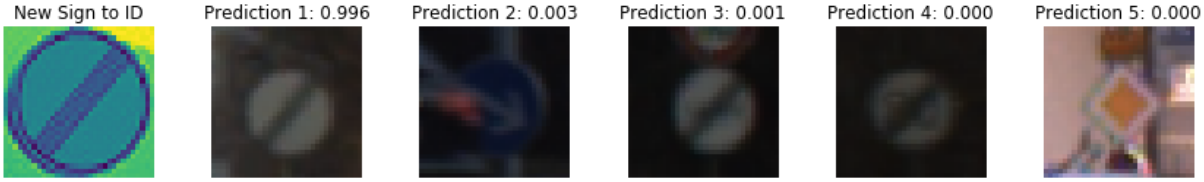

Image #2: Correctly classified with 100% confidence, despite image having a watermark.

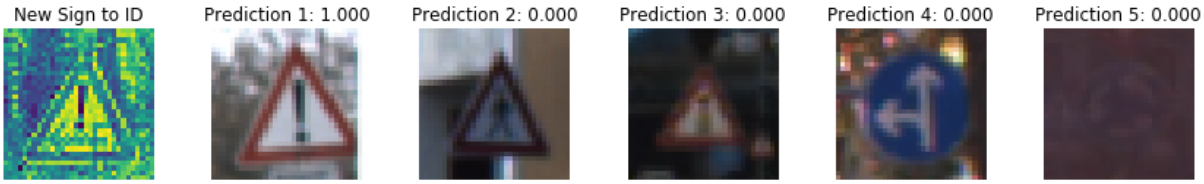

Image #3: Correctly classified with 96.6% confidence. Unlike most training images, the sign in the image was partially clipped, which may have contributed to the slightly lower confidence. Lighting was poor, but as discussed in the grayscaling operation, that may not have too adverse an impact. All of the top choices were triangular signs.

Image #4: Correctly classified with 99.9% confidence. I had chosen this image to be particularly challenging, as it was obscured, either due to vandalism or it was temporarily inactive. I would have expected a lower confidence. This one surprised me.

Image #7: Correctly classified with a 100% confidence. This one is a relatively easy and clear image, at least to my human eyes. The only challenging part was that it was partly obscured by another sign. Since dropout makes a neural net seek alternate representations for the classification of an image, I am curious if a network trained with zero dropout would have more trouble classifying this image.

Of the four incorrectly classified images, all are distorted and one does not exist in the training set. Also in hindsight, seeing that the traffic signs in the training images generally have plenty of margin around them, classification could have been better if signs were not so closely cropped

Image #6: Wrong classification, but 100% confidence in top choice. Even though this image was quite distorted (probably taken from below & close quarters), and the edges were clipped, I would have expected at least some hesitation in the wrong classification. Even more problematic is that the correct choice does not even show up in the top-5 choices.

Image #6: Wrong classification, with 99.9% confidence of one of the closest options. With this particular sign not present in the 43 trained classes, I certainly was trying to trip up the neural net. There were 5 somewhat similar classes (see Figure 1.), and the neural net chose one of those as its top choice. An ideal network, in my estimation, would have selected all of those 5 most similar signs (visually) as its top-5 choices, each with a confidence of ~20%. In this particular instance, it is possible that leaving color data in could have helped in better classification.

Image #8: Wrong classification, with 99.2% confidence. This was another tricky image—it was a temporary cloth sign placed on the road surface, and the sign had some creases. Four of the 5 top choices were triangular signs, just like the input image.

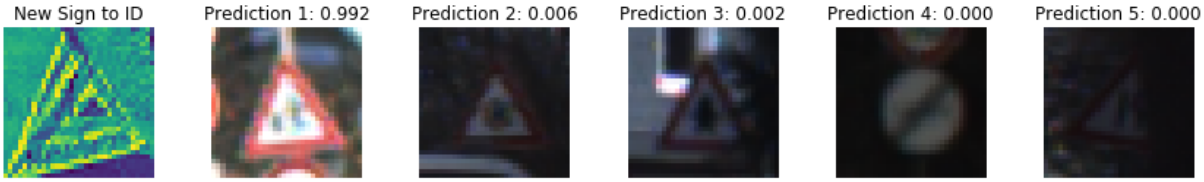

Image #9: Wrong classification, with 93.7% confidence. This was an older style sign, also taken at an angle, which introduces distortion. However, I am a little surprised that the correct choice was not within the top5 choices.

Comparison of results on New Images and original Test set

With an accuracy of 56% on the limited statistic of 9 images, vs. Test set accuracy of 94.9%, the trained network significantly underperformed on the images I fed into the classifier. While I could argue that 2 of the incorrectly classified images were not really fair game (image #6 & #8), that still results in an accuracy of only 71% (even with limited statistics: 7 samples vs. 12600). Clearly this suggests that the trained model does not generalize well enough, and suffers from a mild case of overfitting.

Concluding comments on New Image classification

I had admittedly given somewhat difficult signs for the system to decode, and the neural network didn't do an absolutely horrible job. Given that convolution networks are quite robust in the face of translational variations and rotational variations, the network needs to be made more robust to common distortions and situations where part of the sign is clipped.

Therefore, if I were to **augment the training dataset**, I would introduce **keyhole distortions** on all four sides and randomly crop the traffic sign images. I believe such modifications to training set will make the system more robust. Given that with only slight changes to the neural network architecture used in this

project, other researchers have achieved 99%+ classification accuracy, when the neural net is exposed to “more real-world” type images it can become more fool-proof.

I am also intrigued by the possibility of entirely automated generation of training data sets, at least in the case of traffic sign recognition. While I have certainly heeded the caution to not draw too many parallels between human neural recognition and ANNs, clearly we do not learn to recognize signs by looking at 2000 different images of the same traffic sign. So, given a particular diagram or rendition of a traffic sign, could we create a computer program that can **automatically generate thousands of variations, distortions, and other degradations of the raw sign image to feed into a neural network** for training?

Network State Visualization

While I have **completed this part of the project**, thanks to considerable help of mentors on the Udacity Discussion Forums, I do not fully understand what is happening.

For my particular neural net, I had 24 layers in the output of my 1st convolution layer, and thus I have 24 featuremaps. Based upon my reading, I understand that in DNNs for image recognition, the early layers “look at” or are activated by simple features: edges, lines, shapes, etc, and later layers combine these simple features and are activated by object patterns, more elaborate features, etc. So I would have expected to see common shapes that would be activated by parts of the traffic signs. Perhaps different neurons activated by circles, or triangles, or 45 lines, etc. I may be talking about something different.

Bottomline: I **do not know what these featuremap visualizations mean** & this has opened up more questions than answers about my understanding so far. **I welcome your feedback or pointers**.

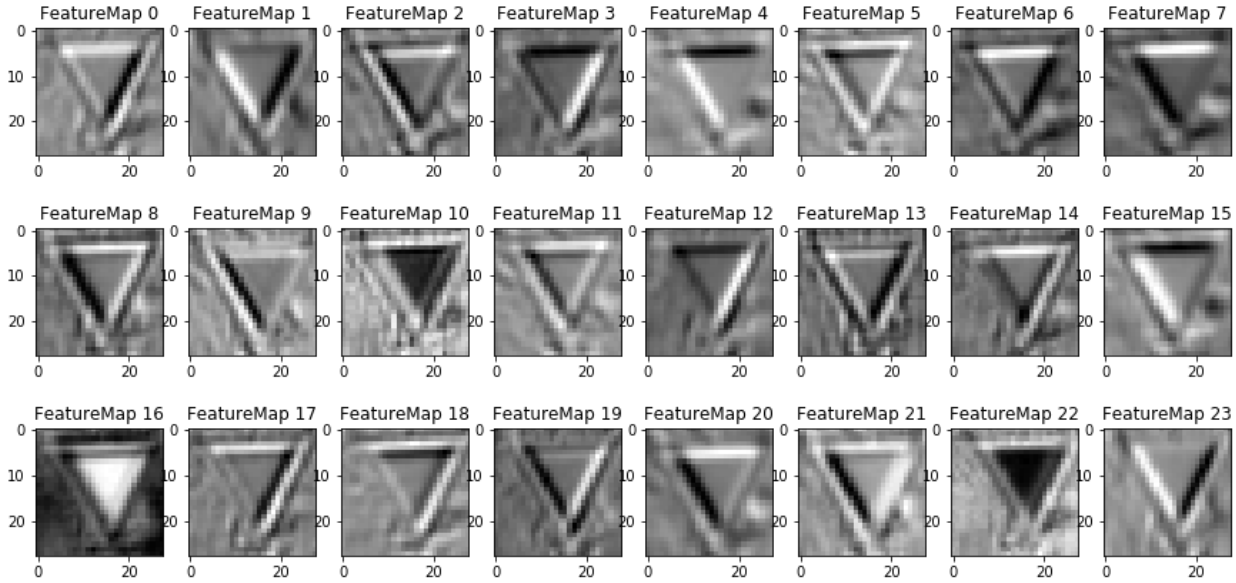

Figure 8. Featuremaps from first convolution layer when evaluating a yield sign from the training set. But what does this mean?