Build a Traffic Sign Recognition Project

The goals / steps of this project are the following:

- Load the data set (see below for links to the project data set)
- Explore, summarize and visualize the data set
- Design, train and test a model architecture
- Use the model to make predictions on new images
- Analyze the softmax probabilities of the new images
- Summarize the results with a written report

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. You can use this template as a guide for writing the report. The submission includes the project code.

You're reading it! and here is a link to my project code

Data Set Summary & Exploration

1. Provide a basic summary of the data set and identify where in your code the summary was done. In the code, the analysis should be done using python, numpy and/or pandas methods rather than hardcoding results manually.

The code for this step is contained in the **2nd code cell** of the IPython notebook.

I used the **Python's numpy library** to calculate summary statistics of the traffic signs data set:

- The size of training set is : **34799**
- The size of test set is : **12630**
- The shape of a traffic sign image is : **(32,32,3)**
- The number of unique classes/labels in the data set is : **43**

2. Include an exploratory visualization of the dataset and identify where the code is in your code file.

4th code cell contains the **bar graph for "No. of examples for each label"** for Training, Validation and Testing dataset.

Fig. Histogram of 'Training data' distribution for various Labels.

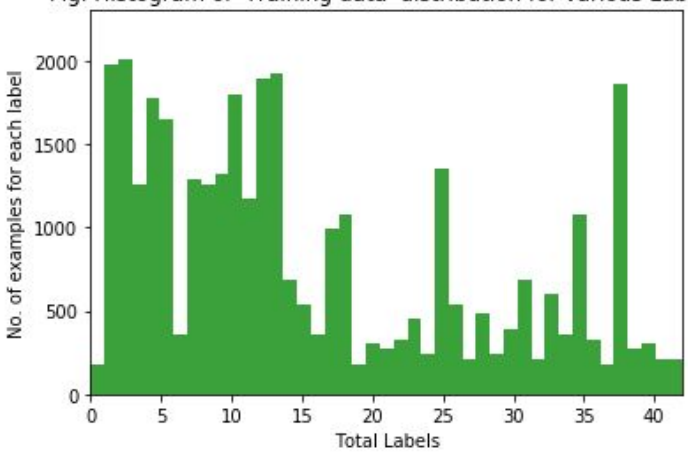


Fig. Histogram of 'Validation data' distribution for various Labels.

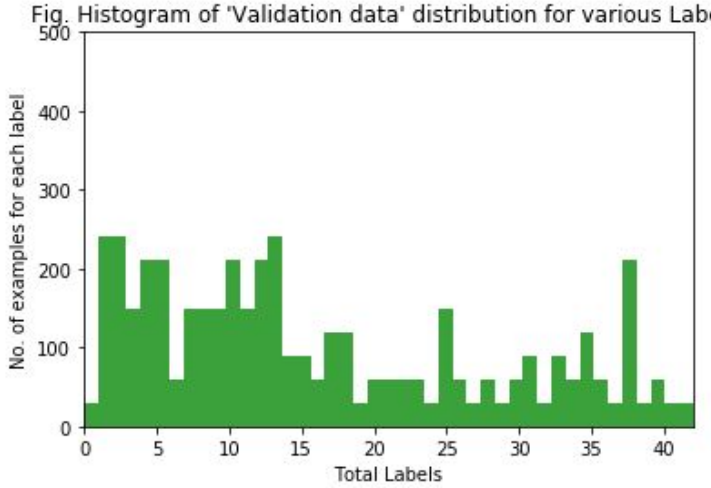
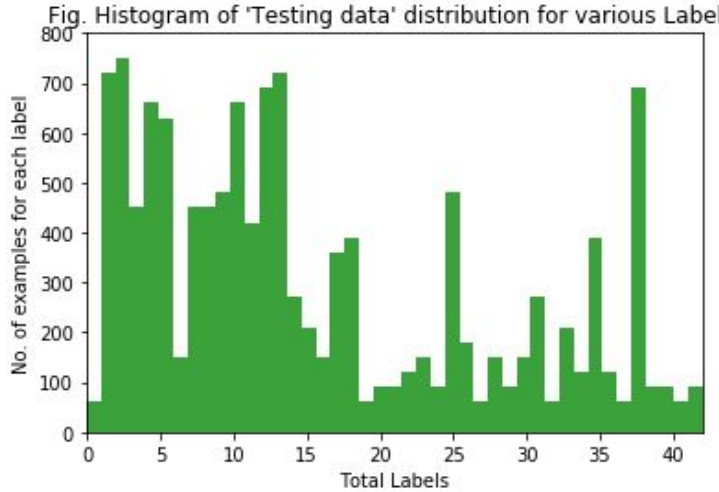
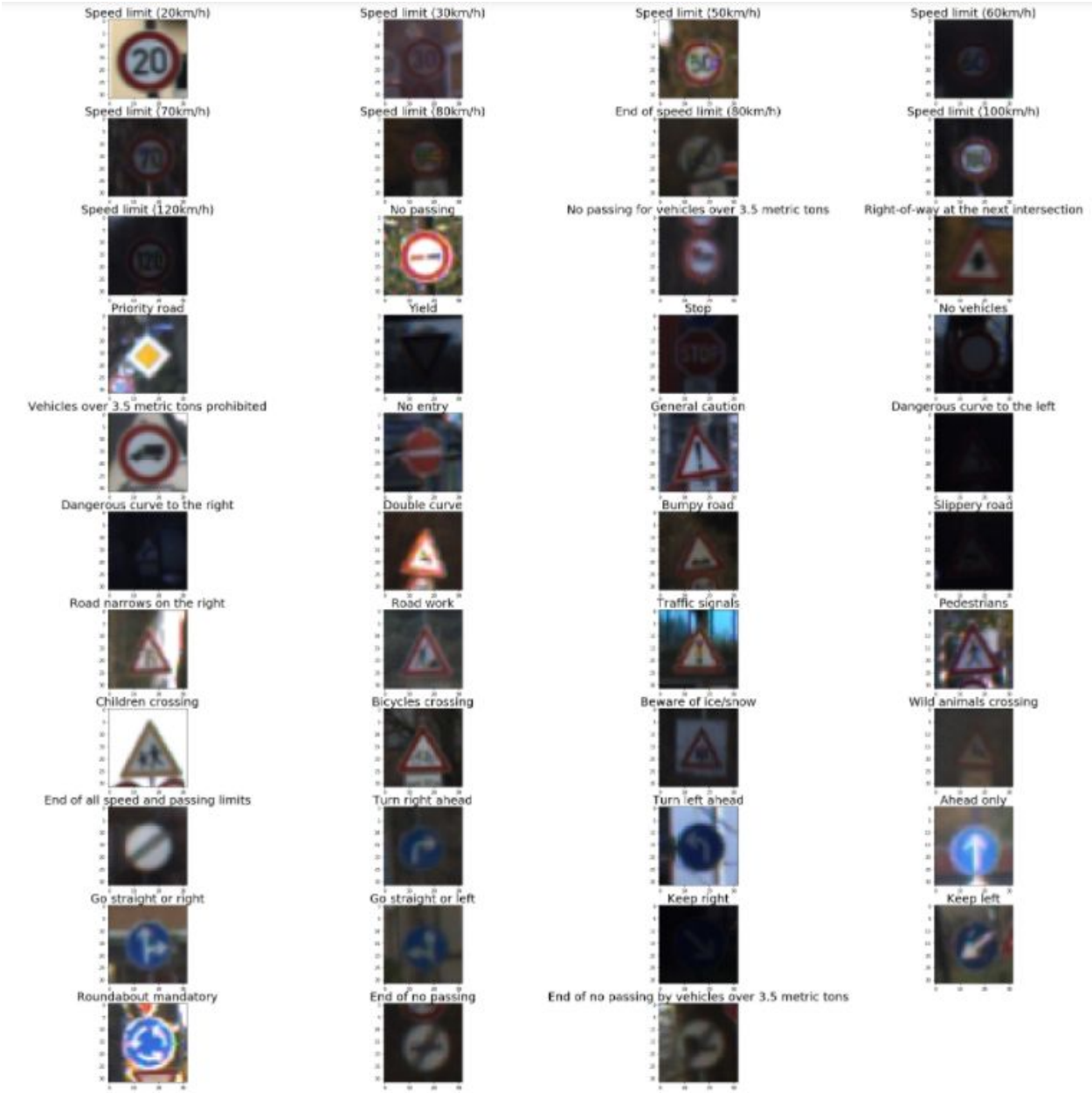


Fig. Histogram of 'Testing data' distribution for various Labels.



5th code cell contains the image **for each label/class** that we have in training data set (Total of 43 images).



Design and Test a Model Architecture

1. Describe how, and identify where in your code, you preprocessed the image data. What techniques were chosen and why did you choose these techniques? Consider

including images showing the output of each preprocessing technique. Pre-processing refers to techniques such as converting to grayscale, normalization, etc.

6th code cell of the IPython notebook contains the code for preprocessing which includes :

(1). Grayscaling the images, to convert from 3 channel RGB to single channel gray image.

(2). Normalization of image data between 0.1 and 0.9 instead of 0 to 255. (Bounding the value b/w small ranges to avoid too large and too small values).

2. Describe how, and identify where in your code, you set up training, validation and testing data. How much data was in each set? Explain what techniques were used to split the data into these sets. (OPTIONAL: As described in the "Stand Out Suggestions" part of the rubric, if you generated additional data for training, describe why you decided to generate additional data, how you generated the data, identify where in your code, and provide example images of the additional data)

The unzipped '**traffic-signs-data**' dataset folder already contained 3 pickle files.

```
(carnd-term1) Swapans-MacBook-Pro:traffic-signs-data Swapan$ ls
test.p  train.p  valid.p
```

So, for **training** used : **train.p**, for **validation** used : **valid.p** and for **testing** used : **test.p**

train.p has: **34799**

valid.p has: **4410**

test.p has: **12630**

The code for reading these picke file is in **1st code cell.**

3. Describe, and identify where in your code, what your final model architecture looks like including model type, layers, layer sizes, connectivity, etc.) Consider including a diagram and/or table describing the final model.

The code for my final model is located in the **7th, 8th and 9th code cells** of the ipython notebook.

My final model consisted of the **5 layers using the LeNet architecture**:

Layer	Description
Input	32x32x3 RGB image
Pre-processed and Normalized	32x32x1 grayscale
LAYER 1 Convolution 3x3	I/P: 32x32x1, Stride: 1x1 VALID padding, output: 28x28x6
RELU	Applied RELU
Average pooling	I/P: 28x28x6, Stride: 2x2 VALID, Output: 14x14x6
LAYER 2 Convolution 3x3	I/P: 14x14x6, Stride: 1x1 VALID padding, output: 10x10x16
RELU	Applied RELU
Average pooling	I/P: 10x10x16, Stride: 2x2 VALID, Output: 5x5x16
Flatten	I/P: 5x5x16. Output :400
LAYER 3 Fully connected	I/P: 400, Output : 120
RELU	Applied RELU

Dropout	Probability of 0.7
LAYER 4 Fully Connected	I/P: 120, Output : 84
RELU	Applied RELU
Dropout	Probability of 0.7
LAYER 5 Fully connected	I/P: 84, Output : 43
Softmax	Applied on logits with one hot encoding.

4. Describe how, and identify where in your code, you trained your model. The discussion can include the type of optimizer, the batch size, number of epochs and any hyperparameters such as learning rate.

The code for training the model is located in the **10th code cell** of the ipython notebook.

Optimizer: AdamOptimizer

Batch Size = 100

Epochs : 27

Hyperparameter - mu = 0, sigma = 0.1, learning rate: 0.001, dropout_rate = 0.7

The above values worked well for me.

5. Describe the approach taken for finding a solution. Include in the discussion the results on the training, validation and test sets and where in the code these were calculated. Your approach may have been an iterative process, in which case, outline the steps you took to get to the final solution and why you chose those steps. Perhaps your solution involved an already well known implementation or architecture. In this case, discuss why you think the architecture is suitable for the current problem.

Ans:

The code for calculating the accuracy of the model based on **Training** and **Validation** set is located in the **10th code cell** of the Ipython notebook.

The code for **testing model on Test dataset** is located in **11th code cell**.

My final model results were:

- training set accuracy of : **1.000**
- validation set accuracy of : **0.949**
- test set accuracy of : **0.920**

A Well known architecture was chosen.

What architecture was chosen?

I choose the **LeNet** architecture, as discussed and **LeNet** discussion and solutions during lectures. And I used **Adam's Optimizer** which is an improved version over **Gradient Descent**.

By doing multiple training iterations by changing **'learning_rate'** and **'epochs'**, I felt that **'learning_rate'** of **0.001** and **iterations** around **24-25** were the points where the learning matured and didn't change. Thus, chose the **EPOCH** as **27**. This way, my **training dataset accuracy** was : **1.000** and **validation** dataset accuracy was : **0.9495**. This gave **testing dataset** accuracy as : **0.920**, which is fine i believe.

Given that my learning accuracy came to be : 100 % and testing dataset accuracy at 92 %, I believe that "***techniques like rotation, translation, zoom, flips, and/or color perturbation***" would help to increase the **validation** and **testing** dataset accuracy. I haven't tried these things as of now. :-(

- Why did you believe it would be relevant to the traffic sign application?

I chose **LeNet** architecture code solution as starting point with few parameter tuning because as per my prior art search, **LeNet** architecture has been found to be really good for Traffic Sign classification.

Paper reference : http://publications.lib.chalmers.se/records/fulltext/238914/238914.pdf

Snippet:

In order to bench-test the Conv.NET library, in the second part of this work different CNNs were trained for the task of classifying traffic signs, using the German Traffic Sign Recognition Benchmark data set. Specifically, the attention was focused on two network architectures, respectively denoted by LeNet and VGGNet. The best single-model classification accuracy (96.2%) on the official GTSRB test data was obtained by training a LeNet architecture with dropout and early stopping on RGB images.

- How does the final model's accuracy on the training, validation and test set provide evidence that the model is working well?

Ans: Training set accuracy for model is 100%, Validation one is : 94.9 % and testing dataset is : 92.0 % (Classified 92% of 12630 images in dataset correctly). This gave me a fair idea that classifier has been trained well and classifier is working.

Test a Model on New Images

1. Choose five German traffic signs found on the web and provide them in the report. For each image, discuss what quality or qualities might be difficult to classify.

Here are five German traffic signs that I chose from the web:

Image	Prediction
50km.jpg	Speed limit (20km/h)
Ahead only.jpg	Ahead only
Keep left.jpg	Keep left
Stop.jpg	Stop
Yield.jpg	Yield

50km.jpg:

This got wrongly classified as **“Speed limit (20km/h)(0)”.** It may be that more training example for **50km sign** boards will help, or a different/improved NN model can help extract the further features in y dimension to help here.
Difficulty can be (1). detecting and learning the features related to number, correctly. (2). Other reason can be the much distortion seen by image, as I took it from net and resize it to 32x32, to make it not recognizable for features by the model. (3). Less number of images present in data set for training, in order to extract features.

Ahead only.jpg :

This got correctly classified. The current trained model finds its comfortable to classify “Ahead Only” signs.
Not difficult because it is one connected shape.

Keep left.jpg:

This got correctly classified. The current trained model finds its comfortable to classify “Ahead Only” signs.
Not difficult because it is one connected and line shape.

Stop.jpg :

This got correctly classified. The current trained model finds its comfortable to classify “**Stop**” signs.
Not difficult because it is one connected shape.

Yield.jpg :

This got correctly classified. The current trained model finds its comfortable to classify “**Yield**” signs.
Not difficult because it is one connected shape.

2. Discuss the model's predictions on these new traffic signs and compare the results to predicting on the test set. Identify where in your code predictions were made. At a minimum, discuss what the predictions were, the accuracy on these new predictions, and compare the accuracy to the accuracy on the test set (OPTIONAL: Discuss the results in more detail as described in the "Stand Out Suggestions" part of the rubric).

Ans. The code for making **predictions using my model on new Test images** is located in the **15th code cell** of the Ipython notebook.

The results of the prediction for 5 images:

```
=========================================
No. of Test Images =  5
Correct Test Images predicted count =  4
Test Images Accuracy prediction :  80.0 %
=========================================
```

PREDICTED : Speed limit (60km/h) , ACTUAL : Speed limit (50km/h)

PREDICTED : Ahead only , ACTUAL : Ahead only

PREDICTED : Keep left , ACTUAL : Keep left

PREDICTED : Stop , ACTUAL : Stop

PREDICTED : Yield , ACTUAL : Yield

The model was able to correctly guess 4 of the 5 traffic signs, which gives an accuracy of 80%.

This is less compared to the accuracy on the test set of 92.2 %. *But then it all depends on what all range of images were there in the Testing set, as it is not exhaustive.*

3. Describe how certain the model is when predicting on each of the five new images by looking at the softmax probabilities for each prediction and identify where in your code softmax probabilities were outputted. Provide the top 5 softmax probabilities for each image along with the sign type of each probability. (OPTIONAL: as described in the "Stand Out Suggestions" part of the rubric, visualizations can also be provided such as bar charts)

The code for making predictions on my final model is located in the 11th cell of the Ipython notebook.

Image 1: '50km.jpg'

For the first image, the model is relatively sure that this is a **"Speed limit (60km/h)"** (probability of 0.991), but the image if for **Speed limit (50km/h)**, and **prediction is wrong**.

The top five soft max probabilities were:

Probability	Prediction
9.91876960e-01	Speed limit (60km/h)
7.94608612e-03	End of speed limit (80km/h)
1.10908477e-04	Ahead only
3.46321685e-05	End of all speed and passing limits
2.27282562e-05	End of no passing

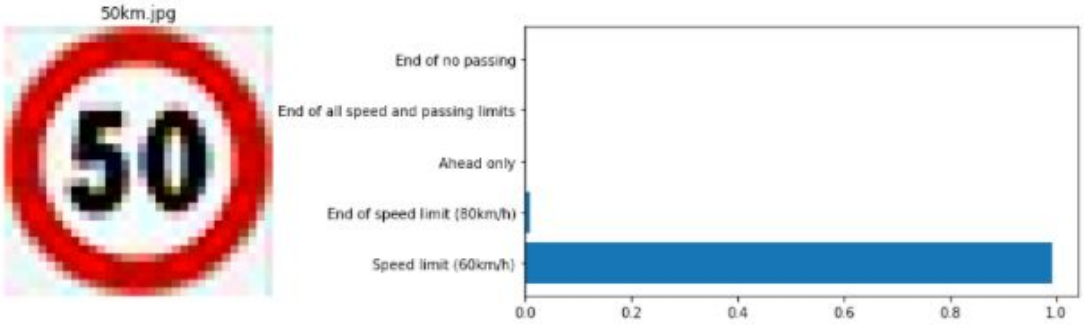


Image 2: Ahead only.jpg

For the 2nd image, the model is pretty sure that this is a **“Ahead only”** sign.(probability of 0.998), and this is **correct**.

The top five soft max probabilities were:

Probability	Prediction
9.98933733e-01	Ahead only
1.06459169e-03	Yield
1.28758177e-06	Go straight or right
3.72263486e-07	Speed limit (60km/h)
2.16206408e-09	Bicycles crossing

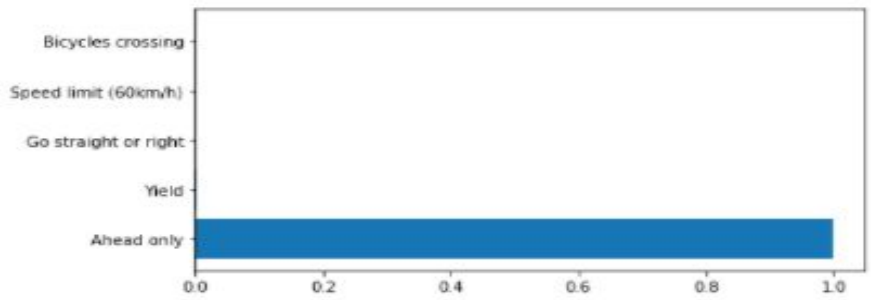


Image 3: Keep left.jpg

For the 3rd image, the model is relatively sure that this is a **“Keep left.jpg”** (probability of 0.999), and the prediction is **Correct**.

The top five soft max probabilities were:

Probability	Prediction
9.99962687e-01	Keep left

3.73129915e-05	Turn right ahead
1.65280324e-11	Double curve
7.70539101e-12	Yield
4.30874260e-12	Wild animals crossing

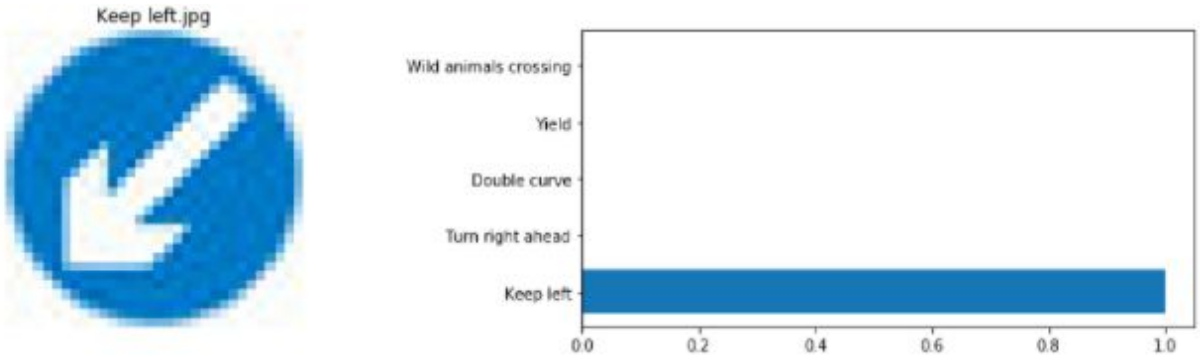


Image 4: Stop.jpg

For the first image, the model is relatively sure that this is a **“Stop.jpg”** (probability of 0.997), and the prediction is **Correct**.

The top five soft max probabilities were:

Probability	**Prediction**
9.97405350e-01	Stop
2.18952983e-03	Keep right
1.98700785e-04	Yield
5.50896693e-05	Turn left ahead
4.72589891e-05	Speed limit (30km/h)

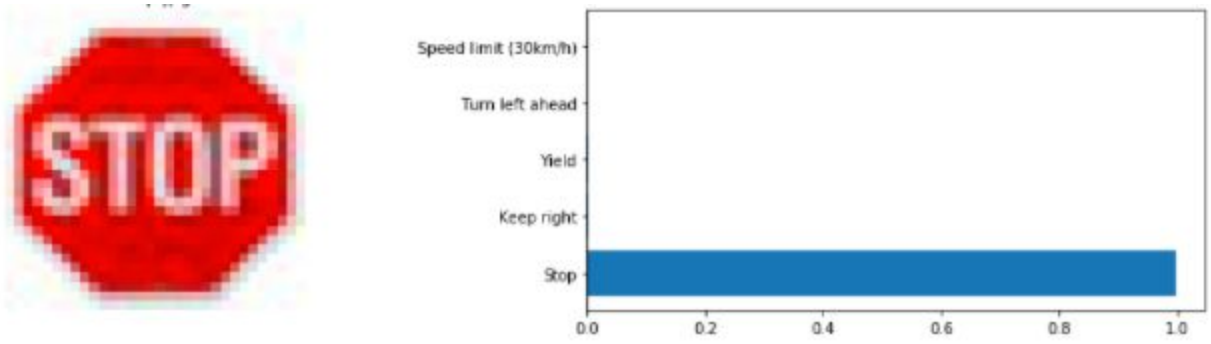


Image 5: Yield.jpg

For the first image, the model is relatively sure that this is a **“Yield.jpg”** (probability of ~ 1.000), and the **prediction is Correct**.

The top five soft max probabilities were:

Probability	Prediction
1.00000000e+00	Yield
1.71548836e-26	Ahead only
5.09981388e-37	Turn left ahead
2.36893922e-37	Keep right
0.00000000e+00	Speed limit (20km/h)