Traffic Sign Recognition

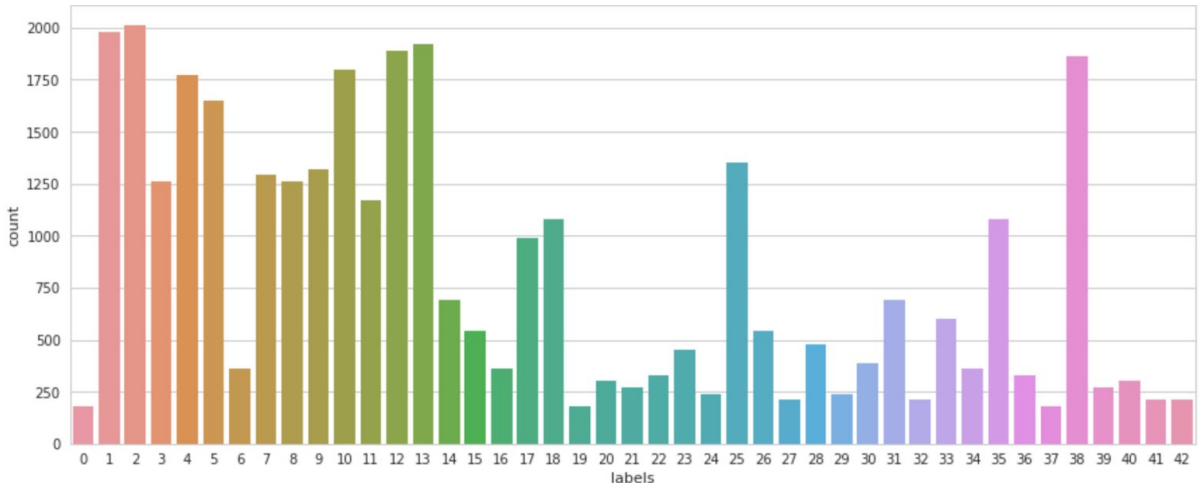
Introduction

In this project, we will build a traffic sign classifier that will be able to detect German traffic signs. This type of model can be integrated as a part of a self-driving car in other to allow it to recognize traffic signs while driving. The steps of this project are the following:

1. Load the data set.
2. Explore, summarize, and visualize the data set.
3. Design, train, and test the model architecture.
4. Make predictions.
5. Analyze the softmax probabilities of the new images.
6. Summarize the results with a written report.

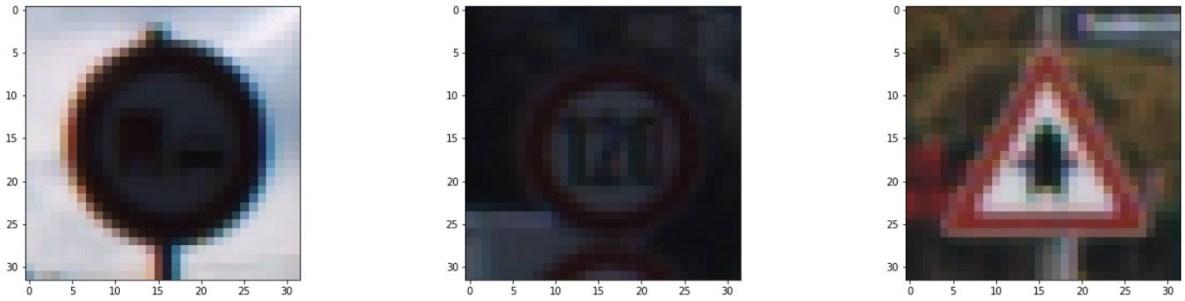
I. Load, explore, summarize, and visualize the data set

The code for this section is contained in the Jupiter notebook 'Traffic_Sign_Classifier.ipynb' located at the root of the project.
Our dataset consists of labeled german traffic signs. It consist of 51839 images of shape (32, 32, 3) with 43 classes. We have futher split of data into three subsets, that we will use for training(34799), testing(12630), validation(4410). (cell 6 of the notebook).
Below is the histogram of classes. (cell 10 of the notebook)



Histogram of classes

Below are some images. (cell 12 of the notebook).



Sample of images

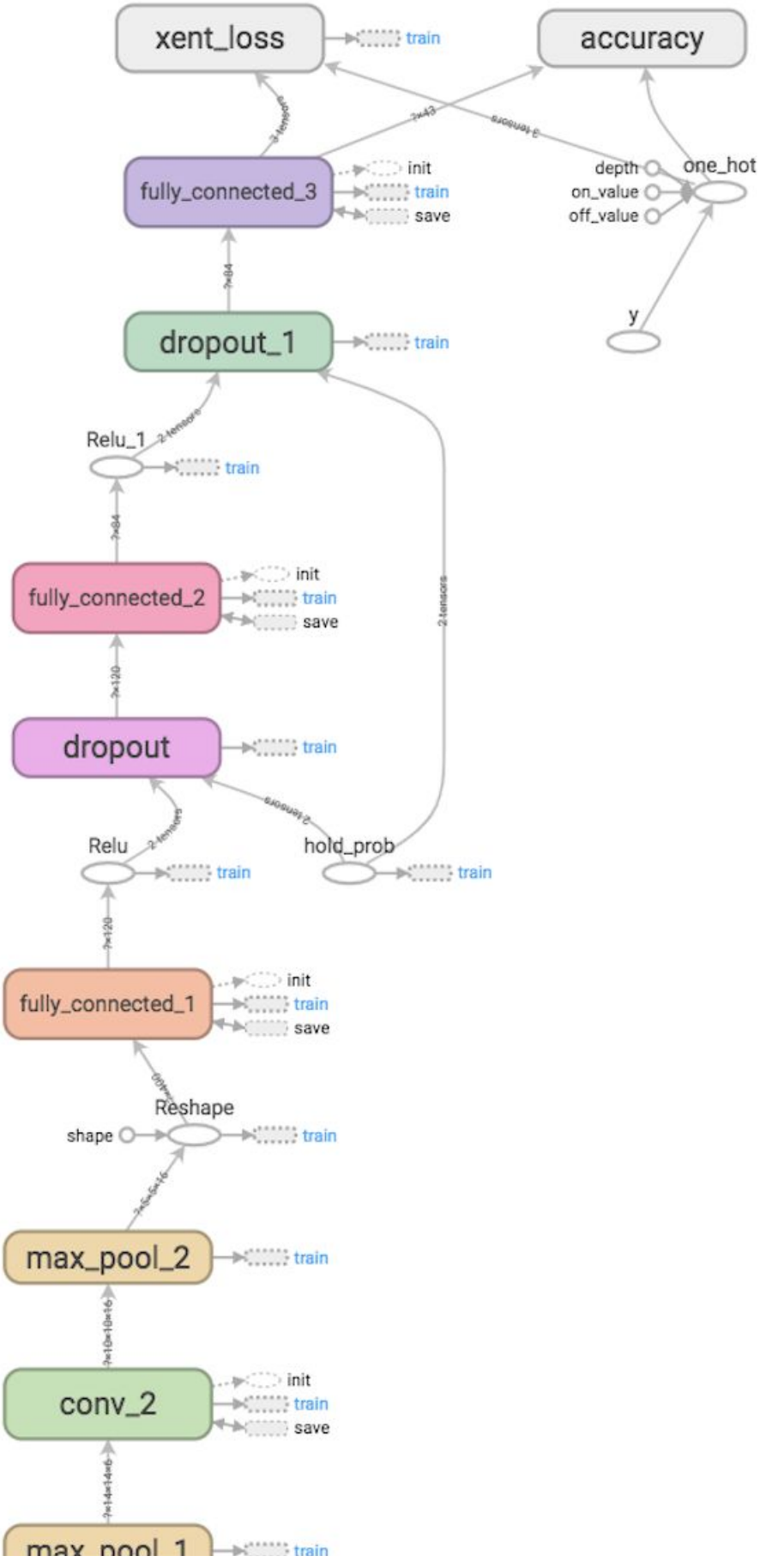
II. Model design and architecture

1. Pre-process the data set

We have grayscaled and normalized the pixel value between 0 and 1.
(cell 13 of the notebook). The function call is located inside the file ‘utils.py’.

2. Model Architecture

The model implementation is located inside the file ‘lenet.py’, at the root of the project.


xent_loss
train
accuracy
fully_connected_3
init
train
save
depth
on_value
off_value
one_hot
y
dropout_1
train
Relu_1
train
fully_connected_2
init
train
save
dropout
train
Relu
train
hold_prob
train
fully_connected_1
init
train
save
Reshape
shape
train
max_pool_2
train
conv_2
init
train
save
train

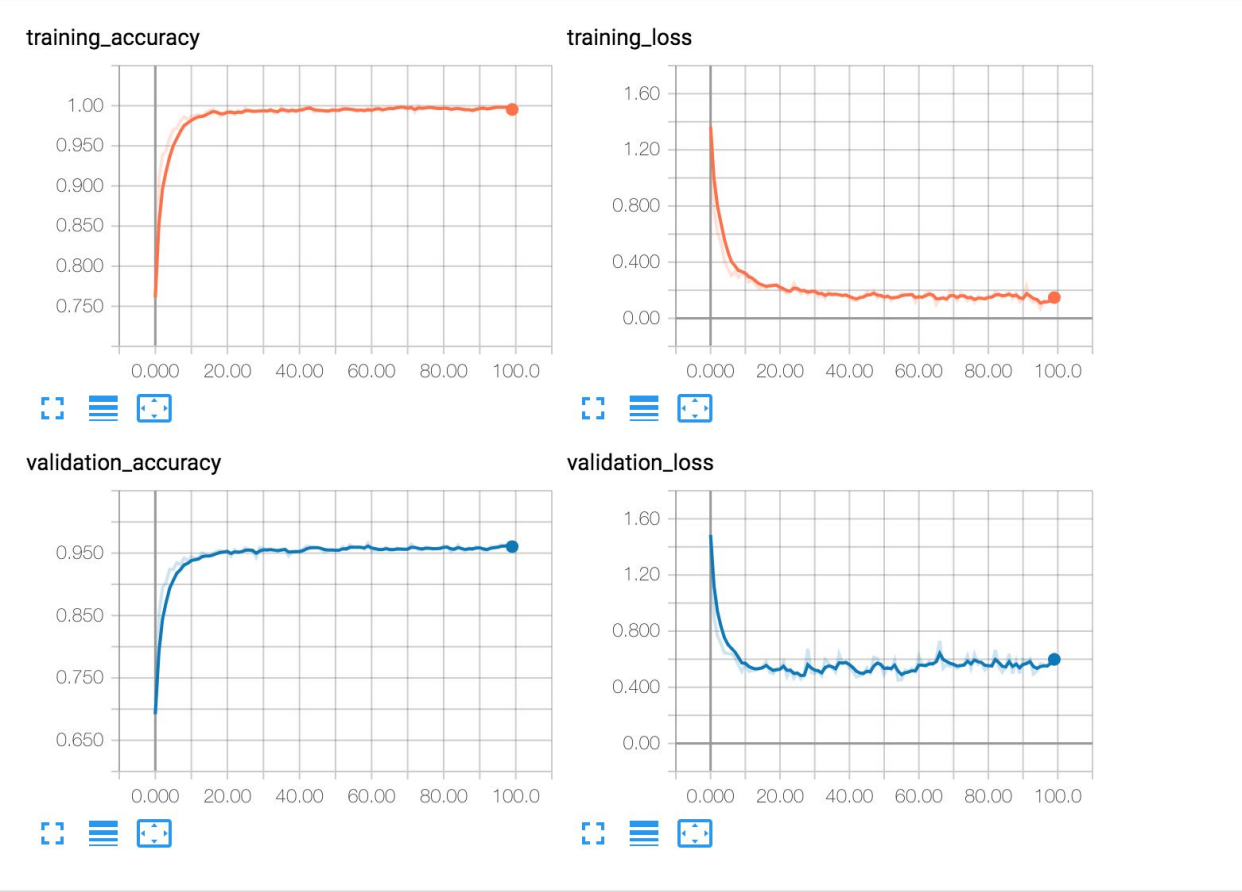
Layer	**Description**
Input	32x32X1 grayscale image
Convolution 5x5 (conv_1)	1x1 stride, valid padding, 27x27x6
Relu	Non-linearity function
Max pooling 2x2 (max_pool_1)	2x2 stride, valid padding, 14x14x6
Convolution 5x5 (conv_2)	1x1 stride, valid padding, 6x6x16
Relu	Non-linearity function
Max pooling 2x2 (max_pool_2)	2x2 stride, valid padding, 3x3x16
Fully connected 120 (fully_connected_1)	Fully connected layer with 120 neurons.
Relu	Non-linearity function
Fully connected 84 (fully_connected_2)	Fully connected layer with 84 neurons
Relu	Non-linearity function
Fully connected 43	Fully connected layer with 43 neurons
Softmax	Output probabilities of each class.

III. Training

The code for this section is located in (cell 14, 15, 16 of the notebook, and the file 'lenet_model.py' at the root of the project)
The first step of the training is to initialize variable weights and biases. We initialized the weights with a normal distribution with mean 0 and a standard deviation of 0.1. We initialized the biases with a constant value of 0.01. For regularization, we used a dropout regularization value of 0.5 for all the fully connected layers. We trained with 100 epochs, a batch size of 128, and a learning rate of 0.003.
The final result is described in the table below.

Training accuracy	100%

Validation accuracy	96.6%
Test accuracy	94.6%

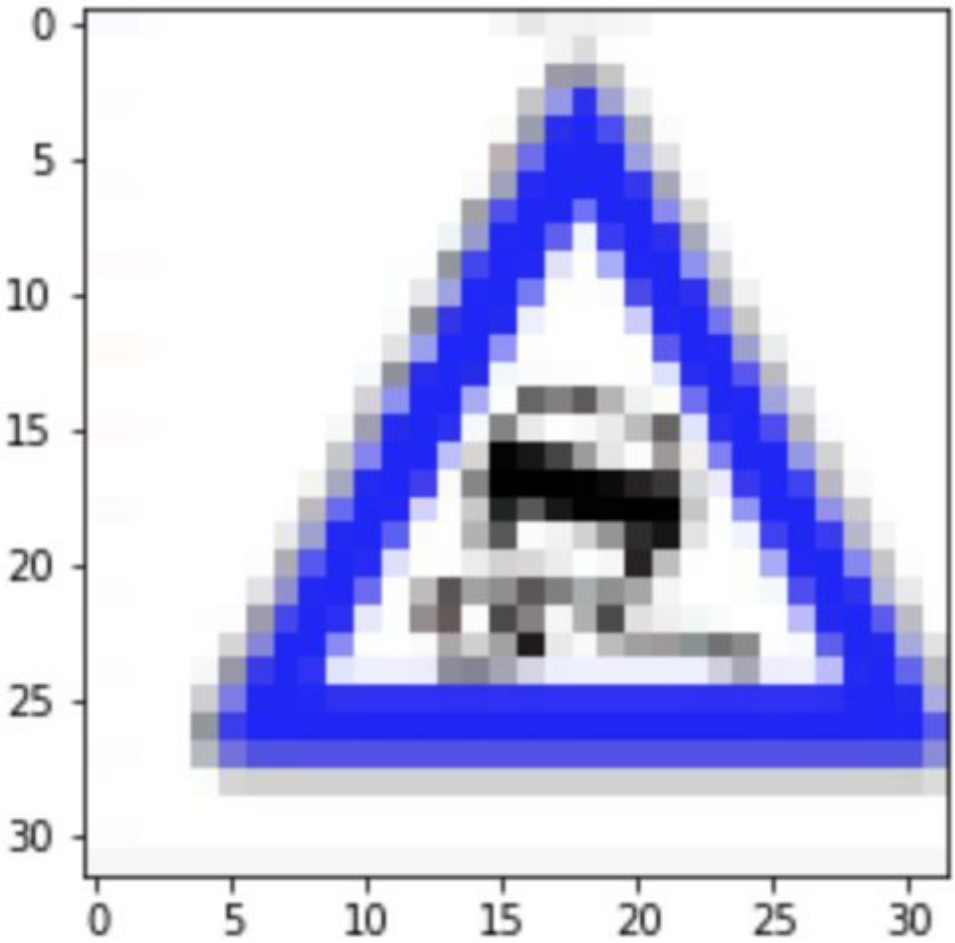


The figure above shows how accuracies(training, and validation) and losses (training, and validation)vary during training. As the number of epochs increase, the accuracies increase, and the losses decrease, hence our model is learning.

IV. Test images on new data

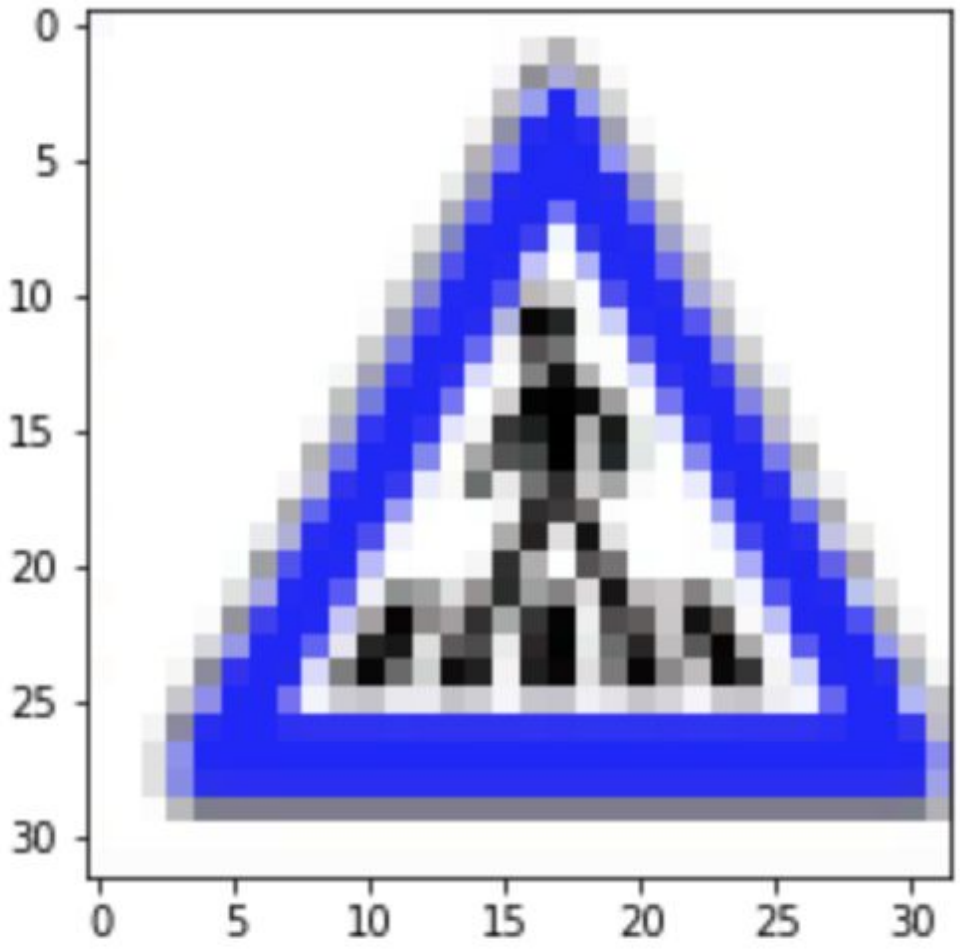
The code in this section is located at the cell 44 of the notebook.

```
23 ['Slippery road']
```



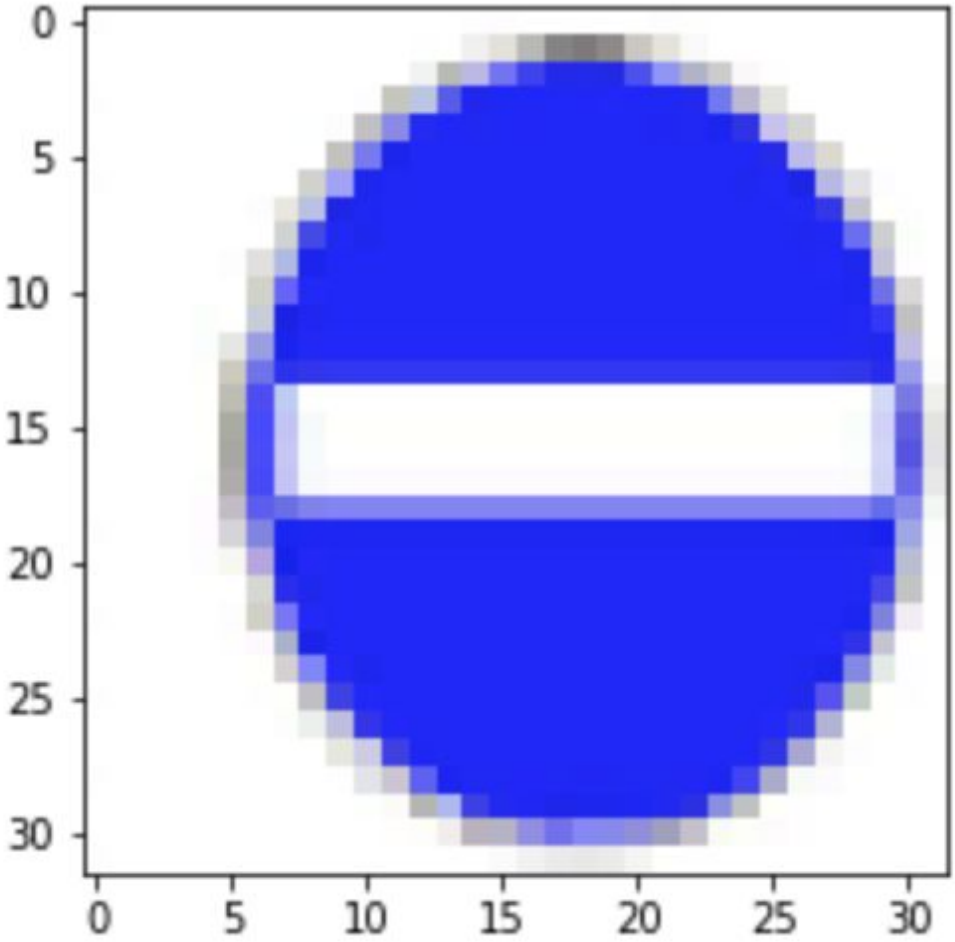
My model successfully predicted this image.

```
27 ['Pedestrians']
```



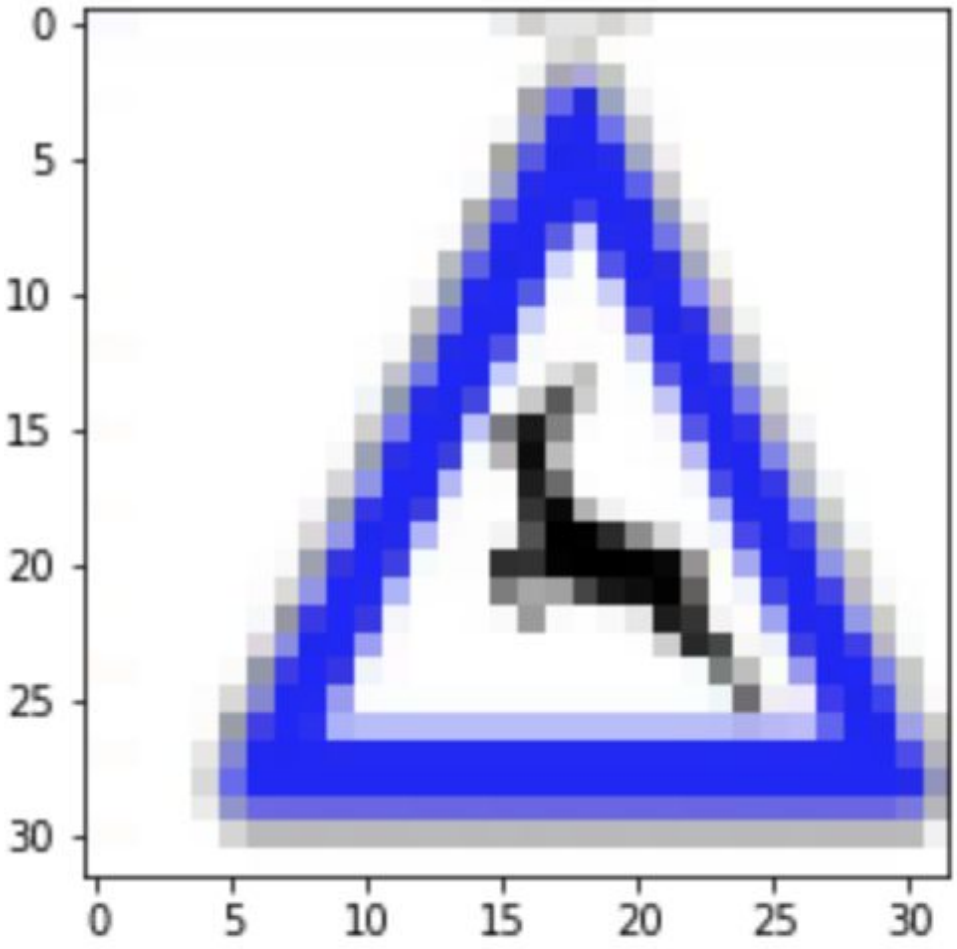
My model almost successfully predicted this image. It didn't predict it exactly because we don't have the class 'pedestrians crossing', but we have the class 'pedestrians' instead.

```
17 ['No entry']
```



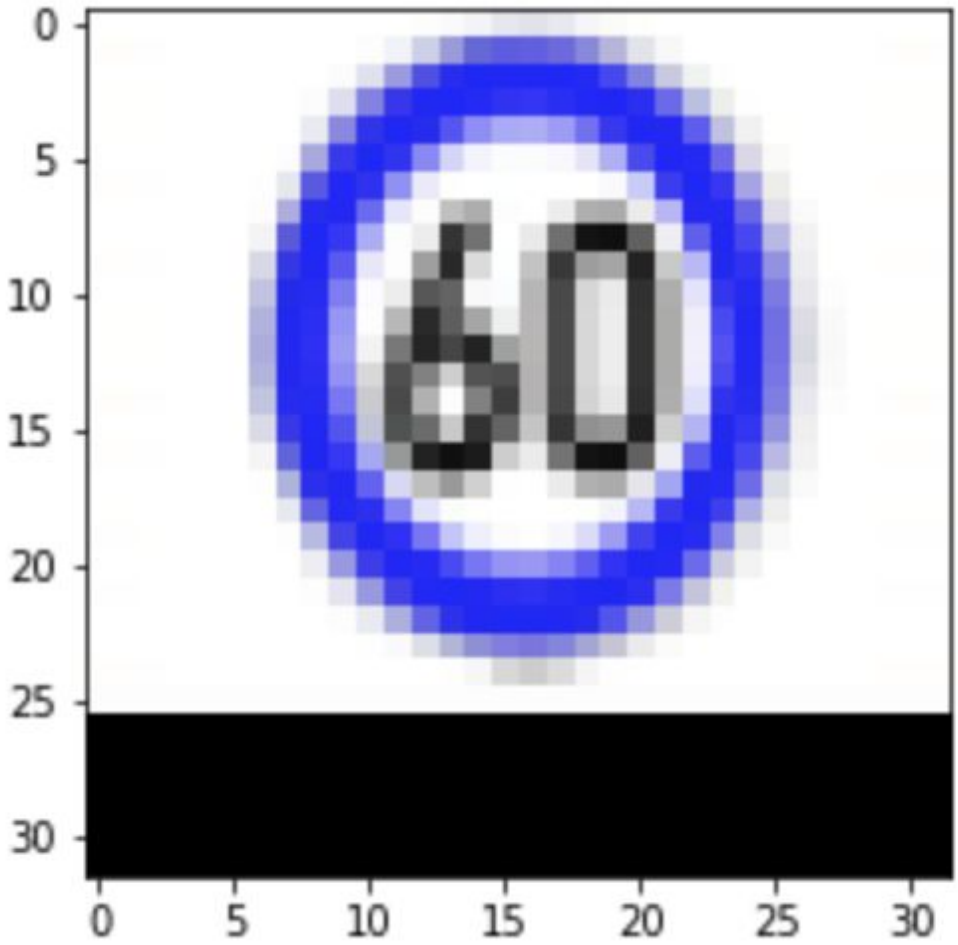
My model has successfully predicted this image.

```
1 ['Wild animals crossing']
```



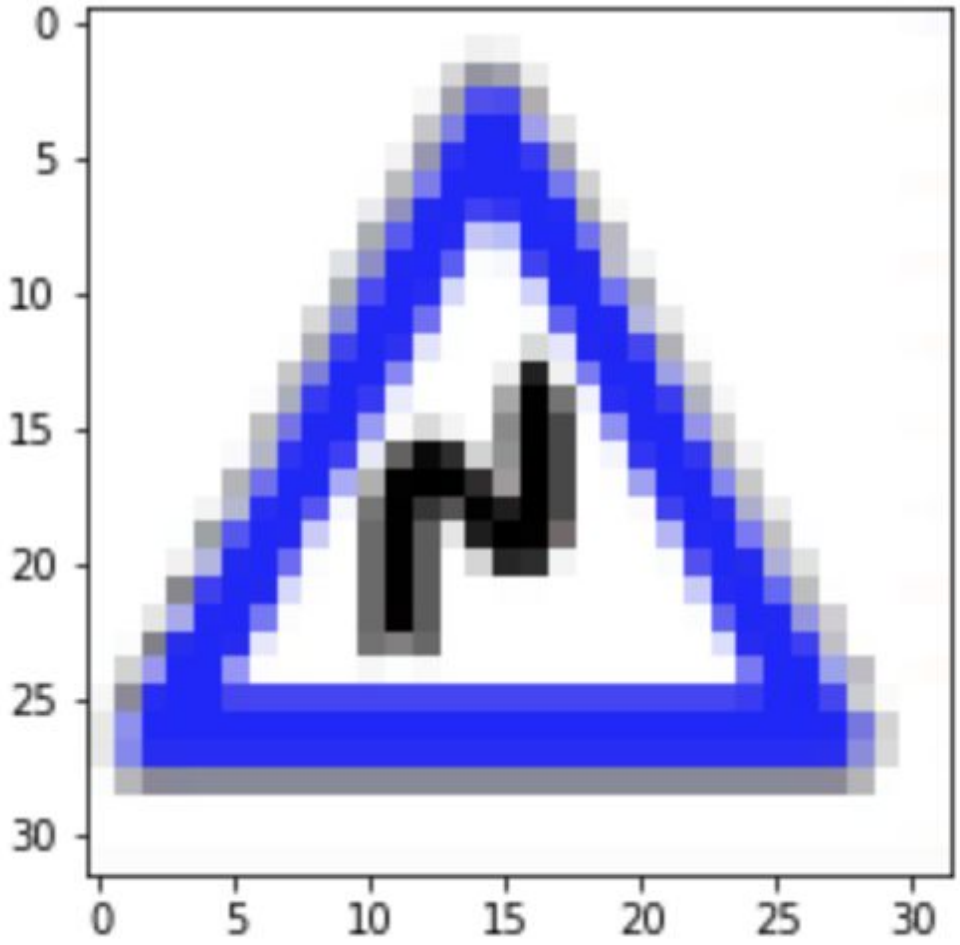
My model successfully predicted this image.

```
12 ['Priority road']
```



My model did not successfully predict this image. I think my model got fooled by the black strip at the bottom.

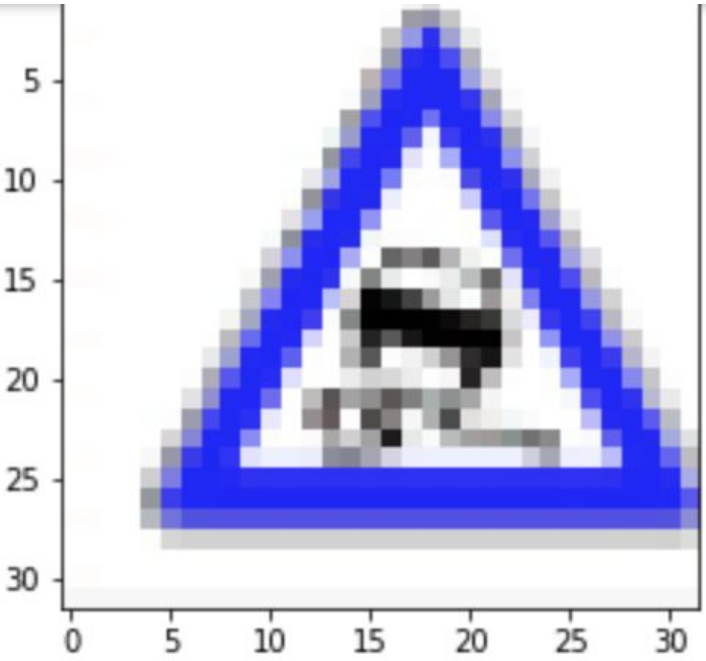
```
28 ['Children crossing']
```



My model did not successfully predict this image. But the predicted class looks similar to the true class.
Overall, my model successfully predicted 4 out of 6 images, hence an accuracy of 66%.

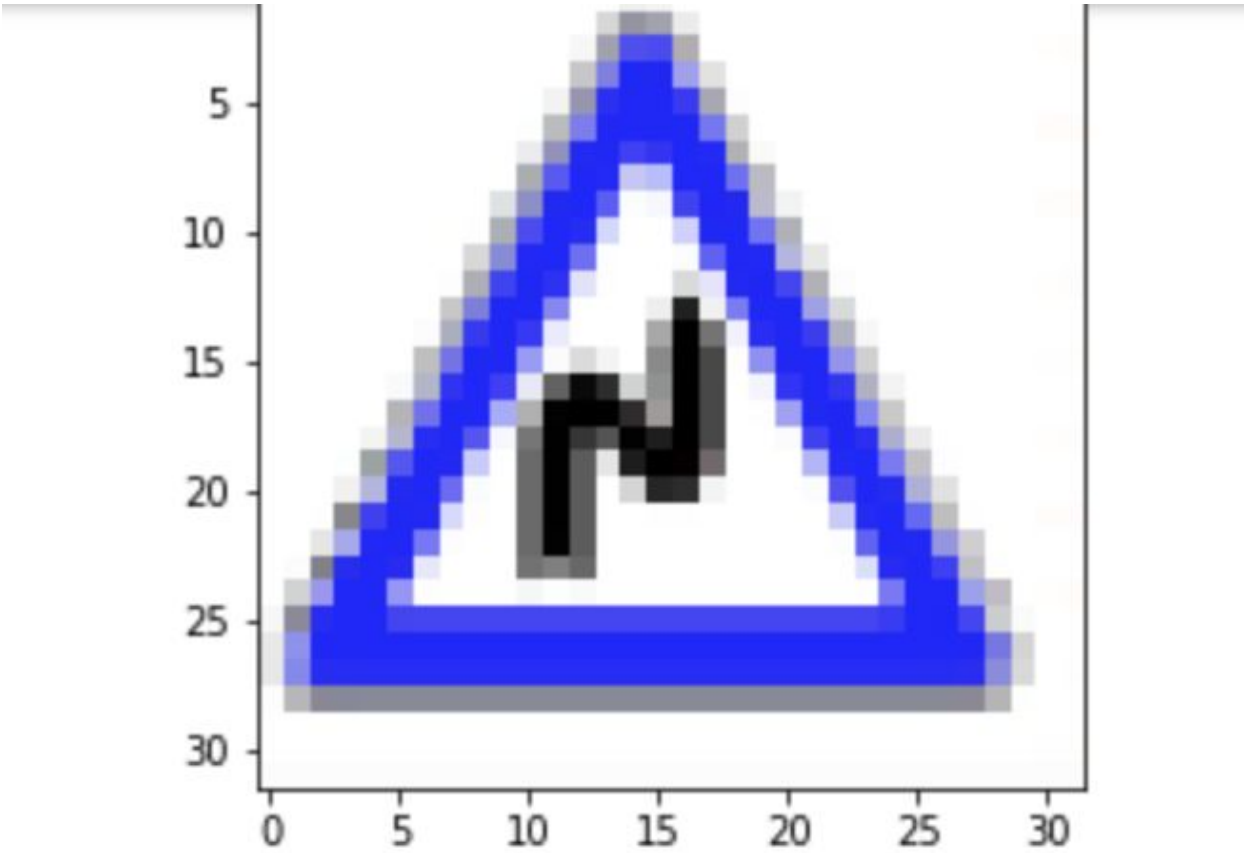
V. Top 5 predictions

The code in this section is located at the cell 56 of the notebook.
Now, let's evaluate the top 5 predictions.




```
['Slippery road']
['Dangerous curve to the left']
['Beware of ice/snow']
['Right-of-way at the next intersection']
['No passing']
```

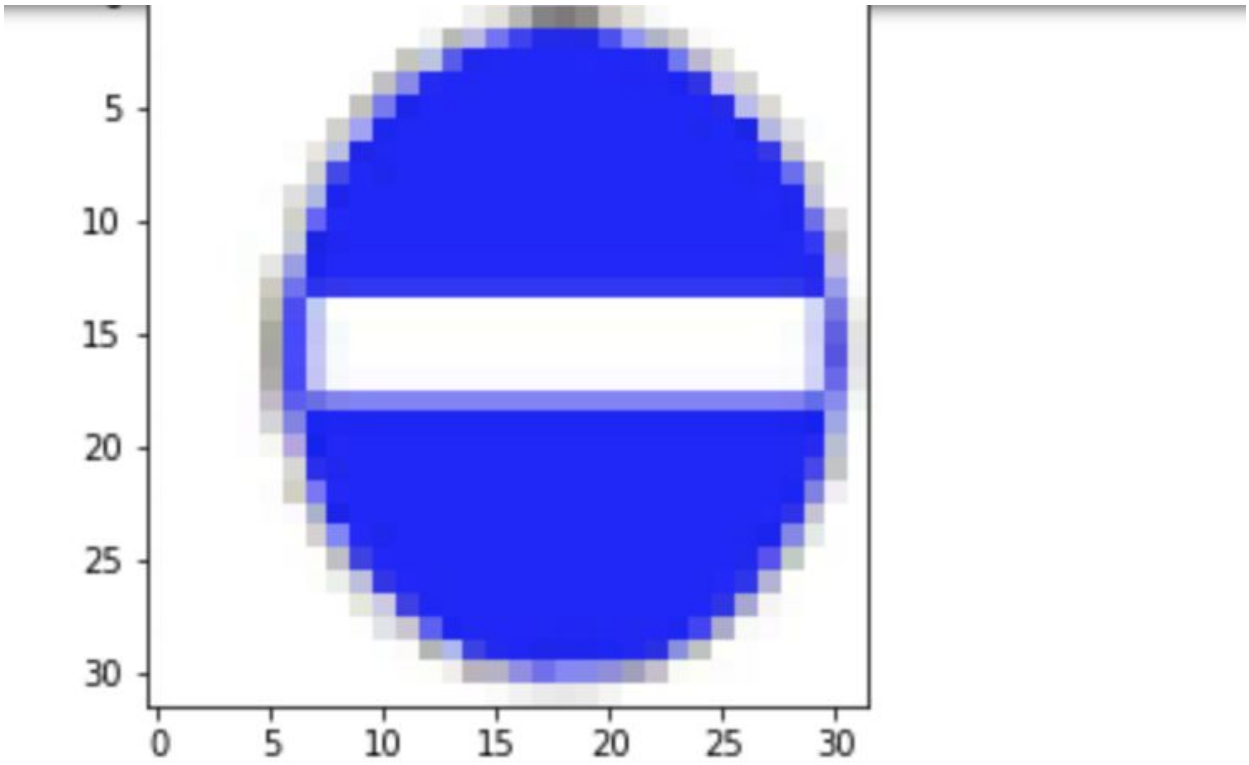
Predictions	**True value**	**Probabilities**
Slippery road	Slippery road	0.99
Dangerous curve to the left	Slippery road	6.15e-06
Beware of nice/snow	Slippery road	9.19e-07
Right of way at the next intersection	Slippery road	2.5e-07
No passing	Slippery road	1.02e-07




```
['Children crossing']
['Turn left ahead']
['Ahead only']
['Speed limit (60km/h)']
['Bicycles crossing']
```

Predictions	**True value**	**Probabilities**
Children crossing	Double curve	0.99
Turn left ahead	Double curve	9.42e-07
Ahead only	Double curve	1.61e-10
Speed limit (60km/h)	Double curve	3.96e-11

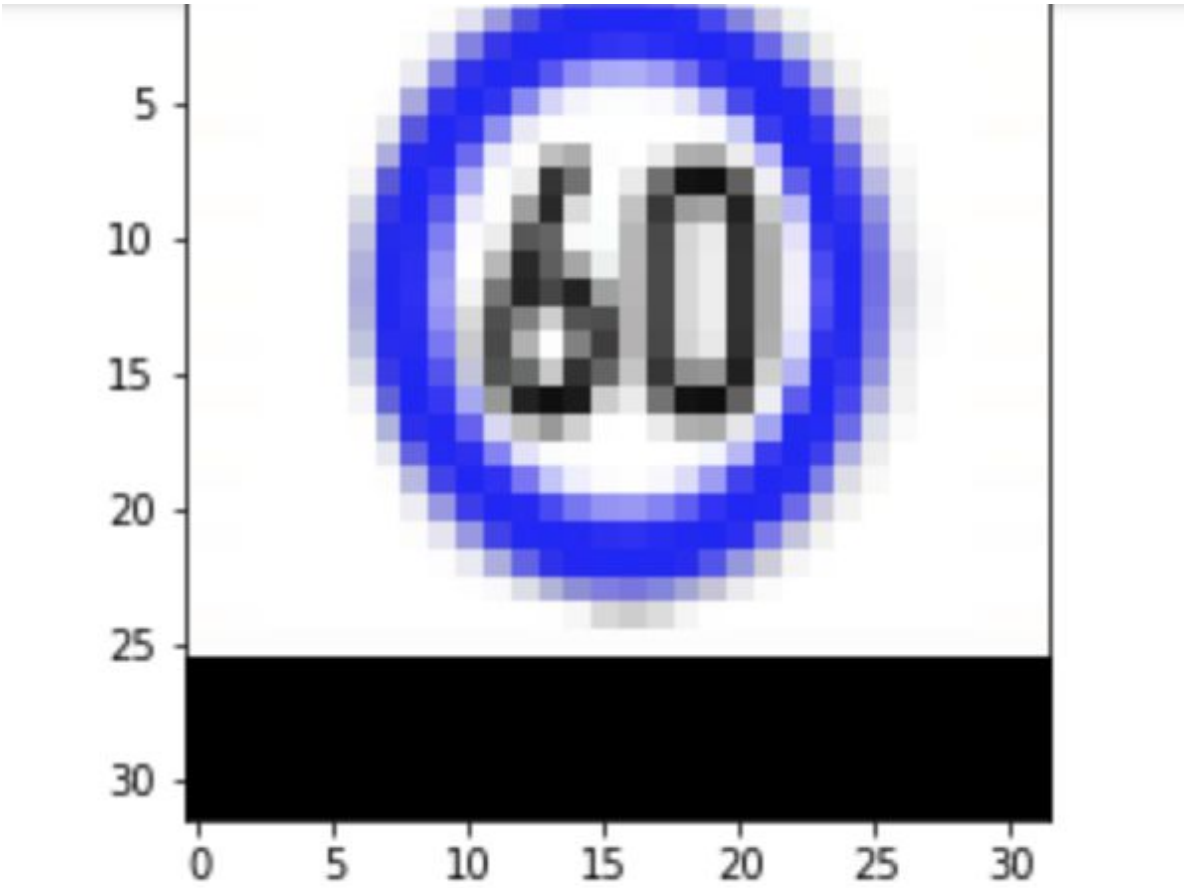
Bicycles crossing	Double curve	2.65e-11




```
['No entry']
['Speed limit (20km/h)']
['Stop']
['End of all speed and passing limits']
['Speed limit (30km/h)']
```

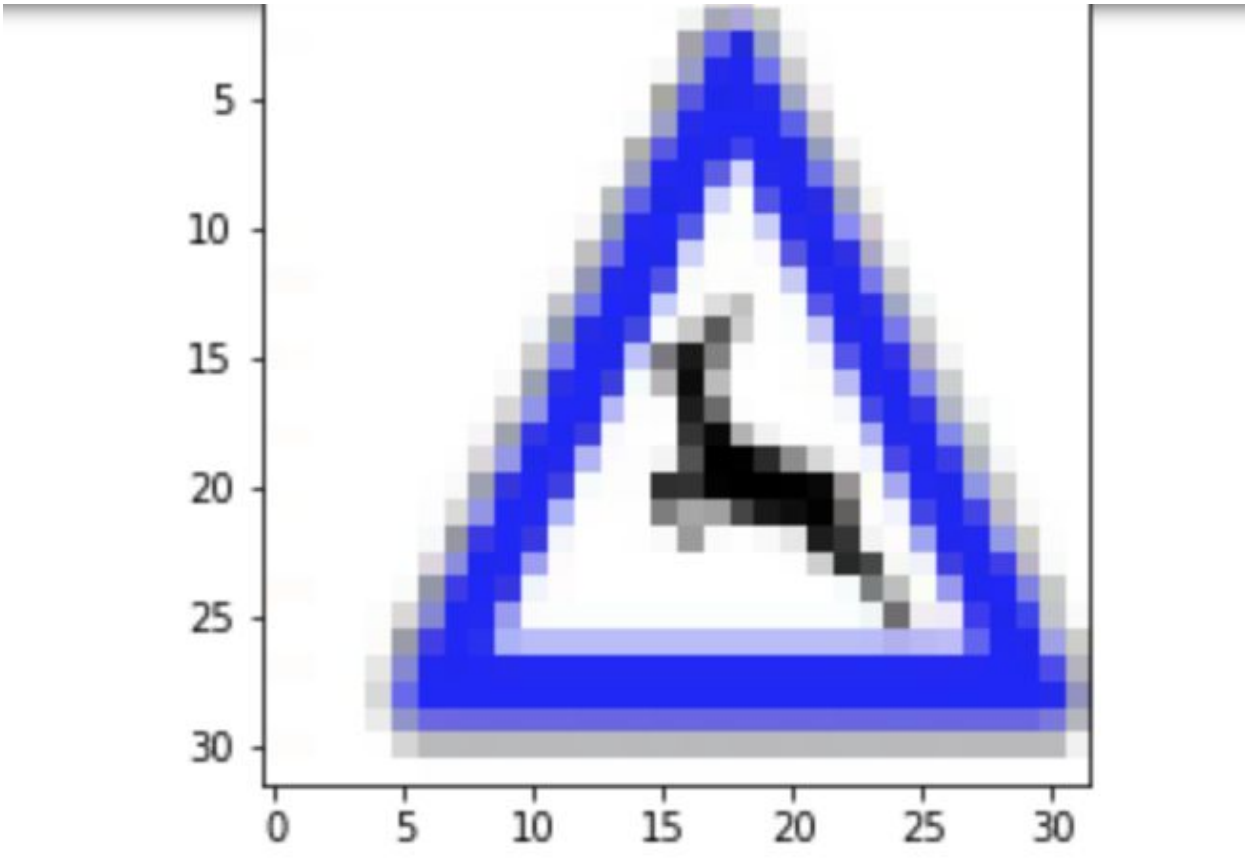
Predictions	**True Value**	**Probabilities**
No entry	No entry	1.00
Speed limit (20km/h)	No entry	0
Stop	No entry	0

End of all speed and passing limits	No entry	0
Speed limit (30km/h)	No entry	0




```
['Priority road']
['Stop']
['Road work']
['Keep right']
['Roundabout mandatory']
```

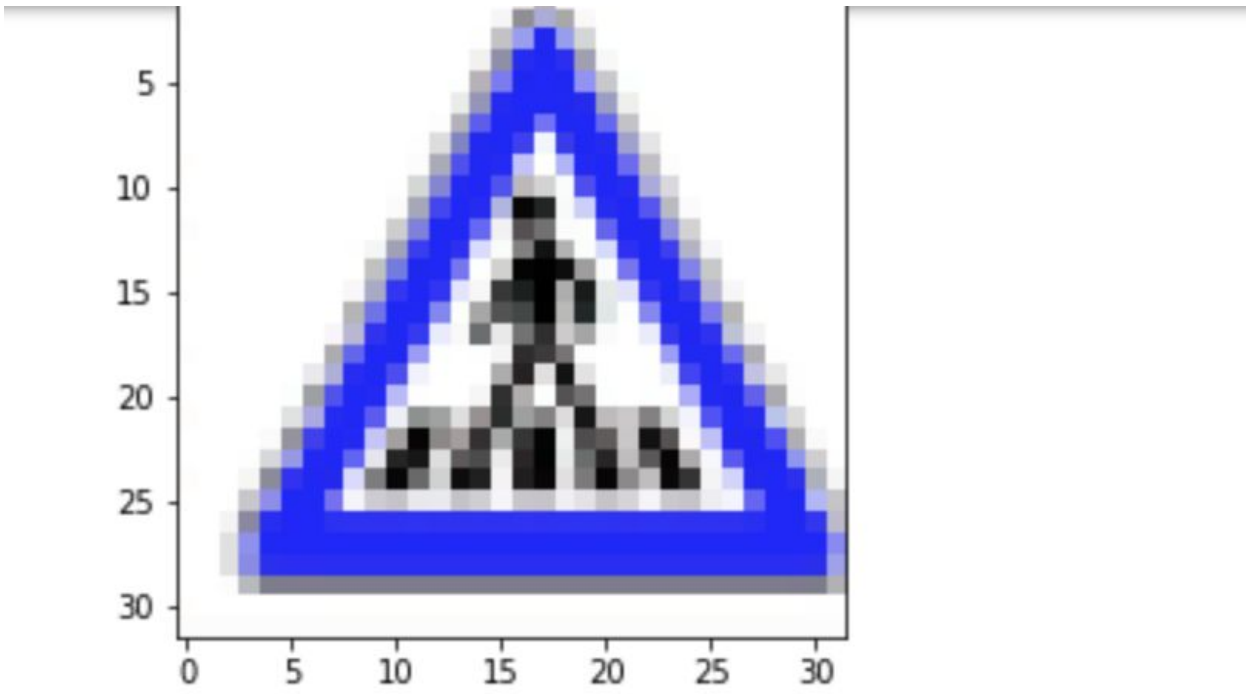
Predictions	**True value**	**Probabilities**
Priority road	Speed limit (60km/h)	0.307
Stop	Speed limit (60km/h)	0.147
Road work	Speed limit (60km/h)	0.138
Keep right	Speed limit (60km/h)	0.138
Roundabout mandatory	Speed limit (60km/h)	0.130




```
['Wild animals crossing']
['Road work']
['Speed limit (80km/h)']
['Speed limit (60km/h)']
['General caution']
```

Predictions	**True value**	**Probabilities**
Wild animals crossing	Wild animals crossing	0.99
Road work	Wild animals crossing	6.66e-07
Speed limit (80km/h)	Wild animals crossing	1.15e-12

Speed limit (60km/h)	Wild animals crossing	2.02e-13
General caution	Wild animals crossing	7.67e-15




```
['Pedestrians']
['Right-of-way at the next intersection']
['General caution']
['Dangerous curve to the right']
['Beware of ice/snow']
```

Predictions	**True value**	**Probabilities**
Pedestrians	Pedestrian crossing	0.585
Right-of-way at the next intersection	Pedestrian crossing	0.215
General caution	Pedestrian crossing	0.132
Dangerous curve to the right	Pedestrian crossing	5.065e-02

Beware of ice/snow	Pedestrian crossing	6.847e-03]

Overall, my model successfully predicted 4 out of 6 images, hence an accuracy of 66%.

VI Improvements

The model can be improved by doing the following:

1 - Add additional images, and classes data.

2 - Train with higher value of epochs.

3 - Try different value of hyperparameters such as learning rate.

4 - Improve the images resolution.