Traffic Sign Recognition

Build a Traffic Sign Recognition Project. The goals / steps of this project are the following:

◊ Load the data set
◊ Explore, summarize and visualize the data set
◊ Design, train and test a model architecture
◊ Make predictions and Analyze the SoftMax probabilities of the new images
◊ Summarize

1. **Load Data Set:**
   - We have three pickled data sets in which we've already resized the images to 32x32.
     - train.p: The training set.
     - test.p: The testing set.
     - valid.p: The validation set.

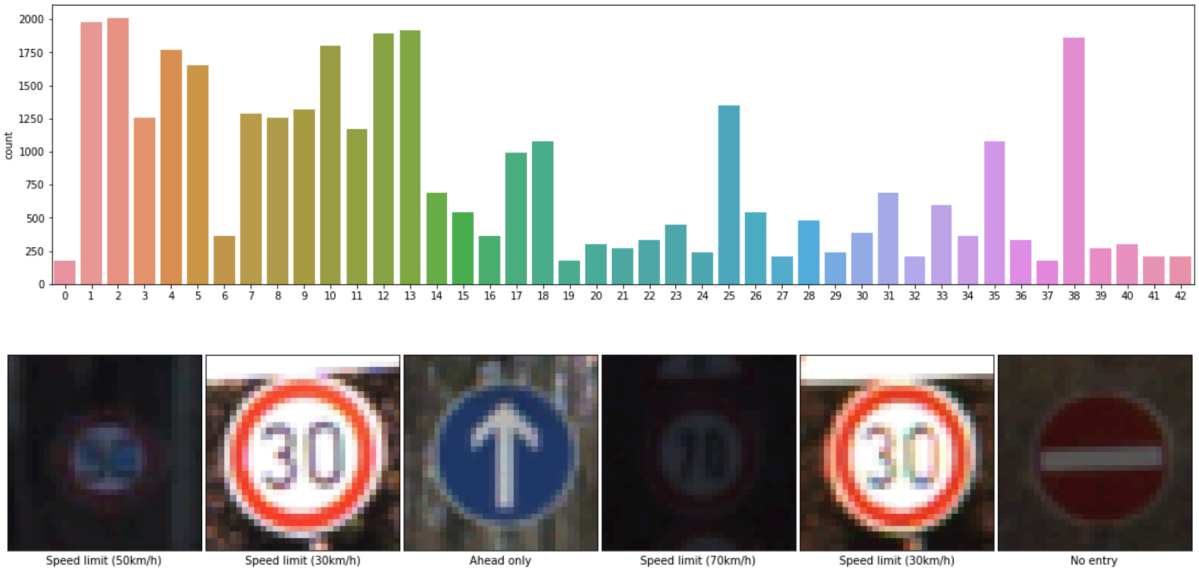
   - The pickled data is a dictionary with 4 key/value pairs:
     - 'features' is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
     - 'labels' is a 1D array containing the label/class id of the traffic sign. The file signnames.csv contains id -> name mappings for each id.
     - 'sizes' is a list containing tuples, (width, height) representing the original width and height the image.
     - 'coords' is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image.

2. **Explore, summarize and visualize the data set:**
   - I used the NumPy and Pandas library to calculate summary statistics of the traffic signs data set:

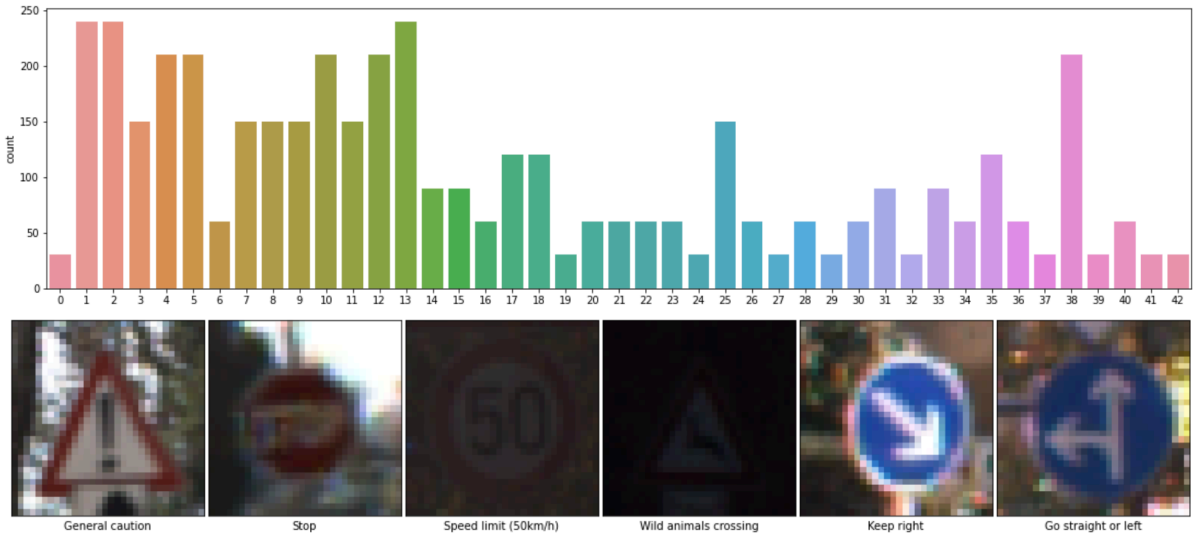
```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

- Here is an exploratory visualization of the data set. It is a bar chart showing how the number of labels and sample images from each data set:

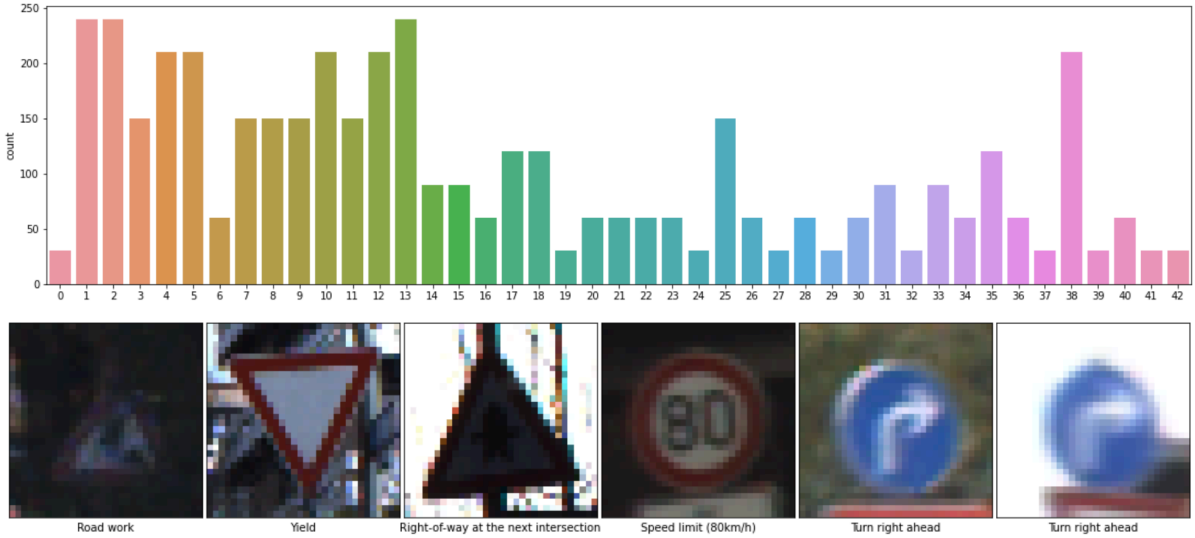
Training Data



Validation Data



Test Data



3. Design, train and test a model architecture:

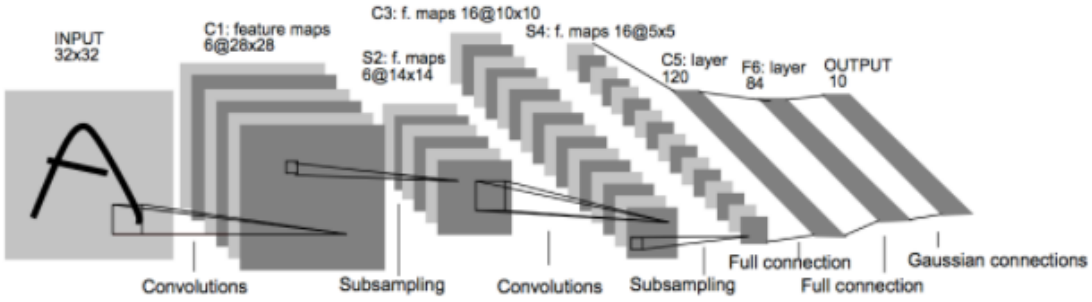
a. **Preprocess Data Steps:**

- Shuffling: In general, I use **sklearn** shuffle the training data to increase randomness and variety in training dataset.
- Grayscaling: Using grayscale images instead of color improves the ConvNet's accuracy. I use **OpenCV** to convert the training images into grey scale.
- Local Histogram Equalization: This technique simply spreads out the most frequent intensity values in an image, resulting in enhancing images with high contrast. Applying this technique will be very helpful in our case since the dataset in hand has real world images, and many of them has low contrast.
- Normalization: Normalization is a process that changes the range of pixel intensity values. Usually the image data should be normalized so that the data has mean zero and equal variance.

b. **Build model:**

- LeNet-5 is a convolutional network designed for handwritten and machine-printed character recognition. It was introduced by the famous Yann LeCun in his paper Gradient-Based Learning Applied to Document Recognition in 1998. Although this ConvNet is intended to classify hand-written digits, we're confident it have a very high accuracy when dealing with traffic signs, given that both hand-written digits and traffic signs are given to the computer in the form of pixel images.

- LeNet-5 architecture is used to build the training model:



- This ConvNet follows these steps:
    - Input => Convolution => ReLU => Pooling => Convolution => ReLU => Pooling => FullyConnected => ReLU => FullyConnected
    - **Layer 1 (Convolutional):** The output shape should be 28x28x6.
    - **Activation.** Your choice of activation function.
    - **Pooling.** The output shape should be 14x14x6.
    - **Layer 2 (Convolutional):** The output shape should be 10x10x16.
    - **Activation.** Your choice of activation function.
    - **Pooling.** The output shape should be 5x5x16.
    - **Flattening:** Flatten the output shape of the final pooling layer such that it's 1D instead of 3D.
    - **Layer 3 (Fully Connected):** This should have 120 outputs.
    - **Activation.** Your choice of activation function.
    - **Layer 4 (Fully Connected):** This should have 84 outputs.
    - **Activation.** Your choice of activation function.
    - **Layer 5 (Fully Connected):** This should have 10 outputs.

c. **Train model:**

- Now, we'll run the training data through the training pipeline to train the model.
    - Before each epoch, we'll shuffle the training set.
    - After each epoch, we measure the loss and accuracy of the validation set.
    - And after training, we will save the model.
    - A low accuracy on the training and validation sets imply underfitting. A high accuracy on the training set but low accuracy on the validation set implies overfitting.
- We've been able to reach a minimum accuracy of **93%** on the validation set over 20 epochs, using a learning rate of 0.001.
- Below is the accuracy rate in each set:
    - Validation Set:

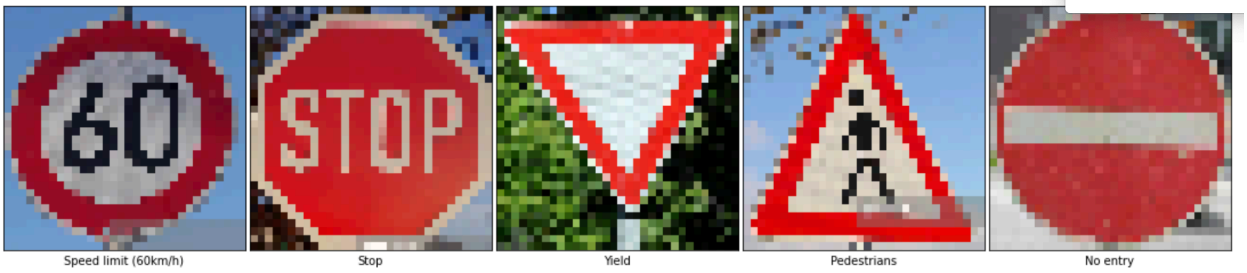
```
EPOCH 17 ...
Validation Accuracy = 0.927

EPOCH 18 ...
Validation Accuracy = 0.936

EPOCH 19 ...
Validation Accuracy = 0.933

EPOCH 20 ...
Validation Accuracy = 0.937

Model saved
```

    - Test Set:

```
Test Accuracy = 0.908
```

4. Make predictions and Analyze the SoftMax probabilities of the new images:

- Here are 5 German traffic signs that I found on the web:



- These test images include some easy to predict signs, and other signs are considered hard for the model to predict. For instance, we have easy to predict signs like the "No Entry" and the "Yield ". The two signs are clear and belong to classes where the model can predict with high accuracy.

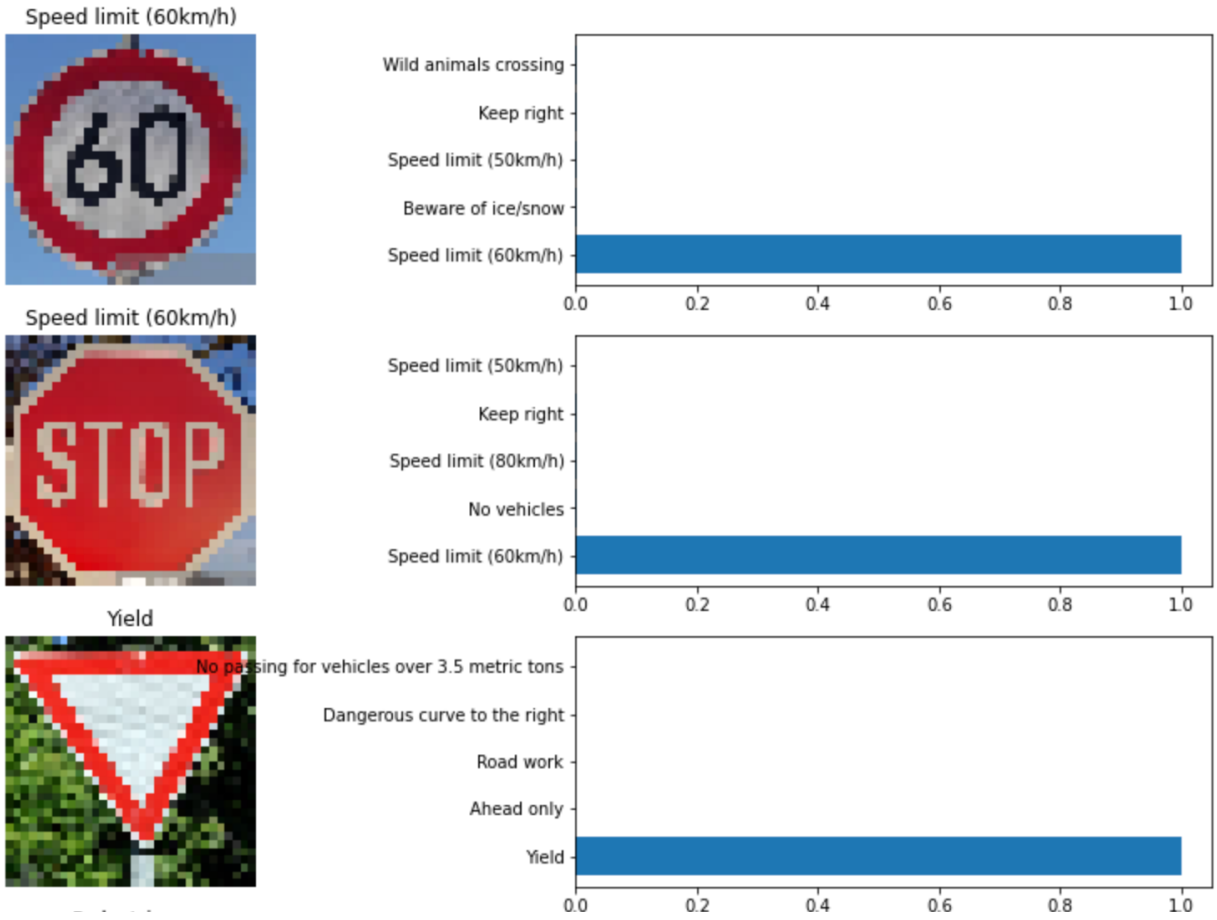
- On the other hand, we have signs belong to classes where has poor accuracy, like the "Stop" sign, because as stated above it turns out that the various speed limits are

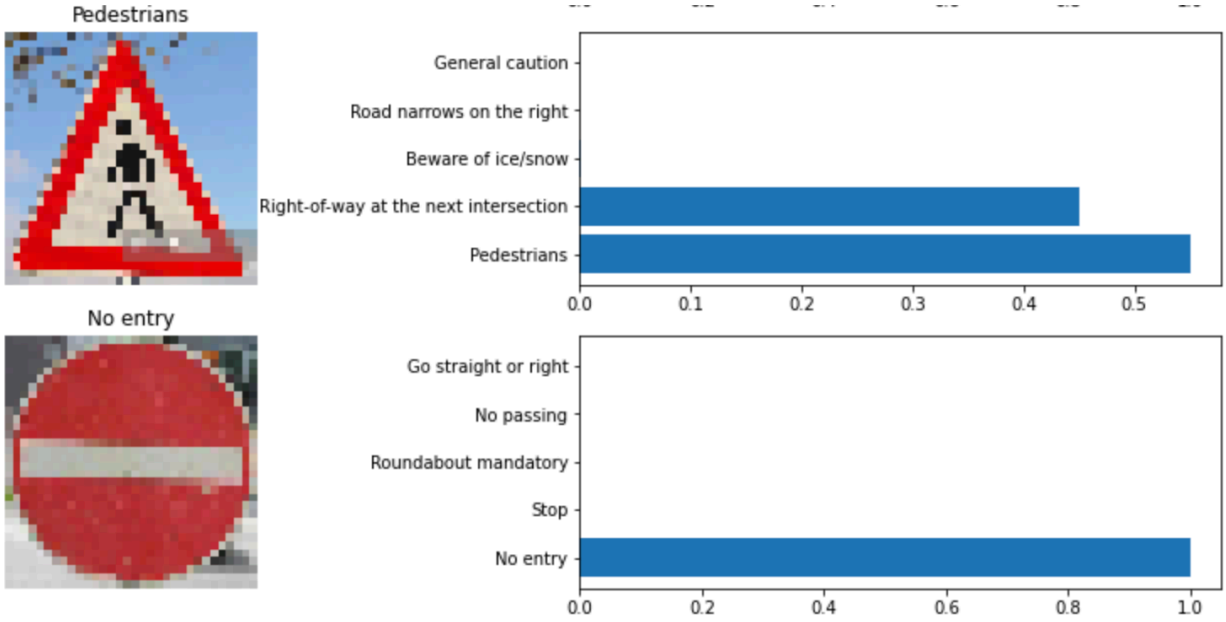
sometimes misclassified among themselves, and the "Pedestrians" sign, because traffic signs with triangular shape are misclassified among themselves.

- As we can notice from the top 5 SoftMax probabilities, the model has very high confidence (100%) when it comes to predict simple signs, like the "No Entry" and the "Yield" sign

- On the other hand, the model's confidence slightly reduces with more complex triangular sign in a "pretty noisy" image, in the "Pedestrian" sign image, we have a triangular sign with a shape inside it, and the images copyrights adds some noise to the image, the model was able to predict the true class, but with 80% confidence.

- Finally, the “Stop” sign is predicted 100% wrong which indicate that we still have to increase the model performance or increase number of “Stop” sign samples.





5. Summarize

- Using LeNet, we've been able to reach a very high accuracy rate. We can further improve on the model using hierarchical CNNs to first identify broader groups (like speed signs) and then have CNNs to classify finer features (such as the actual speed limit).
- We also need to increase the number of samples that are not enough to train in our model to prevent the imbalanced dataset. This model will only work on input examples where the traffic signs are centered in the middle of the image. It doesn't have the capability to detect signs in the image corners.