Belgian Traffic Sign classification with NN

1. Load the data into Matlab environment.

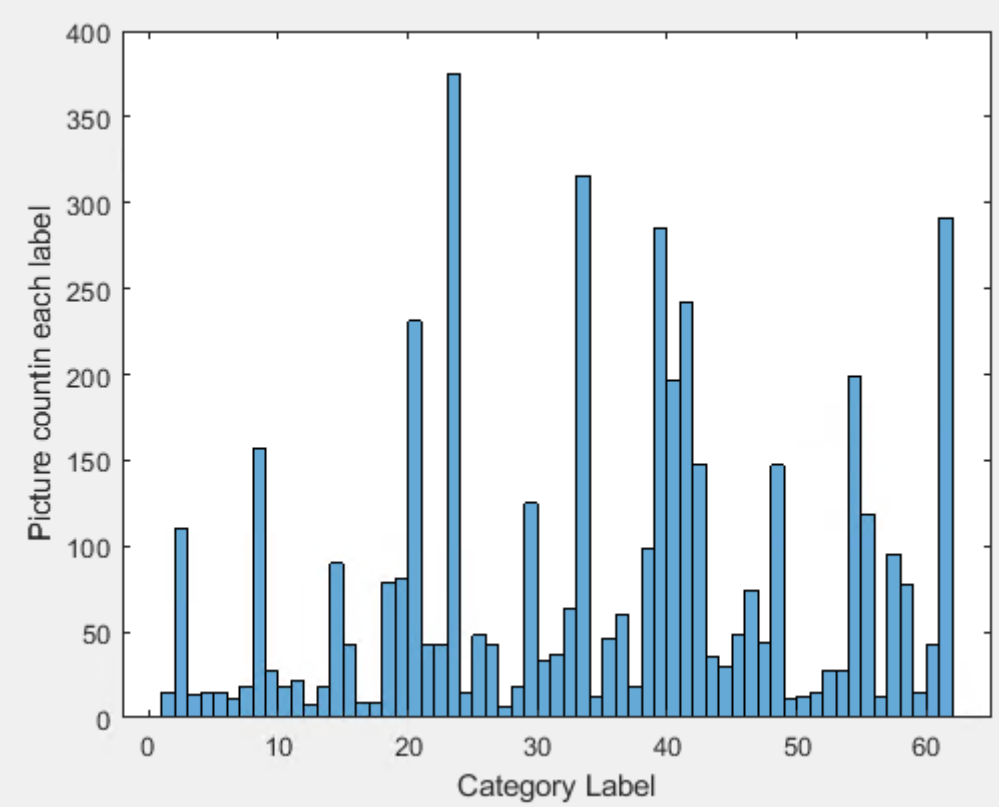
   The Belgian traffic sign images are categorized into 62 types and placed in separate folders. The following 'load_data' script will read the '.ppm' files in each folder and add it to the cell arrays. Creating a composite matrix (like in Python) in Matlab is not possible due to the vectorization limits. So, a cell array is created to provide a better indexing.

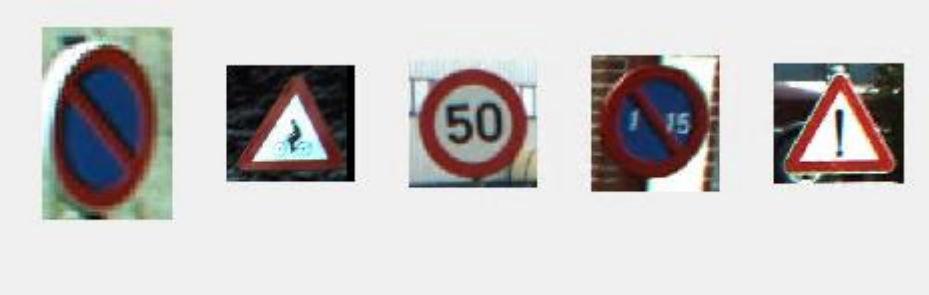
   After the script is executed, cell arrays for images and labels are created. They will be of the size 1x4575, with each as a RGB image a x b x 3.

2. Data Statistics

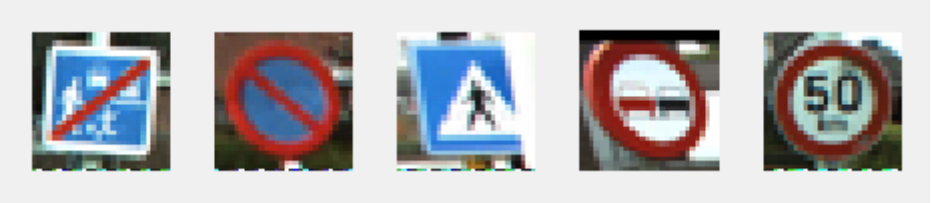
   It is better to visualize the data and related metadata. So, the script 'dataStats' would generate a histogram, an image from each label and few sample pictures.

   The image distribution for each label is not same and the size of the images are also different. This is a real world!



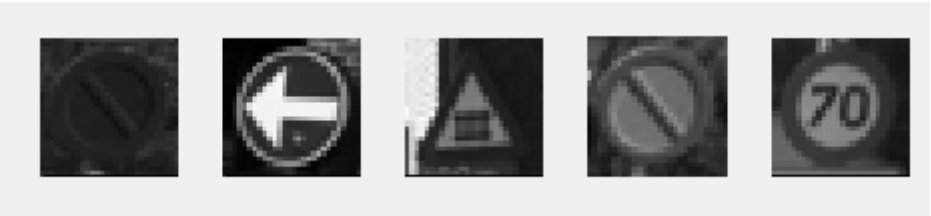


3. Feature extraction.
   a. We will now reshape images to 28x28x3 for uniform size. Please note, that the image may get distorted because of this. So, you can try and add some methods to the 'rescale' script for adjusting the pixel quality. After this, rerun the 'dataStats' script to check the images.



   b. As the images are discrete and mutually exclusive, it is better to convert the images in RGB to gray. This would also reduce the training time and training feature size. We can increase the features later in case of poor accuracy. The

script ‘conv2Gray’ converts the RGB image (28x28x3) to gray (28x28). After this, rerun the ‘dataStats’ script to check the images.



c. Now, we can convert the cell array to matrix of dimensions 4575 X 784. The features 784 is 28x28 pixels that are just made into a row vector.
d. Now we should normalize the feature for better accuracy. The script ‘featureNormalize’ will return the normalized gray images matrix (4575 X 784), mean of each feature in a row vector (1 X 784) and standard deviation in a row vector (1 X 784). This normalized matrix will be our NN training matrix.

4. Neural Network training.

In the current model, I have used only 3 layers (input + hidden + output). The accuracy depends on the number of layers, hidden layer features, regularization and so on. The NN uses backpropagation updated for computing the gradients. The NN is as shown in the picture below.

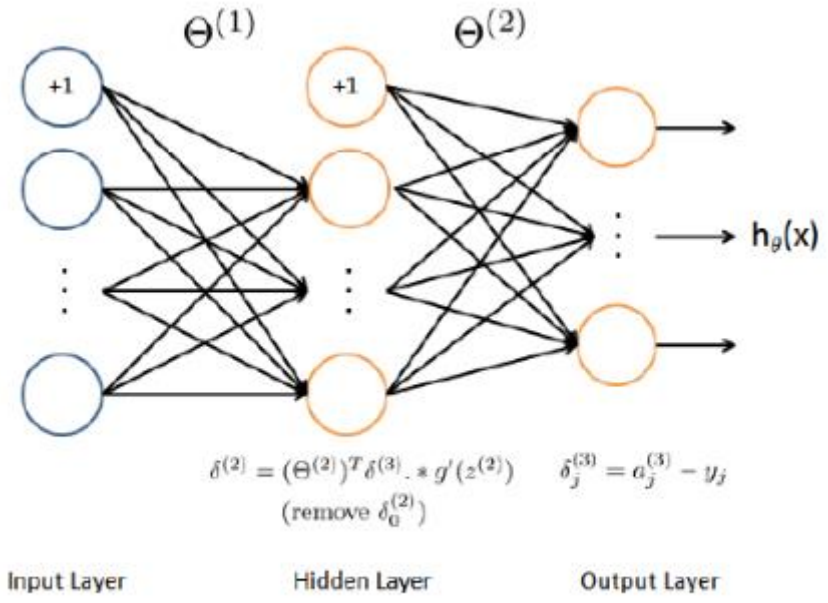


Figure 3: Backpropagation Updates.

The gradient calculation is a 5-step process.

1. Set the input layer's values $(a^{(1)})$ to the t-th training example $x^{(t)}$. Perform a feedforward pass (Figure 2), computing the activations $(z^{(2)}, a^{(2)}, z^{(3)}, a^{(3)})$ for layers 2 and 3. Note that you need to add a $+1$ term to ensure that the vectors of activations for layers $a^{(1)}$ and $a^{(2)}$ also include the bias unit.

2. For each output unit k in layer 3 (the output layer), set $\delta_k^{(3)} = (a_k^{(3)} - y_k)$ where $y_k \in \{0, 1\}$ indicates whether the current training example belongs to class k $(y_k = 1)$, or if it belongs to a different class $(y_k = 0)$.

3. For the hidden layer $l = 2$, set $\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$

4. Accumulate the gradient from this example using the following formula: $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$. Note that you should skip or remove $\delta_0^{(2)}$.

5. Obtain the (unregularized) gradient for the neural network cost function by dividing the accumulated gradients by $\frac{1}{m}$: $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$

   With regularization, the penalty terms are computed and added to the gradients as shown below. Please note that the regularization is not to be applied for bias unit.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \text{ for } j = 0,$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)} \text{ for } j \geq 1$$

With my basic setup with 784 + 378 + 62 features and lambda as 1, I could get an accuracy of around 79.34%. This can be changed as you wish.
The script 'train' will run the NN model using a 'fmincg' optimization script and will give the accuracy and NN weights for the hidden and output layers (Theta1, Theta2).