Traffic Sign Recognition Project

Following pipeline was made to train a model for classifying traffic signs using CNN in Keras.

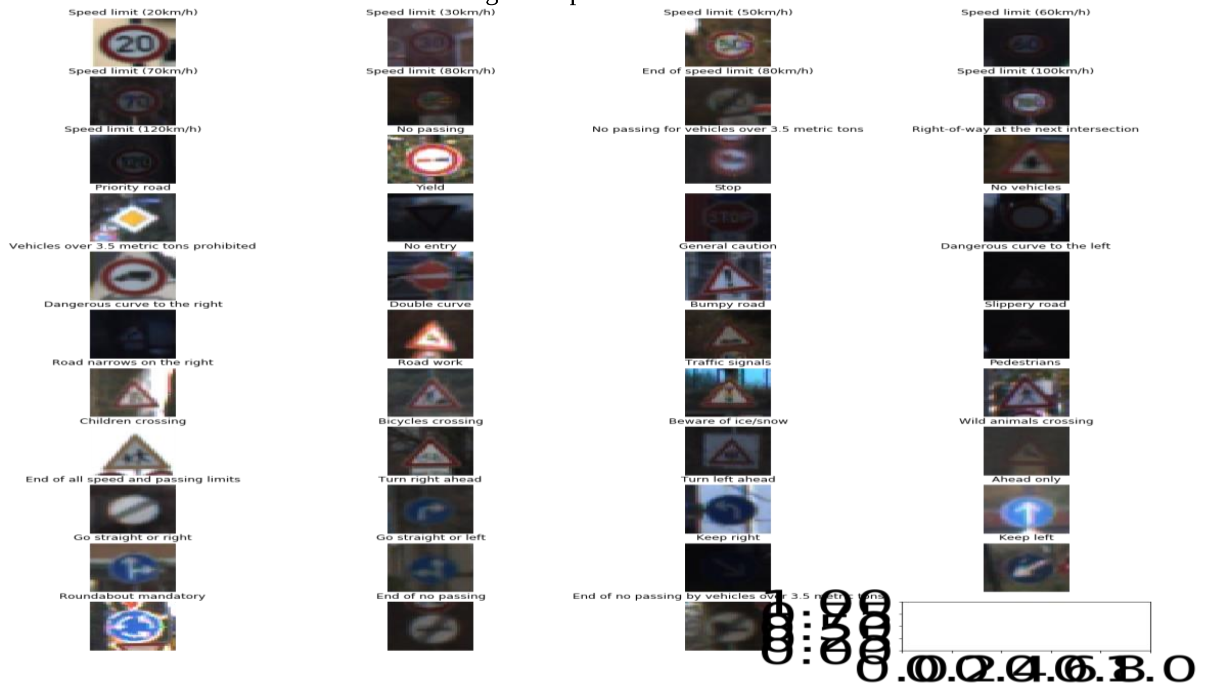
1. Load the data set from given link.
2. Check for classes, size of image for type of train/valid/test dataset.
3. Check for numbers of images given for each class.
4. Use image augmentation to increase test images.
5. Design a CNN model to classify images using train data.
6. Train and validate model.
7. Make predictions using trained model on test set.
8. Predict on new images from web.

1. Load the data set from given link.

This data is like hello world for traffic signal classifications. Nearly all research papers use this dataset as benchmark for their models. Data for this project was shared in form of pickle files splitted in train/valid/test data.

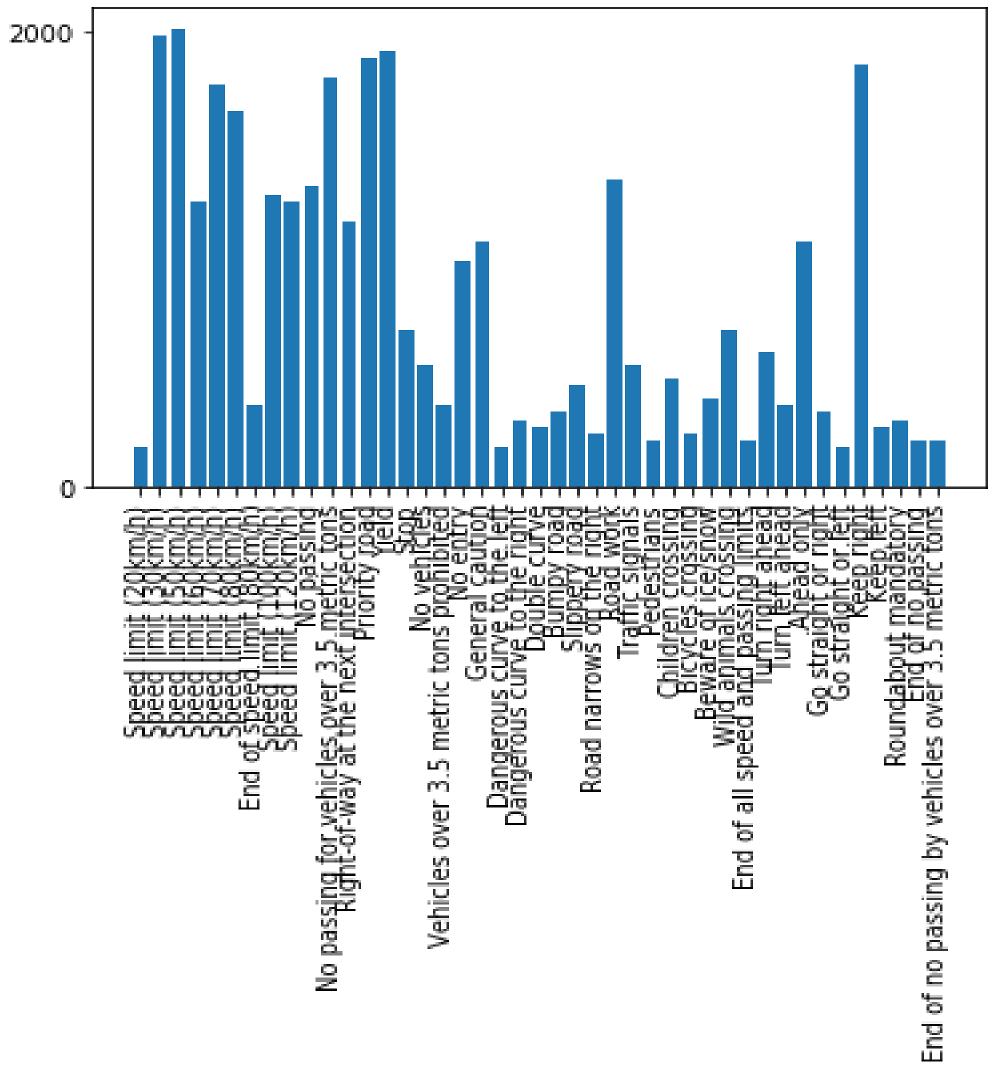
2. Check for classes, size of image for type of train/valid/test dataset.

Given dataset has 43 classes of images. The image ranges from signal for speed limits (round shape), warning signs (triangle shape) and others which are square in shape. Each of the given image is in RGB format with 32x32 size. There is a total of 47429 images given with 34799 in training set, 12630 in test set. Training set images are used to train model and test set is used to check how good is performance on unseen data for model.



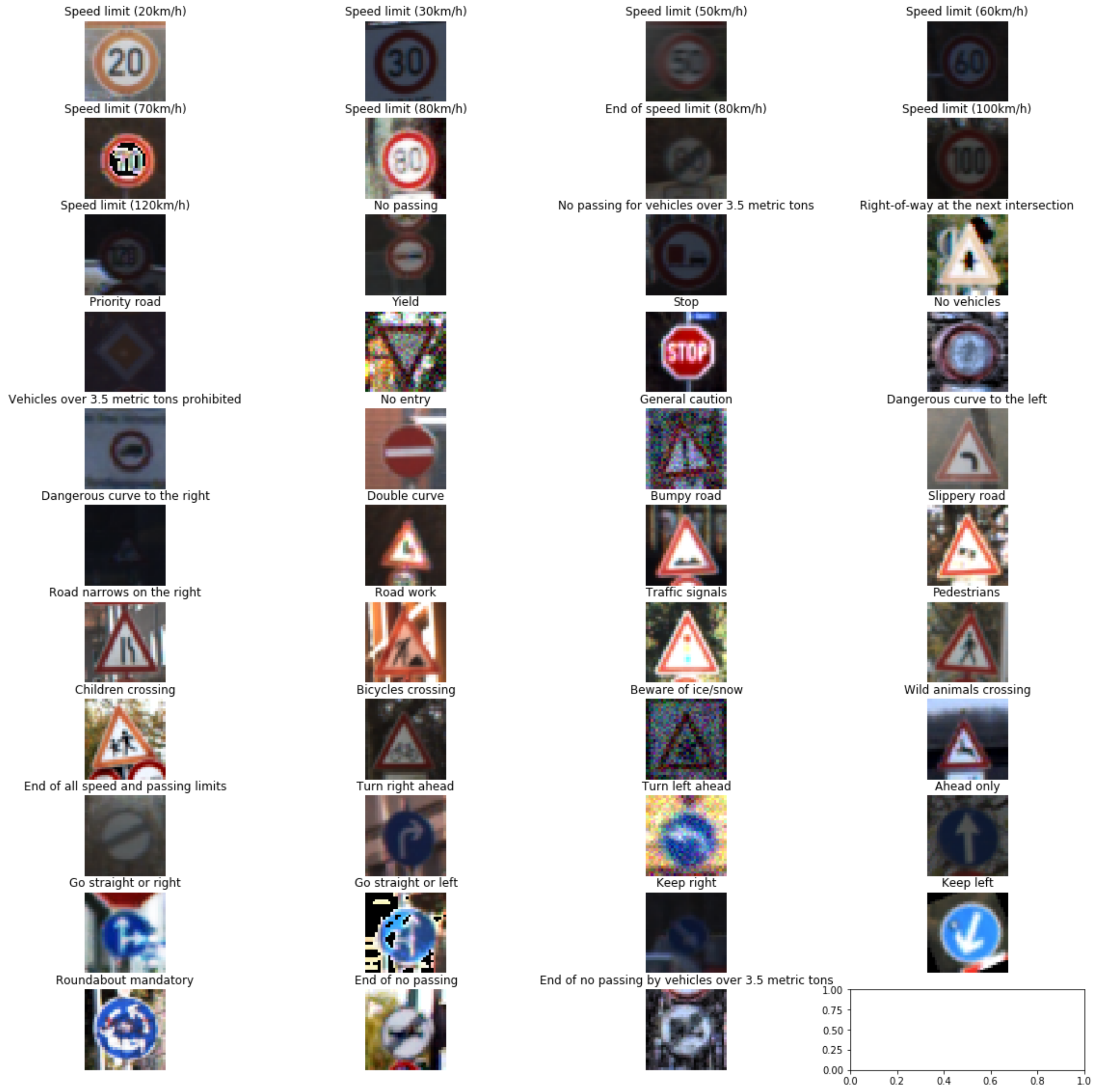
3. Check for numbers of images given for each class.

Once training dataset is loaded, we plotted number of images in each class. This helps us to check if there is uneven distribution of images. In case of uneven distribution, we need change weights of each class so that model is not biased towards a particular class, which in turn will help for better generalization of model.



4. Use image augmentation to increase test images

To get more training images and increase generalization of model, one of the most common practice is to augment given images to generate new images. For this we have used Image contrast to limit min and max intensities in an image, image histogram equalization to evenly distribute intensities across images, local contrast enhancement using Adaptive Equalization, change brightness of image randomly by increasing/decreasing it, randomly rotating images between -25 to 25 degrees, and finally adding some random noise to image. Each of this process is done an all images one by one and added to processed image list. Once it is done for all images, we shuffle the list. This increase our training set by 6 times (by 1 for each operation). Due to limit training time and model generalization we double our initial training data by selecting same size from processed images.



5. Design a CNN model to classify images using train data.

Model used for classifying images uses 6 layers of convolution each followed by batch normalization with relu activation and 2 layers of Max pooling. After convolution layers it has 4 fully connected layers with 3 having relu as activation and final layer having softmax. Model uses Adam as optimizer, cross entropy as loss function and accuracy as metric. Below is architecture of model.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 12, 12, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 3, 3, 128)         512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 1, 128)         147584
_________________________________________________________________
batch_normalization_5 (Batch (None, 1, 1, 128)         512
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_3 (Dense)              (None, 43)                5547
=================================================================
Total params: 688,715
Trainable params: 687,819
Non-trainable params: 896

_________________________________________________________________
```

We have used CNN's for this as this they are best in class for image classification. As general flow, as we go down in CNN layers number of filters increase by a factor of 2 and max pooling is used to reduce image size. After each convolution operation we need to normalize our data so that few nodes do not overshadow others in training model. The selected model has about 700,000 parameters and 19 layers deep. We have kept padding as **valid** and strides as **1** as we do not want to lose information during convolution operations. Although we have used max pool layers but they properly placed to reduce computation time and memory requirements for GPU/CPU.

6. Train and validate model.

We used default learning rate of 0.001. For batch_size we used 128 (power of 2), epochs 16.

Once our model is defined, we ran it on CPU for 16 epochs using class weights from sklearn.utils, to have equal weights for all classes. On training data, we have reached accuracy of more than 99.3% (near perfection). Once training is completed, we saved our model for any future reference.

7. Make predictions using trained model on test set.

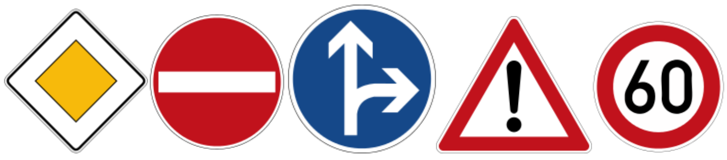
On train data we get accuracy of 99.44 %.

On test data we get accuracy of 95.61%.

On validation data we get accuracy of 97.48%.

8. Predict on new images from web.

Below 5 images were downloaded from web and fed to network. Out of this 5, first 4 were predicted correctly wile last one was wrongly classified. Original classes are 12, 17,36,18,3 for each one while predicted are 12, 17, 36, 18, 38.



Future Improvements:

- Use more deep neural networks like RESNET
- Use multiple kernel size at each layers like in Inception model
- Increase training data size by collecting images from other data source as well.
- Use better image resolution to get same working on higher resolution images.
- Try with different kernel size and other layered architecture.
- Add layers with skip connections to back propagate errors to upper layers.