# COSC2673 – Assignment 2

## Student Details:-

- Vikas Virani : **s3715555**

## Project:-

- Classifying Road traffic signs using modified version of Belgium traffics sign data taken from real world vehicles. The objective of the project is to classify these traffic images based on 2 different categories: Traffic **Sign Type** and **Sign Shape**.

## Approach:-

The dataset of road traffic signs was given in the form of grayscale images of size 28x28 dimensions. To tackle this problem, I have decided to go with two machine learning approaches, Support Vector Machines (SVM) & Convolutional Neural Networks (CNN), as these two are vastly use in research areas and applications like computer vision, object detection**[1], [6]** etc. The data divided across folders and subfolders with directory name as labels. It was processed to make a dataframe of image paths and labels of both category sign type and shape. For all the models trained, the data was divided into 3 parts, 1) training data, 2) validation data to validate the model & 3) testing data to predict on final model.

## (SVM – Sign Type):

For the sign type classification, I have implemented SVM algorithm first. Images are resized to (28,28, 1) for input to SVM and scaled by dividing each pixel value with 255 to improve processing efficiency. As it was already converted to grayscale, no other processing needed on data. Here, it is important to note that **stratified split** was made of the train, validation and test data to **handle class imbalance**, and to make sure each class example is present in all 3 datasets**.** I tried to find best parameters to fit the model by using **Grid Search with 5-fold** cross validation and parameters found are as follows.

```
Best score for training data: 0.959864807773553

Best C: 100

Best Kernel: rbf

Best Gamma: 0.001
```

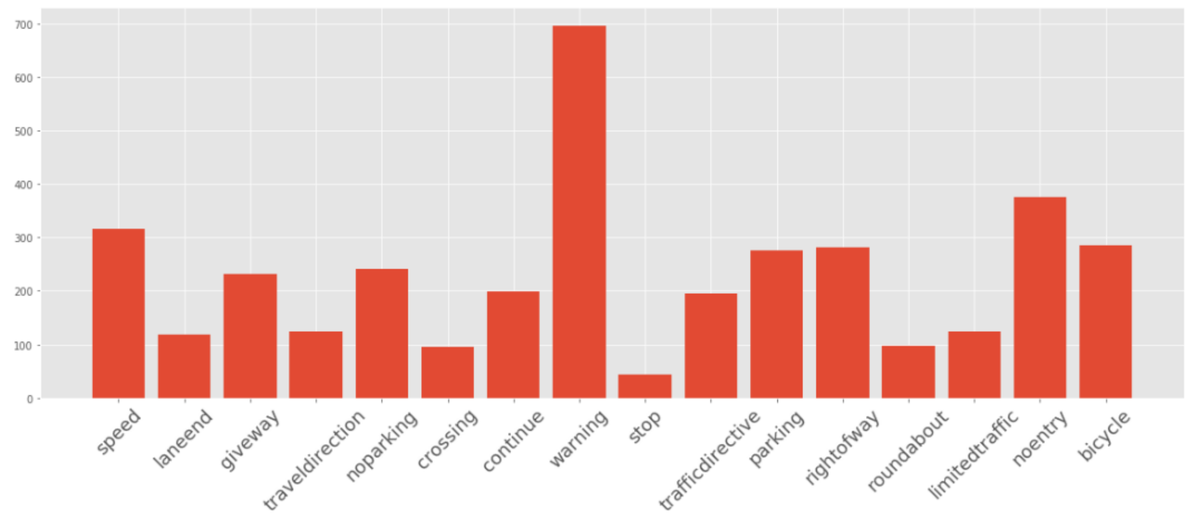
The final accuracy on **test data** was **96%**, which is pretty good. Also, since the target class was imbalanced in terms of data, accuracy is not a good measure, so I checked precision, recall and f1-score of each individual class and they are quite high as well which can be seen in Jupyter file.

This is a quite good result, but there are lots of competitions and research online suggest Convolutional Neural Networks can do better on image classification **[1].** So next, I have tried to implement CNN on sign type classification to compare it with SVM.

## (CNN – Sign Type): Steps followed for this model are as below:

**Data loading:** I have leveraged the data loaded into memory for SVM to start with CNN explore it. For actual training, I have made a dataframe which is passed to image generators.

**Data summary & Exploration:** The dataset has in total **3699** grayscale images with the dimension of **28x28**. I have visualized first image of each sign type class to visually explore it. Which is as per **[I]** which has a label in form of "Label: Name (# of images for that label)". I have also plotted a bar chart of frequency count of each label class shown below,



Now, we can use different things to deal with imbalance, like **alternative accuracy metrics, image augmentation** to add new images, **Random Under/Over sampling** or **SMOTE** technique depending on use case, calculate **weights of class** and feed it to network, **Thresholding with prior class probabilities [2] & [3]**etc. Since the external image augmentation is not allowed and Random sampling has repeated images (which can make model more robust also), but the same can be achieved by providing the class weights of each class in our data and our model give importance accordingly. So, I have decided to leverage that use case of class weights.

**Data Preprocessing:** As, label classes are imbalanced, we will use **stratified split** on our train, test & validation data. Apart from that, when training a final model, I have used various augmentation techniques on data like shearing, zooming the data, rescaling, horizontal flips, rotation of images **[4]** etc. Gray scaling was already done for train data, it will be used to preprocess external test data when creating data generators. Now **for sign type** prediction, rotation can be set to any value, but **for sign shape** prediction, we need to be careful about what value is set for rotation, as certain rotations might change the shape of sign and misclassify it while training for example, **square can become diamond** for rotation around 90 or if original image is tilted then even for rotation of around 40 to 50.

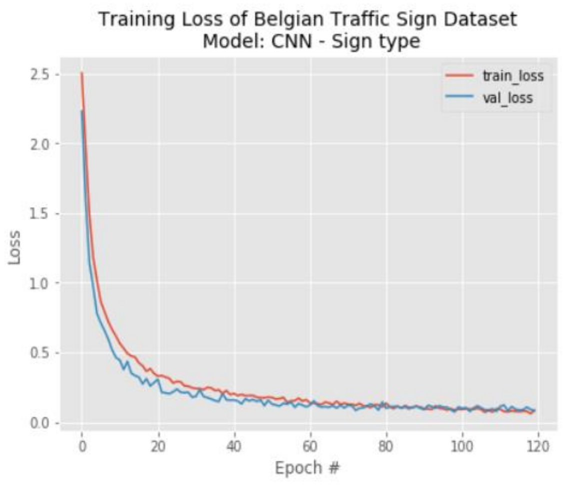
**Model Architecture**: I have started with basic MLP and then iteratively added layers just to visualize how it flows from MLP to CNN. It is to note here that I am discussing only 2 strategies

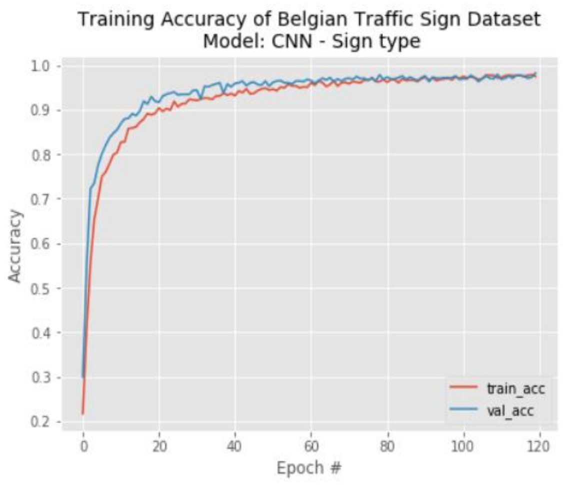
SVM and CNN and so my approach will not have a code for the basic MLP or fully connected network only, this is just an idea of how I reached to CNN iteratively from baseline model. I have made a function called **“cnn_model_sign_test”** which created a model, adds all the different layers in model architecture, this function has various different arguments which controls the hyper parameter that are used inside this architecture. Since our data is imbalanced, accuracy only won’t be a good measure to look at when validating our model, so, I’ve made 3 more functions named **“recall_m”, “precision_m”** and **“f1_m”** & used them as metrics inside my model architecture when compiling a model. There are a lot of inbuilt metrics to choose from as well, provided by keras **[5],** I have selected, precision, recall and f1-score as other measure to look at. I have followed a VGGNet model like architecture; which has more layers which allows it to identify low level/local features more accurately, as it achieved high accuracy in 2014 competition when it was first introduced **[6],** but with modified version to best suit on our problem and data size. After further fine tuning, I have also added dropout to regularize the network and prevent overfitting **[8]**.

**Model training & evaluation**: Selecting right **loss function** (area of research Inn itself) & **optimizer** can boost the performance of model significantly. Since, cross_entropy is commonly used loss function in classification, I have decided to go wit it, but our problem is multilabel classification, in our case, it will be “**categorical_crossentropy**”. I have first started with using Grid search on my model architecture to find the best combination of parameters **[7]** which gives me highest accuracy. But since the combinations were too many, it would have taken a lot of time to execute, so I decided to separate combinations in 2 groups & ran grid searches separately on both groups iteratively. I found the best parameters as below,

```
{'activation': 'tanh', 'epochs': 120, 'init': 'uniform', 'optimizer': '
adam'} & {'pool_size': (2, 2), 'kernel_size': (3, 3), 'filters': 16, 'e
pochs': 120, 'dense_layer_sizes': [1024, 512]}
```

Using this combination of parameters, I have built a final model to train on our dataset which got the accuracy of **98.20%** on validation data. Below is the loss/accuracy graph of my sign type model.





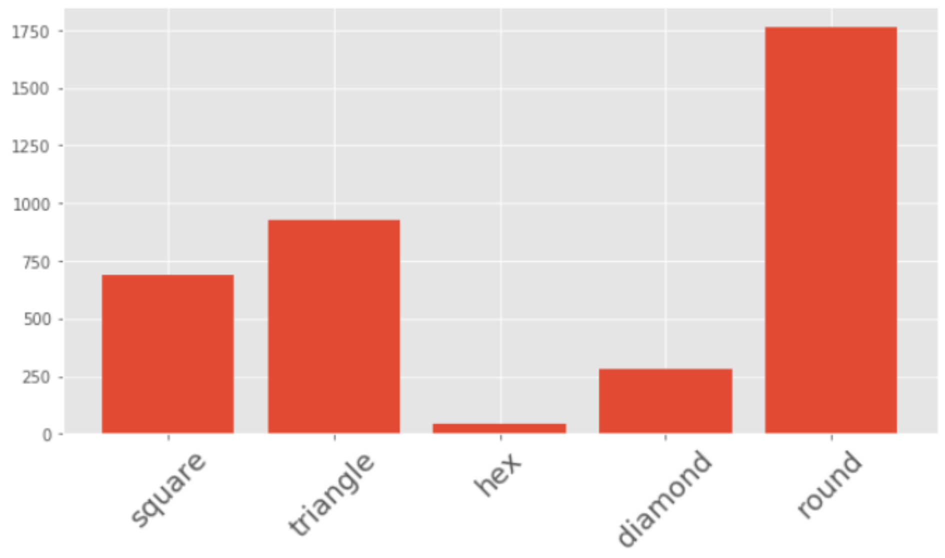
This model got accuracy, precision, recall and f1- score of **0.9789, 0.9786, 0.98, 0.9772** respe ctively on test data. As can be seen in classification report in Jupyter file, **each class** has **higher precision and recall** values, which **eliminates** our model’s **bias to predict high frequency** class more.

## **(CNN – Sign Shape): Steps followed for this model are as below** (in terms of changes in process from CNN - Sign Type)**:**

I have not used SVM again on sign shape, instead I decided to with CNN directly.

**Data loading:** is done same as earlier.

**Data summary & Exploration:** As earlier, I have visualized first image of each sign shape class to visually explore it. Which is as per **[II]** which has a label in form of "Label: Name (# of images for that label)". The bar chart again shows the same imbalance in classes of sign shape as well, so we will consider same approach as per sign type in terms of handling imbalanced class as below,



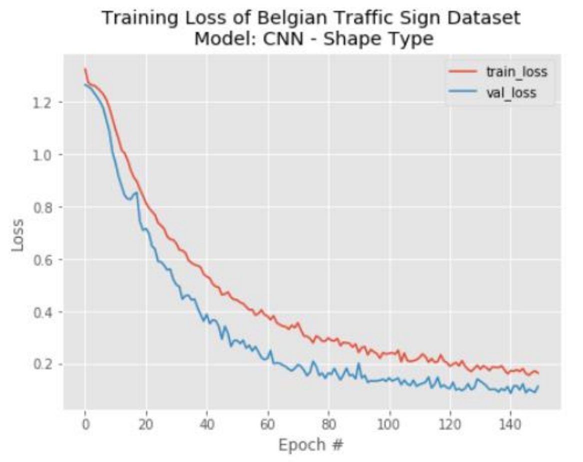
**Data Preprocessing:** Again, this was done same as sign type class, but with some different augmentation techniques, as said earlier, like **rotation,** which is set to **180** which also preserves shape as well as considers vertically flipped signs like **triangle,** which I have used in my **independent evaluation** to **make it harder for our model to predict** those external test data shapes.

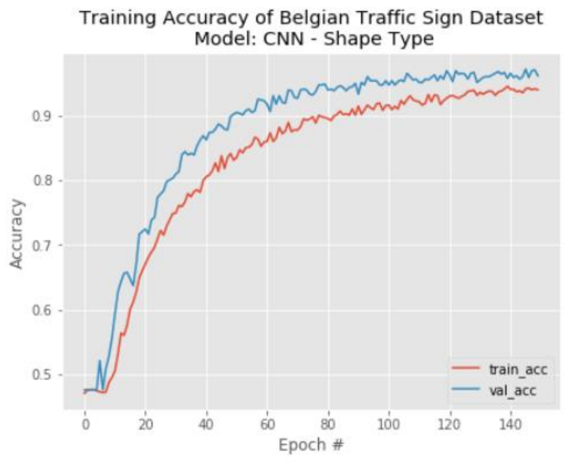
**Model Architecture**: At first, I have decided to use less layers than my sign type model, as sign shape has a smaller number of classes than sign type. But by doing so, it decreased the accuracy, so I decided to go with the same architecture, but not using the same parameters obtained by Grid Search in earlier steps.

**Model training & evaluation:** Since grid search was taking more time, I have used Randomized search for when combinations were more than 10 and grid search for second group of parameter combinations. By iteratives executing these two multiple times, I found the best parameters as below,

```
{'pool_size': 2, 'learn_rate': 0.0001, 'kernel_size': 3, 'filters': 16,
'epochs': 150, 'dense_layer_sizes': [1024, 512], 'activation': 'relu'}
& {'epochs': 150, 'init': 'uniform', 'optimizer': 'adam'}
```

By using these parameters, I have built a final sign shape model & evaluated it in test images. The loss/accuracy graph of model looks like below,





Now, this graph has validation accuracy more than train accuracy, but this could be because o f trained model is built with difficult training data with different augmentations (to make it robust) while validation was easy to predict. It was not because of the class imbalance and m odel predicting majority class more, as can be seen from classification report in Jupyter file, p recision, recall & **f1-score** are **higher for individual classes**.

This model got accuracy, precision, recall and f1- score of **0.9789, 0.9759, 0. 9759, 0. 9759** respectively on test data.

## (Independent Evaluation on Test Images):

Apart from provided train data, I have also done independent evaluation **on external test dat a** as well for **both sign type** & **sign shape** classification. The test data was gathered from co mbining DITS (Italian traffic data) **[9]**, GTSRB
(german traffic dataset) **[10]** & original Belgium traffic test data (from web) **[11]** and **labeled manually**.

**Challenges faced:** During the handling and gathering of this external data and combining it with model, a lot of challenges arise to make it predictable by our model. The dataset needed to be **labeled manually** because not every country has same label for every sign. Also, **Imag e formats** of the data were
differed. For example, Italian data was in **PNG** format, while German and Belgium data was i n **PPM** format. The **resolutions and aspect ratio** of these images are also different. All imag es were in **RGB** format. I have gathered all images together and then processed all of those to **change their formats to PNG, made them of 28x28 dimensions** and **changed to grayscale**.

This was the case for European countries only, had it been a different country, even the sign Types will be different, which is harder to make predictions on. The same thing I have done additionally, for sign shape in next section.

Then I used generators to load these images and made prediction on that.

The sample images for prediction on sign type and sign shape with correct labels are given in Jupyter file. For sign type, our model got an average f1-score of approx. **0.87** and for sign shape, average f1-score was **0.85**. So, it works well on European data which have almost similar traffic signs.

**(Additional Evaluation):** I have also done **additional evaluation** on **other countries**, with combining **Australian** and **Indian** images taken from **[12], [13]** & **[14]**, together to predict their sign shape.

As sign type of these countries are different, it is of no use to predict for them because trained model does not know these types.

The sample data is plotted in Jupyter file. As can be seen from it, it is trickier to deal with it. For ex. **Square sign border** is the **image border**, **additional things with traffic sig**n in an image, **Inverted triangles**, hex boundaries are at the image boundaries, whole **hex filled with same red color** etc.

Our Sign shape model has an **accuracy** of approx. **0.43** and **f1-score** of **0.503** on this dataset. This was **intentional** to **show the limitations of the model**. Because different countries have different ways of representing traffic sign types & sign shapes, it is **harder to generalize a model** based on **single country's data**.

## (Ultimate Judgement):

**In terms of accuracy:** Our CNN model gives **higher accuracy** on general data as well as distorted, blurry, or augmented data compared to other algorithm because of the way it works with local features.

**In terms of background knowledge & evaluation:** As per the various researches going on currently in computer vision field**[1], [6], SVM** and other algorithms can also gain **higher accuracy** but they requires **additional feature extraction and engineering**(histogram of oriented gradients, Linear discriminant Analysis, Iteratives Nearest neighbors-based Linear Projects etc.), while the architecture of CNN enables it to **automatically detect the features** on **various levels** and predict accurately, which makes it more generalizable and they scales well depending on additional layer added for new prediction types.

Of course, **when considering a large dataset** or **large number of classes** , this is not a perfect model, it can be fine-tuned further with SMOTE or over sampling or using actual large collection of images, use different number of layers/neurons in model, change weights initialization & optimization techniques to predict more accurately. But for **our use case** and **predicting Belgium traffic sign** based on sign type and sign shape, my **ultimate judgement** for the model to **use on real world settings** would be to use the **CNN model** that we have trained for both sign type & sign shape.

## References: -

[1] Radu Timofte*, Markus Mathias*, Rodrigo Benenson, and Luc Van Gool, Traffic Sign Recognition - How far are we from the solution?, International Joint Conference on Neural Networks (IJCNN 2013), August 2013, Dallas, USA. (* equal contributions)

[2] https://medium.com/x8-the-ai-community/solving-class-imbalance-problem-in-cnn-9c7a5231c478

[3] https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

[4] https://machinelearningmastery.com/image-augmentation-deep-learning-keras/

[5] https://keras.io/api/metrics/

[6] VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION, Karen Simonyan∗ & Andrew Zisserman+, Visual Geometry Group, Department of Engineering Science, University of Oxford

[7] **https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/**

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov - Dropout: A Simple Way to Prevent Neural Networks from Overfitting.

[9] **http://users.diag.uniroma1.it/bloisi/ds/dits.html**

[10] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In Proceedings of the IEEE International Joint Conference on Neural Networks, pages 1453–1460. 2011.

[11] Radu Timofte*, Markus Mathias*, Rodrigo Benenson, and Luc Van Gool, Traffic Sign Recognition - How far are we from the solution?, International Joint Conference on Neural Networks (IJCNN 2013), August 2013, Dallas, USA. (* equal contributions).

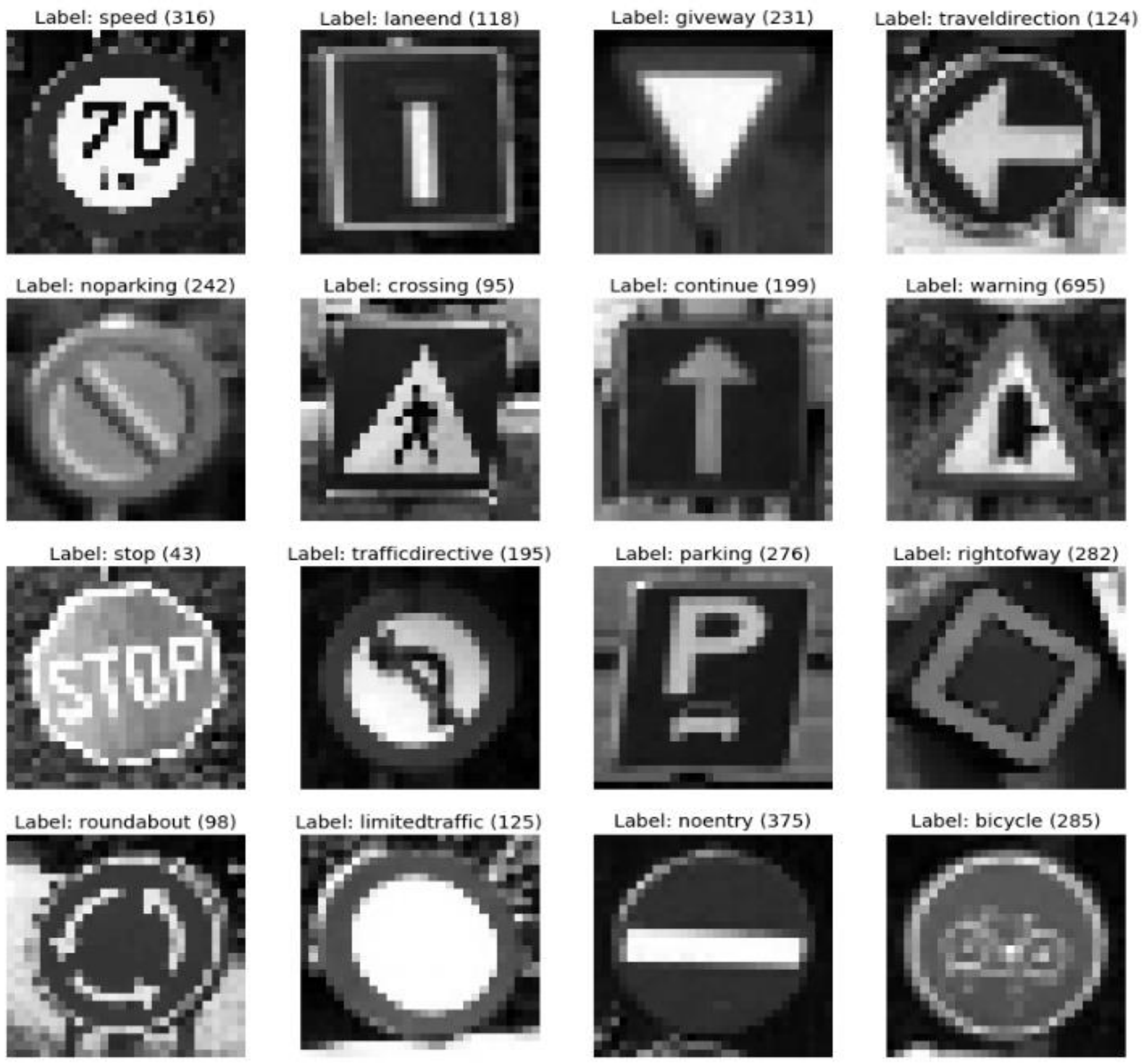
Obtained from - http://people.ee.ethz.ch/~timofter/traffic_signs/index.html

[12] **https://en.wikipedia.org/wiki/Road_signs_in_Australia**

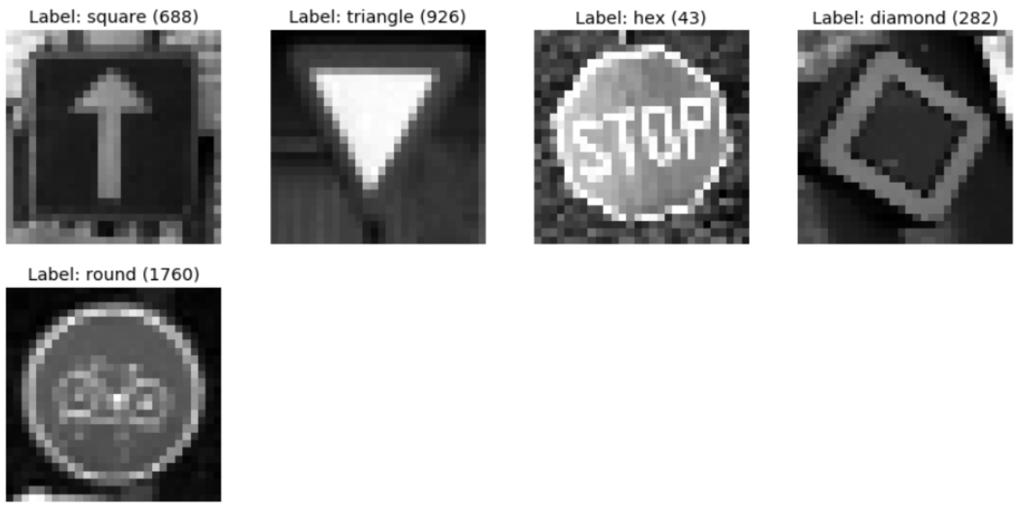
[13] **https://en.wikipedia.org/wiki/Road_signs_in_India**

[14] **https://www.pinterest.com.au/talentudumalpet/road-signs/**

## Appendix: -



I.



II.