PROJECT REPORT
CP 318: Data Science for Smart City Applications
Project 1: Indian Traffic Signs Prediction

- **Project statement:** Develop a machine learning model/framework for classifying various Indian traffic signs images.
- **Dataset:** (30 classes)
  - Training data: 932 data points with 2500 features (932 grayscale images of dimension 50*50)
  - Test data: 400 data points with 2500 features (400 grayscale images of dimension 50*50)

Some Random training data images (obtained from converting the feature vectors into images) are as follows:

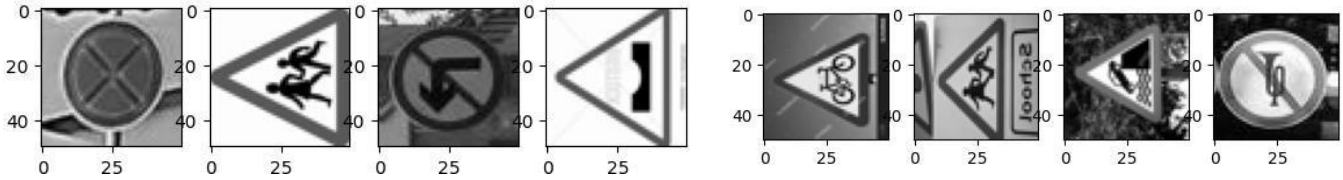


Fig.1: Grayscale images of training data It is well understood that it is a multi-class image classification problem.

- **Training Data Analysis:**

After analysing the training dataset properly, we found it to be unbalanced

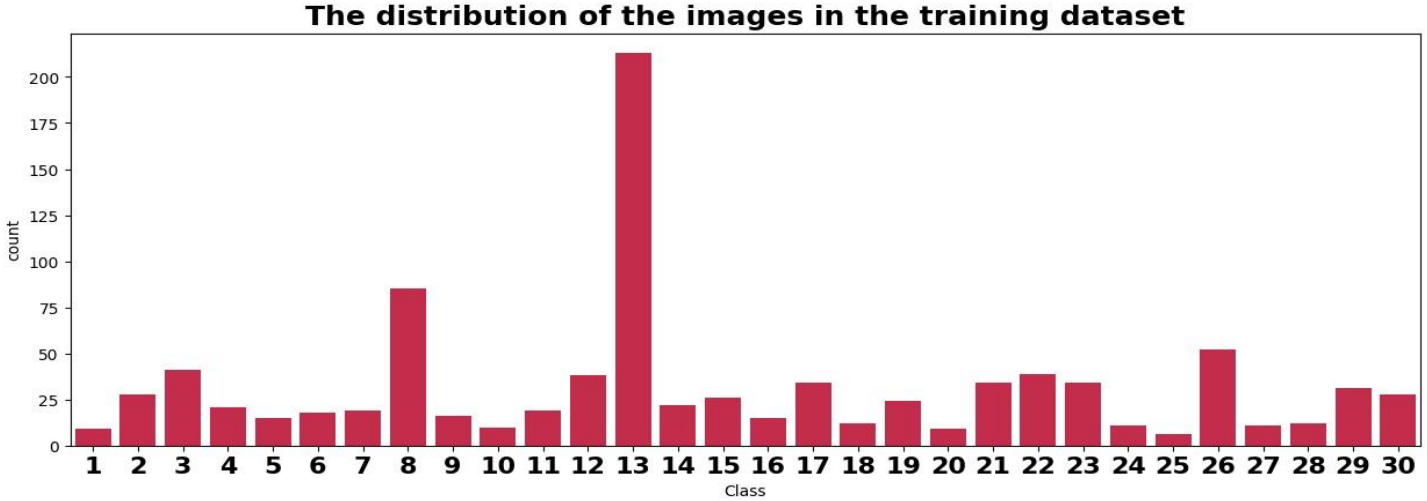
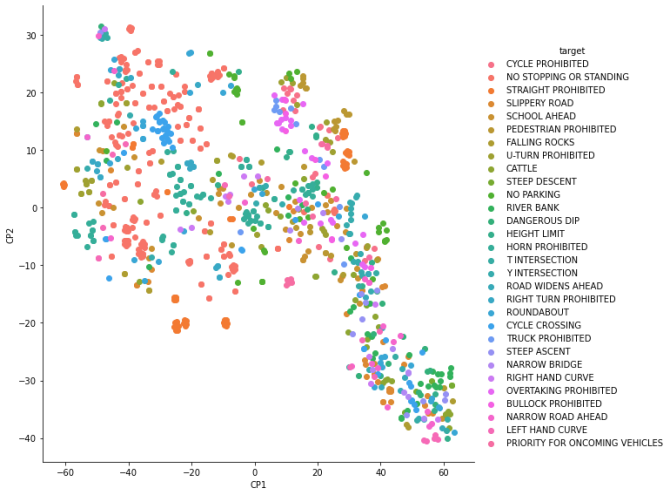


Fig.2: The Distribution of the images in the training dataset

We can infer from the above plot that Class-13 (NO STOPPING OR STANDING) had the maximum occurrence of about 22.85% and class-25 (STEEP DESCENT) had the lowest occurrence of about 0.6% in the training data.



We also plotted a t-SNE plot to analyse the separation among the different classes:

- t-distributed stochastic neighbour embedding (tSNE) is a statistical method for visualising highdimensional data by giving each data point a location in a two or three-dimensional map.
- It models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modelled by nearby points and dissimilar objects are modelled by distant points with high probability

From this plot we can infer that the multiple classes are overlapping and we cannot find a linearly separable boundary.

- **ML Models deployed for classification: -**

1) **Logistic regression**: This did not perform well because it is challenging for the model to extract the features by just looking at pixel values. The image pixels share a very complex and hierarchical relationship, and therefore it gets hard to deduce them just with a simple Logistic Regression.

2) **Decision Tree Classifier:** Decision trees as a classifier do not consider the local relationship between features, such as pixels in images which are close to each other; therefore, the results were not that great.

3) **SVM:** As expected, it was memory efficient but also did not perform well as the target classes were overlapping (more noise in data samples), and the dataset is imbalanced and the number of features for each of the data points exceeded the number of training data samples and these reasons caused it to underperform.

4) **KNN:** It is a distance-based algorithm and does not take the relationship between different features into account. However, imbalanced and overlapping datasets reduced the model's performance. Furthermore, due high dimensionality of images, the cost of distance computation between two points is very high

5) **Naive Bayes:** It gave quick but bad results which are because it assumes the features of the data to be independent which is not in the case with images, as features (nearby pixels) are very much correlated

As these basic classification models could not generalise well on images, we got motivated to use something that does not treat an image as a simple vector but can transform it into a form that is easier to process without losing critical features.
So, till now, we had faced two issues

- Classes are not linearly separable and overlapping
- Features are correlated

These problems motivated us to use something that can deal with these issues and capture critical information present in an image. CNN is one type of Model that deals with these issues very well.

- **Convolutional Neural Network (CNN):** We used several different architectures and observed that the model was very easily getting overfitted, so to tackle that we used data augmentation and synthesised new data using ImageDataGenerator in Keras

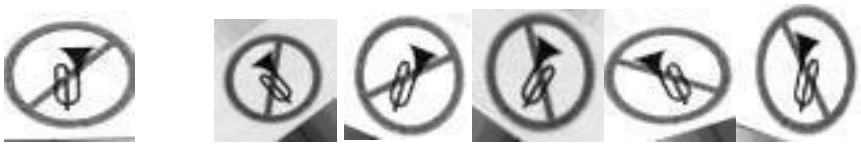


Fig.4: An example of 5 images generated from a single image (Data Augmentation:1 original + 5 generated)

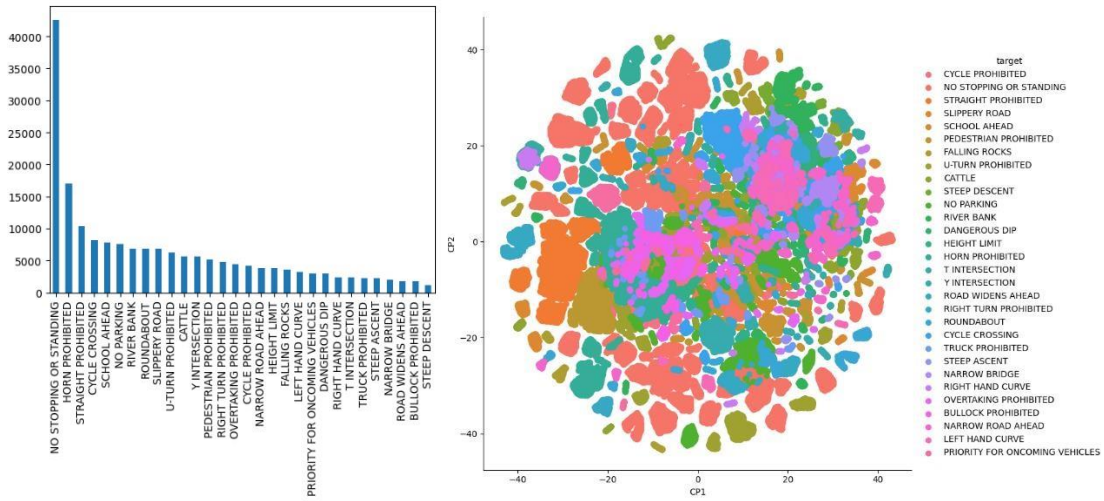
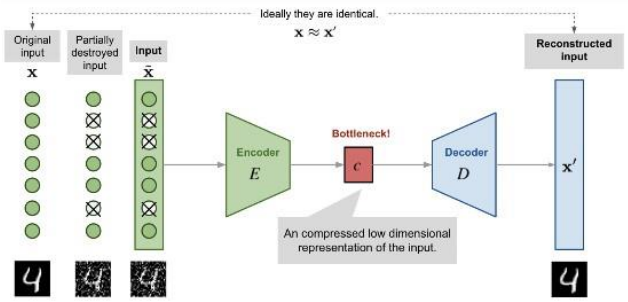


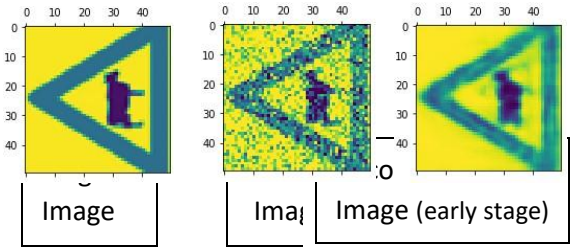
Fig.5: Class count bar graph and t-SNE plot for the new data set after augmentation

We tried to balance the data using oversampling during augmentation. Still, it resulted in lousy generalisation, so we finally decided to maintain each class's relative probability in the augmentation process and this resulted in above dataset.

By using the data augmentation trick, we reached 94.99% accuracy on naïve CNN.
We also noticed some noise in multiple samples of our training data and used **Denoising Autoencoder** to avoid this issue.

- **Denoising Autoencoder:**





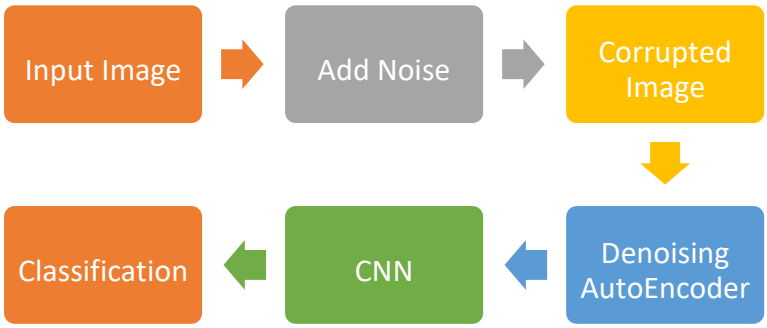
This is a type of Autoencoder which tries to learn only the important features of an images avoiding the noise, to train it we add some gaussian noise to original image and then pass it to the Autoencoder and try to generate back the original image and in this process of reconstruction the model learns to remove noise from the data

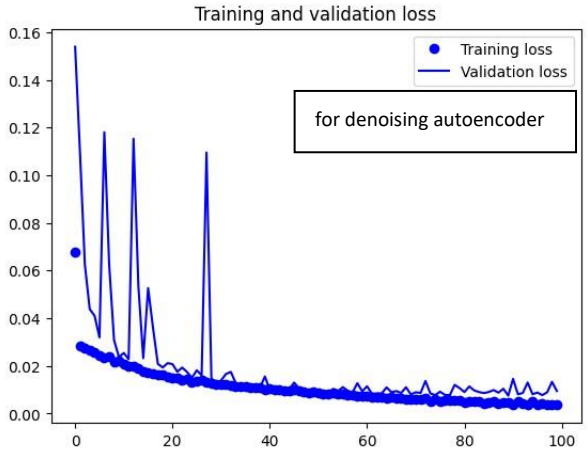
- **OUR FINAL MODEL:**

We used denoising autoencoder stacked with the CNN such that the denoised image that we get from the output of decoder is passed to CNN.

This approach of focusing more on extracting important features of the image worked well for this dataset

giving an accuracy of up to 97%





- **Disadvantages of our approach:**
  - The process of adding noise to an original image is stochastic so the model is not fully deterministic and therefore results can vary depending on noisy randomness introduced in the data
  - The model is computationally very expensive as it requires a lot of data and training of encoder and decoder network is time consuming process

- **Further Scope:**

Due to computational deficiency, we were not able to train it for longer time but it can lead to more good generalization if trained for longer time

- **References:**

- https://www.geeksforgeeks.org/python-data-augmentation/
- https://arxiv.org/pdf/2207.11771.pdf