Traffic Violation Detection System

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Motivation

The rise in **accidents** due to reckless driving led us to adopt technology, specifically **computer vision** and **machine learning**, for a Real-time Traffic Violation Detection System. Our aim is to quickly spot traffic violations using effective models like 3 layer **CNN**. This project stemmed from our mutual worry about road safety and the power of technology to make a difference. We developed this solution to match technological advancements with real safety issues in the world.

Literature Review

1. The paper on "[Automatic Traffic Red-Light Violation Detection Using AI]":

b. Detect and classification

Objective: The paper aims to develop a machine learning-based system for detecting traffic signal violations, particularly red-light violations, to address the growing number of road accidents caused by rule infractions.

Methodology: YOLO v5s recognize vehicles and optimize the system for accuracy using region-of-interest and vehicle location analysis.

Results: The system achieves high accuracy rates, with 82% for vehicle identification, 90% for detecting traffic signal status changes, and up to 86% for violation detection. Real-time results include detecting red-light violations, vehicle classification, and license plate extraction.

Literature Review

2. "Traffic Surveillance System and Criminal Detection Using Image Processing and Deep Learning":

Objective: The paper addresses the challenges of traffic violations and criminal activities on roads, aiming to enhance security and safety. It proposes a system that utilizes image processing and deep learning to intelligently identify traffic offenders by recognizing license plates, detect accidents, and identify criminals.

Methodology: The system employs RetinaNet, a one-stage object detection technique and OCR for plate reading.

Circuit and Experimental Results: The paper provides details of the system's circuit diagram, highlighting the integration of Arduino, GSM, and GPS modules. Experimental results demonstrate the effectiveness of the proposed system, including helmet detection, accident alerts, and criminal identification. RetinaNet is found to perform well, and the system shows promise for real-time applications.

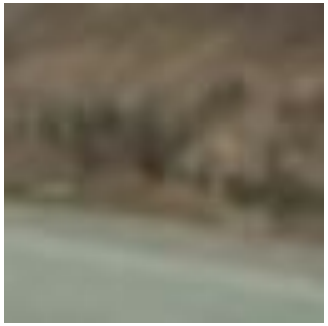
Dataset

Dataset used before mid-sem evaluation

Vehicles: https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip

Non-vehicles: https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip

The dataset consists of frame images that were taken from a section of CCTV video. These pictures show both automobiles and non-automobile objects. The photographs are displayed without any precise categorization and in the manner in which the CCTV system originally recorded them. The dataset's content consists of numerous scenes with both automobiles and non-vehicle objects in view.



Dataset

Dataset used after mid-sem evaluation

License plate: https://www.kaggle.com/datasets/sarthakvajpayee/ai-indian-license-plate-recognition-data/download?datasetVersionNumber=1

The dataset consists of images of numericals, alphabets (to be used for training models for identifying license plate number i.e. converting image data to a number which is predicted to be license number of that vehicle (cars in our case). Then we have sql data of registered license numbers in India.

Traffic signs: https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-preprocessed

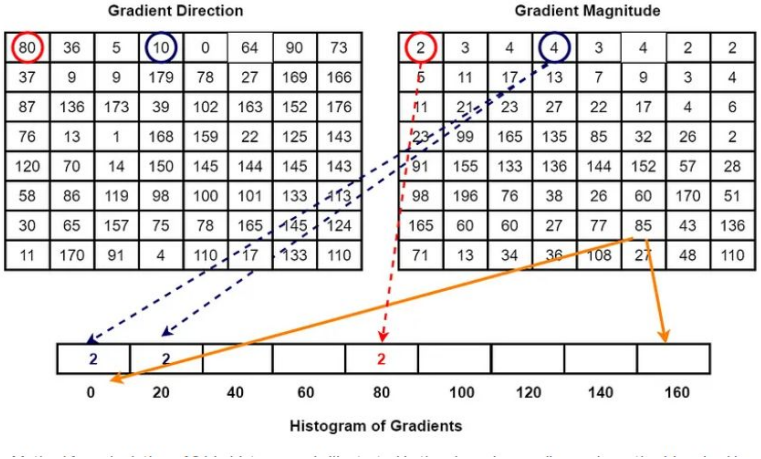
Pre - processed data for traffic signs.

Preprocessing of vehicles dataset

- **Histogram of Oriented Gradients (HOG)** is a key technique used for feature extraction from images. It works by capturing the distribution of gradients (edges) in different orientations within localized regions of an image.
    - Computing Gradient:

$$G_x(r,c) = I(r,c+1) - I(r,c-1) \quad G_y(r,c) = I(r-1,c) - I(r+1,c) \qquad Magnitude(\mu) = \sqrt{G_x^2 + G_y^2} \quad Angle(\theta) = |\tan^{-1}(G_y/G_x)|$$

where r, c refer to rows and columns respectively. (Image by author)



Reference

Flow of Processing

- **Color Conversion:** converts the input image from the RGB color space to the specified color space
  feature_image = cv2.cvtColor(image, cv2.COLOR_RGB2X)

- **Spatial Binning:** resizes the image to a smaller size and flattens it into a one-dimensional array.
  spatial_features=bin_spatial(feature_image,size=(32,32))

- **Color Histogram**: computes the histogram of color channels separately and concatenates them into a feature vector.
  hist_features=color_hist(feature_image,nbins=32)

- **Histogram of Oriented Gradients (HOG)**: features are extracted from a specific channel of the image using specified parameters like orientation, pixels per cell, and cells per block
  hog_features = get_hog_features(feature_image[:,:,hog_channel], orient, pix_per_cell, cell_per_block)

- **Feature Vector Extraction:** extracts spatial, color histogram, and HOG features from a single image based on specified parameters
  img_features = single_img_features(image, color_space, spatial_size, hist_bins, orient, pix_per_cell, cell_per_block, hog_channel)

- **Batch Feature Extraction:** iterates through a list of images and extracts features for each image, creating a feature matrix
  features = extract_features(imgs, color_space, spatial_size, hist_bins, orient, pix_per_cell, cell_per_block, hog_channel)

Methodology tested before mid-sem evaluation

- We use the **Linear Support Vector Machine (SVM)** and **Logistic Regression** machine learning classifiers to identify vehicles.

- **SVM** is trained using a dataset that includes features taken from car and non-car photos for vehicle detection. Once trained, it can quickly categorise fresh photos, identifying whether or not a vehicle is there. The LinearSVC class from scikit-learn is used in the code to enable the SVM to learn and predict the presence of vehicles in the supplied photos.

- **Regression** uses a logistic function to describe the likelihood that an input belongs to the positive class (vehicle). The input is categorised as a positive example if this probability is higher than a predetermined threshold, which is commonly 0.5; otherwise, it is categorised as a negative example.

Methodology tested after mid-sem evaluation

We use the **CNN** model for training the dataset of vehicles, as they are theoretically more reliable for classification based prediction.

- Model Architecture:
  - The model starts with a dense layer of 512 neurons using ReLU activation.
  - A dropout layer is added to prevent overfitting by randomly dropping 50% of the neurons during training.
  - Another dense layer with 128 neurons and ReLU activation follows.
  - Finally, a dense layer with a single neuron and sigmoid activation is added for binary classification.
- Compilation:
  - The model is compiled using the Adam optimizer and binary crossentropy loss, which is suitable for binary classification tasks.
- Training:
  - The model is trained on the training data (X_train and y_train) for 10 epochs with a batch size of 32.
- Evaluation:
  - The test set (X_test and y_test) is used to evaluate the model's performance.
  - Test loss and accuracy are printed.
- Prediction and F1 Score:
  - The model predicts labels for the test set, and predictions are thresholded at 0.5.
  - F1 score is computed using the predicted and true labels.

Methodology tested after mid-sem evaluation

Model for training the dataset of License plate

- pre-trained Haar Cascade classifier (plate_cascade) to detect license plates.
- The segment_characters function takes a detected license plate image, processes it to enhance character visibility, and then uses contour detection to segment individual characters.
- CNN (convolutional layer, max pooling, dropout, and dense layers) function of Keras library
- The model is compiled using the sparse categorical cross-entropy loss function and the Adam optimizer with a specified learning rate.
- Custom F1 score functions (f1score and custom_f1score) are defined for model evaluation.
- The CNN model is trained using the fit_generator method with training and validation data generators. Training stops if the validation custom F1 score exceeds 0.99.

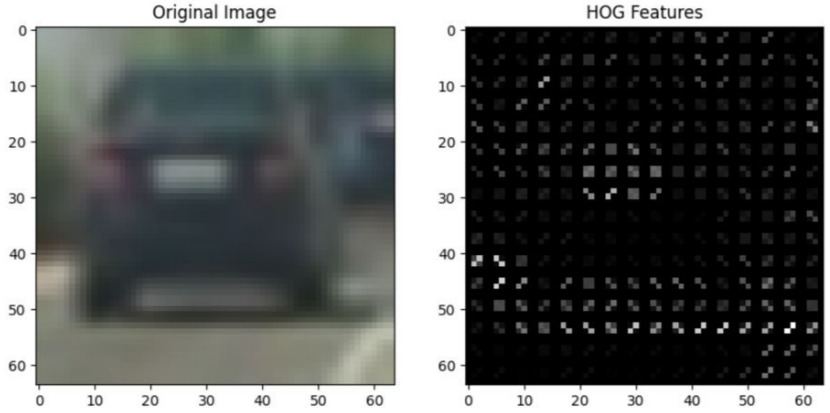
We use the **CNN** model for training the pre-processed dataset of Sign Recognition

Model Architecture:

- Creates a Sequential model using Keras.
- Adds a 2D convolutional layer with 32 filters, a kernel size of 3x3, 'same' padding, and ReLU activation.
- Adds a 2D max pooling layer with a pool size of 2x2.
- Flattens the output to a 1D array.
- Adds a dense layer with 500 units and ReLU activation.
- Adds the output layer with 43 units (assuming it's a classification task) and softmax activation.
- Compiles the model using the Adam optimizer and categorical cross-entropy loss.

Results & Analysis for SVM and Regression

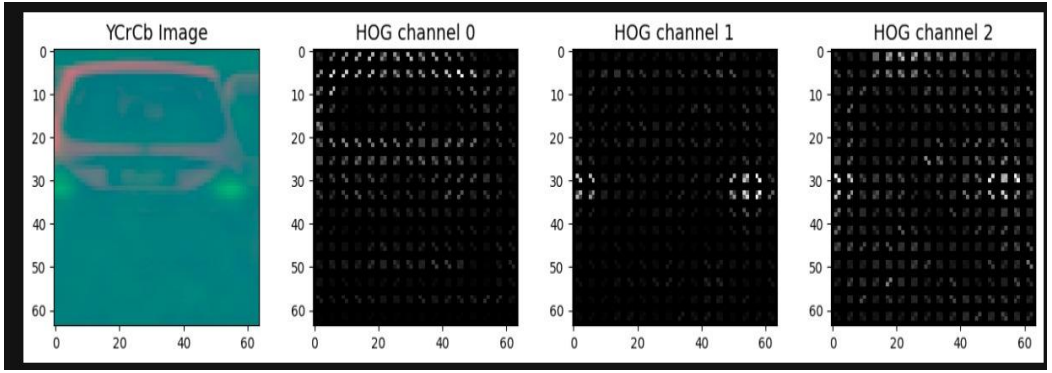
```
5.93 Seconds to train SVC...
Test Accuracy of SVC =  0.6962
F1 Score of Linear SVC = 0.6833
0.79 Seconds to train Logistic Regression...
Test Accuracy of Logistic Regression =  0.6124
F1 Score of Logistic = 0.6044
Classifier parameters were saved to file
```



- LR accuracy = 0.6044
- SVC accuracy = 0.6962
- SVC has better accuracy than LR

- The HOG descriptor will capture information about the edges and gradients in different parts of the image, emphasizing features that are characteristic of the object (vehicle)

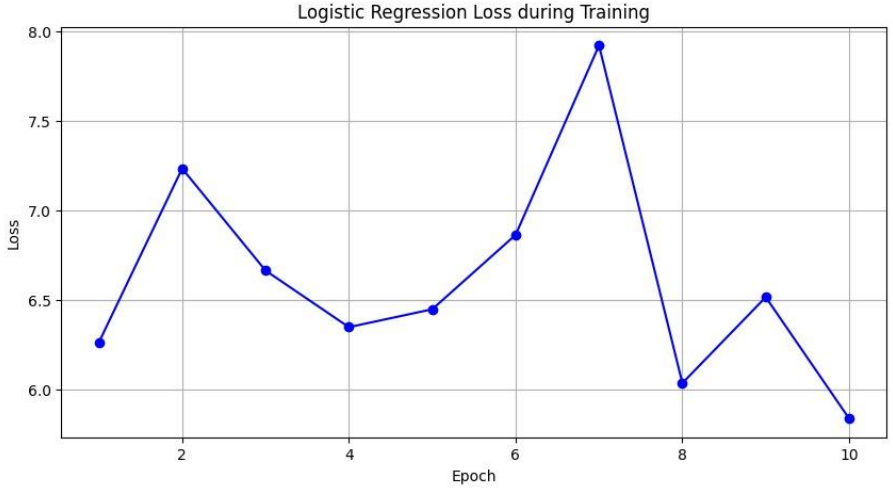
Results & Analysis for SVM and Regression

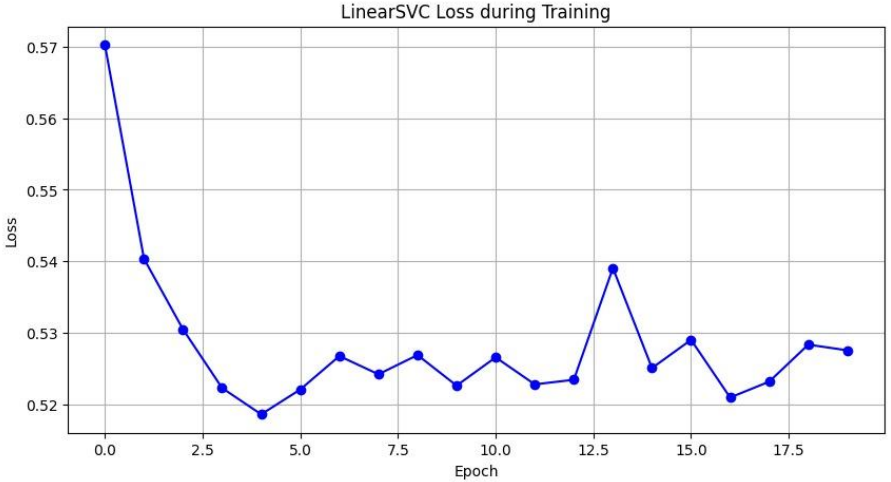


- For color images, each color channel can be treated independently, and HOG features can be computed for each channel separately. This means that for an RGB image, you would compute HOG features for the Red, Green, and Blue channels separately.

- Computing HOG features independently for each color channel in a color image provides additional information about the spatial distribution of gradients and edges in different color channels. This can be beneficial in various computer vision tasks.

Results & Analysis for SVM and Regression



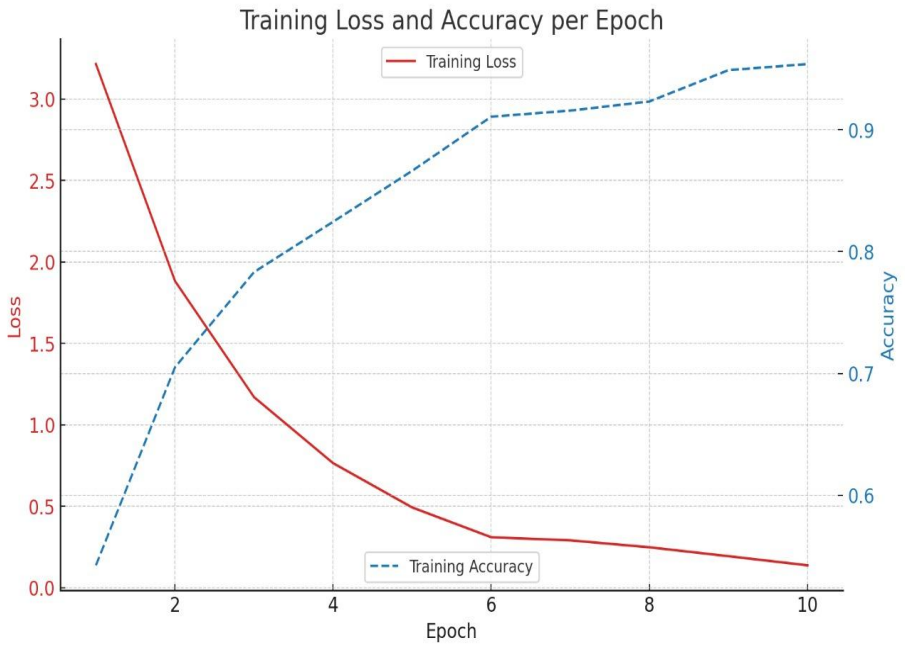


Loss in Linear Regression is increasing may be due to noisy data and model's complexity may be inadequate to capture the underlying patterns

Small variations in individual data points affect the model's decision boundary and causes slight changes in loss in Linear SVC, dataset may be noisy.

Results & Analysis for CNN of Vehicles and license detection

```
Epoch 1/10
50/50 [==============================] - 9s 158ms/step - loss: 3.2139 - accuracy: 0.5425
Epoch 2/10
50/50 [==============================] - 8s 158ms/step - loss: 1.8827 - accuracy: 0.7050
Epoch 3/10
50/50 [==============================] - 8s 160ms/step - loss: 1.1701 - accuracy: 0.7831
Epoch 4/10
50/50 [==============================] - 8s 152ms/step - loss: 0.7649 - accuracy: 0.8244
Epoch 5/10
50/50 [==============================] - 8s 153ms/step - loss: 0.4932 - accuracy: 0.8662
Epoch 6/10
50/50 [==============================] - 8s 153ms/step - loss: 0.3102 - accuracy: 0.9106
Epoch 7/10
50/50 [==============================] - 8s 154ms/step - loss: 0.2913 - accuracy: 0.9156
Epoch 8/10
50/50 [==============================] - 8s 154ms/step - loss: 0.2487 - accuracy: 0.9231
Epoch 9/10
50/50 [==============================] - 8s 157ms/step - loss: 0.1940 - accuracy: 0.9488
Epoch 10/10
50/50 [==============================] - 8s 158ms/step - loss: 0.1381 - accuracy: 0.9538
Time to train DNN: 79.14 seconds
13/13 [==============================] - 0s 12ms/step - loss: 0.5322 - accuracy: 0.8500
Test Loss for DNN: 0.5322
Test Accuracy for DNN: 0.8500
13/13 [==============================] - 0s 11ms/step
F1 Score for DNN: 0.8592
```

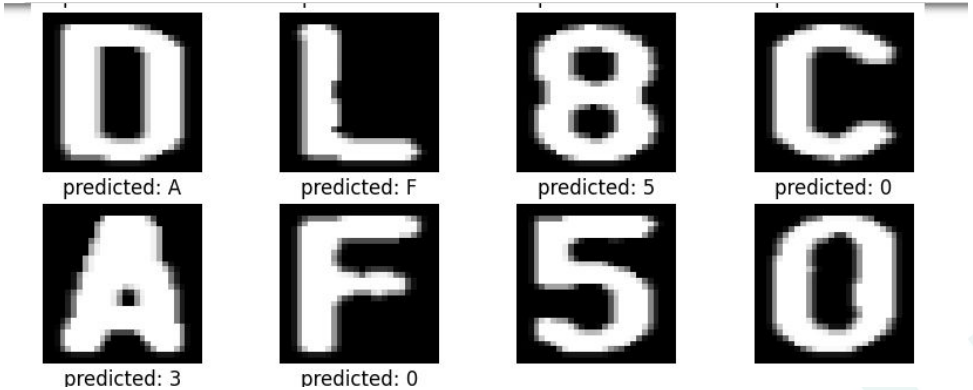


Results & Analysis for CNN of Vehicles and license detection

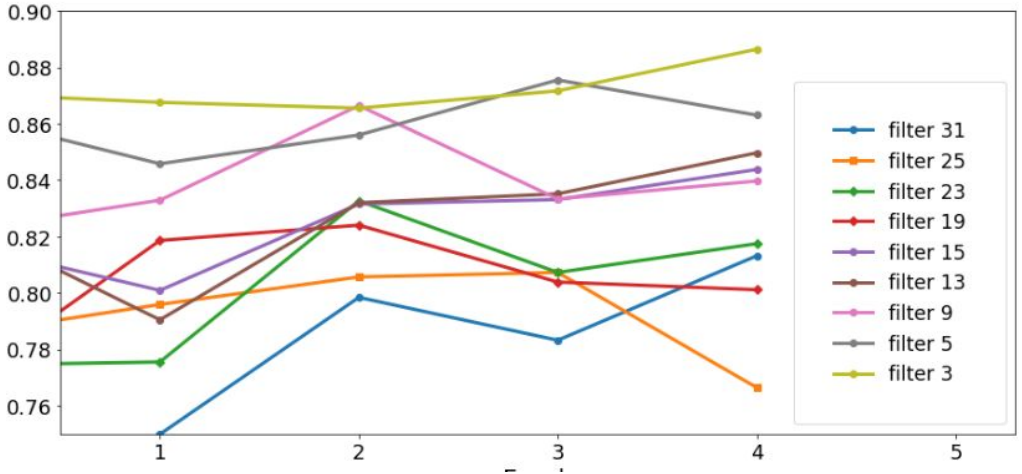
```
Epoch 16/80
864/864 [==============================] - 22s 26ms/step - loss: 0.0966 - custom_f1score: 0.9688 - val_loss: 0.0756
- val_custom_f1score: 0.9554
Epoch 17/80
864/864 [==============================] - 18s 21ms/step - loss: 0.0956 - custom_f1score: 0.9664 - val_loss: 0.2206
- val_custom_f1score: 0.9375
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	23248
conv2d_1 (Conv2D)	(None, 28, 28, 32)	131104
conv2d_2 (Conv2D)	(None, 28, 28, 64)	131136
conv2d_3 (Conv2D)	(None, 28, 28, 64)	65600
max_pooling2d (MaxPooling2D)	(None, 7, 7, 64)	0
dropout (Dropout)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 128)	401536
dense_1 (Dense)	(None, 36)	4644

Total params: 757,268
Trainable params: 757,268
Non-trainable params: 0



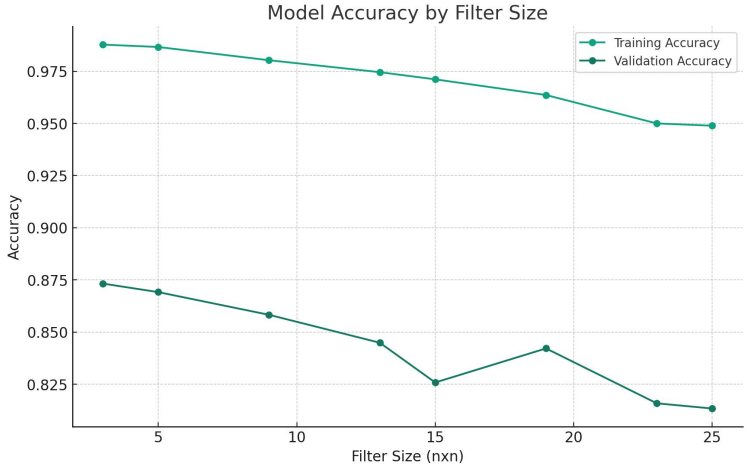
Results & Analysis for CNN of traffic signs



Output on terminal for this:

```
ClassId: 3
Label: Speed limit
```

Results & Analysis for CNN of traffic signs




```
Model with filters 3x3, epochs=5, training accuracy=0.98776, validation accuracy=0.87324
Model with filters 5x5, epochs=5, training accuracy=0.98661, validation accuracy=0.86916
Model with filters 9x9, epochs=5, training accuracy=0.98027, validation accuracy=0.85828
Model with filters 13x13, epochs=5, training accuracy=0.97454, validation accuracy=0.84490
Model with filters 15x15, epochs=5, training accuracy=0.97111, validation accuracy=0.82585
Model with filters 19x19, epochs=5, training accuracy=0.96360, validation accuracy=0.84218
Model with filters 23x23, epochs=5, training accuracy=0.94998, validation accuracy=0.81587
Model with filters 25x25, epochs=5, training accuracy=0.94891, validation accuracy=0.81338
```

Future Scope

- We have models for vehicle detection, identifying license plate numbers for those images. And another model for identifying traffic signs
- We can create a pipeline in future where we first received footages of camera, break that into frames and then apply the traffic sign model to know constraints on vehicles for the given scenario
- Then, apply vehicle detection and license plate models on culprit frames to identify violations.

Timeline after midem evaluation

16 Oct - 21Oct: Dataset review of license plate and its pre-processing
21 Oct - 31 Oct: CNN on previous datasets
1Nov - 10Nov: CNN for license detection
10 Nov - 20 Nov: CNN for traffic signs
20 Nov - 22 Nov: Model Testing and evaluation

Contribution

- Project Planning: Lakshya Sharan, Madhur, Ankush, Anubhav
- Data preprocessing: Lakshya Sharan, Madhur
- Model Creation and development: Ankush, Anubhav, Lakshya Sharan, Madhur
- Performance metrics: Anubhav, Ankush
- Documentation: Lakshya Sharan, Madhur, Ankush, Anubhav