# Live identification of vehicles with fake number plates using Machine Learning Algorithms

Regional Remote Sensing Center, West

Team members:

- Kshitij Upadhyay
- Akshat Agrawal
- Aman Sharma

## Introduction to the problem statement

In India and most parts of the world, it is mandatory for all vehicles running in public places to have a number plate with their unique registration number. Unfortunately, number plates are often stolen or faked by people trying to conceal criminal activity. A system that can identify fake number plates on vehicles by processing the data from police cameras can be of great help for law enforcement in dealing with crime.
The system has to be able to read the number plate on a vehicle. The details of the vehicle with that number plate have to be obtained from the m-Parivahan API. At the same time, the system has to recognize the color and model of the vehicle. The details have to be matched, and an alarm has to be generated in case of discrepancy.
The system has to be fast enough to process data in real-time. In addition, since the amount of data to be processed is massive, accuracy must be reasonably high to avoid frequent false positives.

## Abstract

We read number plates using Automatic number-plate recognition (ANPR). ANPR uses a series of image manipulation techniques to detect the number plate and then optical character recognition (OCR) to extract the alphanumeric of the license plate. We used a combination of TensorFlow Object Detection with OpenCV to detect the number plates and PyTorch and EasyOCR or pytesseract to extract the text out of it. We checked the database of the RTO to get the details of the vehicle associated with the number extracted from the number plate. Finally, we used transfer learning to predict the model of the car, where a pre-trained Convolutional Neural Network, VGG-16 from the Keras Application, is used to solve the problem under consideration.

We will compare the two results, and in case of a mismatch, alert the nearby police station or headquarters for every mismatch.[Fig1]

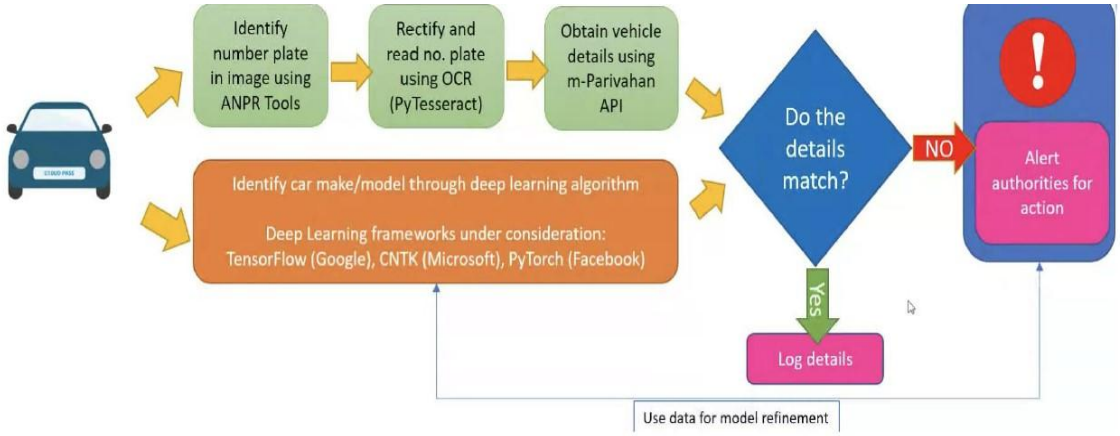


Fig 1: Methodology

## Progress up to Mid Semester

We have gone through Youtube tutorials to learn about Tensorflow and its usage. We have also developed a hand gesture recognition model to implement the learning on a less demanding problem in terms of the training set. [Fig2] It recognizes the ‘thumbs up’ and ‘thumbs down’ gestures.

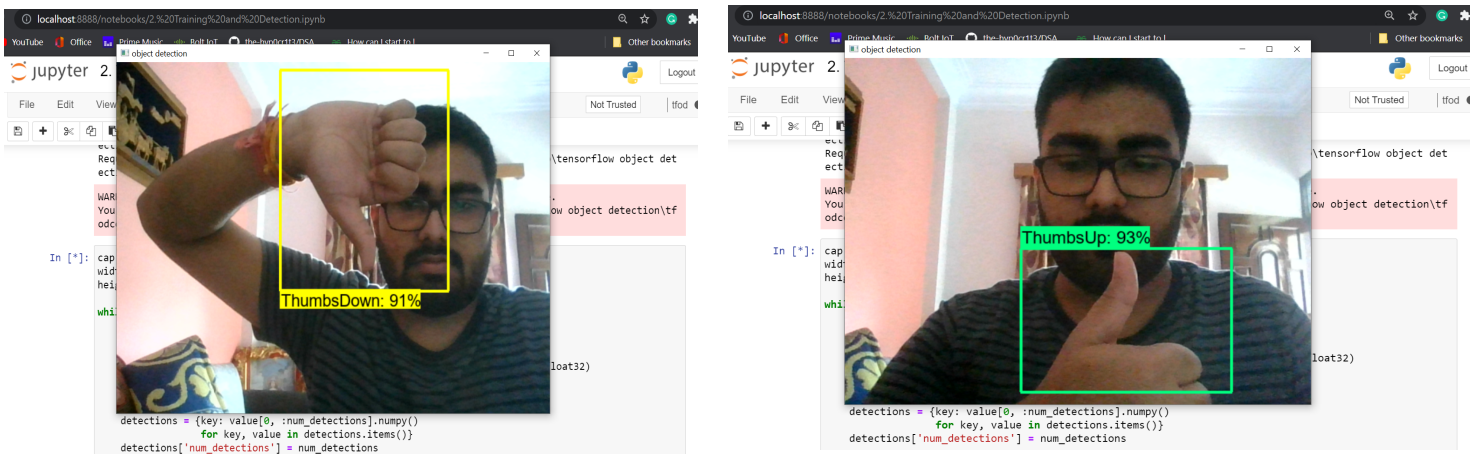


Fig 2: hand gesture prediction model

Later on, we have also extended the learning to the actual problem we have to solve. We have made a model that predicts the exact location of the number plates from the given image. We used the online available Kaggle dataset [1]. Attached are the pictures of the performance of the model on test images.[Fig 3]

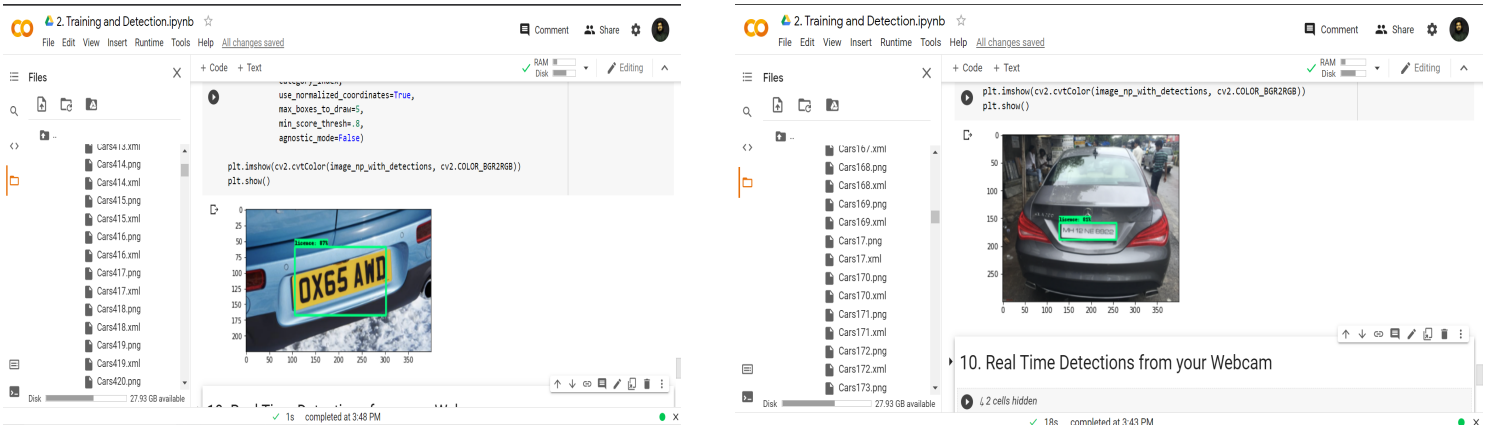


Fig 3: Detecting number plates on test images

## Plans for the remaining four weeks

We have started with an object detection course where we detected hand gestures (live, when using Jupyter Notebook, and with images when using Google Colaboratory), then used that model to detect where the number plate is in the image. Next, we will extract text/license numbers from the number plate using Optical Character Recognition(OCR). Finally, after successfully pulling the license number, we will recognize the car's model and then match the license data with the RTO data using m-Parivahan API.
After we have all the data, we will report the mismatch to the corresponding stations.
If time permits, we also plan to develop a website where people can upload their car's image and verify with our model, giving better results and avoiding their hassle due to an error that misclassification by the model might cause.

# Implementation

### Number Plate Detection

We used a TensorFlow Object Detection (Tensorflow Model Zoo) to detect the number plates and PyTorch and EasyOCR to extract the text out of it. An issue with this approach was that the model could not recognize the edges of the number plate and left out some parts of the number plate in the final image.

To overcome this challenge, we tried to extract the number plate from the image using OpenCV. Using OpenCV, we extracted a contour map of the image. Then we extracted the coordinates of the most prominent rectangle present in the contour map. Using the coordinates of the rectangle, we crop out the image of the number plate.


Fig 4: Original image

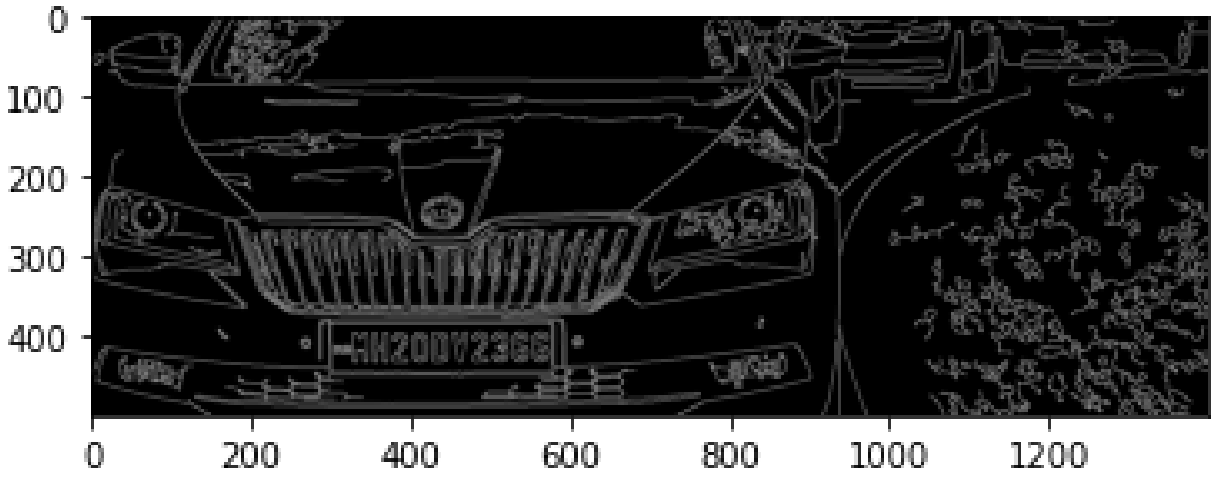


Fig 5: Contour map


Fig 6: The extracted image of number plate

We tried using EasyOCR, an OCR tool, to extract text from the image of the number plate. The results from EasyOCR were not accurate. It often misclassified characters like ‘2’ as ‘Z’, ‘W’ as ‘M’ and ‘0’ as ‘O’. On the recommendation of our mentor, we tried using Pytesseract, which is a wrapper for Google’s Tesseract OCR engine, instead of EasyOCR to extract text. The results from Pytesseract were more accurate than EasyOCR.


Fig 7(a): Text result using EasyOCR


Fig 7(b): Text result using pytesseract

## Image Scraping

Since there was no dataset available on the internet, we used data scraping scripts using BeautifulSoup4 and python. We could download only 80 images at a time due to the limit set by the browser. To tackle this problem, we ran the scripts with similar queries. For example, to collect 'model A' images, the queries were 'Model A India, 'Model A front view, 'Model A used'. In this manner, along with some manual removal of duplicate images, we collected around 600 images for each model. We worked on three models, namely Nissan Micra, Ford Ecosport, and Suzuki Baleno. We selected the models keeping in mind many factors. Firstly, all of these models have only one version, or even in different versions, the build has not changed a lot. Secondly, Baleno and Micra are both in the hatchback segment, so it would test the model's ability to classify car models in the same segment. We have not restricted ourselves to specific colors; each model has images of cars in all the company's colors.

## Car Model Detection

In the third part, which consisted of predicting the model of the car, we used a Convolutional Neural Network (CNN).

Convolutional Neural Network is a deep learning algorithm that takes an image as an input, and then weights and bias are trained on various objects, features in the image. It uses filters in stages to reduce the image size and combine similar features.

Convolutional Neural Networks use convolution layers that use a filter, which helps in recognizing essential features.

We used Keras Application's VGG-16 model, which has pre-trained weights of the neurons in the neural network and 138,357,544 parameters. We used the same weights except for the input and the output layer. However, from this depth, the model derives its state of the art 92.4% precision on the ImageNet dataset.

We trained the model on Ford Ecosport, Maruti Suzuki Baleno, and Nissan Micra and achieved an accuracy of 73.75 percent.

We converted our image dataset into the dimensions of the ImageNet input size. We then used softmax activation to classify the image and determine which class the object belongs to.

The model is scalable, and we can extend it to classify any number of car models in the future. We just have to add test and train sets in google drive and run the program, and we will get the probabilities of class in an array to which the car belongs.

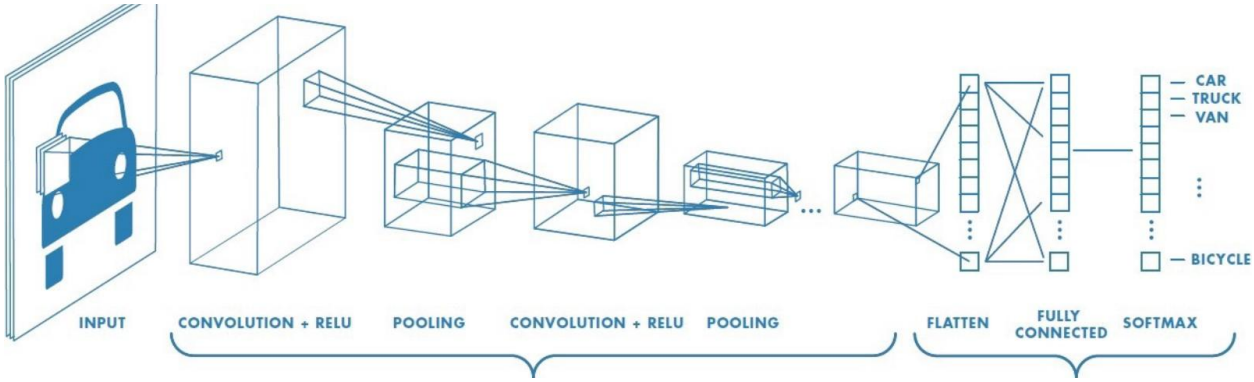


Fig 8: Convolutional Neural Network Architecture

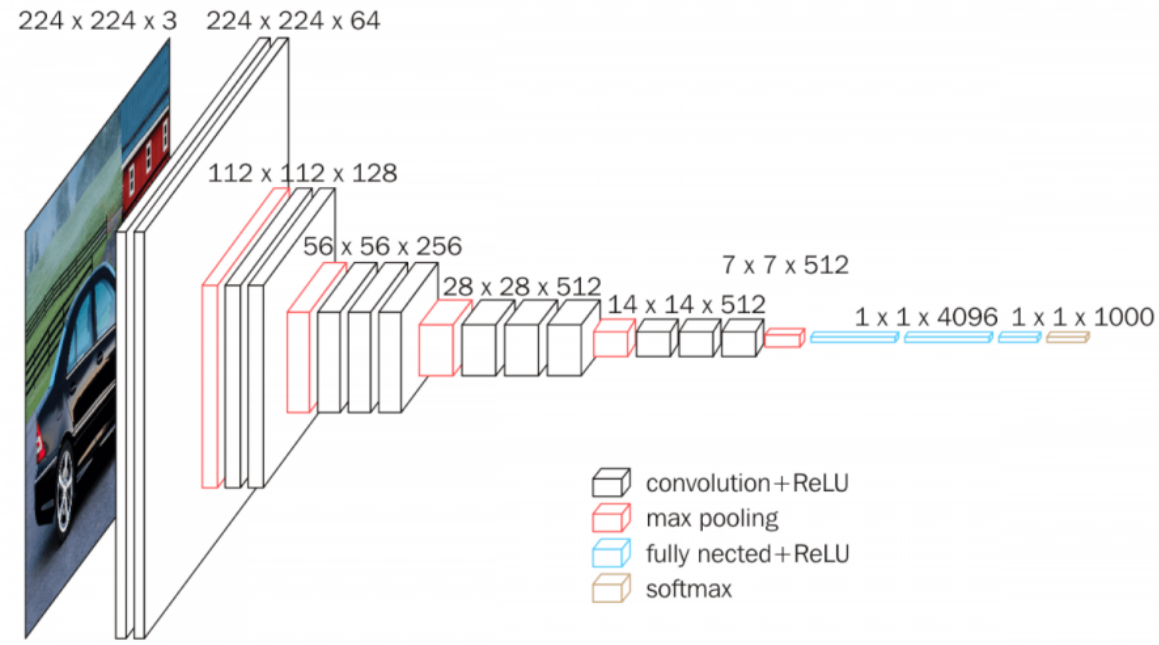


Fig 9: VGG-16 Model Architecture

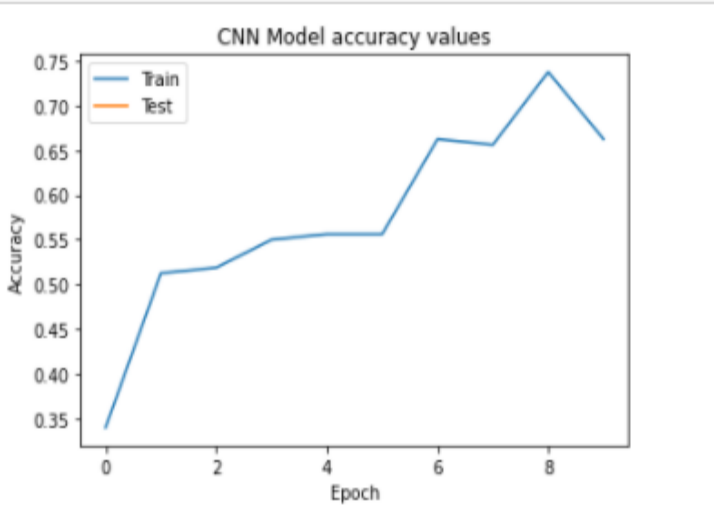


Fig 10(a): CNN model accuracy iterations

```
Epoch 00001: val_loss improved from inf to 6.09521, saving model to mymodel.h5
Epoch 2/10
5/5 - 43s - loss: 6.4166 - accuracy: 0.5125
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 3/10
5/5 - 36s - loss: 6.2784 - accuracy: 0.5188
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 4/10
5/5 - 33s - loss: 5.0142 - accuracy: 0.5500
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 5/10
5/5 - 31s - loss: 4.5178 - accuracy: 0.5562
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 6/10
5/5 - 23s - loss: 4.2477 - accuracy: 0.5562
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 7/10
5/5 - 24s - loss: 3.7797 - accuracy: 0.6625
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 8/10
5/5 - 22s - loss: 3.0715 - accuracy: 0.6562
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 9/10
5/5 - 17s - loss: 2.5442 - accuracy: 0.7375
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 10/10
5/5 - 13s - loss: 3.0993 - accuracy: 0.6625
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Training completed in time:  0:08:03.473215
```

Fig 10 (b): CNN model training

```
In [23]: img_path=('/content/drive/MyDrive/Dataset/TEST/BAL-TEST/baleno11.jpg')
         img = image.load_img(img_path, target_size=(224, 224))
         img_array = image.img_to_array(img)
         img_batch = np.expand_dims(img_array, axis=0)
         img_preprocessed = preprocess_input(img_batch)
         prediction = model.predict(img_preprocessed)
         prediction

Out[23]: array([[1., 0., 0.]], dtype=float32)

In [22]: img_path=('/content/drive/MyDrive/Dataset/TEST/ECO-TEST/ecosport11.jpg')
         img = image.load_img(img_path, target_size=(224, 224))
         img_array = image.img_to_array(img)
         img_batch = np.expand_dims(img_array, axis=0)
         img_preprocessed = preprocess_input(img_batch)
         prediction = model.predict(img_preprocessed)
         prediction

Out[22]: array([[0.00265376, 0.9960194 , 0.00132687]], dtype=float32)

In [26]: img_path=('/content/drive/MyDrive/Dataset/TEST/MICRA-TEST/micra12.jpg')
         img = image.load_img(img_path, target_size=(224, 224))
         img_array = image.img_to_array(img)
         img_batch = np.expand_dims(img_array, axis=0)
         img_preprocessed = preprocess_input(img_batch)
         prediction = model.predict(img_preprocessed)
         prediction

Out[26]: array([[2.3450502e-14, 2.5967483e-26, 1.0000000e+00]], dtype=float32)
```

Fig 11: Testing CNN model and getting results in an array

## Challenges

The biggest challenge we faced has been to train our model. We do not have the latest GPUs in our laptops, and if we are trying to train it locally, it takes about 4 hours to train the model with 10000 steps.
Using online platforms was the option we have restored to, and the results mentioned above are run on Google Colaboratory. However, they come with their restrictions. For example, we cannot test our model on the live feed in these platforms, which fails the project's purpose. Moreover, these platforms have their time limits and clear the progress once the boundary is crossed.

## Limitations

Challenges that the model might face:

- Non-standard number plates.
- Vehicles with damaged/modified bodies might be misclassified

Weather and poor lighting conditions might substantially reduce the quality of the camera feed, making it difficult for our model to classify correctly.

- Whenever a new car model is launched, the model has to be trained again with its data.

## Scope of Improvement

- Four-point perspective transformation can be used before extracting the text out of number plates for better accuracy.
- An allowlist of characters can be made, and an Algorithmic layer ahead of the OCR can be used to filter undesirable characters.
- The number of test and train images can be increased to improve accuracy.

## References


1. https://www.kaggle.com/andrewmvd/car-plate-detection
2. Tensorflow 2.0 crash course
3. https://levelup.gitconnected.com/a-beginners-guide-to-tesseract-ocr-using-pytesseract-23036f5b2211
4. https://www.coursera.org/learn/machine-learning
5. https://keras.io/api/applications