# VEHICLE CLASSIFICATION
## And License Plate Recognition

**CS771A Course Project :**
**Under Prof. Harish Karnick**

Amlan Kar
Nishant Rai
Sandipan Mandal
Sourav Anand
Group 26
Indian Institute of Technology Kanpur

# Introduction

**Project Aim:**

Detect and Classify relevant objects in the video. In case of 4-Wheelers localise and read the license plate.

**Steps Involved:**

- Pre-Processing Video Stream
- Detection of Objects
- Object Classification
- Localising License Plate
- Recognizing Text (OCR)

## Steps Involved (Detailed)

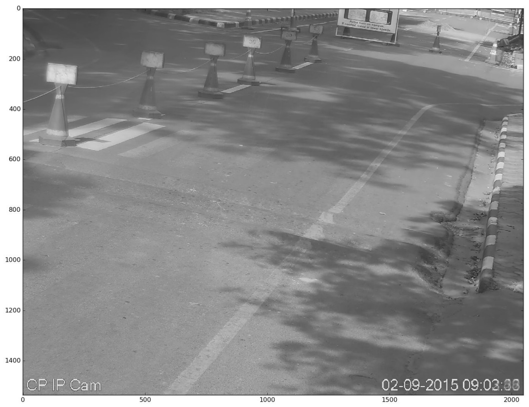
The detailed steps involved are as follows (The later slides discuss the topics in detail),

- Background detection
- Background Subtraction
- Identifying Objects
- Tracking Objects
- Object Classification
- Number Plate Localization
- OCR Text Detection

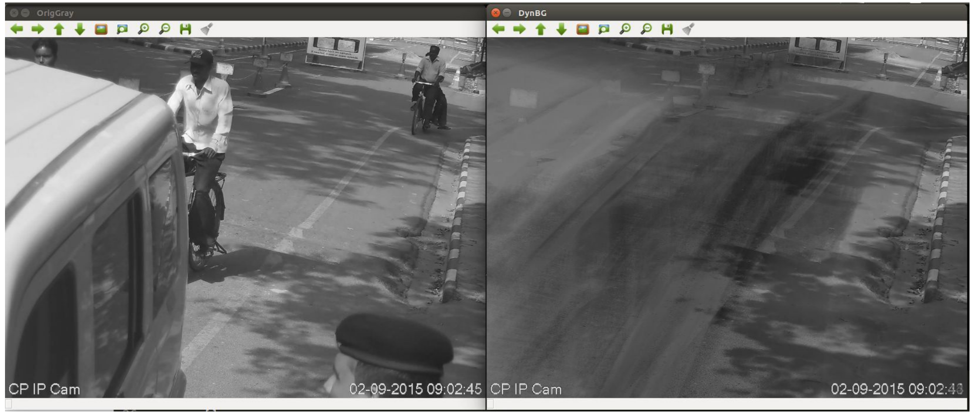
# Background Detection

## Background Computation:

### 1. Static background



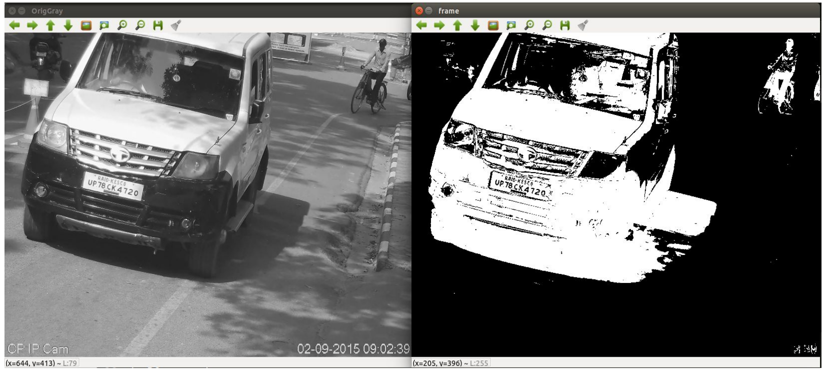
### 2. Dynamic Background

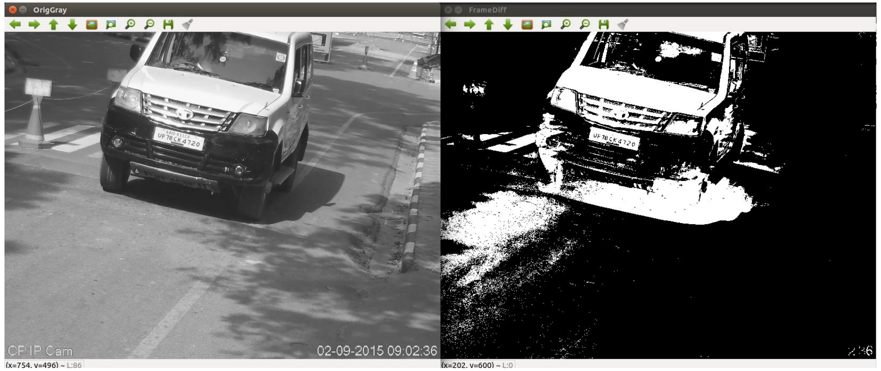


# Background Subtraction

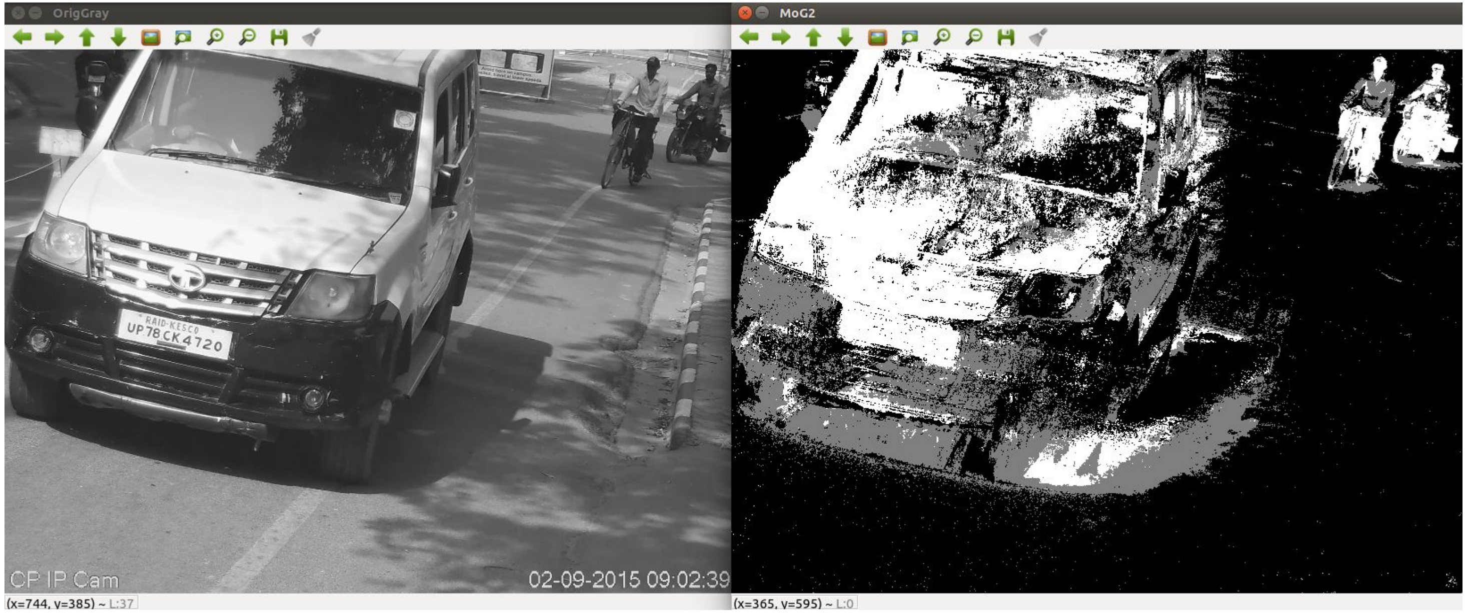
1. Subtracting the frames from the static background computed at start.

2. Subtracting the frames from the background which is dynamically updated with the video





## Background Subtraction



3. Using the Gaussian Mixture-based Background/Foreground Segmentation Algorithm (MoG2 of OpenCV)

# Object Detection

As mentioned in the previous slide. Object ‘detection’ is one of the most important parts in the workflow. As all the consequent stages depend upon it.

The popular object detection methods are mentioned below,

1. Interest Point based Detection: Find interesting points in images/objects express their respective localities. A few popular examples: SIFT features, Harris features.
2. Background Subtraction - Object detection can be achieved by building a representation of the scene called the background model and then finding deviations from the model for each incoming frame.
3. Segmentation - The aim of image segmentation algorithms is to partition the image into perceptually similar regions. There has been work showing the closeness to multi object tracking (Region-based Segmentation and Object Detection)

## Object Detection
### (Using Image Processing Operations)

We construct a background model and extract objects using the constructed model by considering deviations from it. Objects are detected using morphological operations.

The main reason for choosing to construct a background model is the fact that the security camera is still and thus a reasonable (accurate and nearly constant) background can be easily computed.

We use morphological operations because of the fast computation time involved and robustness and effectiveness of the proposed method (i.e. Invariance to Camera angle and Illumination)

The next slide contains the steps involved.

## Object Detection
### (Using Machine Learning Techniques)

We ran the LPO (Learning to Propose Objects) code (CVPR ‘15) by Phillipe Kranhebuhl from UC Berkeley’s BVLC group to act as the object proposer in our videos.

-- An image on average took 25 seconds for processing which was too high for our requirements.

-- The results were very impressive

-- The method uses an ensemble of jointly trained binary classification models (Global and Local CRFs, global and local in terms of connectivity) that are locally trained at locations decided using combinatorial optimization techniques.

## Steps Involved

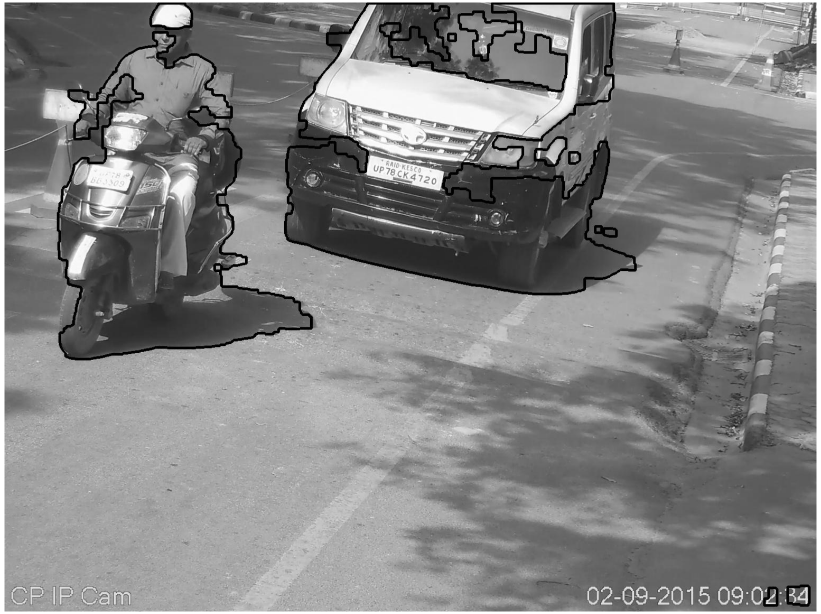
The aim during this stage is identifying blobs and proposing a bounding contour for it.

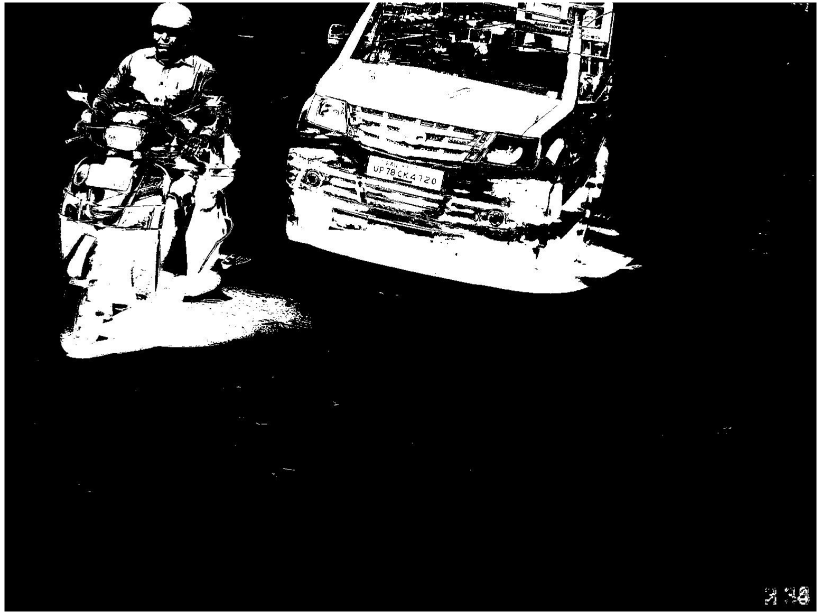
We extract the dominant blobs using the following steps:

- Background Subtraction (Discussed earlier)
- Thresholding and Binarizing the image
- Gaussian Blur (To smooth image and reduce the noise)
- Closing and Dilation operations (Converting objects to blobs)
- Finding contours (Of computed blobs)
- Eliminating Noisy Contours (Remove Noisy Blobs)

All the above operations can be performed in real time (Around 33 fps).

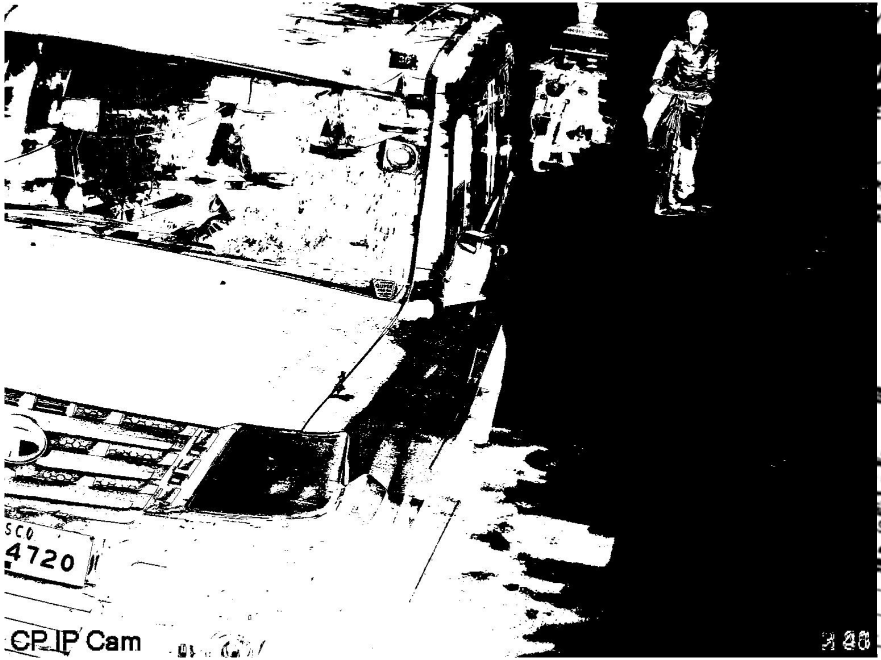
## Intermediate Results

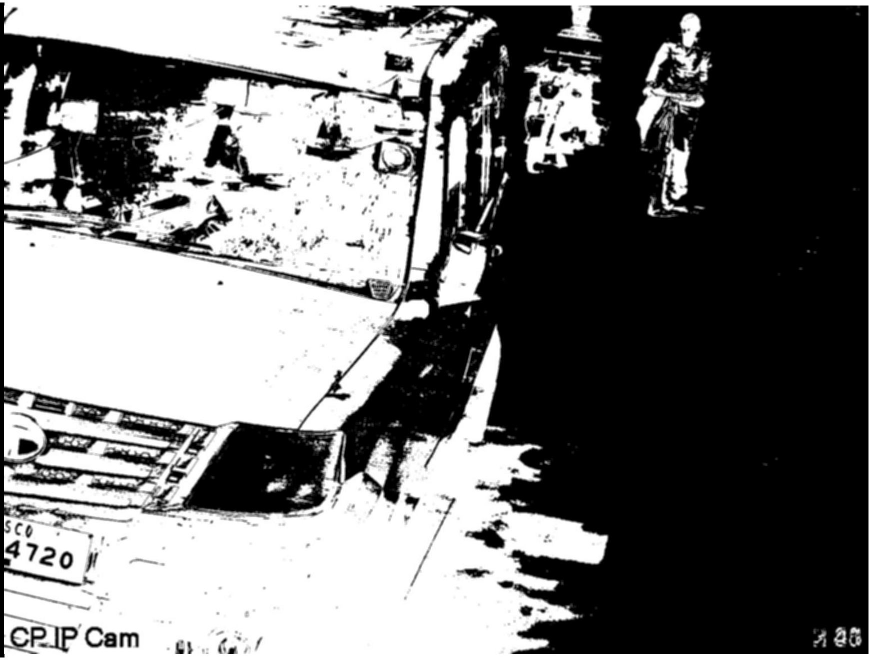




**Thresholding and Binarizing the Image**
**Left Image shows the final result (i.e. Detected Contour)**

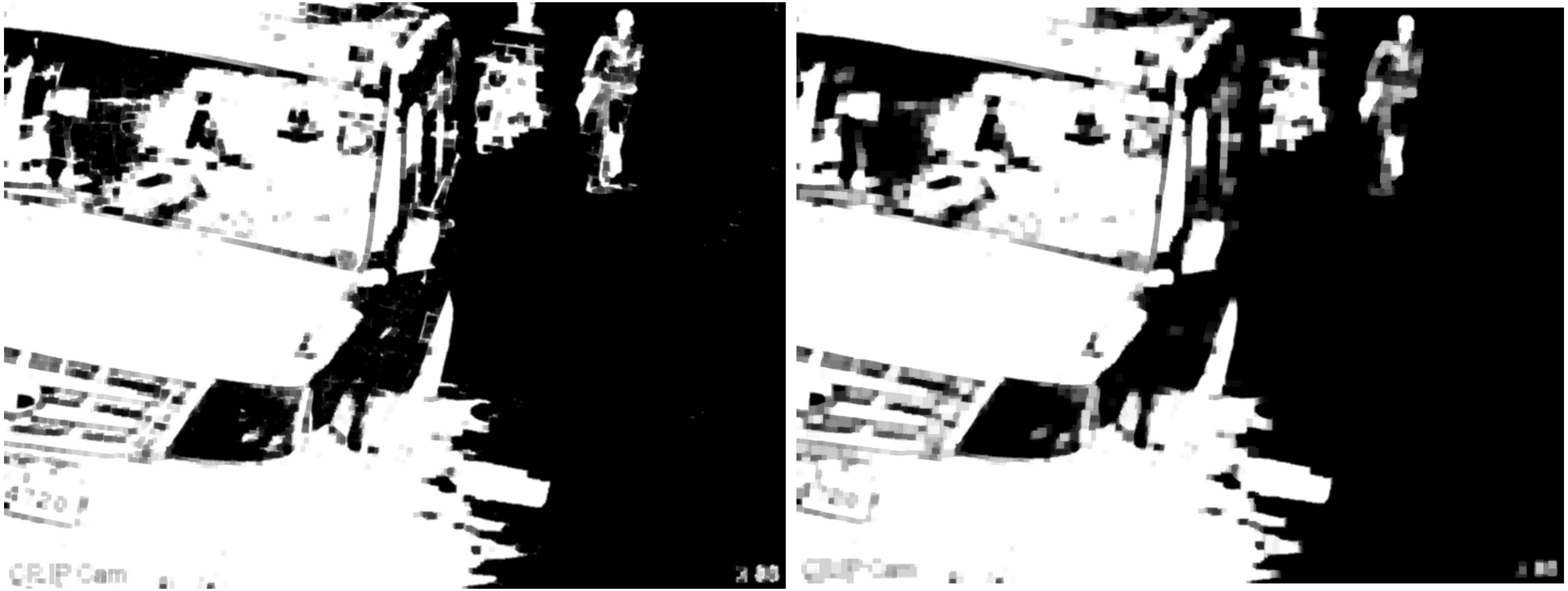
# Intermediate Results



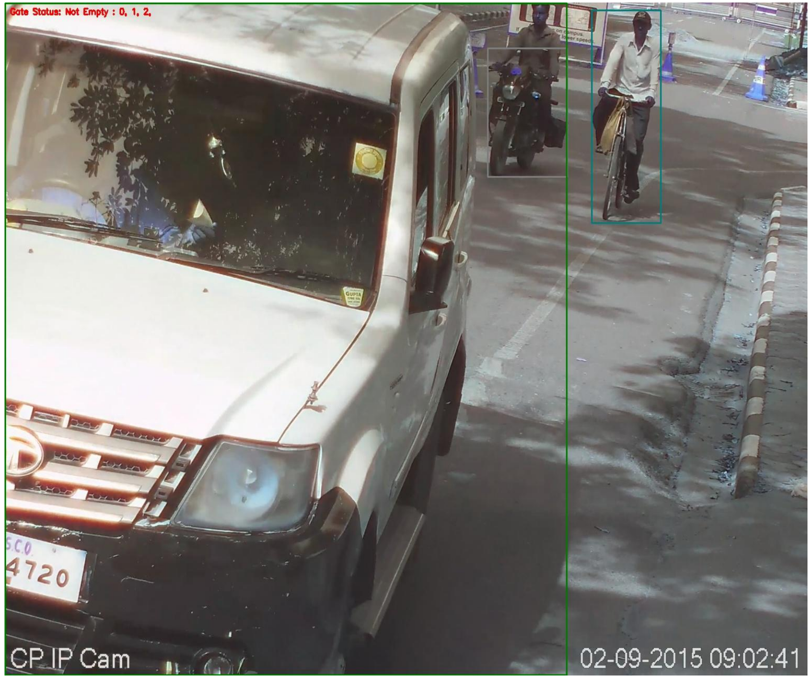


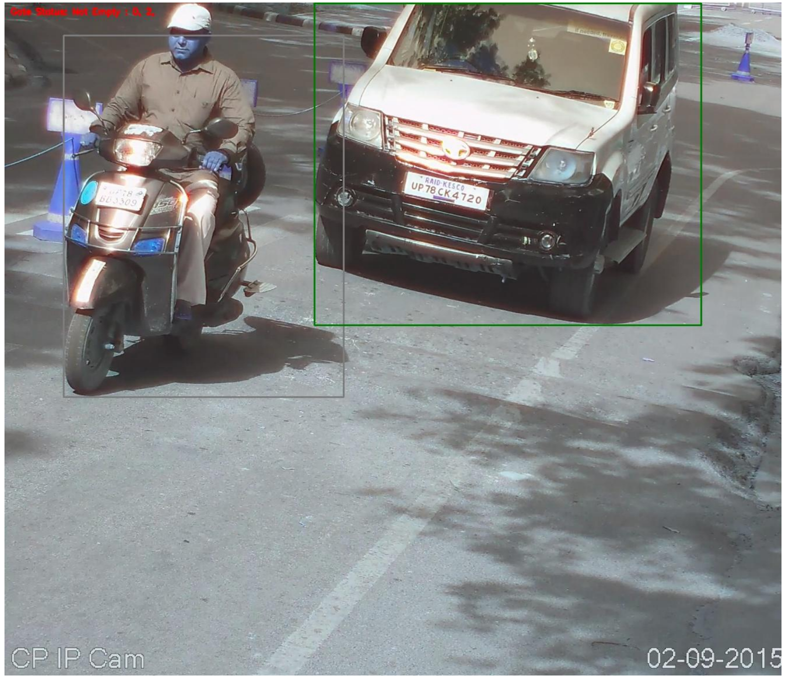
**Original Image (Left), Result after Blurring (Right)**

# Intermediate Results


Closed Image (Left), Opened Image (Right)

# Intermediate Results





**Results (Boxes indicate Objects)**

# Object Tracking

The previous section covered the Object Detection Stage. Note that we've not used the sequential information of the input i.e. it is a continuous video stream.
We name this section Object 'Tracking'. The reason it is different from the previous section is because we ignored the inter frame relationships in Object Detection, while we're using it to infer how the object moves.

We tried the following two methods for the same,

1. Contour similarities (Spatial Closeness)
2. Mean-Shift, CamShift based tracking (Very poor results, tuning required)
3. SIFT Based feature Matching

(Point matching variants tried: Least Squared Distance (Multiple), Hungarian based matching (Single))

Other possible models,

1. Region Proposal Networks
2. Region Based Segmentation and Detection

(Not used due to data constraints, Pre trained Models available but defeat the purpose of the project)

## Contour Similarity Based Tracking

Extremely simple idea: Compute similarities between the contours formed in consecutive frames. The closest contours represent the same objects. In case of a completely new blob formed, we declare it as a new object (And start tracking it).

**Related Issues:**

- Formed object can also be a group of objects.
- High Computation Time. Complexity involved is O(n**2), where n is the number of points in the contour.

**Solutions/Hacks Used:**

- Compute momentum (speed) of each box. Assumption involved is that the objects move with constant velocity (Which is pretty reasonable). Requires additional heuristics.
- Represent (Approximate) contours as rectangles (Bounding rectangles). O(1) computation.

## SIFT Feature Based Tracking

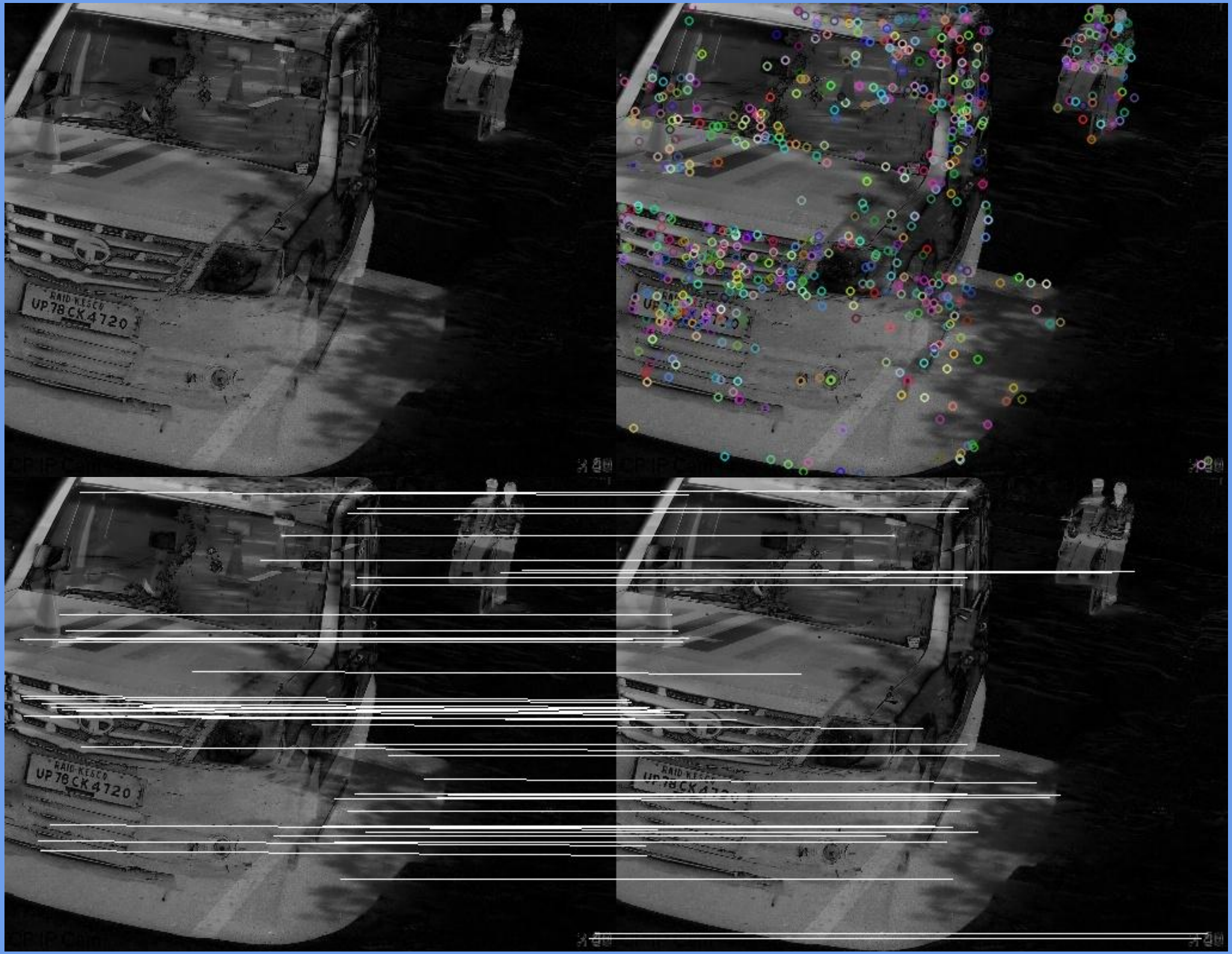
Observe that the objects in consecutive frames are extremely similar. Thus, matching the images on the basis of SIFT vectors (feature points) gives ‘extremely’ good results (Almost an exact match).

**Related Issues:**

- Computation time increases (Computing SIFT vectors takes around 0.09 seconds), The matching part takes around 0.4 seconds (Which is the main bottleneck).
- In case of multiple objects, the interest point matching algorithm sometimes matches different objects (Can be rectified by changing the distance parameter)

**Possible Solutions:**

- Consider only the prominent SIFT vectors. This can reduce the time to allow processing around 10fps.
- The incorrect matches can be removed after ignoring the absurd matches (Based on the expected translation) (Explained Later)

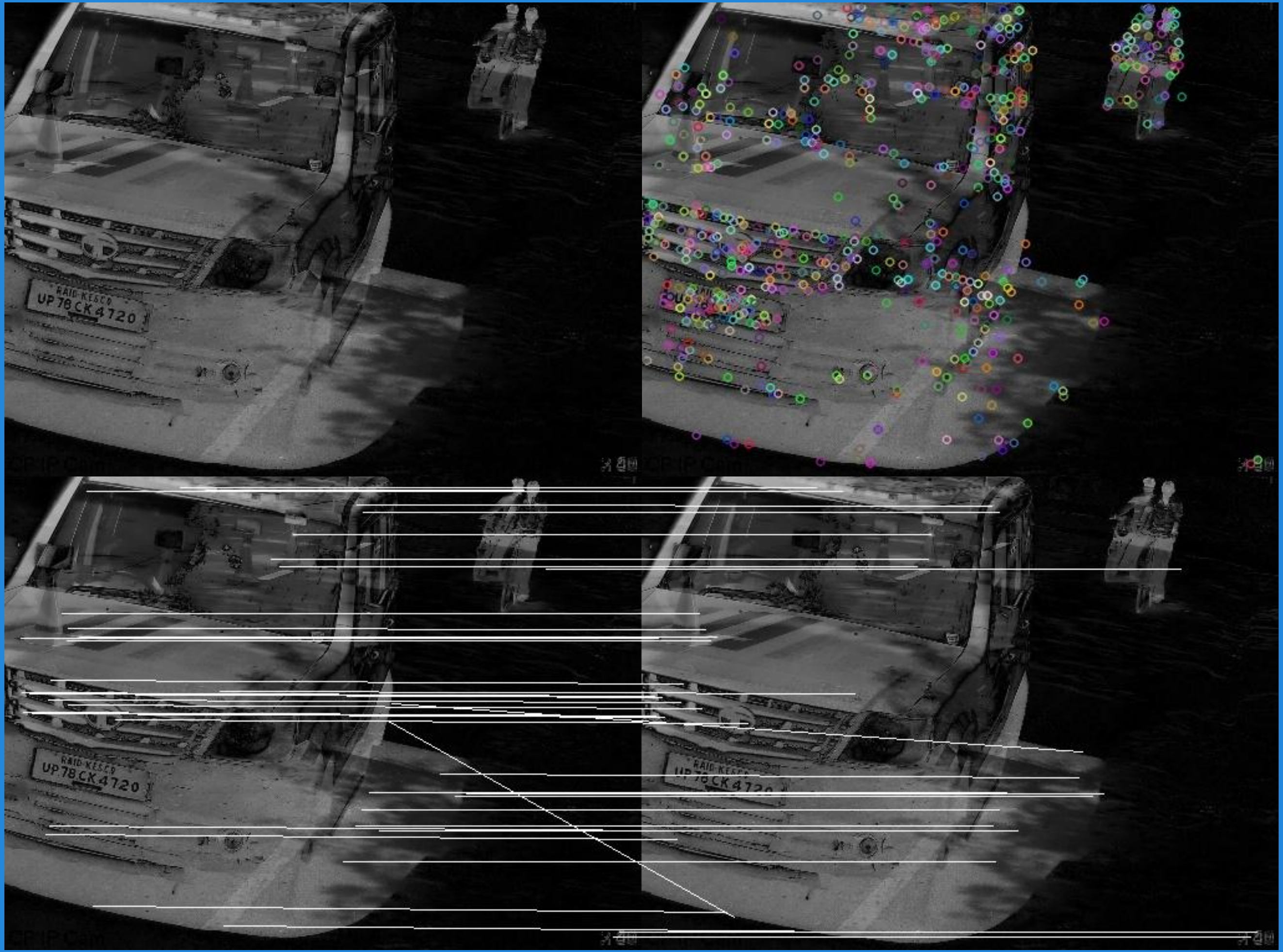


**SIFT interest points (Top)**

**Interest Point matches (Bottom)**

Notice that the matching lines are parallel (Can be used to remove incorrect matches)

(Showing an exact match and translation of the object)



Notice that the correct matching lines are parallel

This can be used to remove the incorrect matches. Since, the slope of the incorrect match is very different from the correct matches.

This is due to the fact that there is only linear (No rotational) translation of the object in small intervals.

## Dataset Description

Data collected by crowd sourcing (Using the architecture made by UC Irvine (VATIC))

Consists of 7 classes:

Car, Person, Motorcycle, Bicycle, Rickshaw, Auto-rickshaw, Number-Plate

Dataset Issues:

- Large number of Outliers (Incorrectly marked objects)
- Extremely similar Images (Discussed later)
- Poorly labelled objects (Occlusion, excessive translation, Overlap with others)
- Low number of images, Low diversity

We discuss a few results in the later sections. The train and test set used are described in the next slide. We also discuss a few points about the dataset.

## Data Pre-Processing

Our **Training** set consists of around 80% of the total data.
Our **Testing** set contains the rest of the data (Ensuring that we choose completely different objects from the training set)

Failing to ensure the above mentioned condition led to **absurdly high** accuracies (Around 96-98%). This is due to the earlier mentioned reason of extremely similar images, which makes the dataset equivalent to a small set multiplied by a constant.

We (try to) minimise incorrectly labelled data using a few heuristics during image extraction.
We also consider only those images of an object which are sufficiently different.

## Data Pre-Processing (Cont.)

As mentioned earlier, the (cleaned) dataset contains a low number of images and diversity. In order to increase the data set size, we can perform the following,

- Flipping the image (Mirroring it)
- Taking a slightly reduced part of the image (Reference)
- Varying the intensities (Through Gamma correction or other methods)

Our motivation for taking a reduced part of the image comes from the fact that a model should be able to decide on an object based on a few distinctive parts of the object.

We use the above mentioned methods to obtain a larger dataset (Around 16 times larger) and train a few models upon it. We observe an improvement in the results (Around 3-4%) which is discussed later and further supports the earlier claim.

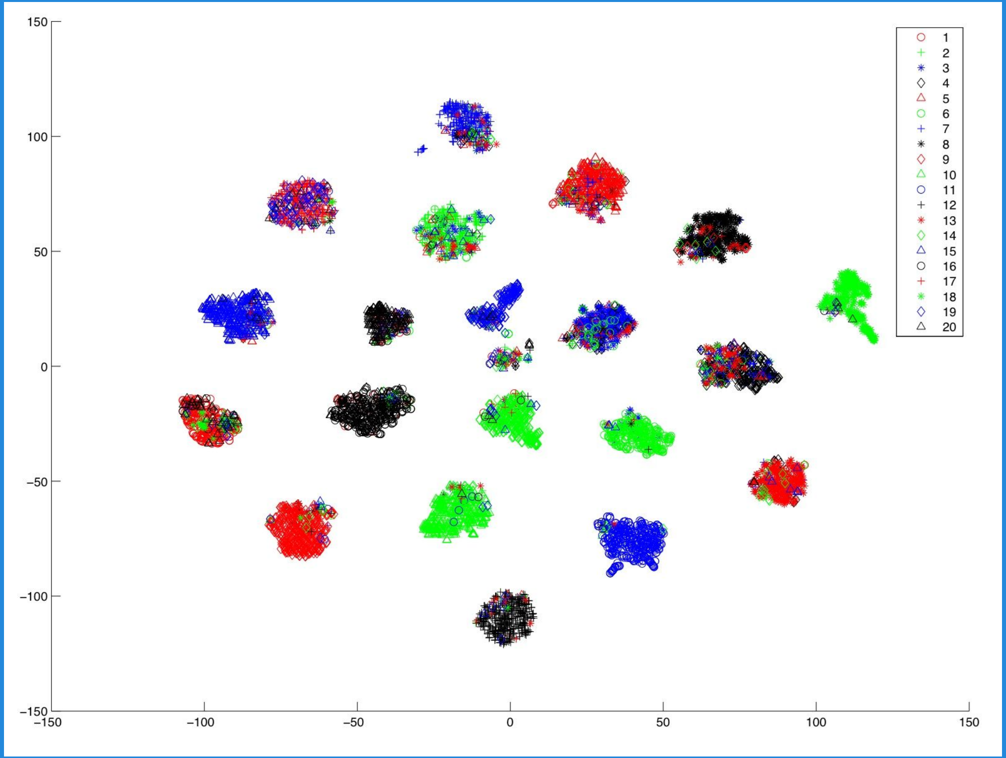
## Understanding the Dataset

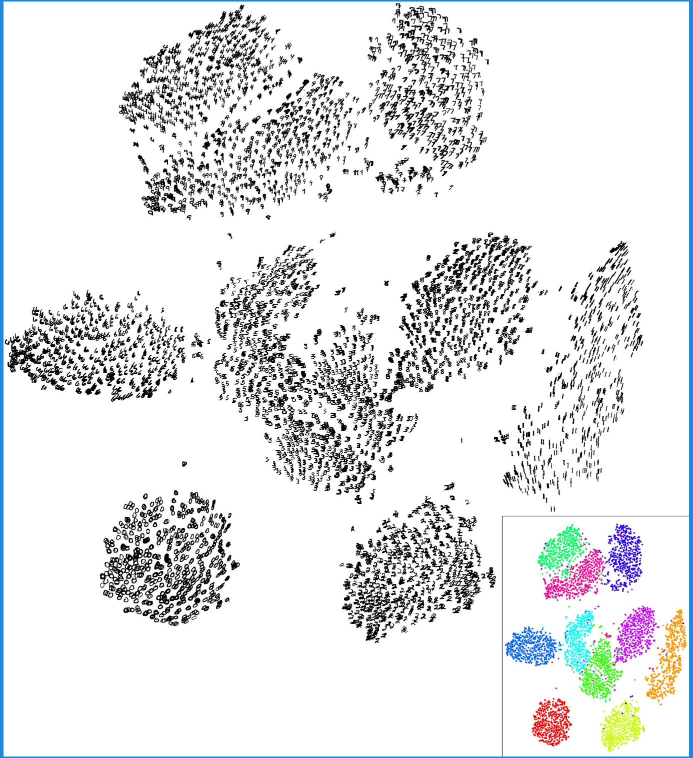
Before discussing the approaches used for classification. We discuss a few features of the dataset in the following slides.

This is an extremely important part which is generally ignored. This section focuses on a few features which motivates further discussions (Or proposals).

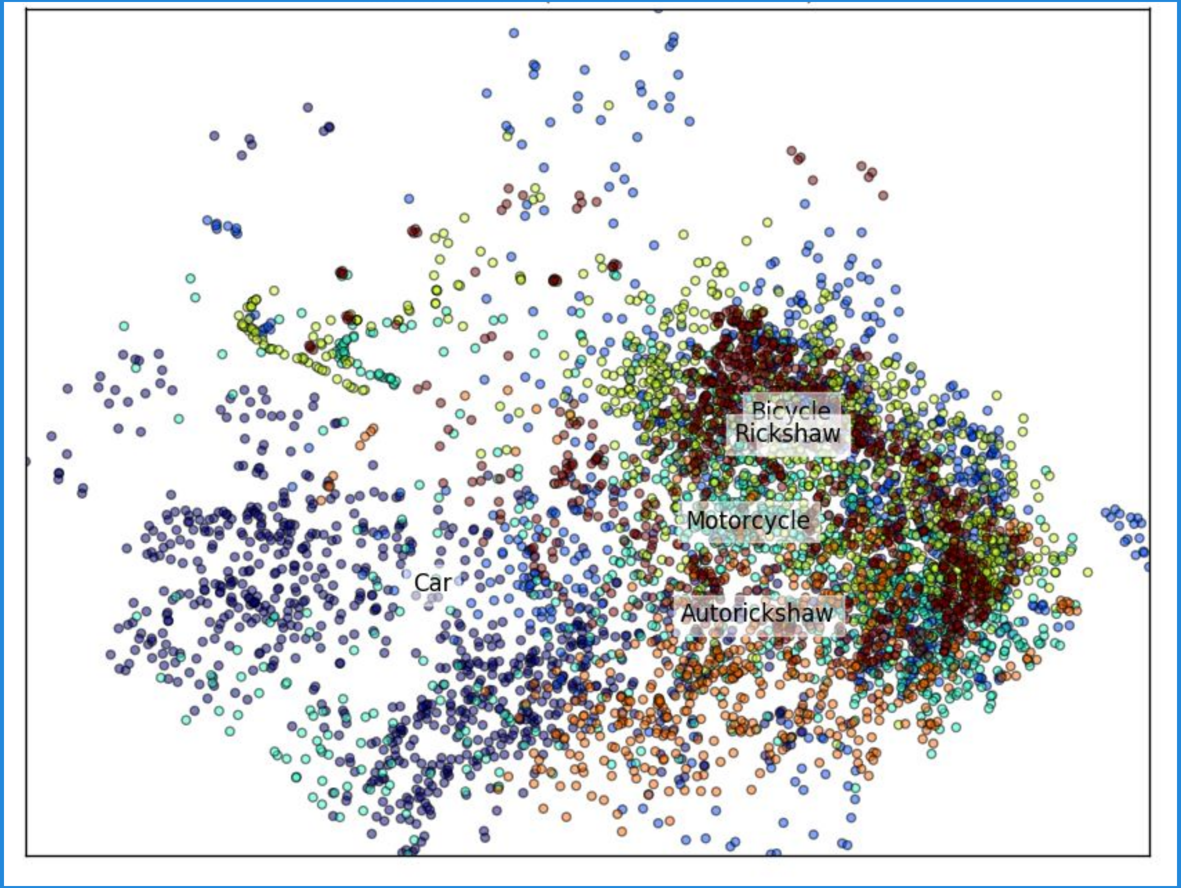
We plot the distribution of the data points in 2 (and 3) dimensions using dimensionality reduction algorithms namely PCA and tSNE (Which has been shown to perform extremely well for visualizing high dimensional data).

Assuming the classes to be easily separable, we should expect the cluster centers to be visibly different. (Examples shown in later slides)





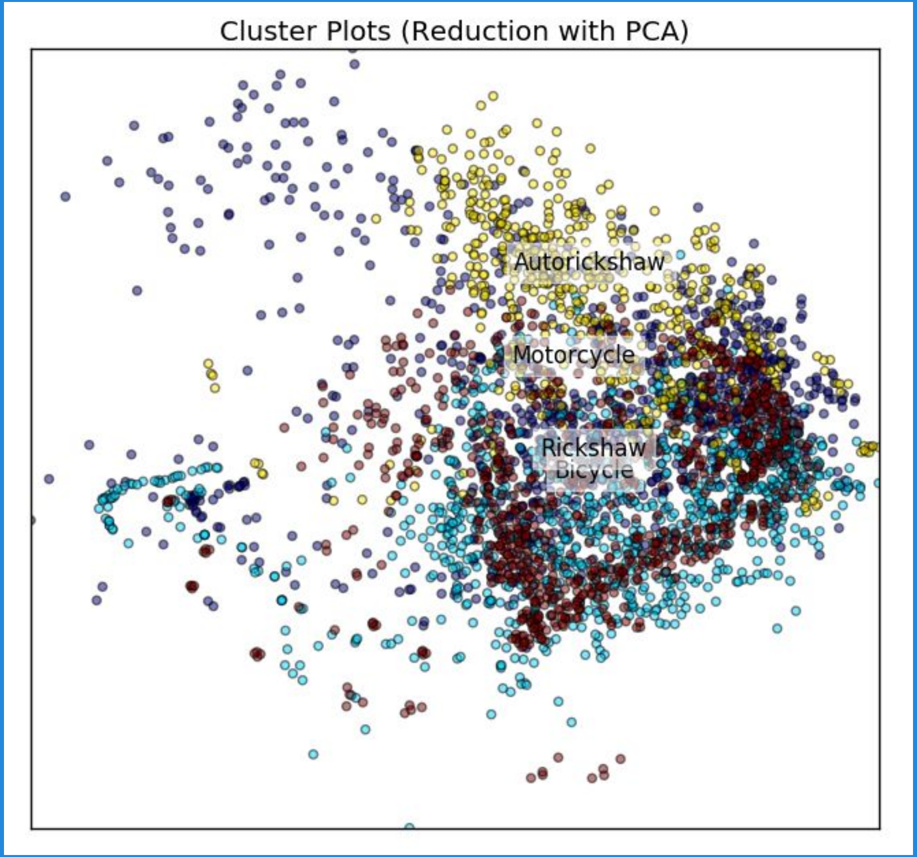
**20-NewsGroups Dataset (Left), MNIST Dataset (Right)**
**Examples of Datasets visualized using tSNE ([Refer])**



**The given image is reduced using PCA (Used due to speed for a primary inference)**

**We can see that the only reasonably separated cluster is Cars. Note that 'Rickshaw' and 'Bicycle' are extremely similar, which should be expected.**

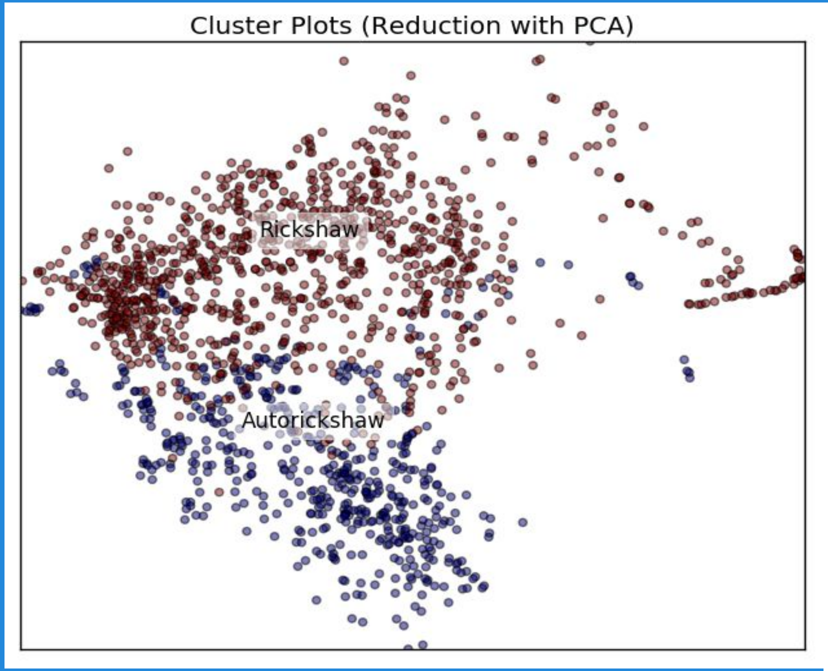
**This motivates us to consider a two class problem i.e. 'Car' and 'Not Car'. Then use another classifier to make further predictions.**

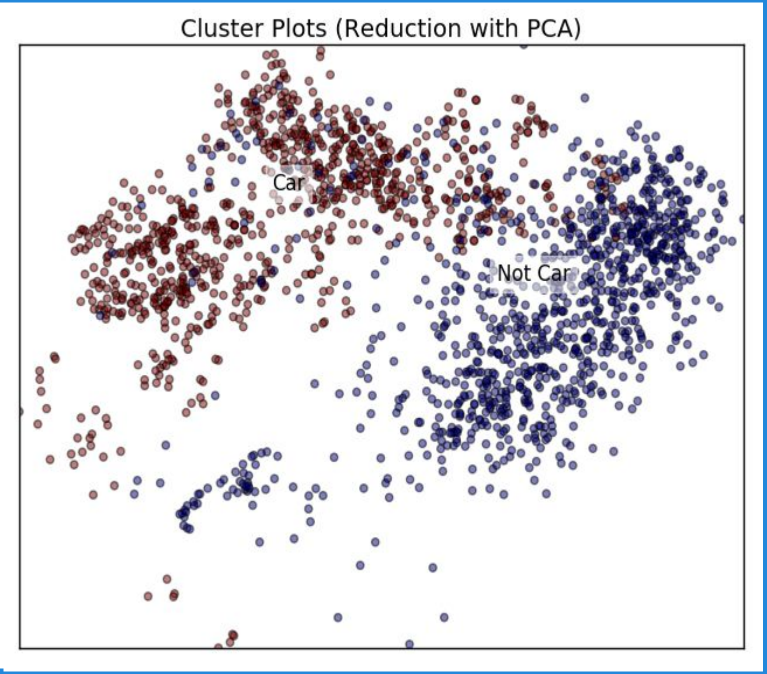


The given plot consists of classes after removing ‘Car’ and ‘Person’.

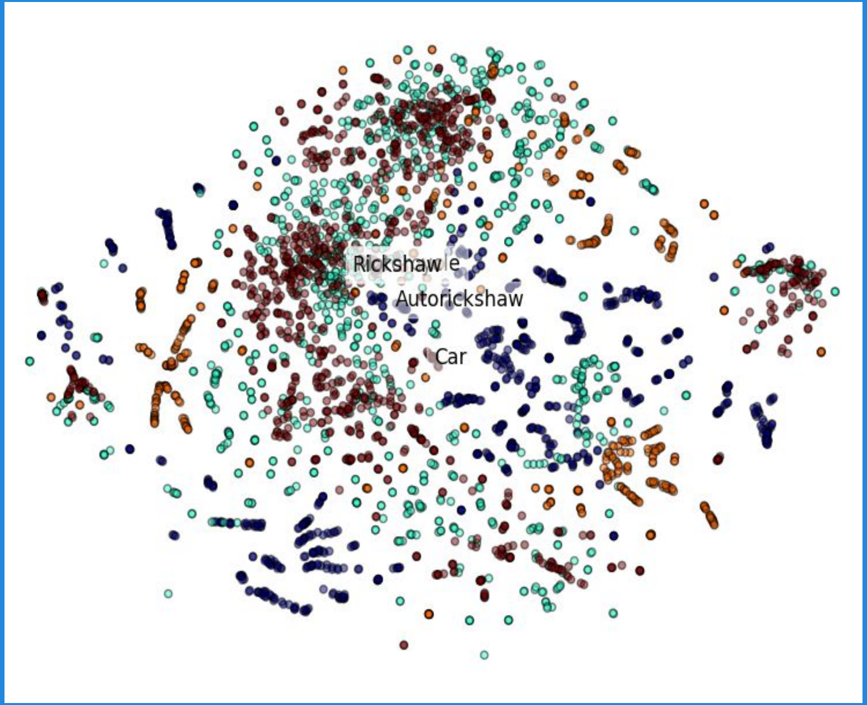
We can see that again ‘Rickshaw’ and ‘Bicycle’ are extremely similar. Which also motivates to merge the two classes together into one.

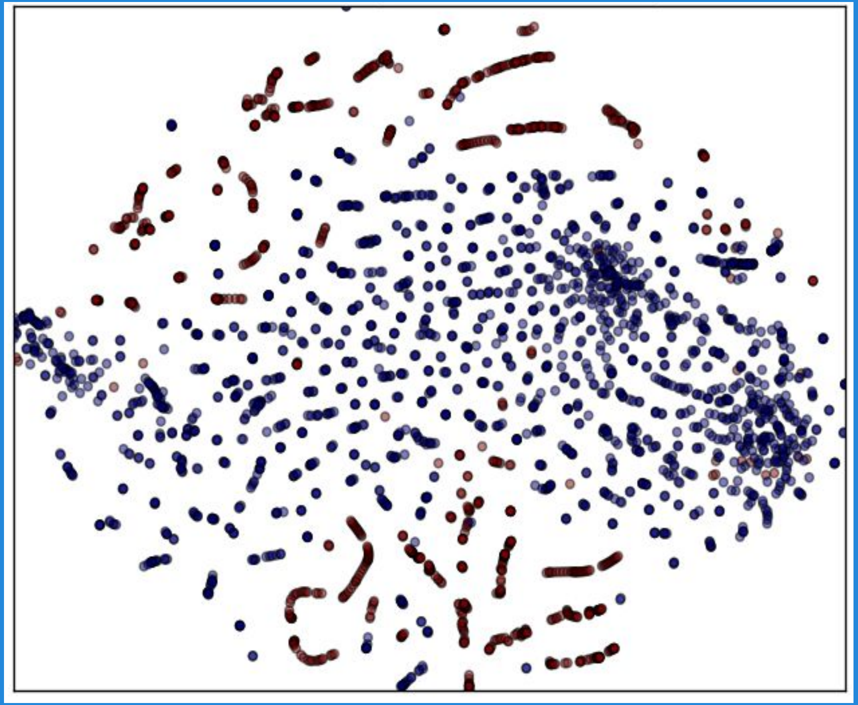
We can also see that ‘Auto-Rickshaw’ and ‘Rickshaw+Bicycle’ are pretty different (Far apart), thus can be discriminated nicely.





**Two Class Plots (Rickshaw, Auto-Rickshaw) (Left)**
**Two Class Plots (Car, Not Car) (Right)**
**Results supporting this are shown in the results section**





**tSNE Plots (All classes except Person) (Left)**
**Two Class Plots (Car, Not Car) (Right)**
**The plots obtained are quite complicated (Even though they do have some inherent structure)**

## Haar Cascade Classification

The very first classifier we tried using was HAAR cascade classifier for vehicle classification. We tried using an external dataset. (From Stanford) http://ai.stanford.edu/~jkrause/cars/car_dataset.html

The results were very poor. So later we tried using the same classifier with data extracted from the video. But we could get only limited number of training samples for HAAR cascade. Hence the classifier could not be properly trained. So again the results were not satisfactory.

This made us move to other common classifiers like LinearSVC, SVC, Random Forest and Adaboost.

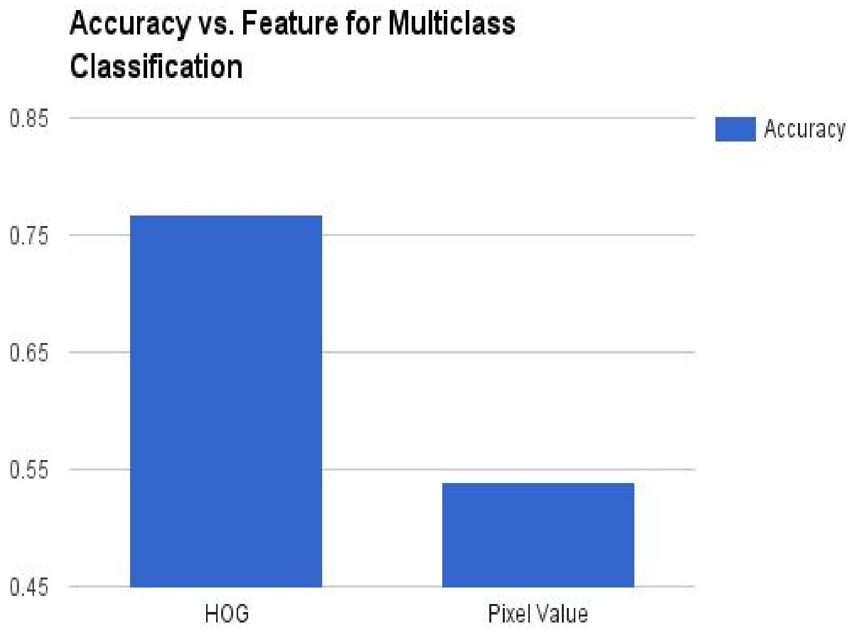
## Analysis of various features

We experimented with two types of features. We used the following features:-

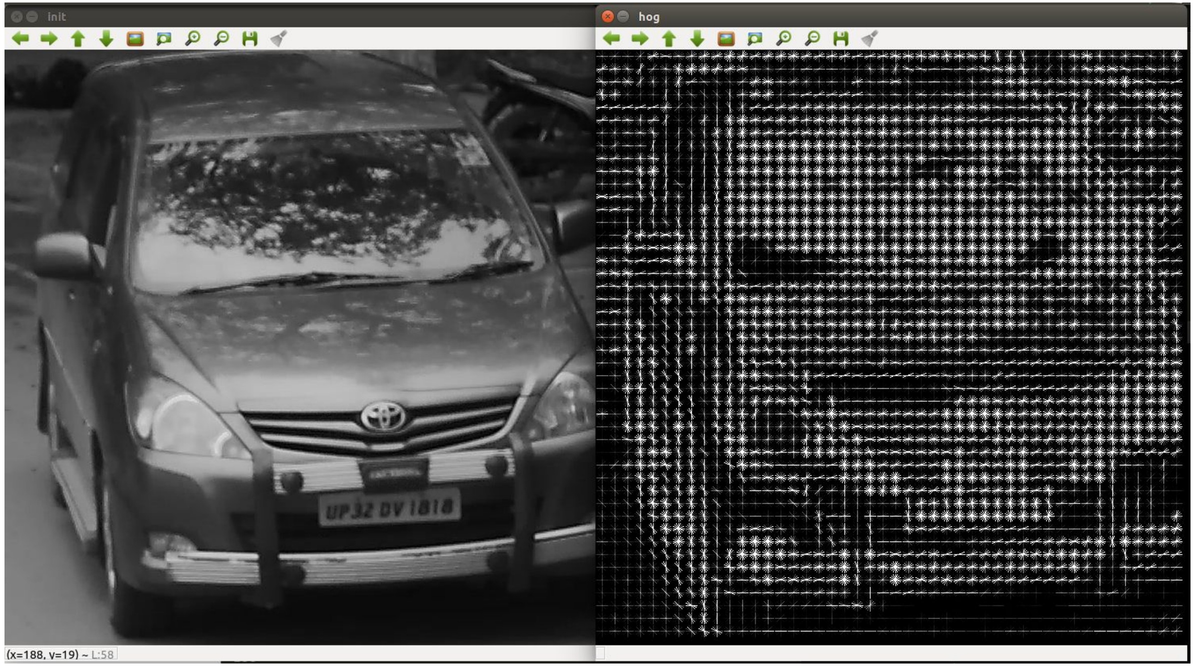
1. HOG
2. Pixel Value
3. PCA on Pixels

As expected, HoG features outperformed raw pixel features by a large margin

-- Figure created using a linear SVM classifier



# Analysis of various features

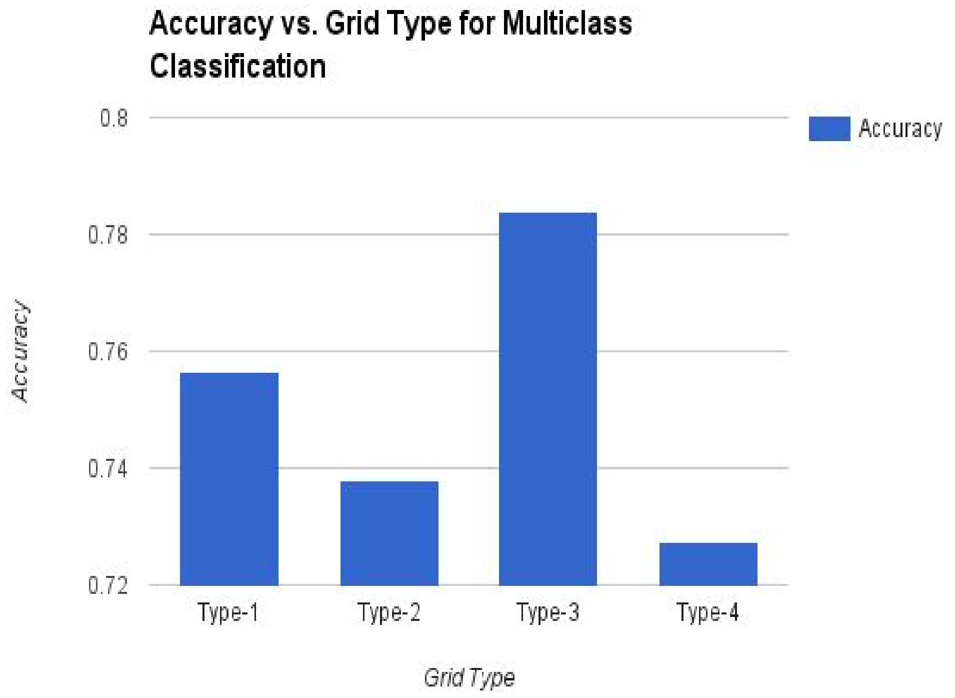


Input image (Left) and Histogram of Oriented Gradients (right)

# Analysis of various parameters (SVC)

Parameter Tuning using Grid Search.
We have used the following grids for SVC :-

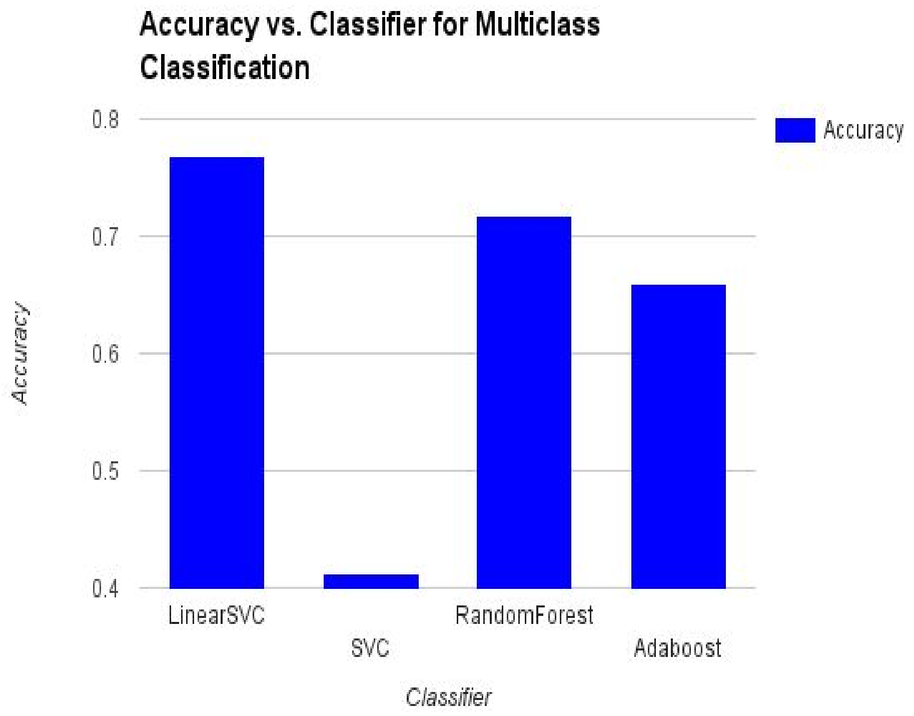
1. {'C': [0.1, 1, 10, 100, 1000], 'kernel': ['rbf']} (Type-1)
2. {'C': [0.1, 1, 10, 100, 1000], 'kernel': ['linear']} (Type-2)
3. {'C': [10,20,30,40,50,60,70,80,90,100], 'kernel': ['rbf']} (Type-3)
4. param_grid = {'C': [10,20,30,40,50,60,70,80,90,100], 'kernel': ['linear']} (Type-4)



## Analysis of various classifiers

For Multiclass Classification we have tried the following classifiers:

1. LinearSVC
2. SVC
3. Random Forest
4. Adaboost

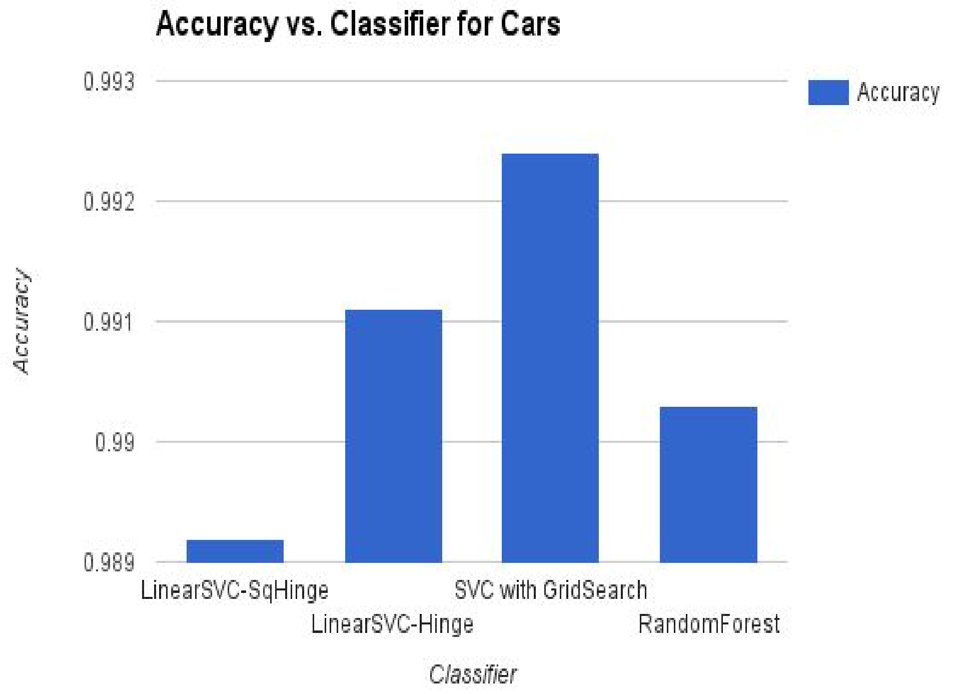


## Analysis of Binary Classification

We also tried binary classification for cars using the following approaches: -

1. LinearSVC with sq. hinge loss.
2. LinearSVC with hinge loss.
3. SVC with Grid Searched parameters
4. Random Forest.

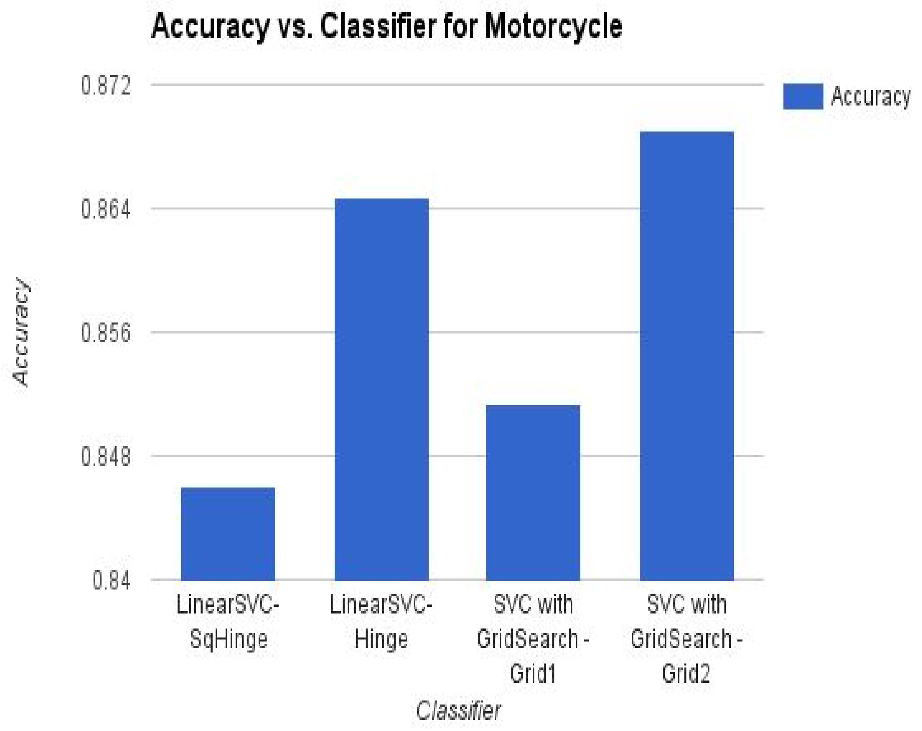
(Mainly because our dataset is skewed in favour of Non-Cars as can be seen in the recall value which is 0.91)



# Analysis of Binary Classification

We also tried binary classification for motorcycles using the following approaches:-

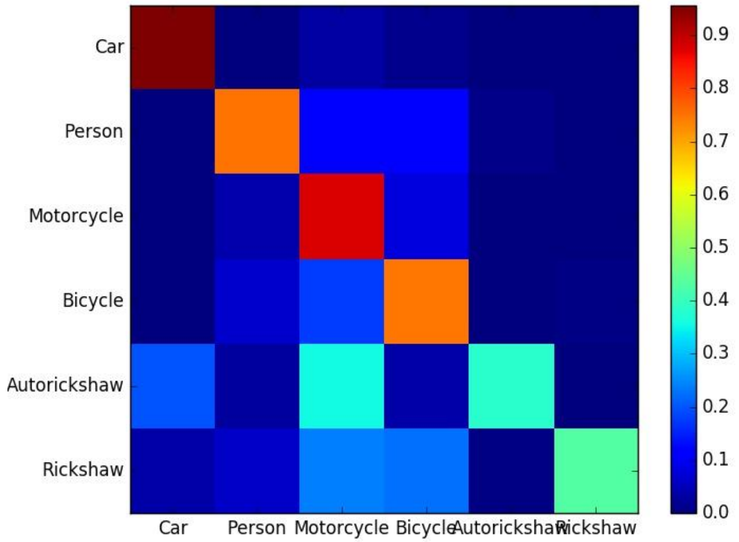
1. LinearSVC with sq. hinge loss.
2. LinearSVC with hinge loss.
3. SVC with Grid Search [Type 1 parameters]
4. SVC with Grid Search [Type 2 parameters]



# Performance of LinearSVC

LinearSVC gave the best results with hinge loss and HoG features for vehicle classification.

The Confusion matrix for the same is:



| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| **Car** | 0.8444 | 0.9559 | 0.8967 | 227 |
| **Person** | 0.6268 | 0.7507 | 0.6832 | 349 |
| **Motorcycle** | 0.7823 | 0.8784 | 0.8276 | 1522 |
| **Bicycle** | 0.8058 | 0.7492 | 0.7765 | 1224 |
| **Autorickshaw** | 0.8615 | 0.3836 | 0.5308 | 146 |
| **Rickshaw** | 0.9074 | 0.4317 | 0.5851 | 227 |

## Convolutional Neural Network

We trained two ConvNet architectures on the Augmented Dataset (~11000 training samples, ~2000 test samples)

**Architecture 1:**

Image[100x100] -> Conv1 [8 filters, 3x3] -> ReLU -> MaxPool(2x2) -> Flatten -> Dense[64] (tanh activation) -> Softmax[6]

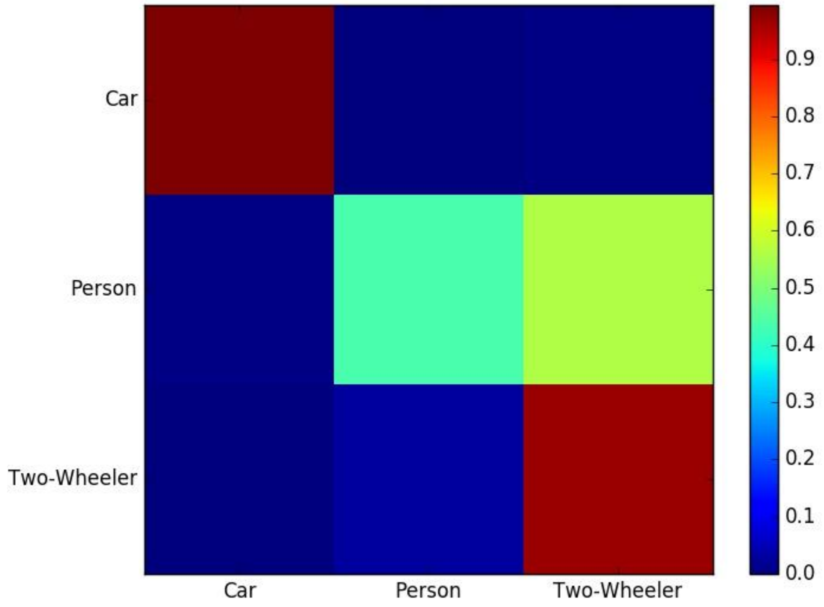
**Architecture 2:**

Image[100x100] -> Conv1 [8 filters, 3x3] -> ReLU -> MaxPool(2x2) -> Conv2 [8 filters. 3x3] -> ReLU -> MaxPool(2x2) -> Flatten -> Dense[64] (tanh activation) -> Softmax[6]

-- Both the architectures were trained with aggressive Dropout rates to prevent Overfitting

# Effect of Data Augmentation

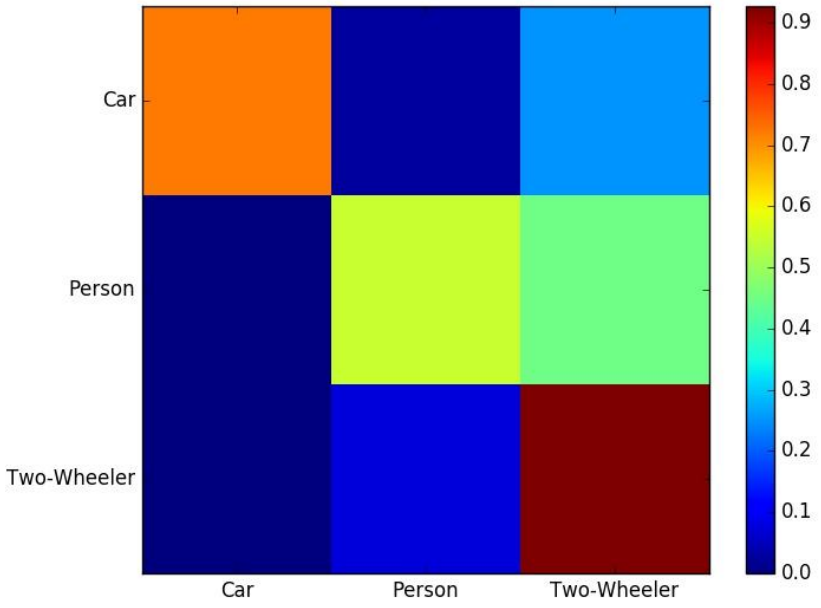
On Augmented Data:



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Car | 0.9831 | 0.9943 | 0.9887 | 176 |
| Person | 0.6243 | 0.4355 | 0.5131 | 248 |
| Two-Wheeler | 0.9366 | 0.9686 | 0.9524 | 2136 |
| avg / total | 0.9096 | 0.9187 | **0.9123** | 2560 |

# Effect of Data Augmentation

On Un-Augmented Data:

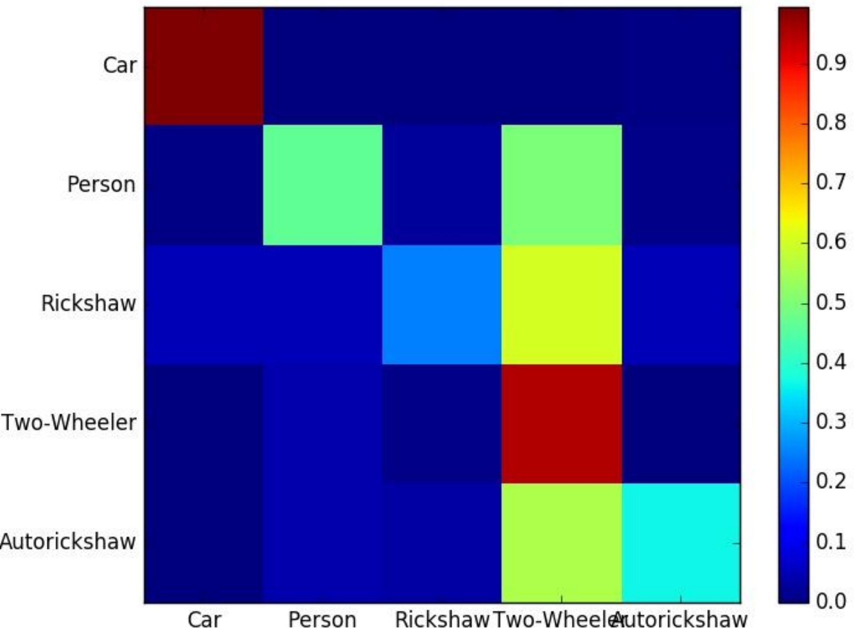


| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Car | 1.0000 | 0.7216 | 0.8383 | 176 |
| Person | 0.4579 | 0.5484 | 0.4991 | 248 |
| Two-Wheeler | 0.9270 | 0.9270 | 0.9270 | 2136 |
| avg / total | 0.8865 | 0.8762 | **0.8794** | 2560 |

**Both tested on same Test Set!**

# ConvNet Performance

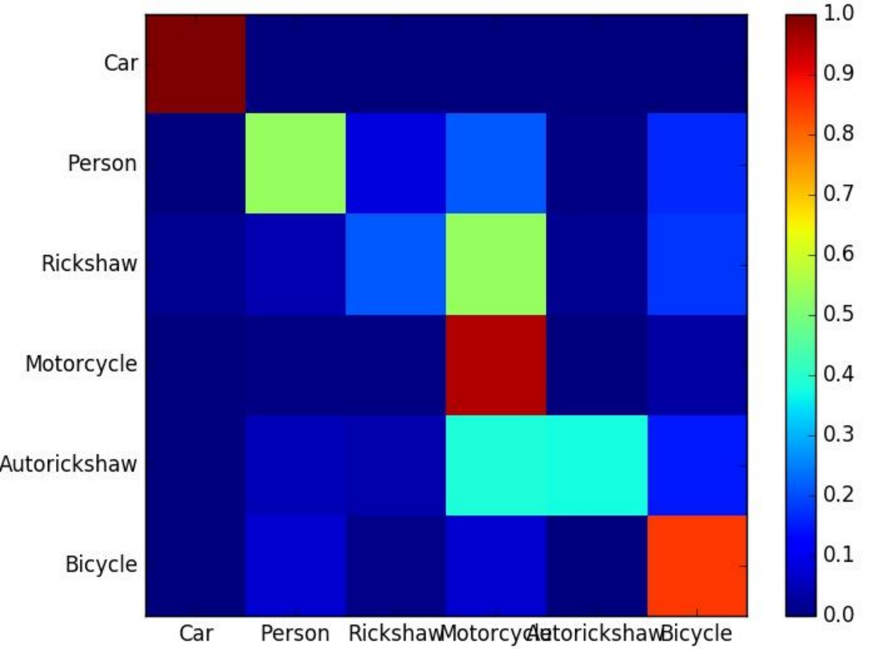
On 5 Classes:



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Car | 0.9358 | 0.9943 | 0.9642 | 176 |
| Person | 0.5425 | 0.4637 | 0.5000 | 248 |
| Rickshaw | 0.5974 | 0.2500 | 0.3525 | 184 |
| Two- Wheeler | 0.8704 | 0.9494 | 0.9082 | 2136 |
| Autorickshaw | 0.7586 | 0.3667 | 0.4944 | 120 |
| avg / total | 0.8238 | 0.8408 | 0.8232 | 2864 |

# ConvNet Performance

On 6 Classes:



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Car | 0.9724 | 1.0000 | 0.9860 | 176 |
| Person | 0.6055 | 0.5323 | 0.5665 | 248 |
| Rickshaw | 0.4875 | 0.2120 | 0.2955 | 184 |
| Motorcycle | 0.8102 | 0.9552 | 0.8767 | 1184 |
| Autorickshaw | 0.9184 | 0.3750 | 0.5325 | 120 |
| Bicycle | 0.8606 | 0.8498 | 0.8552 | 952 |
| avg / total | 0.8030 | 0.8142 | 0.7977 | 2864 |

## License Plate Detection

After the object classification stage, we come to License plate Recognition part. The first stage in this involves localizing or computing regions where the license plate may lie.

The final part is recognizing the text on the license plate. This involves, segmenting the characters and then using OCR to recognize the text on the license plate. We also require lots of data (text information) to train such an OCR model, thus we have used pre-trained models on text from US License plates. Thus, even if the regions proposed are reasonable, the OCR part is not able to recognize the characters.

There are many prevalent methods for localising license plates. We use image processing operations (Mainly gradients and morphological operations) to narrow down the regions.

## License Plate Localization
### (Using Image Processing Operations)

As used in the object detection stage. We use a background model to remove the irrelevant information.

The first stage involves computing (horizontal) gradients. The motivation being that the license plate region will contain a lot of gradients (Due to the text present in it).

We also experimented with vertical and total (Horizontal and vertical) gradients. The improvements were not significant.

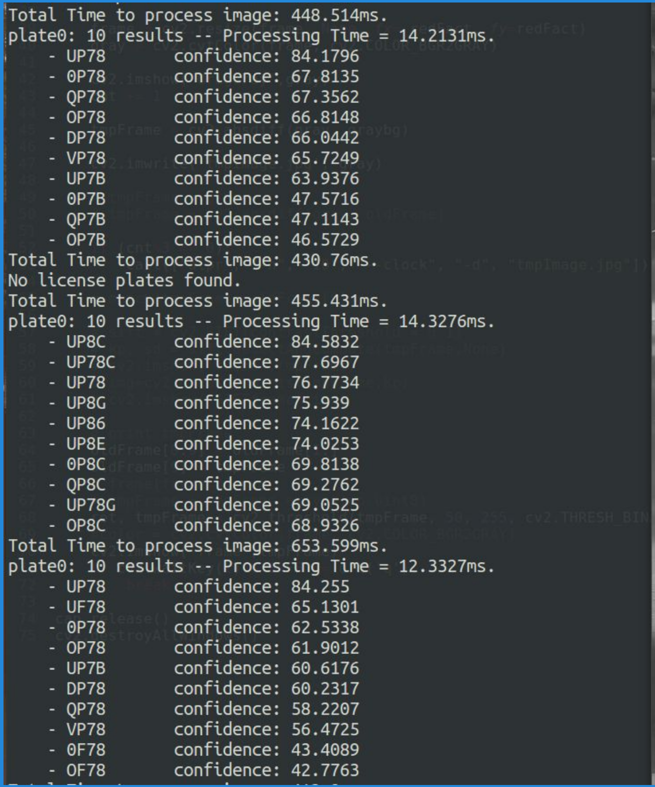
This is followed by morphological operations as discussed earlier (In object detection). The following slide contains the intermediate results.

## License Plate Recognition

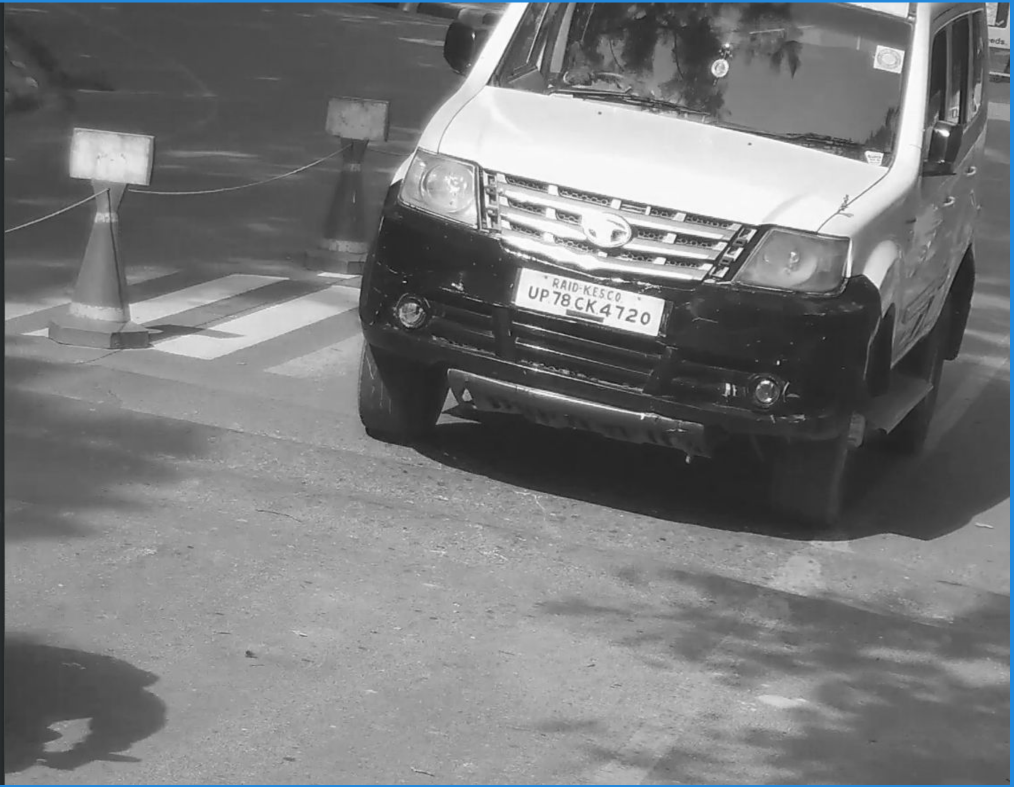
We used the pretrained OpenALPR model (on US License Plates) for the task and tried out their detection pipeline on our proposed number plate regions.

OpenALPR Pipeline --

**Char Analysis** - Find connected character-sized blobs in binarized image
**Deskew -** Affine transformation to change perspective to straight-on view
**Character Segmentation -** Individual character segmentation and cleaning
**OCR -** Analyzing each character given by the Character Segmenter and predicting a character
**Post Processing -** Creating a list of plate possibilities based on OCR confidences



```
Total Time to process image: 448.514ms.
plate0: 10 results -- Processing Time = 14.2131ms.
    - UP78       confidence: 84.1796
    - 0P78       confidence: 67.8135
    - QP78       confidence: 67.3562
    - OP78       confidence: 66.8148
    - DP78       confidence: 66.0442
    - VP78       confidence: 65.7249
    - UP7B       confidence: 63.9376
    - 0P7B       confidence: 47.5716
    - QP7B       confidence: 47.1143
    - OP7B       confidence: 46.5729
Total Time to process image: 430.76ms.
No license plates found.
Total Time to process image: 455.431ms.
plate0: 10 results -- Processing Time = 14.3276ms.
    - UP8C       confidence: 84.5832
    - UP78C      confidence: 77.6967
    - UP78       confidence: 76.7734
    - UP8G       confidence: 75.939
    - UP86       confidence: 74.1622
    - UP8E       confidence: 74.0253
    - 0P8C       confidence: 69.8138
    - QP8C       confidence: 69.2762
    - UP78G      confidence: 69.0525
    - OP8C       confidence: 68.9326
Total Time to process image: 653.599ms.
plate0: 10 results -- Processing Time = 12.3327ms.
    - UP78       confidence: 84.255
    - UF78       confidence: 65.1301
    - 0P78       confidence: 62.5338
    - OP78       confidence: 61.9012
    - UP7B       confidence: 60.6176
    - DP78       confidence: 60.2317
    - QP78       confidence: 58.2207
    - VP78       confidence: 56.4725
    - 0F78       confidence: 43.4089
    - OF78       confidence: 42.7763
```



**Recognizing License Plate Text (Right)**

## Possible Improvements

-- Get more relevant data for digit-wise OCR

-- Use a Convolutional Neural Network for the OCR task

-- Use Semantic Image Segmentation (eg. DeepLab Segmentation Engine). Use Semantic Segmentation to get object proposals

-- The tracking can be done using SIFT correspondence, Mean Shift, learning a small CRNN (Convolutional Recurrent Neural Network (Shi et al., CoRR 2015)) to get corner change values given an images and their corresponding corner positions

-- Using faster-rcnn (Ross Girschick et al.) (Training requires a huge amount of data)

-- Using a Scalable Vocabulary Tree of SIFT Features for fast and efficient class detection (Nister et al.)

## Tools Used

1. Languages Used: Python
2. Libraries Used:
   a. Scikit-Learn
   b. Tesseract
   c. OpenALPR
   d. Keras
   e. Theano
3. Pre-Trained Models Tested:
   a. OpenALPR
   b. LPO
   c. OverFeat