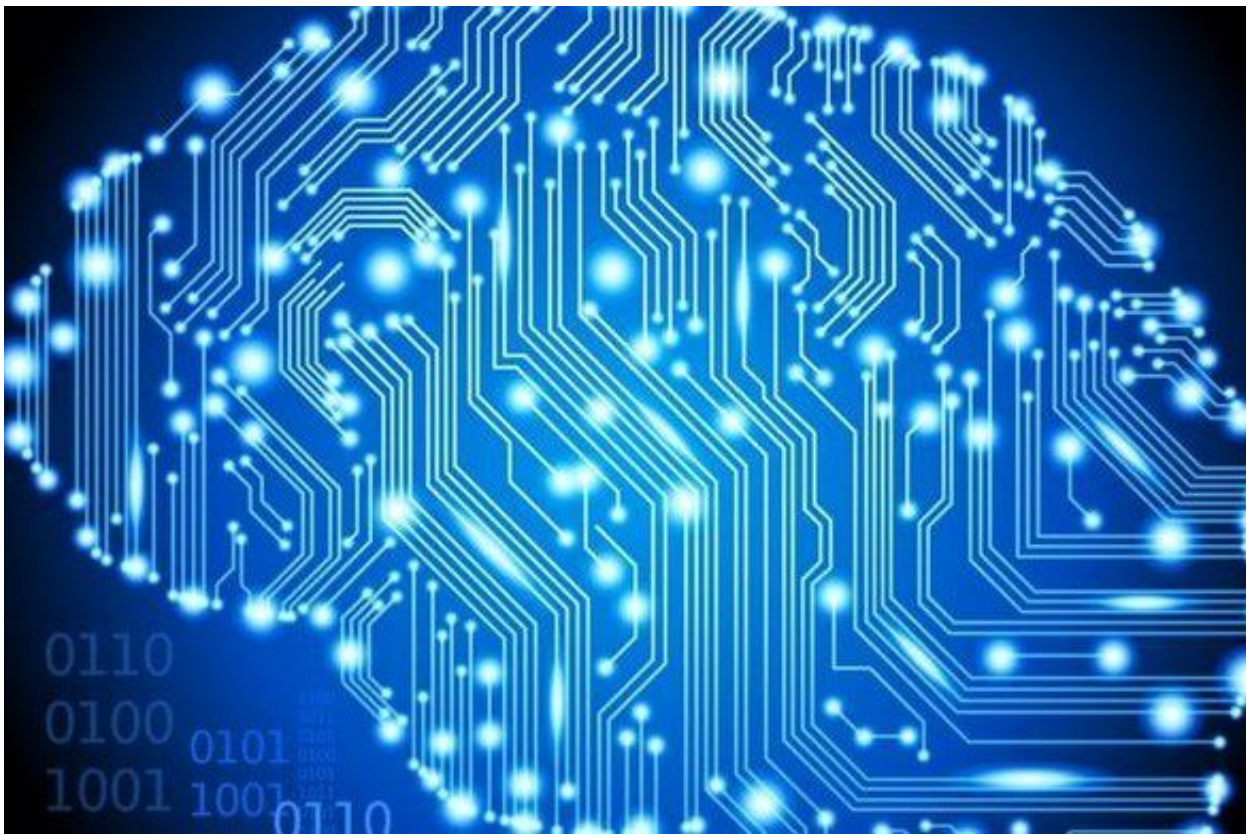


Deep Learning Project

CS 726

Karan Vaidya (130050019)

Shudhatma Jain (130050024)

Anmol Arora (130050027)

Aman Goel (130050041)

Project idea/goals

We aim to build a smart traffic management system. The idea is to use CCTV footage to detect cars and analyze traffic density and flow on the road using deep learning techniques and based on that, manage traffic lights intelligently.

We aim to do the following tasks:

- Take an image and find out the patches of image where cars will be possibly present
- Iterate over the patches found and use deep learning to figure out if a patch actually contains a car

This will eventually give us the number of cars in the image. We can then use this data of car count over time to estimate the flow and hence manage traffic signals on intersections efficiently.

Related Literature

- http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7519418&tag=1
- https://arxiv.org/pdf/1504.01716.pdf
- http://vision.stanford.edu/teaching/cs231b_spring1415/slides/overfeat_eric.pdf
- http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html
- https://github.com/tensorflow/models/tree/master/inception
- https://www.tensorflow.org/versions/r0.11/tutorials/image_recognition/index.html
- https://keras.io/

Approaches tried

As mentioned above, our algorithm has 2 parts:

- Finding possible regions of the image where a car may be present
- Iterating over each region/patch and determine if it actually contains a car

For the former, we tried 2 main ideas:

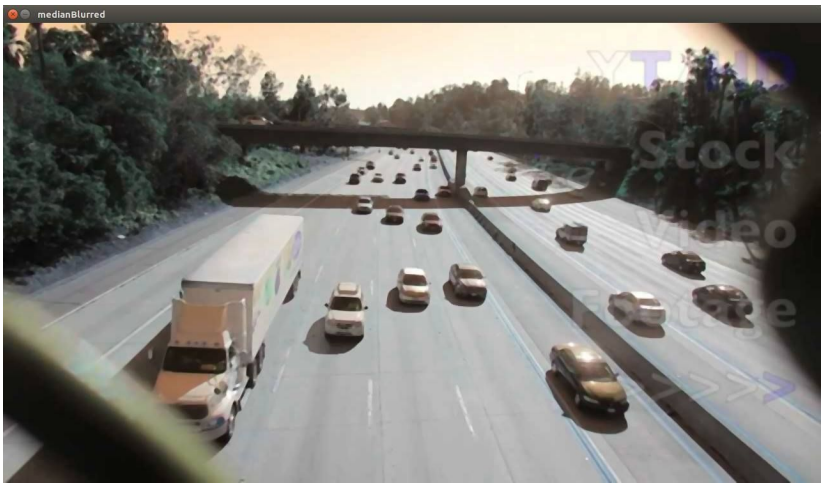
1. **Recursive Division :** We tried basic approaches such as dividing an image into smaller parts and looking for cars in them. Precisely we divided the image into n parts (n varies from 4 - 25) and checked if that part contains a car. If so, we further

divide that part into 4 more parts to find smaller cars and repeat the process. This approach gave us promising results. However it has significant shortcomings

- If a car passes through multiple partitions, it would not be counted (or be counted multiple times), which is an issue.
- The number of partitions was large making the complete process too slow.

Therefore, we tried to look for better techniques.

2. **Contour Detection (or Edge Detection) :** Edge detection is a standard technique of identifying edges in an image at which the image brightness has discontinuities. We used it to find the boxes which showed brightness discontinuity. In the process, we first blurred the image and then converted it into black and white form for better edge detection. An example is below:



Median blurred Image

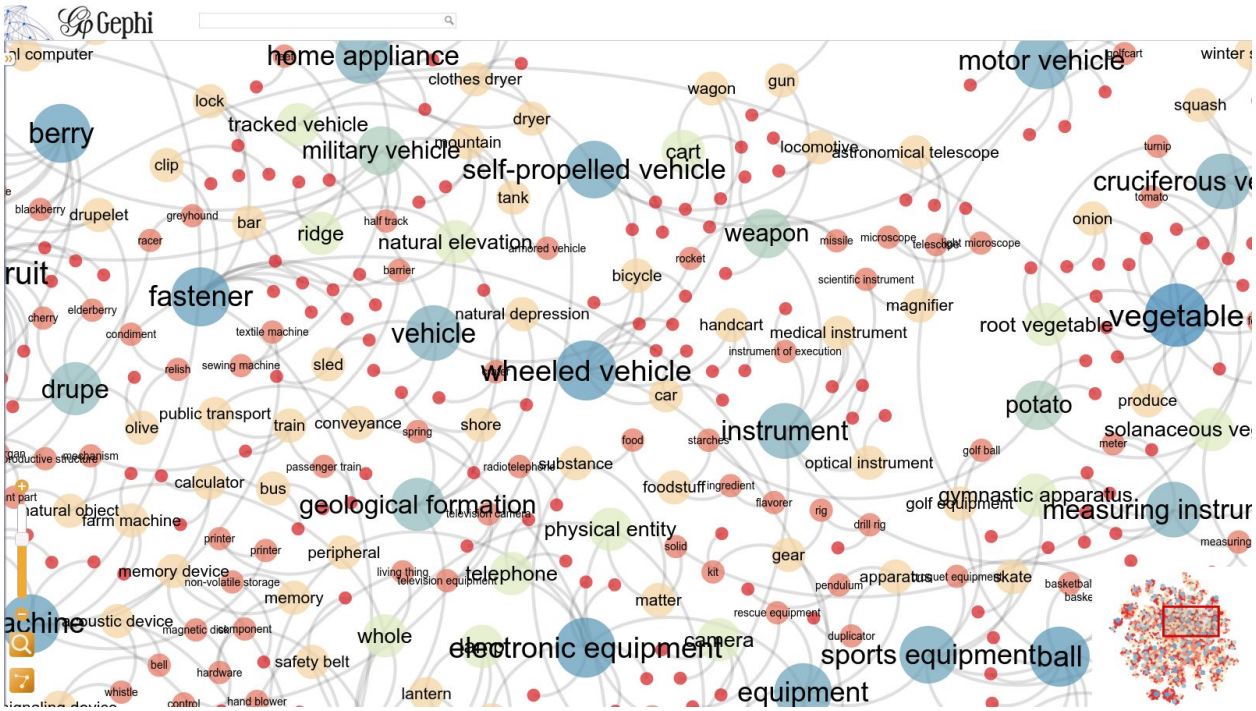
Gray Image

Thresholded Image

At times, double counting was taking place as one box containing a car was completely contained in another box. We took care of all such cases while evaluating the final car count.

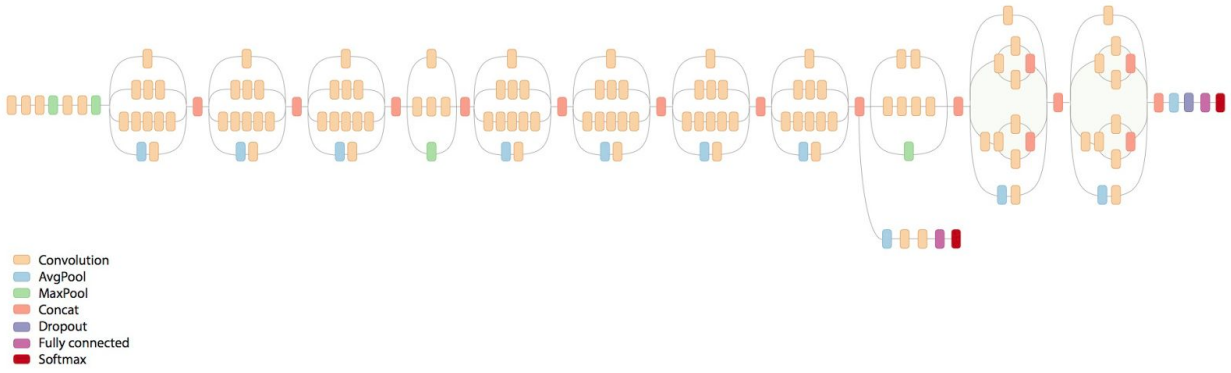
For the latter part, that is, for finding out if an image region represents a car or not, we have used convolutional neural network. We used the following for it:

- **Using imagenet inception model :** ImageNet is a common academic data set in machine learning for training an image recognition system. We used the **pre-trained inception** model trained on it. The output of the model is a probability distribution over the 1000 labels. By manually looking through the distance graph of these labels:

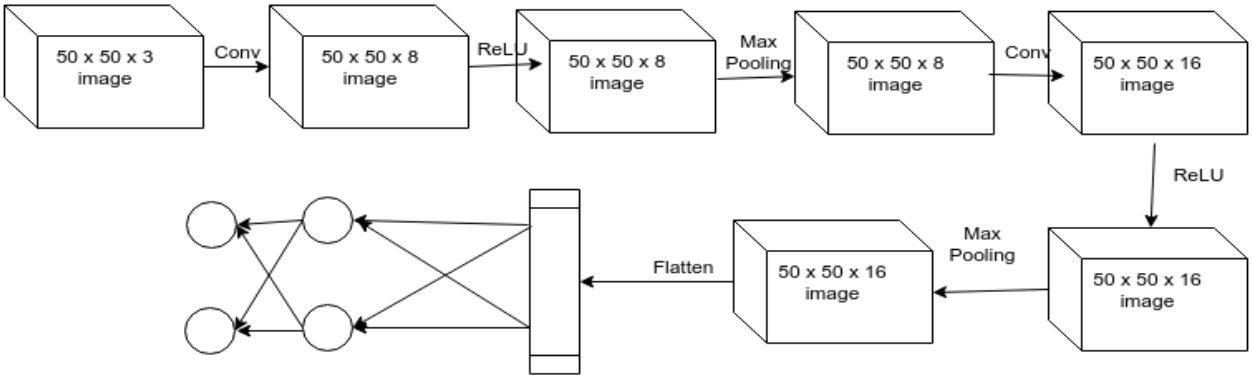


(Src: http://lear.inrialpes.fr/~akata/gexf-js/index2.html)

We manually compiled a list of labels which are related to vehicles, as the figure above shows. We decide that an image is a car if it is among the top 3 predictions of the output probability distribution. Since this model classifies over thousands of objects, it is quite large and takes a lot of time to give results. Also it was not properly tuned to detect cars. One run of our code was taking around **15 seconds per image** to give results. Using this model in the real-time application was not feasible and therefore we trained our own models. The complexity of this neural net is shown by its schematic:



- **Using keras**: keras is an excellent deep learning library in python. In this model, we first transformed each image to 50 x 50 x 3 image (3 is for RGB) and then applied algorithm on the image. This model gave about 95% accuracy. The architecture of this model is as described below:



- **Using Tensorflow**: We created a CNN model to predict if the given image is a car or not. The input was varied size images and their labels and we trained the CNN till 10,000 epochs. We also used a pre-trained model on ImageNet to get the type of various object in the given image and then choose the ones which are vehicle.

Code description

- We have used Python as the programming language
- We have used Tensorflow as the machine learning library for our project
- We also used keras as the machine learning library for one of our CNN models
- We used Opencv for edge detection
- Here is a snippet of one of our CNN algorithms for car identification (keras)

```
20  def CNN():
21      model = Sequential()
22      # 50 x 50 x 3
23      model.add(Convolution2D(8, 3, 3, border_mode='same',
24                              input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))
25      # 50 x 50 x 8
26      model.add(Activation('relu'))
27      model.add(MaxPooling2D(pool_size=(2, 2), border_mode='same'))
28      # 50 x 50 x 8
29      model.add(Convolution2D(16, 3, 3, border_mode='same'))
30      model.add(Activation('relu'))
31      model.add(MaxPooling2D(pool_size=(2, 2), border_mode='same'))
32      model.add(Flatten())
33      model.add(Dense(2))
34      model.add(Activation('softmax'))
35      model.compile(optimizer='adam',
36                    loss='binary_crossentropy',
37                    metrics=['accuracy'])
38      return model
39
```

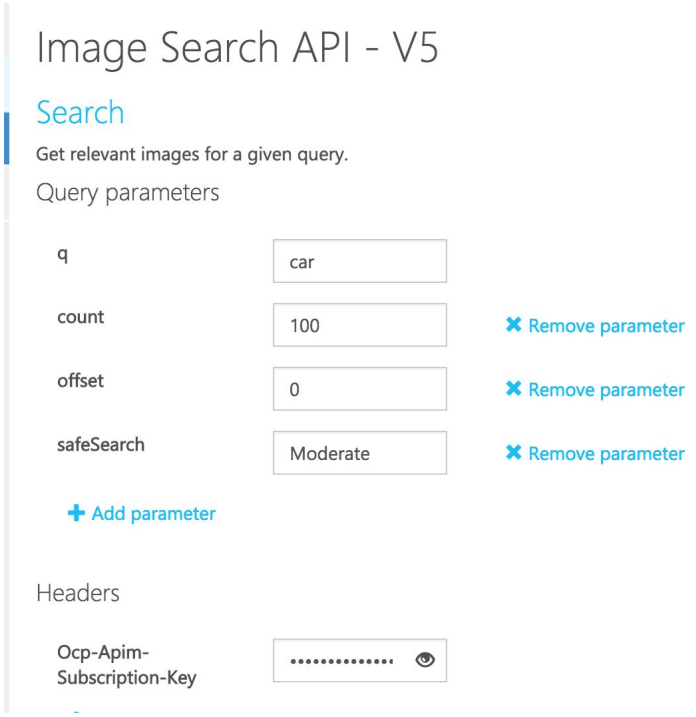
Keras helps developers build Deep learning algorithms quickly. As can be seen from the snippet above, we create a Sequential model and go on adding layers to it. We first add a Conv layer with a depth of 8 and filter size of 3 x 3. Then we add ReLU activation to it. Later a max-pooling layer is added with a pool size of 2 x 2. Note that the spatial dimensions of the network aren't changed and the size still remains 50 x 50. We Repeat this sequence of layers again, but this time we choose a depth of 16 for conv layer. Finally, we flatten the nodes of the network and add a fully connected layer with 2 nodes.

```
# Create model
def conv_net(x, weights, biases, dropout):
    # Reshape input picture
    x = tf.reshape(x, shape=[-1, 64, 64, 3])
    # Convolution Layer
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling)
    conv1 = maxpool2d(conv1, k=2)
    # Convolution Layer
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    # Max Pooling (down-sampling)
    conv2 = maxpool2d(conv2, k=2)
    # Fully connected layer
    # Reshape conv2 output to fit fully connected layer input
    fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
    fc1 = tf.nn.relu(fc1)
    # Apply Dropout
    fc1 = tf.nn.dropout(fc1, dropout)
    # Output, class prediction
    out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
    return out
# Store layers weight & bias
weights = {
    # 5x5 conv, 1 input, 32 outputs
    'wc1': tf.Variable(tf.random_normal([5, 5, 3, 32])),
    # 5x5 conv, 32 inputs, 64 outputs
    'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])),
    # fully connected, 16*16*64 inputs, 1024 outputs
    'wd1': tf.Variable(tf.random_normal([16*16*64, 1024])),
    # 1024 inputs, 10 outputs (class prediction)
    'out': tf.Variable(tf.random_normal([1024, n_classes]))
}
biases = {
    'bc1': tf.Variable(tf.random_normal([32])),
    'bc2': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}
```

We do something similar in tensorflow as we did in keras. However, we changed the filter size to 5 x 5 this time and we also used dropout regularizer to avoid overfitting. Also, the depth of conv layers is 32 and 64 respectively. We then also add a fully connected layer to the network.

- ## Datasets Creation:

We used Bing API for image search.



For training we require two types of images:

1. **Positive car images**: We searched "car aerial side view" to obtain images that are at a similar angle as those of traffic data. For eg.

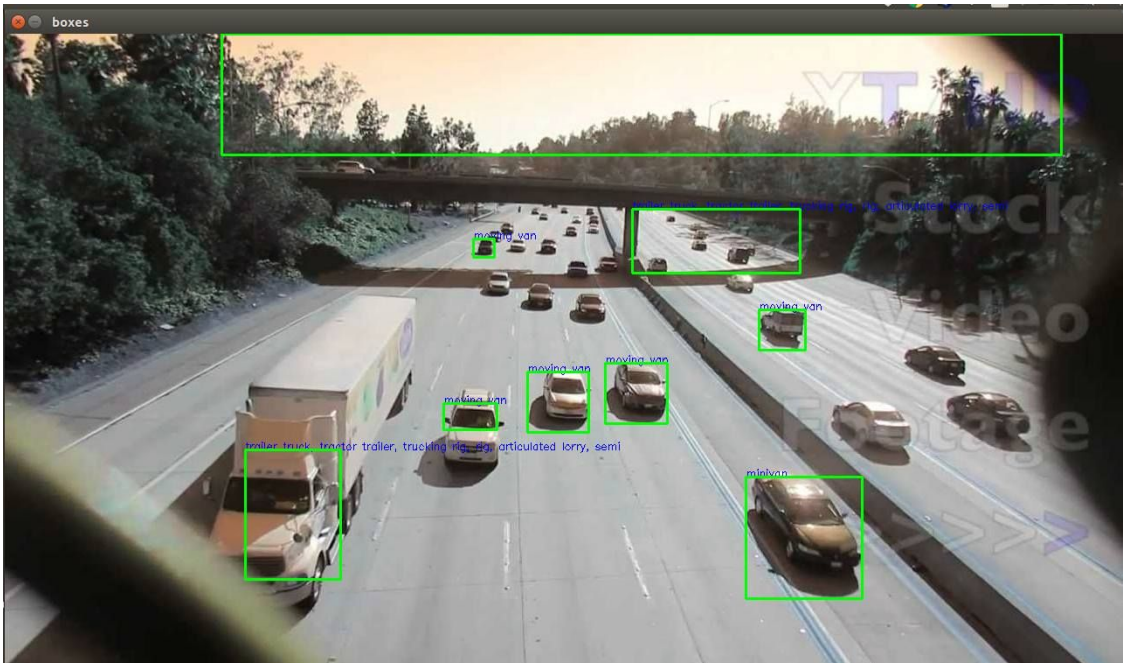
2. **Negative Car images**: For such training we binged "highway" to get empty highway images. We split these videos into 9 parts and then used them for training. The intuition for splitting is as follows. On edge detection, we obtain several boxes which contain the sky, greenery or just the highway. Therefore, training the CNN with these images should give it a better idea of our use case. Eg. The below image is broken down into 9 parts

Results

Recursion one was giving sensible results however had a lot of shortcomings mainly missing cars on boundaries and a large number of partitions. The edge detection method improved the results significantly. It properly detected boundaries and also reduced the number of partitions thus making the job faster.

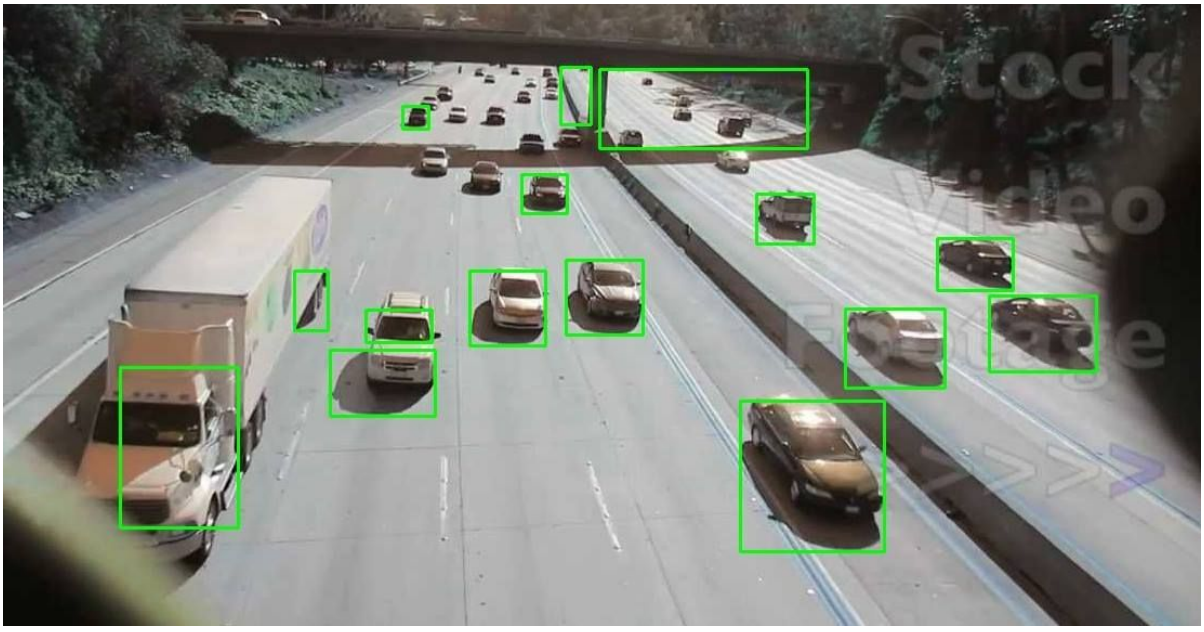
At the start we were using the inception model on imagenet to detect car in an image. It was taking around 10 minutes with recursion technique and 15s with edge detection.

Since the recursion method was very slow at this point, we switched to contour detection and bounding box method to find candidate patches in the image.



Result with inception model

Our own trained models took less than a second to give results and can be used in real-time analysis as well.



Result with our own model

Efforts

1. Time spent in different parts of the project

- Learning how to use a pre-trained model. We spent significant time to understand and use inception model over imagenet dataset. About a week was spent in understanding this.
- Dividing the image into parts to find all the cars. This was one of the challenging aspects and another week was dedicated to it.
- Coding and training our own CNN models took a final week.

2. Most challenging part

- Finding a suitable dataset to train our model.
- Exploring ways to find the most probabilistic parts of the images where we need to apply the model to detect the car. Another aspect was to make it more efficient because our final aim was to perform real-time analysis.

3. Work division

Equal work division for all 4 team members

- Karan Vaidya (130050019) - 25%
- Shudhatma Jain (130050024) - 25%
- Anmol Arora (130050027) - 25%
- Aman Goel (130050041) - 25%

Real-time Analysis

We also implemented the car detection on videos. Our CNN model is fast enough to process images in real time. It detects cars in the videos and shows the images changing

in every second. Results with real time analysis :

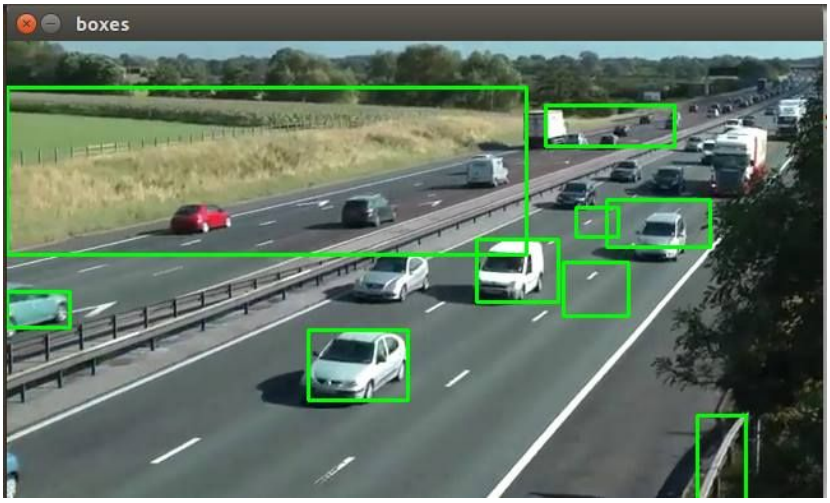
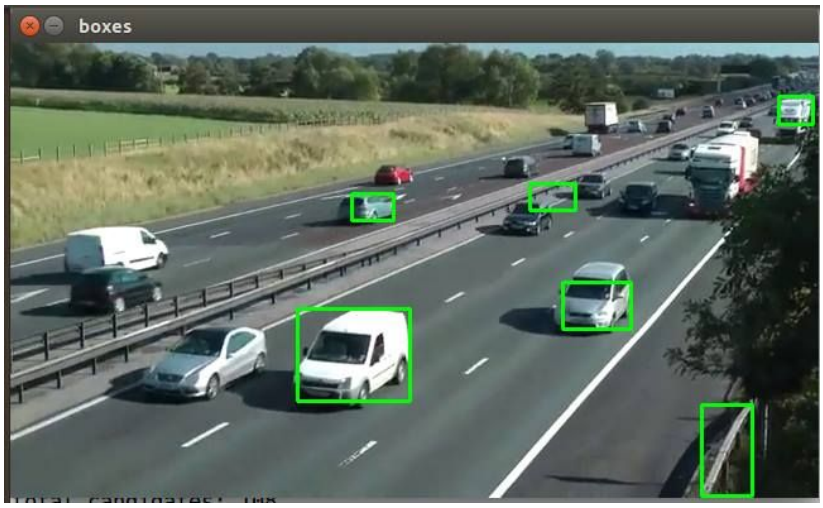


Image after 1 second :



Scope of Future Work

- Better edge detection techniques
- Including the car count in the CNN itself
- Using the car count to build a complete platform for real time traffic management