Machine Learning Engineer Nanodegree

Harshith Mullapudi
November 22nd,2017

Definition

Project Overview

Introduction

People have used Machine Learning in many places to solve many problems by automating systems. From object detection to data analysis and automation in various places.

Scope of this project

We will use machine learning in object detection, specifically to predict cars that is detecting cars in the given image or a video. We will only deal with the cars and not with any other vehicles.

Why this domain?

1. Firstly, there are lot features to be considered if we go with feature extraction such as hog etc.
2. There are lot interesting works coming such as automated cars, analysing speed of cars using computer vision techniques but these require a supervisor to check and is computational expensive.
3. Using Machine Learning scientists and researchers have brought a lot effect as there are lot experiments being done in this area of object detection.

Aim of the project

The aim of this study is to understand the detection of cars in the given image or a video. Performing many models and understanding the performance and accuracy of the detection. Predicting accurately the cars with the exact body is difficult but managing to identify the least rectangle that it fits to and making it efficient is an open challenge to top researchers.

Data used in this project

The dataset consists of total 7325 images of both car and no-car. Both the dataset consists of sub datasets such as

- Far
- Left
- MiddleClose
- Right

This dataset is taken from Vehicle Image Database and is used for our experiment.

Problem Statement

Problem

Build a model to detect the car.

Category	Details
Input	64*64 images of both car and non-car
Output	A binary prediction of presence of car

Challenges

1. The image will not only consist car there will be other objects which may conflict the features of the car thus a model for accurate prediction of the car is hard to make.
2. There is less dataset and not more are from different angles thus generalizing this and identifying car in big image is tough.

Analysis of problem

This is a classification problem because we are prediction the presence of car in the image and this is a binary classification that is it has 1 class “yes” or “no”

If the images are not in all angles and images are always from the back then the neural network doesn’t understand a car which has a front view also if the image is lower brightness and car is not visible then also there would be a problem for prediction. Thus it is very important that our dataset covers all such characteristic images also it is very tough to say just by seeing which model does best.

Characteristic of problem:

- 64*64 image
- Has all far, left, middle close, right images
- Classification problem
- There are lot features to be considered and this is taken in neural network
- Total 7325 images

Strategy

Exploring the dataset and try with a normal machine learning algorithm such as svm, classification algorithms etc.

Then try using a basic CNN model then try increasing and decreasing the number of layers and the size of the layers by checking the accuracy of the prediction. Finally, saving the best model and reusing it.

Expected Solution

The solution will predict the car in a given image, sequence of images and a video.

Metrics

We will measure performance as the percentage of correct predictions for a given test set or say it as a score of test set predictions.

We will divide the dataset into three parts train, validation, test. Both train and validation are used in training model and finally use the test set to get the performance of the model.

Analysis

Data Exploration

The primary dataset contains images of 2 classes Car, Nocar

Image is a .png and size 64*64

First class of cars have 4 categories

- Far – 975 images
- Left – 975 images
- Middle close – 500 images
- Right – 500 images

Second class of nocar have 4 categories

- Far – 975 images
- Left – 975 images
- Middle close – 975 images
- Right – 975 images

Exploratory Visualization

One of each kind are displayed here.

We can see that they are perfectly classified and the images include small images from a big one as we can observe the 4th image consists only the bridge side.

Algorithm and techniques

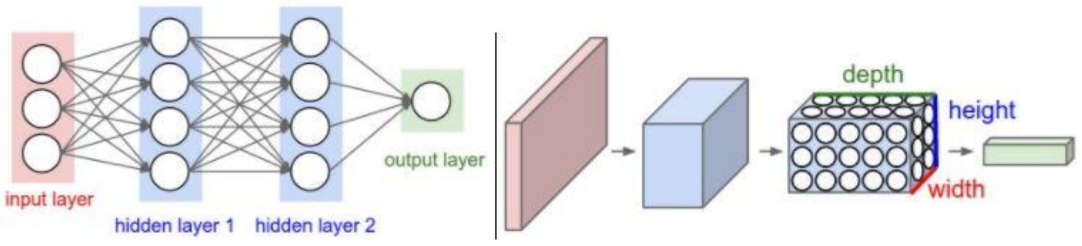
Algorithm

I will use Convolution neural network.

Algorithm Description

Convolution Neural Networks are very similar to ordinary Neural Networks, which are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function; from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

So what does change? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architectures. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.



Algorithm Justification

1. I am using Convolution neural network because this helps in solving more complex problems and also can again be used in learning such as transfer learning.
2. A CNN also works great in classification problems and researchers have solved lot problems using this CNN's

Algorithm Parameters

There are 5 major parameters

- **Validation_split:** Float between 0 and 1: Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last sample in the x and y data provided, before shuffling.
- **Epochs:** Integer. Number of epochs is to understood as initial_epoch, epochs is to understood as “final epoch”. The model is not trained for a number of iterations given by epochs, but merely until the epoch of index epochs is reached.
- **Batch_size:** Integer or None. Number of samples per gradient update. If unspecified, it will be default to 32.
- **Verbose:** 0,1 or 2. Verbosity mode.0=silent, 1=progress, 2=one line per epoch
- **Callbacks:** List of keras.callack.Callback instances. List of callbacks to apply during training.

Techniques

Train-test split

a. We will train our model on what we’ll call the training set, a subset of the data that we have.
b. To make it more generalized we have to have a set on which we can test the accuracy or how well our model has performed.
c. To do such activities we have test dataset.
d. Also we will have validation set which will act as a test set while the model is being trained.
e. We will use sklearn’s train_test_split function ehich automatically shuffles the data.

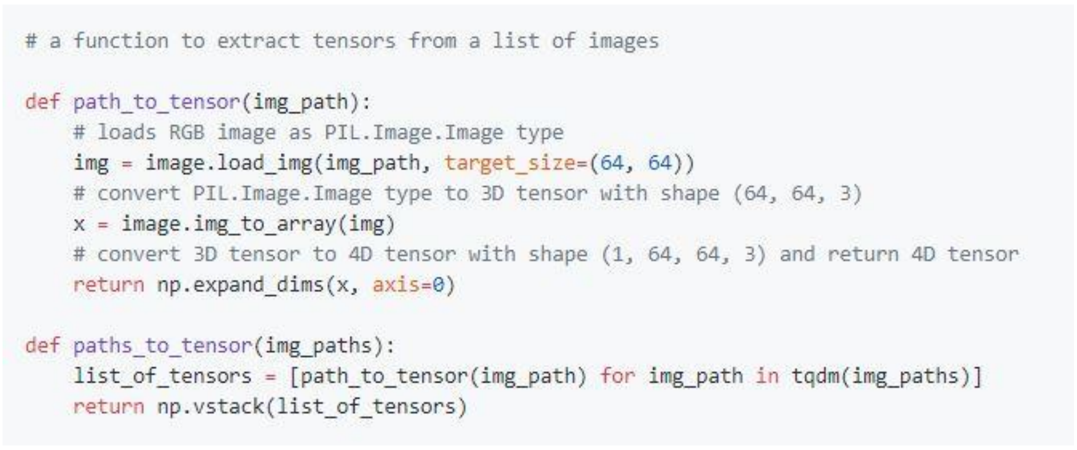
Benchmark

The benchmark given in the project is to detect the car and place it in the rectangle. Thus our benchmark will not have any such root mean squared etc but its just how well we can get the object detected.

Methodology

Data preprocessing

We have to open the images and get the image matrix into a train numpy array this can be done using two functions.

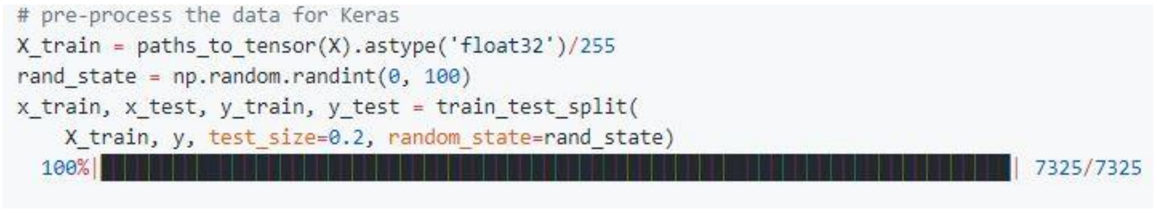


```
# a function to extract tensors from a list of images

def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(64, 64))
    # convert PIL.Image.Image type to 3D tensor with shape (64, 64, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 64, 64, 3) and return 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

Where img_paths will be the an array of all paths to the images and after which img_path will be a single element of the array that is path of a single image. The respective image will be opened as 64*64 image and the image matrix is returned and finally all such are stacked to a x_train array

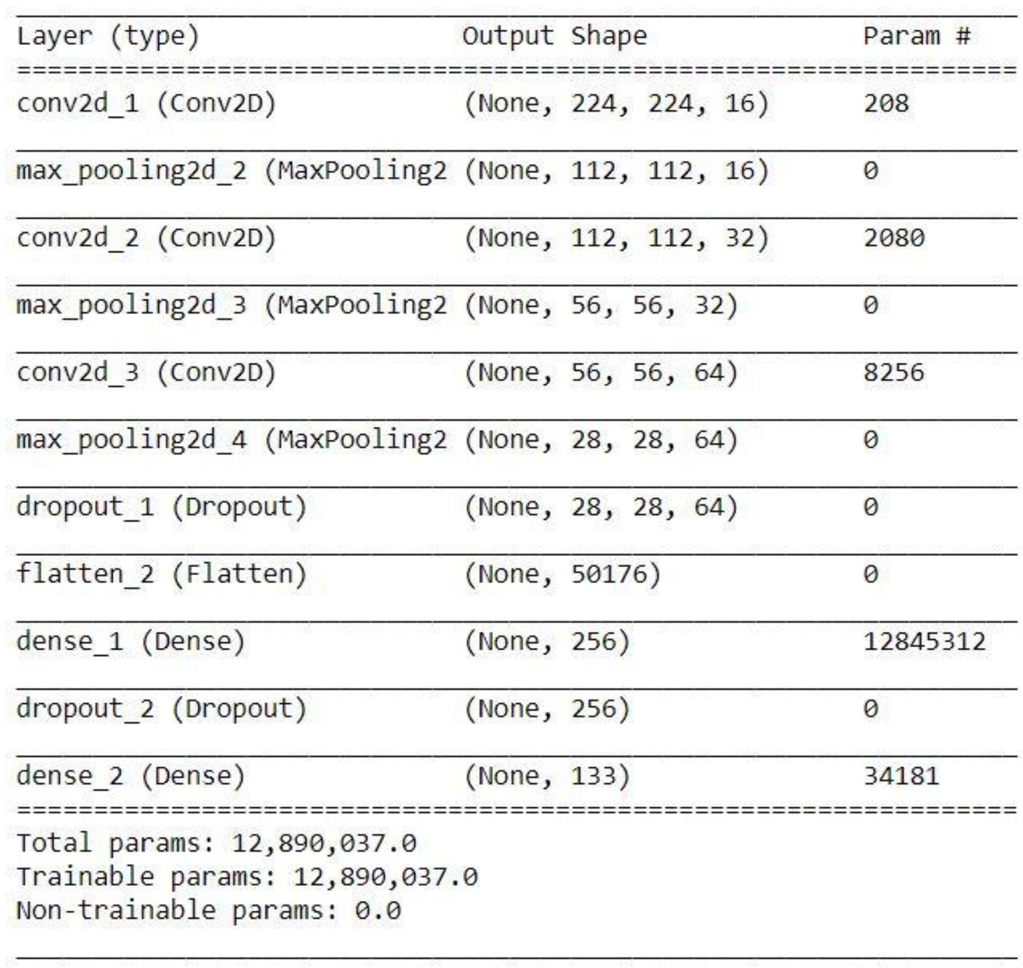


```
# pre-process the data for Keras
X_train = paths_to_tensor(X).astype('float32')/255
rand_state = np.random.randint(0, 100)
x_train, x_test, y_train, y_test = train_test_split(
    X_train, y, test_size=0.2, random_state=rand_state)
  100%|████████████████████████████████████████████████████| 7325/7325
```

Initial Implementation

The initial implementation of CNN’s was a normal CNN with a 5 layered one where the loss value will be very high and thus will not generalize the problem.

Thus it has to refined and refined by adding and deleting both the layers and the size of each layer



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 28, 28, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               12845312
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               34181
=================================================================
Total params: 12,890,037.0
Trainable params: 12,890,037.0
Non-trainable params: 0.0
_________________________________________________________________
```

There are lot changes and lot made have checked for lot models but most of them would give us less percentage of accuracy and the loss would be very high.

Test accuracy: 48.8765%

And the loss would be >4

Here process would be:

1. Construct X_train using function such as path_to_tensors and path_to_tensor and all the values are divided by 255 with float value.
2. Split the respective X_train into 2 parts of which 20 percent is of test set and 80 percent is of training set with random_state of random value.
3. While training model using model.fit() we use validation_set as a metric to reduce both loss and val_loss is calculated for every iteration and using all these it is trained.

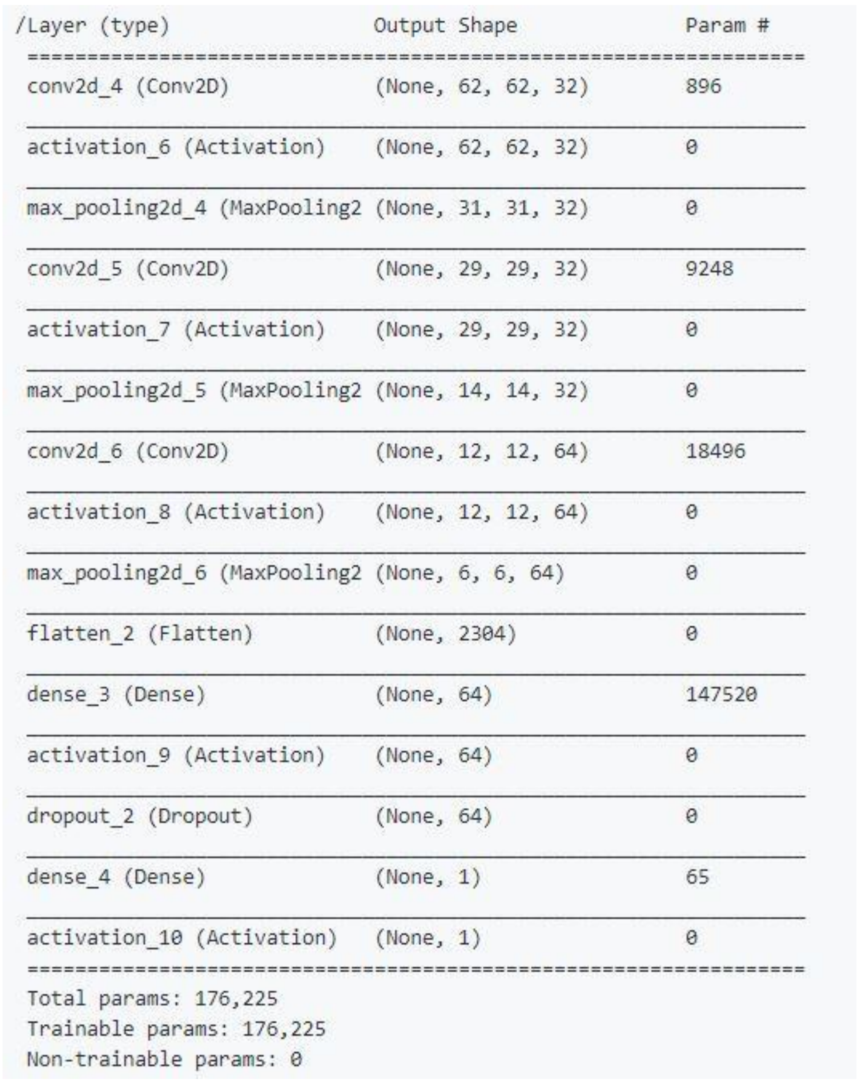
4. Finally using model.predict() function we will predict the presence of car or nocar as a binary value.

Refinement

This refinement can be done in 2 ways

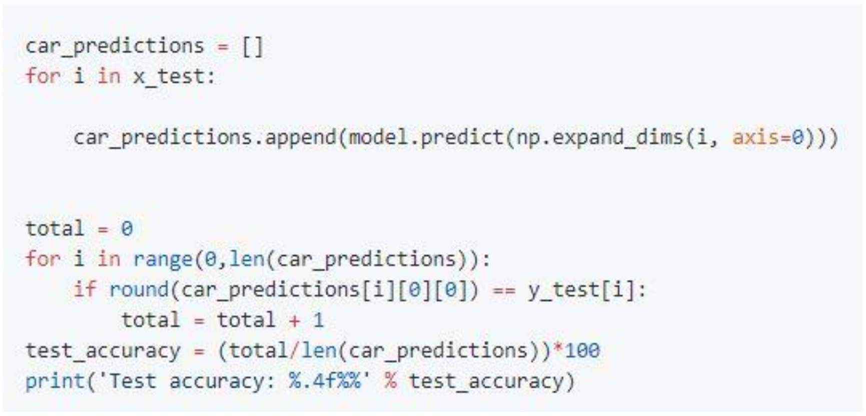
1. Either increasing the layers in the model
2. Increasing the size of each layer

After doing all such activities and many iterations I have come up with a new model



```
/Layer (type)                 Output Shape              Param #
 =================================================================
 conv2d_4 (Conv2D)            (None, 62, 62, 32)        896
 _________________________________________________________________
 activation_6 (Activation)    (None, 62, 62, 32)        0
 _________________________________________________________________
 max_pooling2d_4 (MaxPooling2 (None, 31, 31, 32)        0
 _________________________________________________________________
 conv2d_5 (Conv2D)            (None, 29, 29, 32)        9248
 _________________________________________________________________
 activation_7 (Activation)    (None, 29, 29, 32)        0
 _________________________________________________________________
 max_pooling2d_5 (MaxPooling2 (None, 14, 14, 32)        0
 _________________________________________________________________
 conv2d_6 (Conv2D)            (None, 12, 12, 64)        18496
 _________________________________________________________________
 activation_8 (Activation)    (None, 12, 12, 64)        0
 _________________________________________________________________
 max_pooling2d_6 (MaxPooling2 (None, 6, 6, 64)          0
 _________________________________________________________________
 flatten_2 (Flatten)          (None, 2304)              0
 _________________________________________________________________
 dense_3 (Dense)              (None, 64)                147520
 _________________________________________________________________
 activation_9 (Activation)    (None, 64)                0
 _________________________________________________________________
 dropout_2 (Dropout)          (None, 64)                0
 _________________________________________________________________
 dense_4 (Dense)              (None, 1)                 65
 _________________________________________________________________
 activation_10 (Activation)   (None, 1)                 0
 =================================================================
 Total params: 176,225
 Trainable params: 176,225
 Non-trainable params: 0
```

Thus this model is the final model which would predict the presence of the car. Both the number of layers and the size are changed accordingly to reduce the loss value. After running model.fit() with the respective model we test this using the test set.



```
car_predictions = []
for i in x_test:

    car_predictions.append(model.predict(np.expand_dims(i, axis=0)))


total = 0
for i in range(0,len(car_predictions)):
    if round(car_predictions[i][0][0]) == y_test[i]:
        total = total + 1
test_accuracy = (total/len(car_predictions))*100
print('Test accuracy: %.4f%%' % test_accuracy)
```

Thus giving the accuracy to be 97.1331%. which is good sign that the percentage is above 90 percentage. But this alone cannot decide our solution. As we have trained on only small images what we do we apply a sliding window algorithm on the image and pass the images into the model to detect if the image is car or not for that we have taken the help of opencv functions and pyimagesearch and made our own function to detectcars.

```
def detectcars(imaged):
    (winW, winH) = (128, 64)
    for (x, y, window) in sliding_window(imaged, stepSize=10, windowSize=(winW,
winH)):
        # if the window does not meet our desired window size, ignore it
        if window.shape[0] != winH or window.shape[1] != winW:
            continue
        clone = imaged.copy()
        clone = cv2.resize(clone[y:y+winH,x:x+winW],
(64,64)).astype('float32')/255
        pre = model.predict(np.expand_dims(clone, axis=0))
        if pre[0][0] == 1.0:
                final.append({
                    "x" :x,
                    "y":y,
                    "winH" : winH,
                    "winW": winW,
                    "pred" : pre[0][0]
                })
        clone = imaged.copy()
    for i in final:
        cv2.rectangle(clone, (i['x'], i['y']), (i['x'] + i['winW'], i['y'] +
i['winH']), (0, 255, 0), 2)
    return clone
```

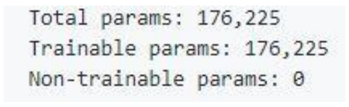
This function will help us in applying sliding window concept which means a rectangle will pass through all the image and the size of the rectangle is decided above as winH and winW as 128*64

Thus on whole finally a model which performs better compared to a model with which we started our project and there is a lot improvement from 48 percent to 97 percent thus concluding that the problem has found a generalized solution according to given dataset.

Results

Finally, the model is chosen with

Parameters (total):

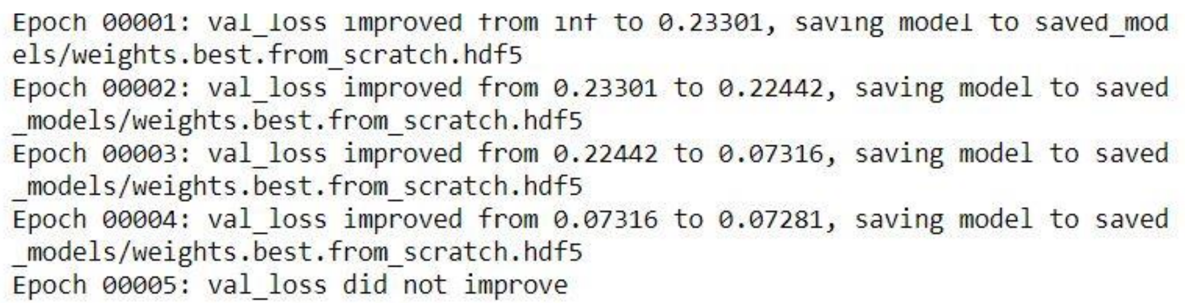


```
Total params: 176,225
Trainable params: 176,225
Non-trainable params: 0
```

Target: predicting the presence of the car. The model has a accuracy percentage of 97 percentage.

Insight: The addition of layers or size, what I thought will make the model more awesome and perfect made it more worse. Understood that as we increase anything it becomes more complex and the data becomes underfit or overfit if we use less size.

Generalizability

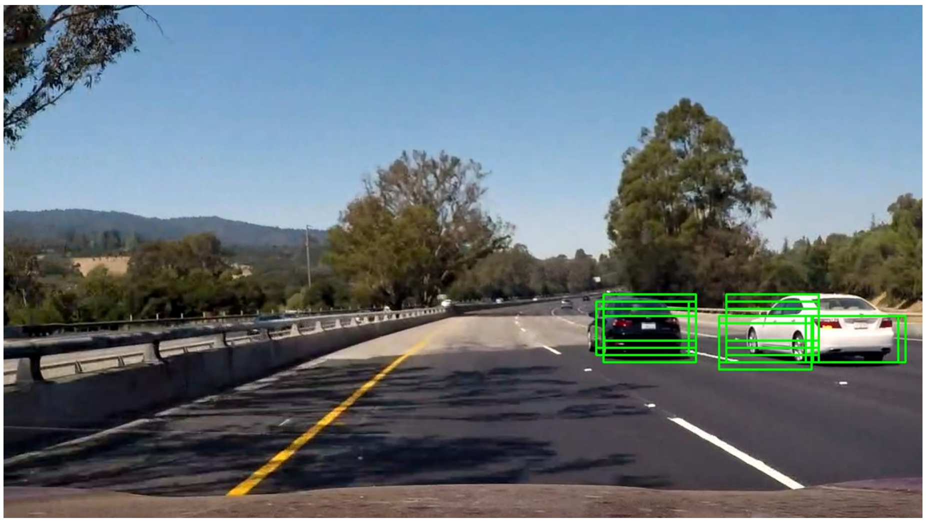


```
Epoch 00001: val_loss improved from inf to 0.23301, saving model to saved_mod
els/weights.best.from_scratch.hdf5
Epoch 00002: val_loss improved from 0.23301 to 0.22442, saving model to saved
_models/weights.best.from_scratch.hdf5
Epoch 00003: val_loss improved from 0.22442 to 0.07316, saving model to saved
_models/weights.best.from_scratch.hdf5
Epoch 00004: val_loss improved from 0.07316 to 0.07281, saving model to saved
_models/weights.best.from_scratch.hdf5
Epoch 00005: val_loss did not improve
```

This is performance metric. When we evaluated the model in the previous section, each iteration of the model was run on training and test sets.

Conclusion

Result Visualization



This is the image output of the model that is trained and after applying sliding window we plot the rectangle where we get the prediction as 1

Reflection

In this project, we predicted the presence of car

Initially we started with a small neural network which gave bad results and after multiple iterations we have got a model which has a good accuracy and predictions.

In the process

- Import the images
- Get the tensors from each image using respective functions
- Create the training dataset

- Divide the dataset into 2 parts
- Train on the training dataset with shuffling and having validation set as 20 percent
- Predict the results
- Evaluate the test set
- Take an image apply sliding window and get the output image and display it

This process was done for all in between model while finding the right one trying different layers and increasing and decreasing the size of layers.

Interesting Aspects

- Making a new model from scratch for the given dataset was way interesting and way challenging.
- Reading multiple articles for to choose the best model and making changes in the model.
- After all done and making the model complex and as a last option getting back to small one where that works amazingly awesome which teaches us that keep things simple.

Difficult aspects

- It was hard to make the neural network and deciding the right one.
- At some point leaving everything as no network works perfectly.
- With lot parameters operating choosing different ones was a difficult task.

Improvement

- Collecting more dataset of more types and from more angles and in different climatic conditions would improve the algorithm.
- Trying more models may result in different learning and more accurate results.

Things to explore

- Using the model to predict the structure of the car and removing it from the image.

- Training the same model for other classification using transfer learning
- Training it in classification of cars brand names or car model names.