CAPSTONE PROJECT – COMPUTER VISION

BOUNDING BOX REGRESSION AND MULTICLASS CLASSIFICATION OF STANFORD CAR IMAGES

Submitted by - AIML Batch-B Jan, 21

Members - Ambika, Kishore, Nitya, Rohith, Sushweta

Summary of problem statement, data and findings

The **Stanford Cars** dataset consists of 196 classes of cars with a total of 16,185 images. The data is divided into almost a 50-50 train/test split with 8,144 training images and 8,041 testing images. Categories are typically at the level of Make, Model, Year. The images are of varying dimensions

The Objective of Multi-class object detection, implies that we are trying to (1) detect where an object is in an input image [Bounding box] and (2) predict what the detected object is [Class].

Data description:

The dataset directory contains two subdirectories, annotations (Train and Test) and images (Train and Test) and a dictionary file.

Train Images: Consists of real images of cars as per the make and year of the car.

Test Images: Consists of real images of cars as per the make and year of the car.

Train Annotation: CSV file consists of bounding box region for training images.

Test Annotation: CSV file consists of bounding box region for testing images.

Car Name and Make csv file consists of dictionary mapping between class number and class name.

Project File Structure

- A configuration settings and variables file.
- Our training script which will load our images and annotations from disk, modify the Model architecture for bounding box regression, fine-tune the modified architecture for object detection, and finally populate the output/directory with our serialized model, training history plots, and test image filenames.
- Prediction script performs inference using our trained object detector. This script will load our serialized model and label encoder, loop over our testing images, and then apply object detection to each of the images.

- <To do>

Summary of the Approach to EDA and Pre-processing

- We analyzed the dimensions and count of images per class
- Class having maximum number of images – GMC Savana Van 2012
- Class having minimum number of images – Hyundai Accent Sedan 2012
- Sample listing of a class

```
# few statistical analysis
train_images.loc[train_images['car'] == 'GMC Savana Van 2012']
```

	Unnamed: 0	image	image_matrix	filename	car	x0	y0	x1	y1	label
4822	4822	[34 36 48 ... 105 102 101]	[[34 36 48]\n [55 67 95]\n [58 71 97]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	12	17	170	79	119
4823	4823	[218 214 205 ... 177 176 174]	[[218 214 205]\n [215 211 202]\n [216 212 203]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	4	9	165	82	119
4824	4824	[93 105 118 ... 230 228 216]	[[93 105 118]\n [93 105 117]\n [93 106 115]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	12	14	164	90	119
4825	4825	[254 255 255 ... 255 255 255]	[[254 255 255]\n [254 255 255]\n [254 255 255]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	15	31	149	81	119
4826	4826	[56 56 56 ... 108 108 108]	[[56 56 56]\n [56 56 56]\n [56 56 56]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	25	10	167	77	119
...	...	...	...	...	...	...	...	...	...	...
4885	4885	[210 210 210 ... 38 38 36]	[[210 210 210]\n [209 209 209]\n [206 206 206]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	18	26	139	85	119
4886	4886	[240 252 254 ... 83 82 75]	[[240 252 254]\n [243 253 254]\n [253 255 255]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	22	13	148	89	119
4887	4887	[207 215 218 ... 92 82 64]	[[207 215 218]\n [211 217 220]\n [211 218 220]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	21	12	147	88	119
4888	4888	[255 255 255 ... 136 136 136]	[[255 255 255]\n [255 255 255]\n [255 255 255]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	8	8	163	88	119
4889	4889	[254 254 253 ... 255 255 255]	[[254 254 253]\n [255 255 254]\n [255 255 255]...	/content/drive/MyDrive//Dataset/Car Images/Tra...	GMC Savana Van 2012	6	16	165	80	119

68 rows × 10 columns

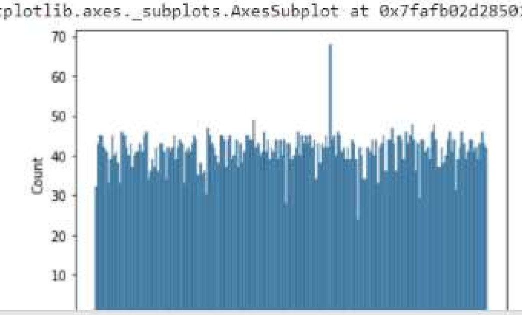
[98]

```
print("car counts:\n")
val_count = train_images['car'].value_counts()
print(val_count)
```

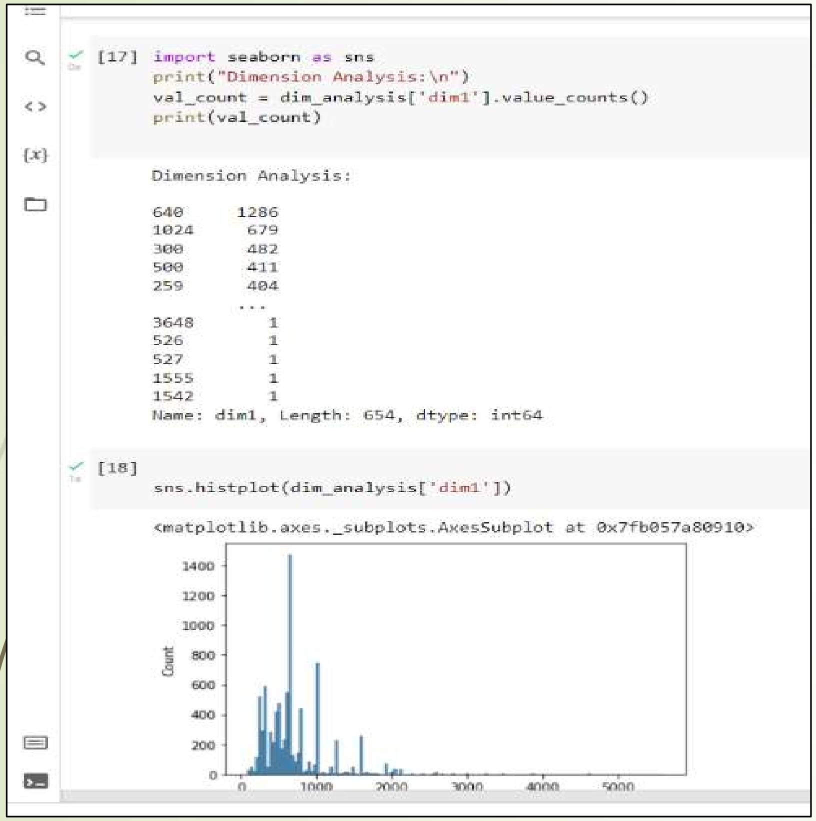
```
car counts:

GMC Savana Van 2012                                  68
Chrysler 300 SRT-8 2010                              49
Mitsubishi Lancer Sedan 2012                         48
Mercedes-Benz 300-Class Convertible 1993             48
Jaguar XK XKR 2012                                   47
                                                     ..
Rolls-Royce Phantom Drophead Coupe Convertible 2012  31
Chevrolet Express Cargo Van 2007                     30
Maybach Landaulet Convertible 2012                   29
FIAT 500 Abarth 2012                                 28
Hyundai Accent Sedan 2012                            24
Name: car, Length: 196, dtype: int64
```

```
sns.histplot(train_images['car'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fafb02d2850>



Maximum and Minimum Size of the image available in dataset



Checking bounding boxes on the image

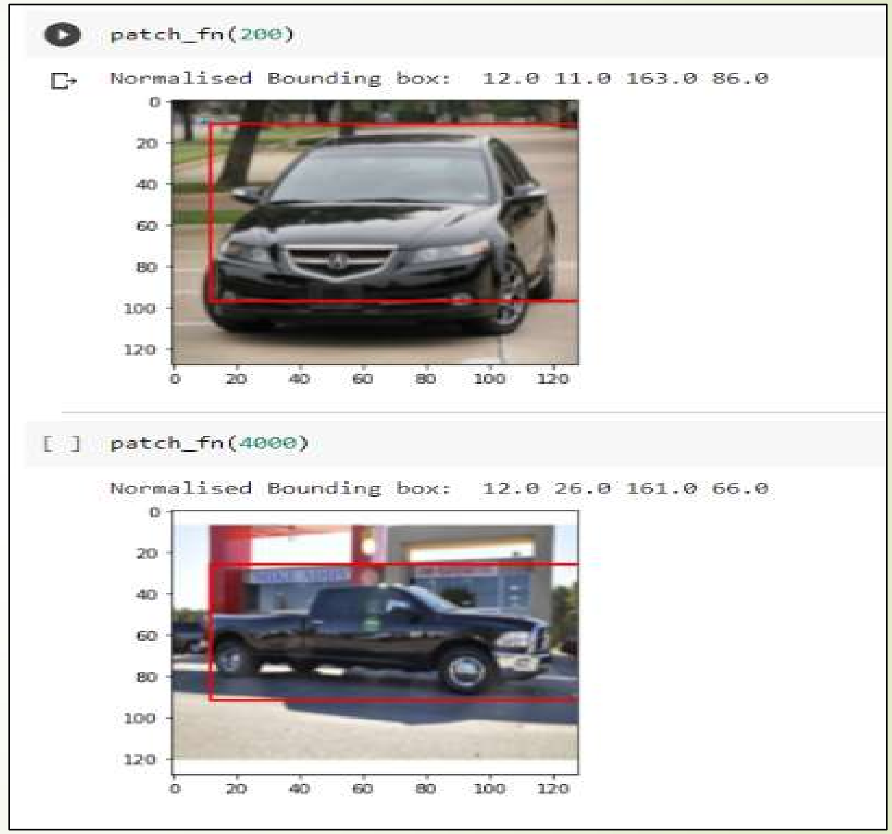


Image Pre-processing

Creating dataset from images folder

- The images subdirectory contains all images in our dataset, with a corresponding subdirectory for the name of the label.

We start with loading the Car Name and Class Name into a pandas dictionary object.

Loading Annotations file in a pandas Dataframe object. Each row in Annotations file consists of consists of six elements:

1. Image Filename
2. Starting x-coordinate
3. Starting y-coordinate
4. Ending x-coordinate
5. Ending y-coordinate
6. Class label number

Looping over our CSV annotation files, we grab all rows in the file and proceed to loop over each of them.

Inside our loop, we unpack the comma-delimited row giving us our filename, (x, y)-coordinates, and class label for the particular line in the CSV. We lagain oop through the annotations dataframe and update new columns for Class Name and build Image path name. Using the imagePath derived from our config, class label, and filename,we load the image and extract its spatial dimensions.

We then scale the bounding box coordinates relative to the original image's dimensions to the range [0, 1] — this scaling serves as our preprocessing for the bounding box data.

Finally we load the image from disk in Keras/TensorFlow format and preprocess it with a resizing step which forces the image to 224×224 pixels for input for model.

Next step is to One-hot encode our labels using LabelBinarizer.

Deciding Models and Model Building

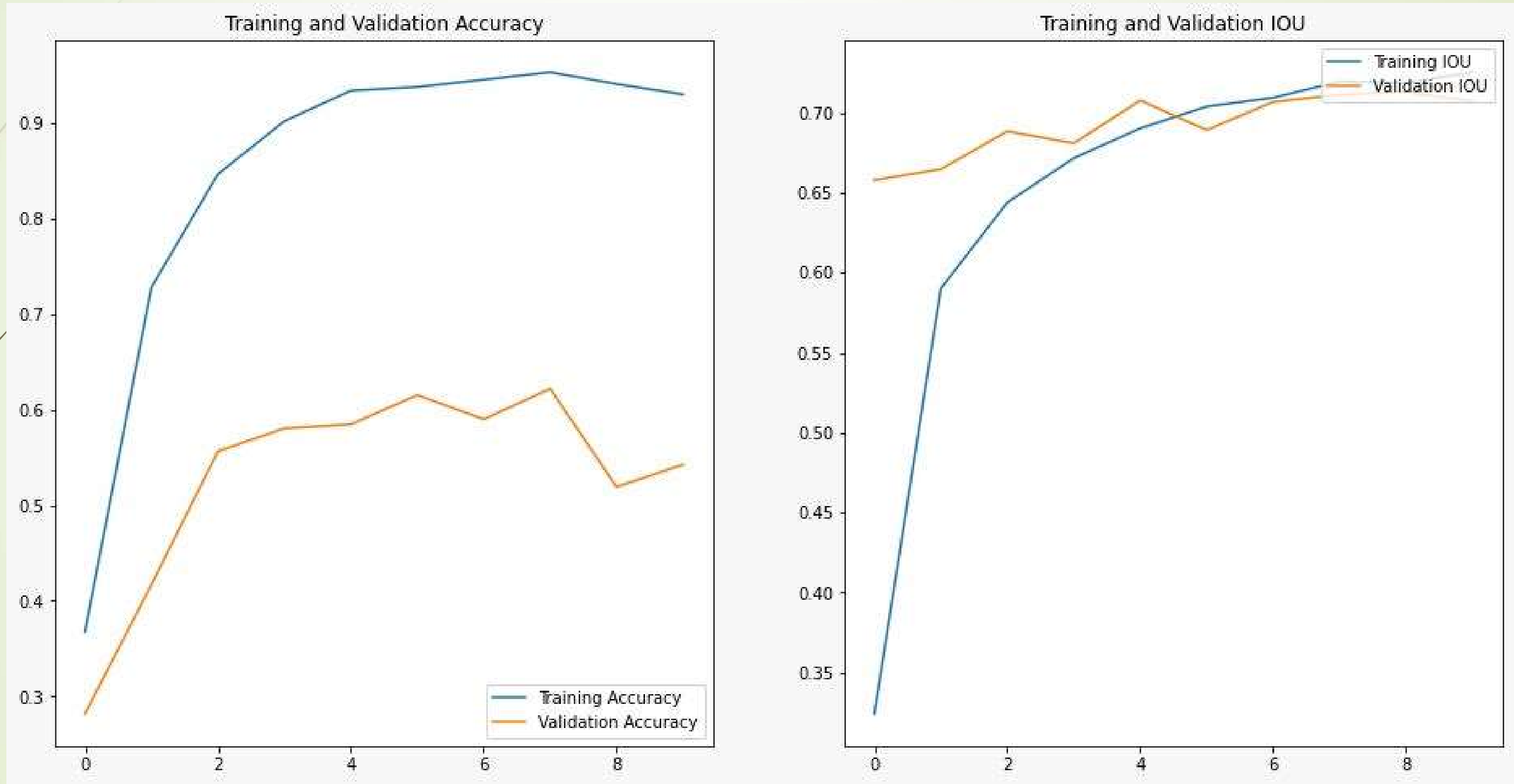
- The problem statement states that it is a Automotive. Surveillance problem and the context states that computer vision can be used to automate supervision and triggering events for images of interest.
- By understanding these statements, we can conclude that the problem can be solved by using deep learning techniques. In Deep learning techniques, we can look into particular sections of convolution neural networks (CNNs). Convolutional Neural Networks (CNN or ConvNet) are complicated feed forward neural networks used in machine learning. Because of its great accuracy, CNNs are employed for image classification, image localization , image detection , etc. The CNN uses a hierarchical model that builds a network, similar to a funnel, and then outputs a fully-connected layer in which all neurons are connected to each other and the output is processed. The benefit of using CNNs is their ability to develop an internal representation of an image by looking at only a subset of pixels in the images.
- We cannot use a Dense neural network for computer vision tasks in deep learning. The fundamental difference between the Convolutional and Dense layers is that the Convolutional layer requires fewer parameters because the input values are forced to share the parameters. The Dense Layer employs a linear operation, which means that the function generates each output based on each input. An output of the convolution layers is formed by just a small size of inputs which depends on the filter's size and the weights are shared for all the pixels

- For this problem statement we have decided to use transfer learning and use a pre-built model in tensorflow. Few layers in pre-built models can be trained to get good accuracy and best prediction for the bounding box. Transfer learning is adaptable, allowing pre-trained models to be used directly as feature extraction preprocessing or integrated into completely new models.
- We have evaluated a few pre-trained models - MobileNet, ResNet50, VGG and efficient net as part of the experiment. Among these evaluations we have got the best result for an efficient net - efficientnet-b5.

Model Building - Efficientnet-b5 model with ImageNet weights is used for developing network for classification and regression tasks for the given problem statement. The efficientnet-b5 model is one of the EfficientNet models designed to perform image classification. This model was pretrained in TensorFlow*. All the EfficientNet models have been pre trained on the ImageNet* image database. Below mentioned are result obtained after training the model.

Prediction of test image

Plot for Training vs Validation Accuracy and IoU



Classification report

Nissan NV Passenger Van 2012	0.93	0.86	0.89	44
Rolls-Royce Phantom Sedan 2012	0.56	0.53	0.55	43
Rolls-Royce Phantom Drophead Coupe Convertible 2012	0.71	0.71	0.71	41
Porsche Panamera Sedan 2012	0.88	0.55	0.68	38
Scion xD Hatchback 2012	0.64	0.70	0.67	30
Ram C-V Cargo Van Minivan 2012	0.77	0.68	0.72	44
Plymouth Neon Coupe 1999	0.70	0.68	0.69	41
smart fortwo Convertible 2012	0.80	0.87	0.83	45
Toyota Camry Sedan 2012	0.80	0.67	0.73	42
Toyota Corolla Sedan 2012	0.68	0.66	0.67	38
Tesla Model S Sedan 2012	0.74	0.67	0.70	46
Spyker C8 Convertible 2009	0.67	0.81	0.73	42
Spyker C8 Coupe 2009	0.55	0.53	0.54	40
Suzuki SX4 Sedan 2012	0.71	0.76	0.73	38
Suzuki Kizashi Sedan 2012	0.88	0.90	0.89	40
Toyota 4Runner SUV 2012	0.85	0.81	0.83	43
Suzuki SX4 Hatchback 2012	0.78	0.67	0.72	43
Suzuki Aerio Sedan 2007	0.73	0.84	0.78	38
Volvo XC90 SUV 2007	0.89	0.95	0.92	42
Volvo C30 Hatchback 2012	0.70	0.85	0.76	46
Volkswagen Beetle Hatchback 2012	0.66	0.88	0.75	43
Volvo 240 Sedan 1993	0.88	0.80	0.84	45
Volkswagen Golf Hatchback 2012	0.78	0.85	0.81	41
Volkswagen Golf Hatchback 1991	0.89	0.72	0.79	43
Toyota Sequoia SUV 2012	0.95	0.95	0.95	40
accuracy			0.74	8041
macro avg	0.75	0.74	0.74	8041
weighted avg	0.75	0.74	0.74	8041

How to improve your model performance?

- ***Data Augmentation***

  - One of the ways of improving the model performance is to train the model on more images. The training of deep learning models usually necessitates a large amount of data. In general, the more data there is, the better the model will perform. The issue with a paucity of data is that our deep learning model may not be able to learn the pattern from the data, and so may result in poor performance on test data.

  - Instead of collecting more data, augmentation techniques can be applied to generate as much data as required. Some of the commonly used augmentation techniques are rotation, shear, flip, etc. While applying data augmentation techniques for the given problem statement, we have to modify the bounding box coordinates as well. For this purpose, we can make use of imgaug or chitra libraries which are readily available and apply few inbuilt functions on images for data augmentation.

One of the augmenter which is available in the library is Affine. Affine transformations involve Translation (“move” image on the x-/y-axis),Rotation, Scaling (“zoom” in/out), Shear (move one side of the image, turning a square into a trapezoid). This Augmenter will affect bounding boxes and hence we need to use BoundingBoxesOnImage function on the bounding box coordinates.

Horizontal flips, Resize and change brightness can also be applied on the images and these function can be wrapped in function which can then be applied which using batch generator for the images.

- ***Transfer learning***
  - Unfreezing a portion of a model and retraining it with a very low learning rate on the new data would give significant improvement in model accuracy. Since the dataset which is provided to us has 8144 images and data similarity is quite low, freezing initial layers of the pretrained model and re-train just the remaining layers will help in accuracy improvement.
  - Here it is recommended to keep the learning rate very low since the layers are unfreezed and model weights are trainable. Because the training will be done on a larger model it's also crucial to utilise a very modest learning rate at this stage. There are substantial weight changes happening here and there is always a danger of overfitting. So incremental readjustment of the pre-trained weights are crucial.

- ***Last layer Classification and regression***
  - If the model is overfitting for given data, dropout layer can be introduced between last dense layers which will help in reduction of overfitting. Dropout is a training strategy in which randomly selected neurons are rejected. They are "dropped-out" randomly. This means that on the forward pass, their contribution to the activation of downstream neurons is removed temporally, and on the backward pass, any weight updates are not applied to the neuron.
  - Using Batch normalization helps in reduction of general errors. Batch normalisation is a technique for standardising network inputs that can be applied to either the activations of a previous layer or the inputs themselves. Batch normalisation reduces generalisation error by speeding up training (in some situations by halving or bettering the epochs) and providing some regularisation.