2021

AIML Capstone Project Computer Vision

Car Detection & Classification

Design a Deep Learning based car detection and identification model.
Clickable UI based interface to automate all tasks

Antara, Deepiga, Gowthaman, Manoj, Shashank
Great Learning
6/12/2021

Contents

Problem Statement

- **DOMAIN:** Automotive, Surveillance.

- **CONTEXT:**
  Computer vision can be used to automate supervision and generate action appropriate action trigger if the event is predicted from the image of interest. For example a car moving on the road can be easily identified by a camera as make of the car, type, color, number plates etc.

- **DATA DESCRIPTION:**
  The Cars dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split. Classes are typically at the level of Make, Model, and Year, e.g. 2012 Tesla Model S or 2012 BMW M3 coupe.

  1. Train Images: Consists of real images of cars as per the make and year of the car.
  2. Test Images: Consists of real images of cars as per the make and year of the car.
  3. Train Annotation: Consists of bounding box region for training images.
  4. Test Annotation: Consists of bounding box region for testing images.

Model Exploration Summary

Cars Classification:

- Custom CNN
  - Using max-pooling, dropout, batch-normalization, optimizer (Adam, SGD)
- ResNet50
  - Transfer Learning, pre-trained model with imagenet weights, top layers trainable
- VGG16
  - Transfer Learning, pre-trained model with imagenet weights, top layers trainable

Cars Localization/Detection:

- MobileNet
  - Transfer Learning, pre-trained model with MobileNet weights
- ResNet50
  - Transfer Learning, pre-trained model with imagenet weights
- VGG16
  - Transfer Learning, pre-trained model with imagenet weights, top layers trainable
- Faster RCNN
  - ResNet50 as backbone network, transfer learning, pre-trained model with all layers trainable
- Yolo
  - Custom backbone, pre-trained model with yolov2 weights and all layers trainable

EDA (Exploratory Data Analysis)

Data health check:

- Unique number of car labels 196

```
[18]: #Loading name of cars
      carNameDF= pd.read_csv("../input/stanford-car-dataset-by-classes-folder/names.csv",header=None)

[19]: carNameDF.shape

[19]: (196, 1)
```

- Total images in train data 8144
- Total images in test data 8041

- NaN check in annotation data
  - No NaNs are present in annotations data

```
train_anno_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8144 entries, 0 to 8143
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Image Name   8144 non-null   object
 1   x0           8144 non-null   int64
 2   y0           8144 non-null   int64
 3   x1           8144 non-null   int64
 4   y1           8144 non-null   int64
 5   Image class  8144 non-null   int64
dtypes: int64(5), object(1)
memory usage: 381.9+ KB
```

```
test_anno_df.info()
```

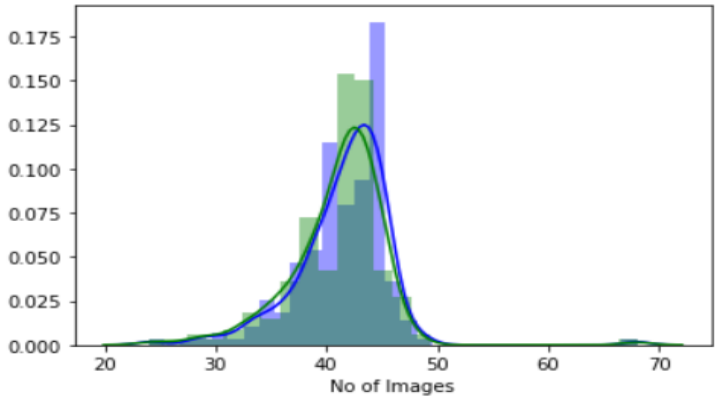
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8041 entries, 0 to 8040
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Image Name   8041 non-null   object
 1   x0           8041 non-null   int64
 2   y0           8041 non-null   int64
 3   x1           8041 non-null   int64
 4   y1           8041 non-null   int64
 5   Image class  8041 non-null   int64
dtypes: int64(5), object(1)
memory usage: 377.0+ KB
```

Data distribution check

- Class Imbalance:
  - Image distribution is skewed
  - Majority car types have image in range of 35-45
  - We have used data augmentation to increase data availability on few classes

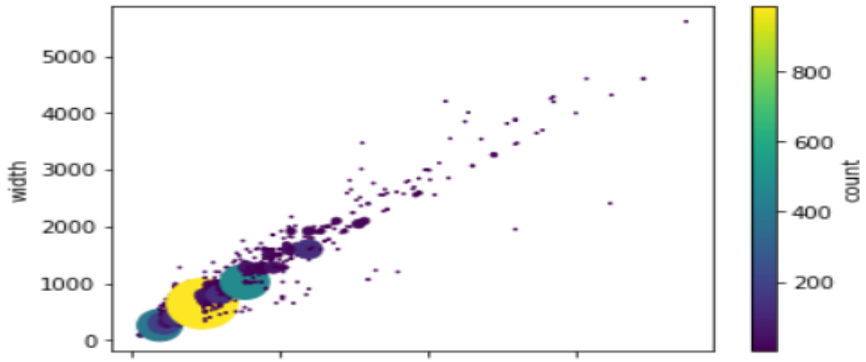
```
sns.distplot(df_overviewTrain['No of Images'],color='b')
sns.distplot(df_overviewTest['No of Images'],color='g')
```

```
<AxesSubplot:xlabel='No of Images'>
```



- Image size:
  - Highly volatile range of image size
  - Major concentration of images have size of 640x480
  - Resize images to sizes close to 600 to capture maximum features if model allows
  - Image sizes range from very small size in 50x50 to 5000x5000 as well
  - To handle class imbalance we have also used fixed number of images per class.

```
<AxesSubplot:xlabel='height', ylabel='width'>
```



- As we are having colored images hence values in image array are ranging from 0 to 255

```
[91]: print('max val ' ,np.max(trainClassifReducedMergedGRPDImageDF['image_array'][0]))
      print('min val ' ,np.min(trainClassifReducedMergedGRPDImageDF['image_array'][0]))
```

```
max val  255.0
min val  4.0
```

- After normalization:

```
[94]: print('max val ' ,np.max(trainClassifReducedMergedGRPDImageDF['image_array'][0]))
      print('min val ' ,np.min(trainClassifReducedMergedGRPDImageDF['image_array'][0]))
```

```
max val  1.0
min val  0.015686275
```

values are normalized between 0 to 1

Object Classification Models

Simple CNN Model (Neural Network Model)

- Model architecture

```
[77]: nnmodel.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 223, 223, 32)      416
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 222, 222, 32)      4128
_________________________________________________________________
flatten_1 (Flatten)          (None, 1577088)           0
_________________________________________________________________
dense (Dense)                (None, 200)               315417800
_________________________________________________________________
dense_1 (Dense)              (None, 196)               39396
=================================================================
Total params: 315,461,740
Trainable params: 315,461,740
Non-trainable params: 0
_________________________________________________________________
```

```
[78]: from keras.optimizers import SGD
      opt = SGD(lr=0.001)
```

```
[79]: nnmodel.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

- Few details:
  - Used input image size 224,224,3
  - Used SGD (i.e simple gradient descendent) optimizer with learning rate 0.001
  - Below is final loss and accuracy

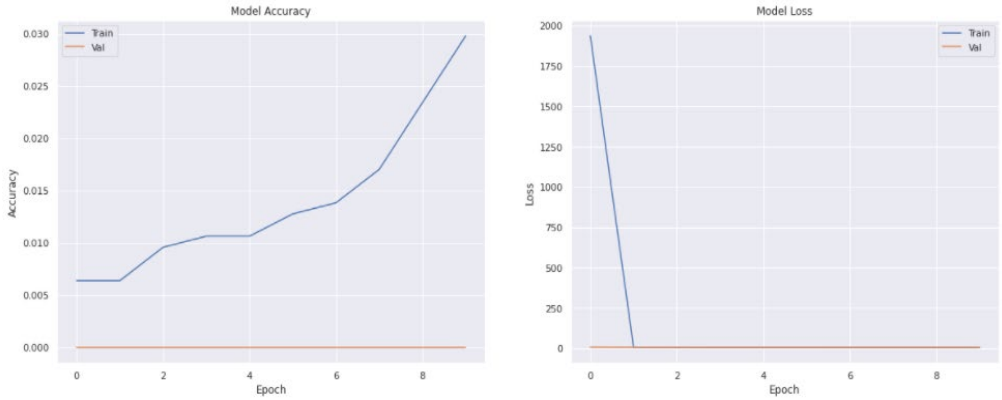
```
)]: nnmodel.evaluate(x_test_classif, y_test_tensor)
```

```
37/37 [==============================] - 2s 49ms/step - loss: 5.4902 - accuracy: 0.0060

0]: [5.490171432495117, 0.0059523810632526875]
```

- Model performance plot is : (It is clear from this plot is that model is too simple to understand validation data, above accuracy is coming may be due to noise in data)

```
'3]:
plotModelPerformanceCurvs(history)
```



CNN model with increasing depth and parameters (nnpmodel)

- Model architecture

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 222, 222, 16)	448
conv2d_15 (Conv2D)	(None, 220, 220, 16)	2320
batch_normalization (BatchNo	(None, 220, 220, 16)	64
zero_padding2d (ZeroPadding2	(None, 222, 222, 16)	0
conv2d_16 (Conv2D)	(None, 220, 220, 32)	4640
conv2d_17 (Conv2D)	(None, 218, 218, 32)	9248
batch_normalization_1 (Batch	(None, 218, 218, 32)	128
conv2d_18 (Conv2D)	(None, 216, 216, 64)	18496
conv2d_19 (Conv2D)	(None, 214, 214, 64)	36928
batch_normalization_2 (Batch	(None, 214, 214, 64)	256
max_pooling2d (MaxPooling2D)	(None, 107, 107, 64)	0
zero_padding2d_1 (ZeroPaddin	(None, 109, 109, 64)	0
conv2d_20 (Conv2D)	(None, 107, 107, 128)	73856
conv2d_21 (Conv2D)	(None, 106, 106, 128)	65664
batch_normalization_3 (Batch	(None, 106, 106, 128)	512
conv2d_22 (Conv2D)	(None, 104, 104, 64)	73792
conv2d_23 (Conv2D)	(None, 102, 102, 64)	36928
batch_normalization_4 (Batch	(None, 102, 102, 64)	256
conv2d_24 (Conv2D)	(None, 100, 100, 64)	36928
conv2d_25 (Conv2D)	(None, 98, 98, 64)	36928
batch_normalization_5 (Batch	(None, 98, 98, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 49, 49, 64)	0
conv2d_26 (Conv2D)	(None, 47, 47, 64)	36928
conv2d_27 (Conv2D)	(None, 45, 45, 64)	36928
batch_normalization_6 (Batch	(None, 45, 45, 64)	256
conv2d_28 (Conv2D)	(None, 43, 43, 128)	73856
conv2d_29 (Conv2D)	(None, 42, 42, 128)	65664
visualized_layer_1 (BatchNor	(None, 42, 42, 128)	512
flatten_2 (Flatten)	(None, 225792)	0

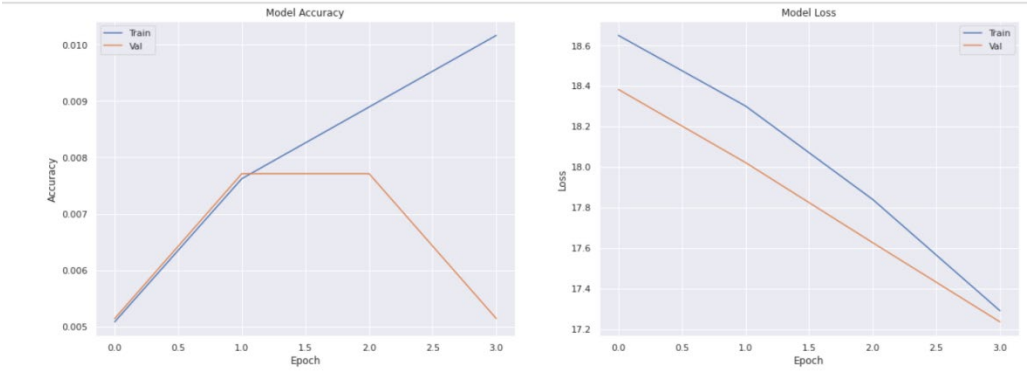
dense_2 (Dense)	(None, 32)	7225376
dropout (Dropout)	(None, 32)	0
dense_3 (Dense)	(None, 32)	1056
dropout_1 (Dropout)	(None, 32)	0
visualized_layer (Dense)	(None, 197)	6501

Total params: 7,844,725
Trainable params: 7,843,605
Non-trainable params: 1,120

- Used input size 224,224,3
- Used Maxpool2d to reduce channeled image size with using max value in a gr id.
- Used zero padding to avid loss of image corner information.
- Used batch normalization , here batch size is of 64
- Used SGD with initial learning rate as below : INIT_LR = 1e-3
- Used adam optimizer as well but with using adam accuracy was not improvin g hence rolled it back to SGD.
- Used call backs like :
  - ReduceLROnPlateau : If gradient is not being captured then reduce le arning rate with patience 2, monitoring val_accuracy
  - EarlyStopping

```
95]: from keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
     Callbacks=[ ReduceLROnPlateau(patience=2,mode="max",monitor="val_accuracy",verbose=1,factor=.01*INIT_LR),
                 EarlyStopping(monitor="val_accuracy",patience=2,verbose=2,mode="max",restore_best_weights=True)]
```

- Model accuracy plots:

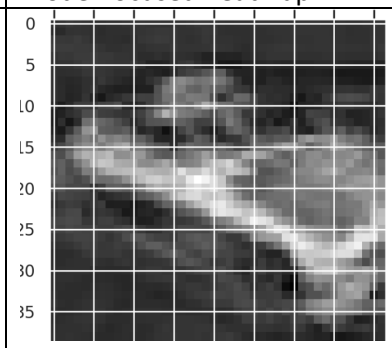
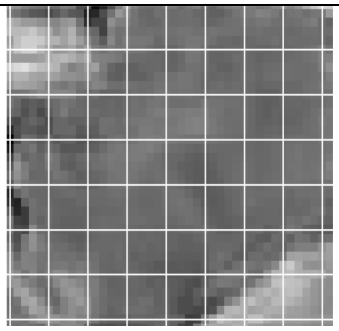
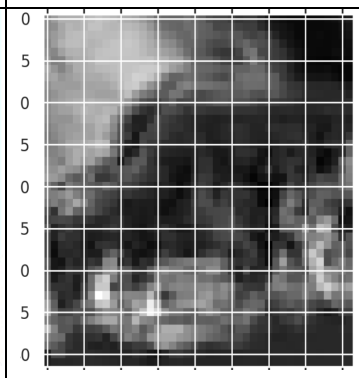


Final loss and accuracy

```
[115]: nnpmodel.evaluate(x_test_classif/255, np.array(y_test_classi))
       37/37 [==============================] - 89s 2s/step - loss: 18.0157 - accuracy: 0.0051
[115]: [18.015714645385742, 0.005102040711790323]
```

Using GRAD-CAM to check model focus area

- Model used is : nnpmodel
- Model layer used for grad cam is : visualized_layer_1
- From below table it is clear that model is focusing on brighter colors.

Actual image	Model focused heatmap	Model focus area
		
		
		

Approaches tried for optimization and tuning

- Used image data generator with below parameter tuning

```
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    #featurewise_center=True,
    #featurewise_std_normalization=True,
    rescale=1. / 255,
    shear_range=0.3,
    zoom_range=0.5,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    #zca_whitening=True,
    channel_shift_range=1.0,
    brightness_range=(0.34,0.68),
    vertical_flip=True,
    fill_mode='nearest'
)
```

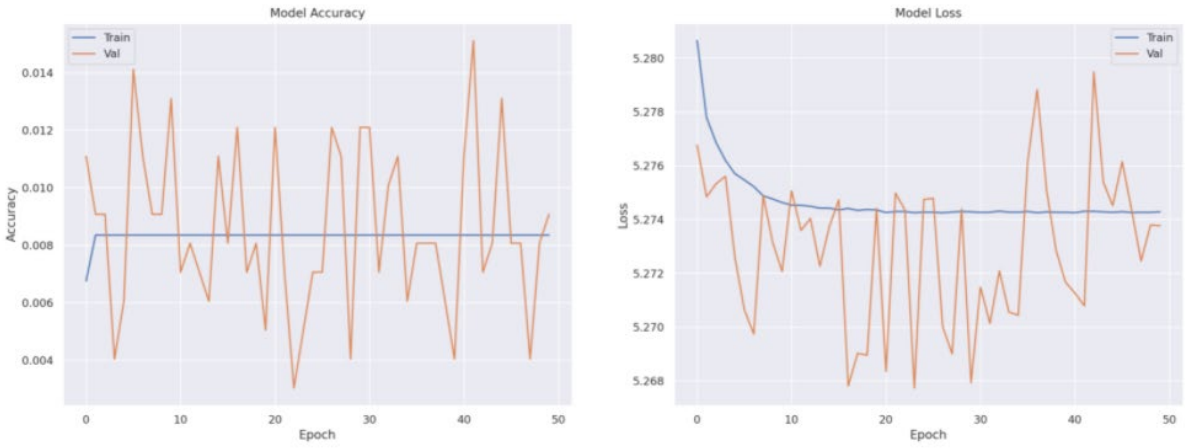
- However this did not improved over all accuracy of model. Model accuracy plot is as well as final accuracy in shown below

```
1]: idgmodel.evaluate(x_test_classif, y_test_tensor)
```

```
37/37 [==============================] - 2s 33ms/step - loss: 5.9321 - accuracy: 0.0043

1]: [5.932104587554932, 0.004251700825989246]
```

```
2]: plotModelPerformanceCurvs(imgdgHist)
```



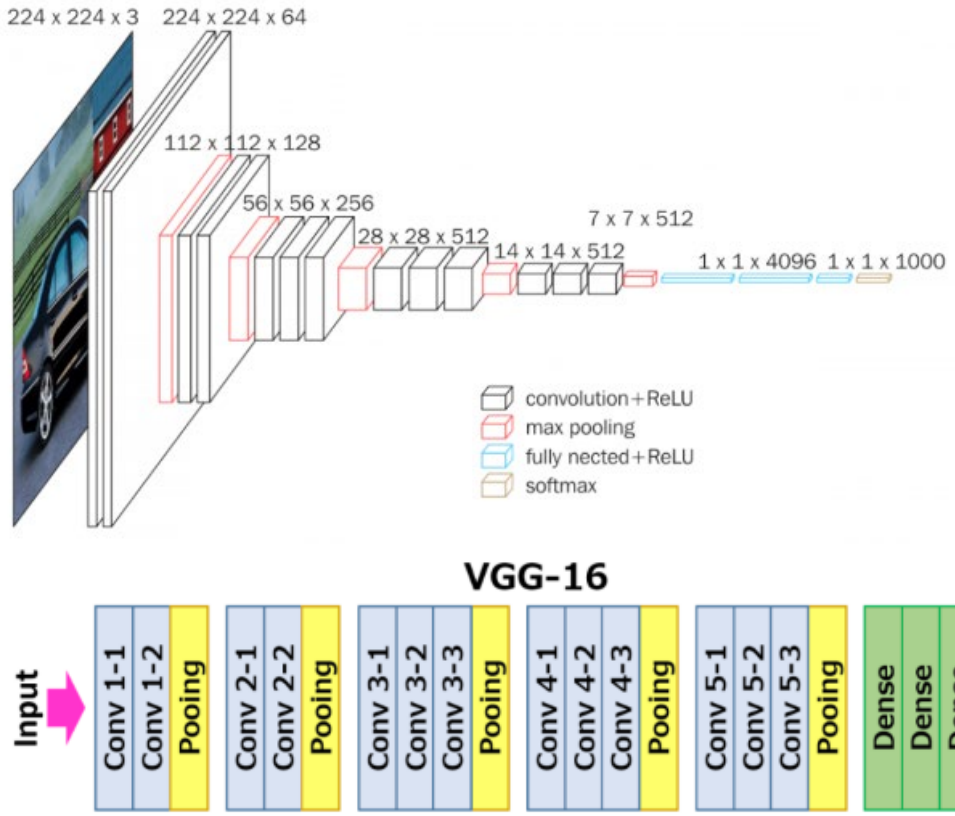
- Tried with other complex model architectures like inception_v1 and residual CNN model, even they are also not able to improve classification accuracy.

Learning /further tuning options for classification

- Use transfer learning for better classification
- Use Tensorboard to visualize gradient movement ,this will help in fine tuning of learning rate ,biases
- Use keras weight regularizes.
- Try with Sigmoid activation function as there is sparsity in output labels

VGG16

- VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. It was one of the famous model submitted to ILSVRC-2014. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU's.
- General VGG16 Model Architecture



Model Key Design Parameters

- Used Transfer Learning Approach
- VGG16 is the Backbone architecture
- Input Image size is 224*224*3
- Used 'Imagenet' weights
- Image Augmentation not done
- Optimizer = Adam
- Loss for box coordinates ='mse'
- Loss for Classification = 'categorical_crossentropy'
- Metric for box coordinates ='IoU'

- Metric for box Classification ='Accuracy'
- Trainable params: 143,200
- Non-trainable params: 14,715,088
- Training & Validation Accuracy for Classification = 0.11 & 0.001
- Training & Validation IoU for Bounding Boxes = 0.004 & 0.004

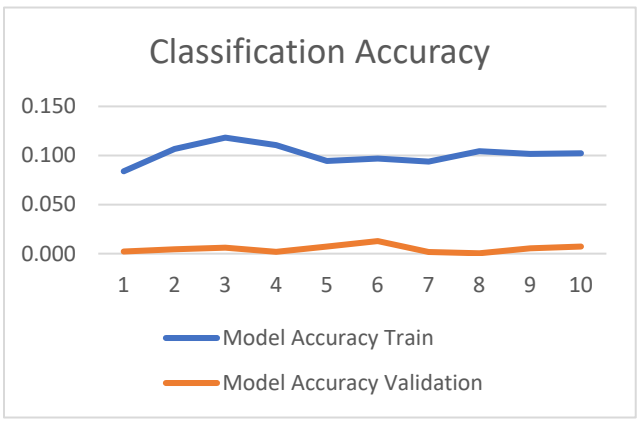
Data Preparation for Modeling

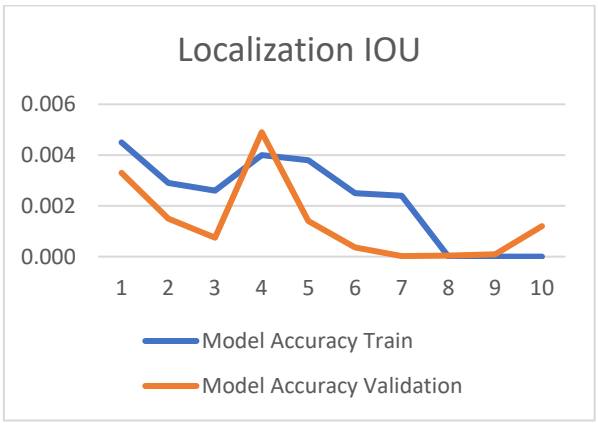
- Mapping of Images in train folder with it class and storing in a dataframe
- Train Images with the classification are merged with respective the bounding box Co-ordinates
- File path has been added against each training images , so that we can create the pipeline to access the images at ease
- Image width and height is mapped against the following function , it will be required when we have to modify the bounding box coordinates based on the resized images
- Cv2 library is used to display the random images with it bounding boxes

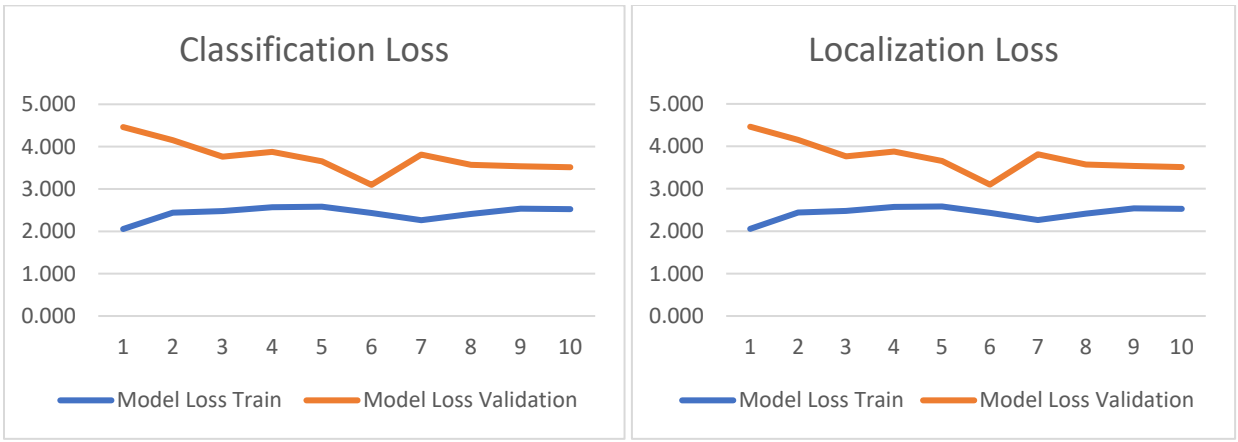
Model Development

- As mentioned earlier base model is taken as VGG16 excluding the top layer & custom top layer has been developed to incorporate 2 outputs (Classification & Localization)
- Batch generator function created to load the images in batches, so that the computer memory can be used in a more efficient manner

Model Performance







- Accuracy is very low both in Training & validation Images of Classification
- IOU also very low both in in Training & validation Images of Localization

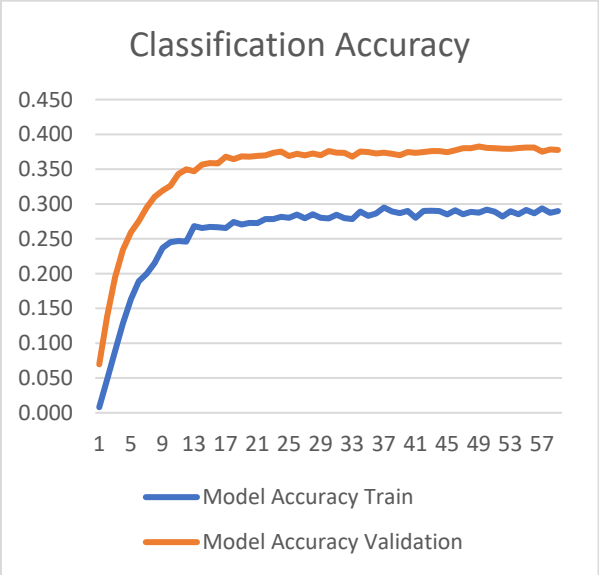
Plans for Model Performance Improvement

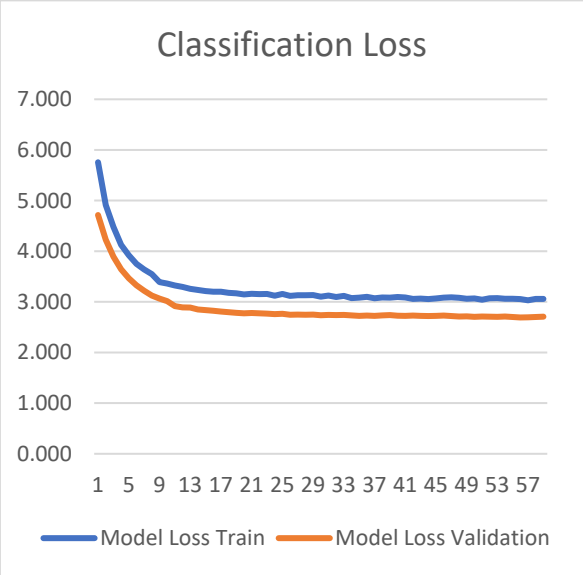
- Building separate models for classification & Localization
- Implementation of Image Augmentation
- Trying out different models
- Top Layer architecture optimization
- Hyper parameter tuning

Model Details Trial 1

- Dedicative model for classification
- Implementation of Image Augmentation through Keras Image Data Generator
- Augmentation used = Rotation, Width shift, height shift & horizontal flip based on the observation from test images
- All other parameter as per base trial
- Significant improvement observed over the base trial (Max Validation Accuracy = 0.383)
- Validation Accuracy is higher than the train accuracy – demonstrates no over-fitting

Model Performance

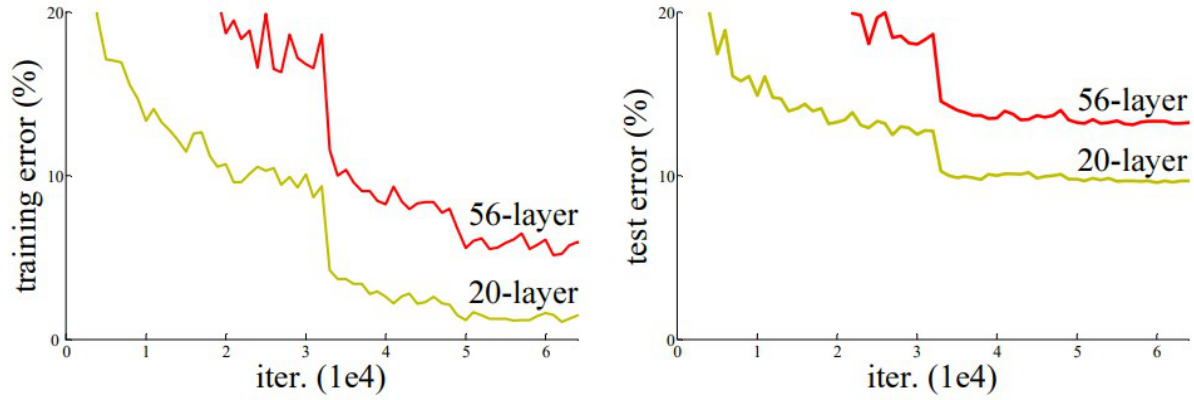




- Significant improvement observed over the base trial (Max Validation Accuracy = 0.383)
- Validation Accuracy is higher than the train accuracy – demonstrates no overfitting
- All other parameter as per base trial

Residual Networks (ResNet) – Deep Learning

After the first CNN-based architecture (AlexNet) that win the ImageNet 2012 competition, Every subsequent winning architecture uses more layers in a deep neural network to reduce the error rate. This works for less number of layers, but when we increase the number of layers, there is a common problem in deep learning associated with that called Vanishing/Exploding gradient. This causes the gradient to become 0 or too large. Thus when we increases number of layers, the training and test error rate also increases.



ResNet, which was proposed in 2015 by researchers at Microsoft Research introduced a new architecture called Residual Network.

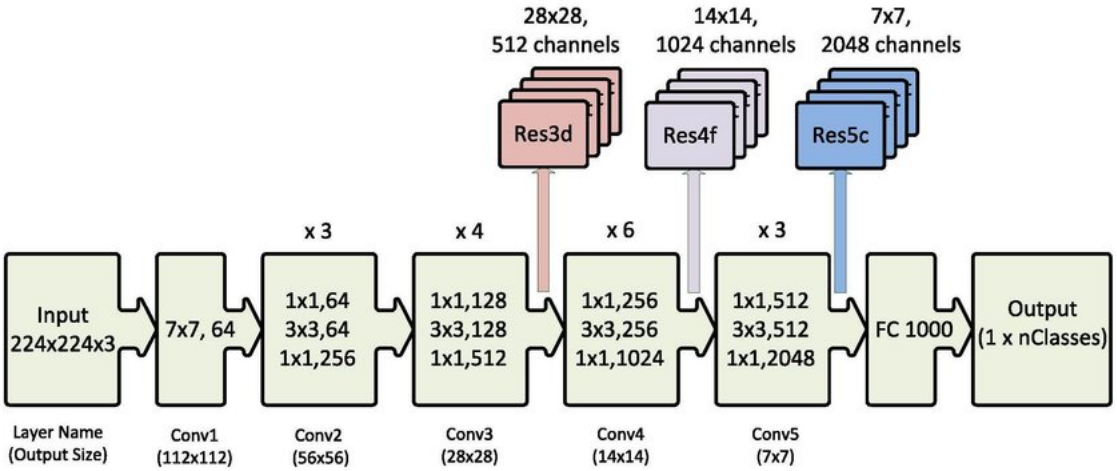
Residual Block:

In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Network. In this network we use a technique called skip connections . The skip connection skips training from a few layers and connects directly to the output.

The advantage of adding this type of skip connection is because if any layer hurt the performance of architecture then it will be skipped by regularization.

There are many variants of ResNet architecture i.e. same concept but with a different number of layers. We have ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-110, ResNet-152, ResNet-164, ResNet-1202 etc.

ResNet50 Architecture



Transfer Learning – ResNet50

Approach 1

- Model Key Attributes
  - ResNet50 is the Backbone architecture
  - Input Image size is 224*224*3
  - Augmentation on image dataset not applied
  - Optimizer = Adam
  - Loss for classification = 'categorical_crossentropy'
  - Metrics for classification = 'accuracy'
  - To implement Transfer learning, we would remove the last predicting layer of the pre-trained ResNet50 model and replace them with our own predicting layers.
  - Weights of ResNet50 pre-trained model is used as feature extractor
  - Weights of the pre-trained model are frozen and are not updated during the training
  - We do not want to load the last fully connected layers which act as the classifier. We accomplish that by using "include_top=False". We do this so that we can add our own fully connected layers on top of the ResNet50 model for our task-specific classification.
  - We freeze the weights of the model by setting trainable as "False". This stops any updates to the pre-trained weights during training We do not want to train ResNet layers as we want to leverage the knowledge learned by the deep neural network trained from the previous data set which in our case is "imagenet"
  - Adding our own layers and the final classifier using softmax activation function.
  - TensorFlow Keras Implementation

```
In [55]: from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer
         from tensorflow.keras.models import Sequential
         from tensorflow.keras import optimizers

         output_nodes=196 #As range of image labels is up to 196

         model = Sequential()
         model.add(restnet)
         model.add(Dense(512, activation='relu'))
         model.add(Dropout(0.3))
         model.add(Dense(512, activation='relu'))
         model.add(Dropout(0.3))
         model.add(Dense(output_nodes, activation='softmax'))
```

```
In [56]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model (Functional)           (None, 100352)            23587712
_________________________________________________________________
dense (Dense)                (None, 512)               51380736
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 196)               100548
=================================================================
Total params: 75,331,652
Trainable params: 51,743,940
Non-trainable params: 23,587,712
_________________________________________________________________
```

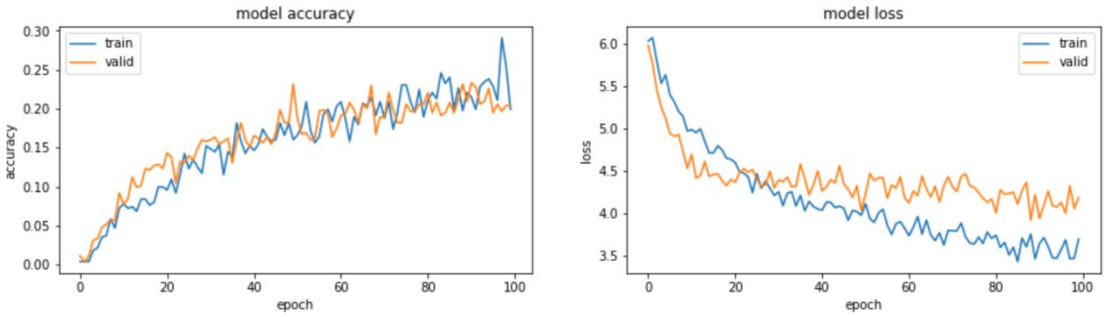
- Data Preparation
  - Data frame creation - merging train / test images and its annotations respectively.

Approach 2

- Unfreezing a few of the last convolution blocks while keeping the first early conv blocks frozen. This will help us to learn very generic features using early layers. Higher layers of pre-trained models will be trainable or fine-tuned

Approach 3

- Image data augmentation implementing TensorFlow ImageDataGenerator
- ResNet50 with pretrained weights
- Normalized test images treated as validation dataset
- Accuracy improved significantly [23%]

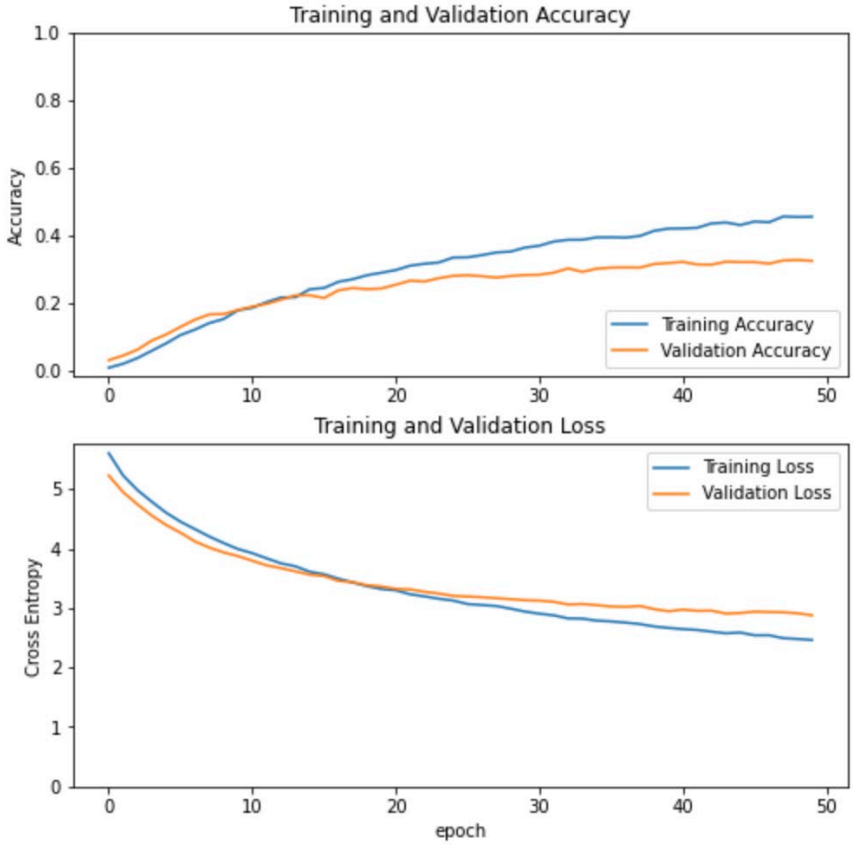


Approach 4

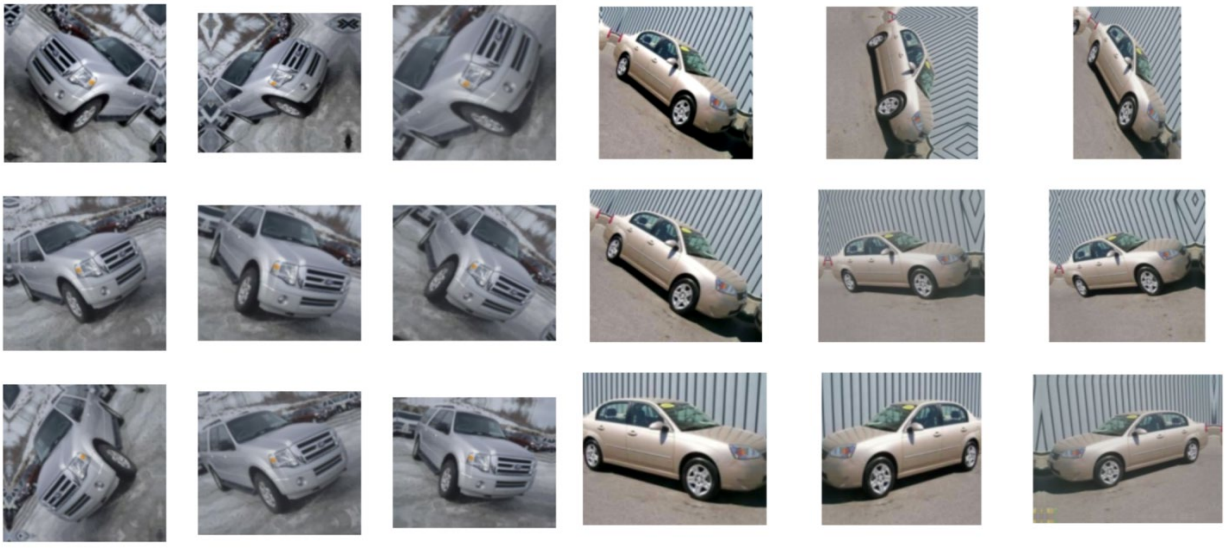
- Validation dataset created from test images
- Added data augmentation layer
- Without unfreezing any layer for training accuracy reached 25%

- Unfreezing the base_model and set the bottom layers to be un-trainable.
- As we are training a much larger model and want to readapt the pretrained weights, it is important to use a lower learning rate at this stage. Otherwise, model could overfit very quickly
- Hyper-parameter momentum accelerates gradient descent in the relevant direction and dampens oscillations.

- Accuracy reached ~40%

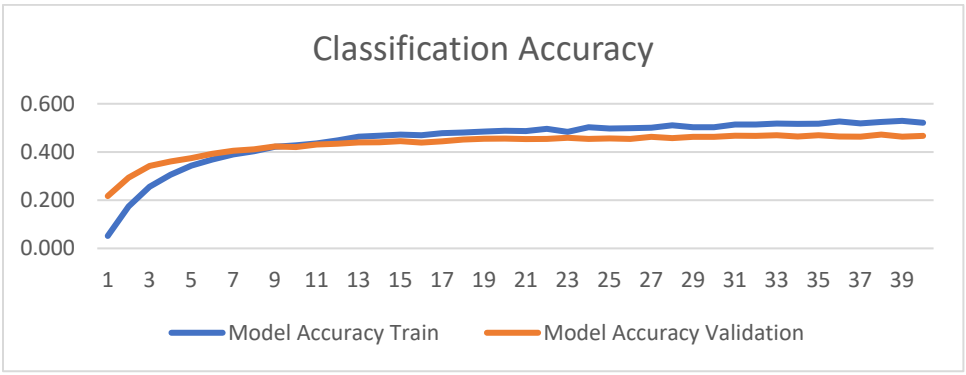


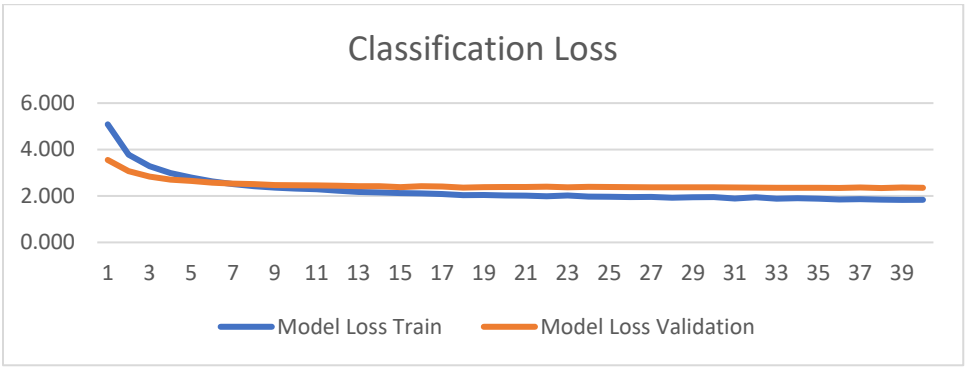
Data augmentation sample



Resnet50: Final Approach

- Except the base model all other parameters kept constant, details taken from VGG16 model mentioned below
- Base Model is changed to ResNet50 instead of VGG16





- Significant improvement observed over the trial1 (Max Validation Accuracy = 0.472)
- Post epoch 9, validation loss is increasing over the training loss
- ResNet50 model works better than the VGG16 in this application hence ResNet50 is considered for further trails

Object Detection Models

Image Augmentation & Bounding Boxes

- **Data/Image Augmentation**
    - Data/Image augmentation works by considering it as a way to artificially expand our dataset. As is the case with deep learning applications, the more data, the merrier.
- We have many deep learning libraries like torchvision, keras, and specialized libraries on Github to support data augmentation for classification training tasks, but we don't have a data augmentation for object detection tasks, which will follow the bounding box along with it.
- Augmentation type options attempted:
    - Resizing with/without letterbox pattern

Resized Image with letter box keeping aspect ratio, Original vs Augmented:

Resized Image without letter box not keeping aspect ratio, Original vs Augmented:

– Horizontal Flip

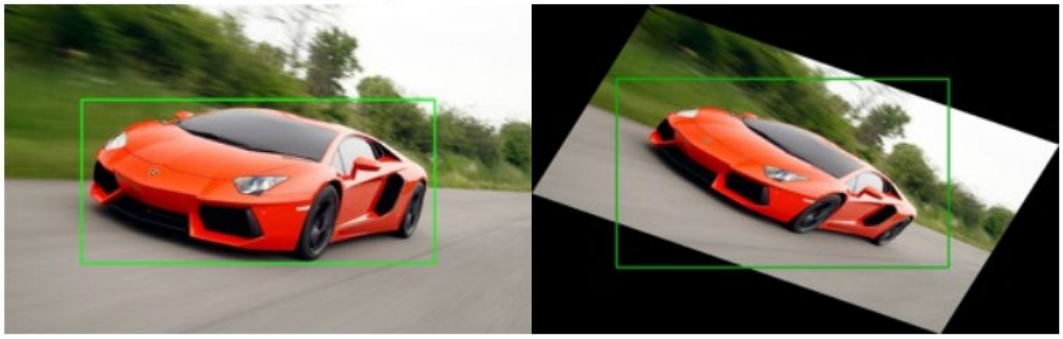
Horizontal Flip, Original vs Augmented:

– Scale

Scale Image, Original vs Augmented:

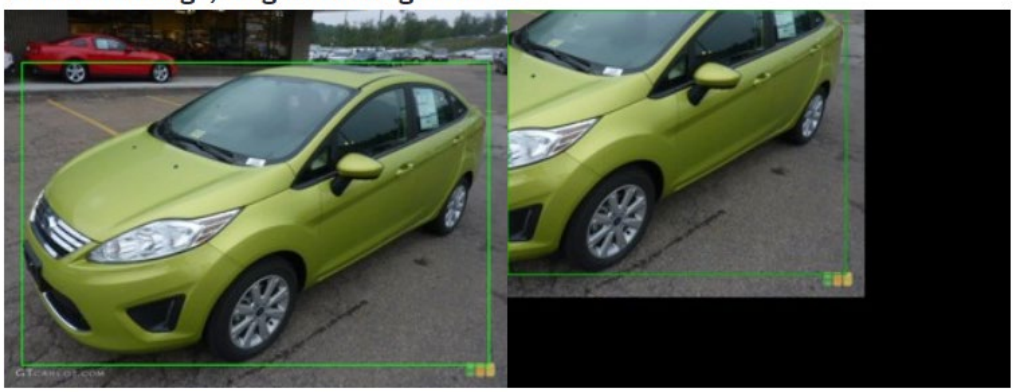
– Rotation

Rotate Image, Original vs Augmented:



– Translation

Translate Image, Original vs Augmented:

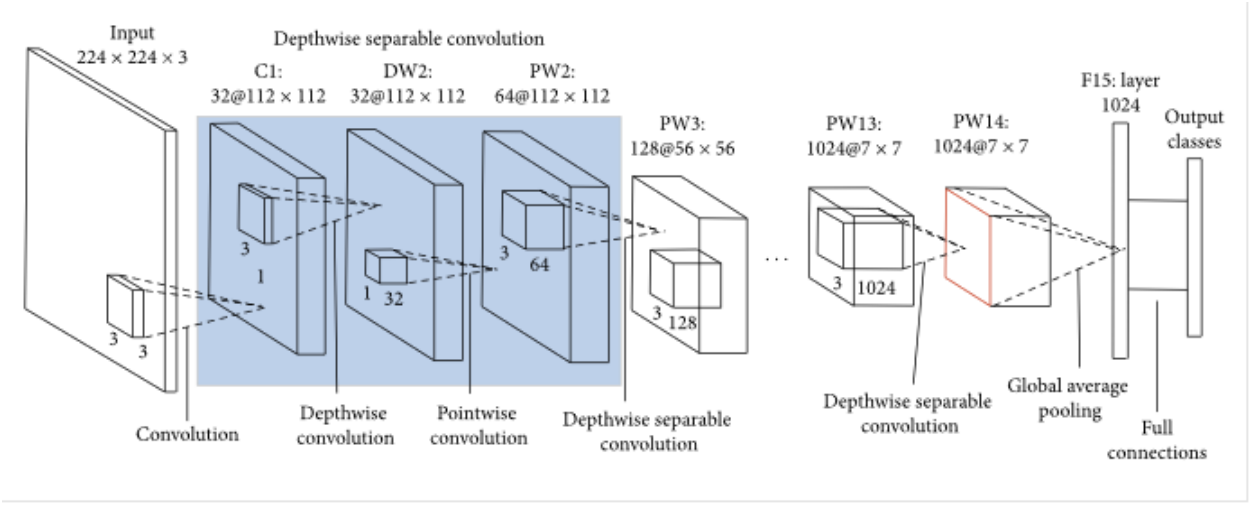


– Shearing

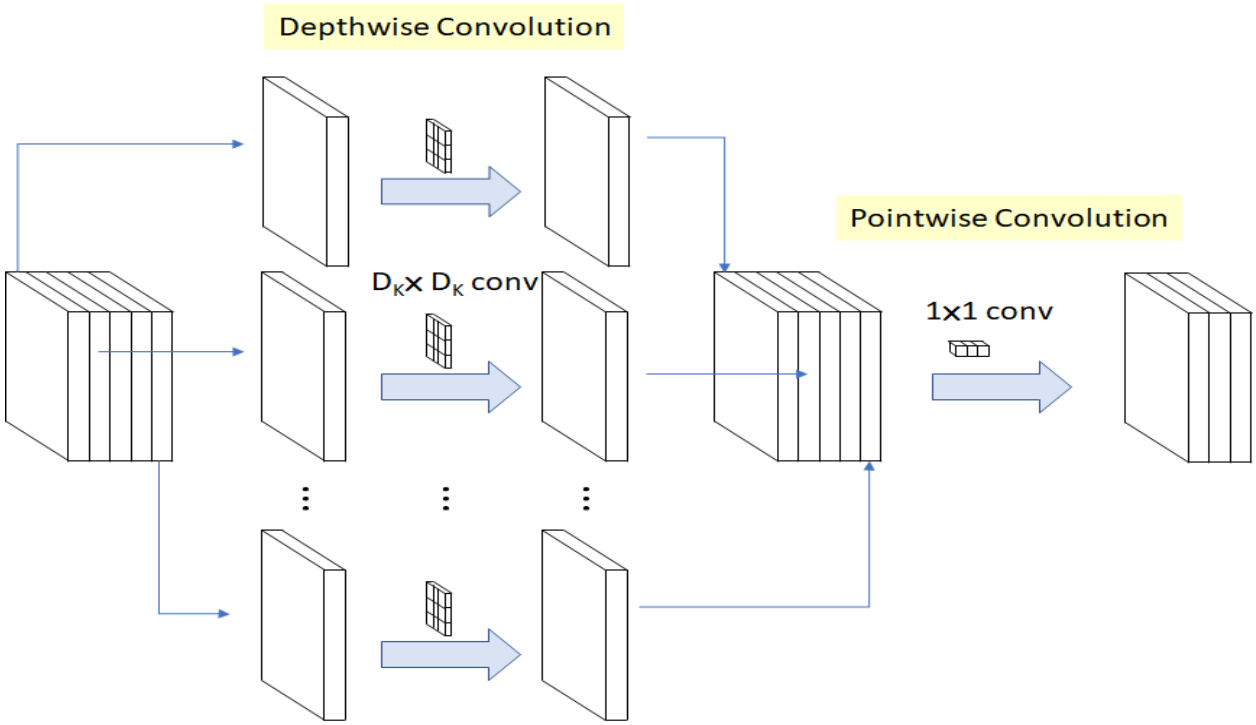
Shear Image, Original vs Augmented:

Mobile Net

- Mobile Net architecture



- Detailed explanation of one unit



- Used pre-trained mobile net model with putting include top layer = false
- Used customized function for Inversion of Unit matrix

Final Model Accuracy

- Model returned IoU of ~81%

Final loss and accuracy

```
[187]: model.evaluate(x_test, y_test)

       31/31 [==============================] - 5s 150ms/step - loss: 54.3525 - IoU: 0.8099
[187]: [54.35245132446289, 0.8098935484886169]
```

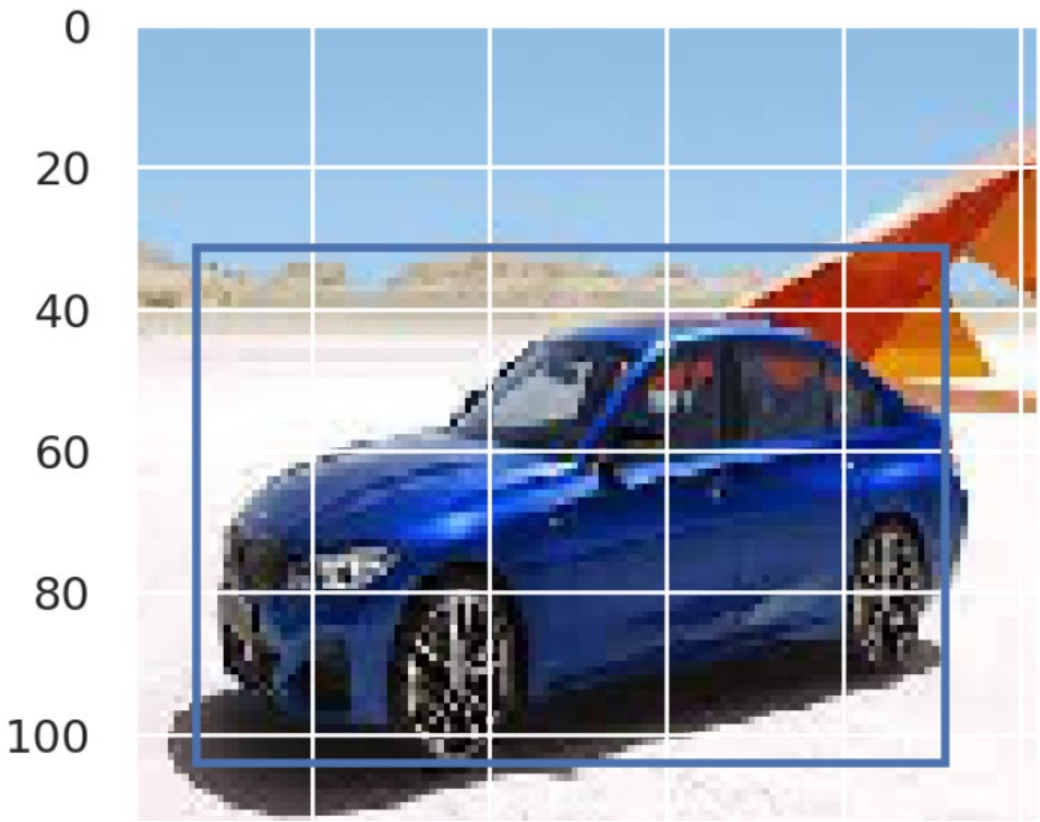
Model output visualization on random image

- Region predicted by model is :

  region co-ordinates predicated by model : [8.798341 31.36415 114.46162 104.061226]

- Bounding box predicted by model is drown below

```
|: filename = '../input/customcarimages/bmw.jpg'
drowPredictedBBOnImage(filename,(IMAGE_WIDTH,IMAGE_HEIGHT,CHANNEL),'image_array','rgb')
```



Resnet50

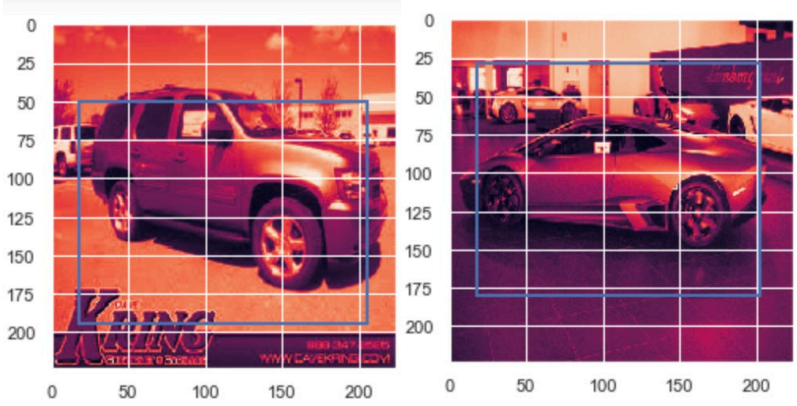
Model Key Attributes

- ResNet50 is the Backbone architecture
- Input Image size is 224*224*3
- Optimizer = Adam
- Loss for bounding box coordinates =‘mse’
- Metrics for object detection = ‘‘IoU’ [Intersection over Union]
- Accuracy ~77%

Final loss and accuracy

```
In [105]: model.evaluate(x_test, y_test)
          43/43 [==============================] - 53s 1s/step - loss: 274.9186 - IoU: 0.7662
Out[105]: [274.9185791015625, 0.7662255167961121]
```

Output sample



Future Plans

- Tried to apply transfer learning InceptionV3 base model, used tensor flow implementation. Even after unfreezing layer and train the weights accuracy much poorer than ResNet50 so further exploration would be around ResNet only
- Started exploring pytorch ResNet34 implementation. Better accuracy observed. Valid / test dataset center cropped. Building understanding on pytorch APIs. CUDA / CPU device type need to be handled thoroughly to load / save parameters / data / weights / model accordingly.
- Cross validation among datasets to close the gap between train and val accuracy.
- Hyper-parameter Tuning - Learning rate, Momentum, Batch Size, Epochs, train / val / test percentage
- Playing around augmentation to avoid over-fitting / high bias concern
- Cloud based deployment pipeline

Faster RCNN

Why not RCNN?

- Introduction of R-CNN was important turn of event in object detection algorithm world. It was topped with Fast–RCNN and then finally by Faster-RCNN.
- R-CNN was proposed to deal with the problem of efficient object localization in object detection
- Earlier method uses exhaustive search to find object
- R-CNN uses the Selective search algorithm which takes advantage of segmentation of objects and Exhaustive search to efficiently determine the region proposals
- Selective search algorithm proposes approximately 2000 region proposals per image
- Issues:
    - Each image needs to classify 2000 region proposals. Might not be an ideal case for all.
    - To store feature map, it requires lot of disk space.
    - R-CNN couldn't be used real time. Each region proposal is fed independently to the CNN for feature extraction. This makes it impossible to run R-CNN in real-time.
    - For our current project objective of car detection, RCNN has a major drawback which leads to rejection. It can not be used in real time.

Why Not Fast RCNN?

- Fast R-CNN overcomes several issues in R-CNN. It justifies its tag ahead of RCNN and is Fast.
- Removes the requirement to store a feature map and saves disk space
- Takes the whole image and region proposals as input in its CNN architecture in one forward propagation
- Softmax layer instead of SVM in its classification of region proposal which proved to be faster and generate better accuracy than SVM
- Softmax layer to predict the class scores
- FC layer to predict the bounding boxes of the detected objects
- Issues:
    - Most of the time taken by Fast R-CNN during detection is a selective search region proposal generation algorithm. Hence, it is the bottleneck of this architecture which was dealt with in Faster R-CNN

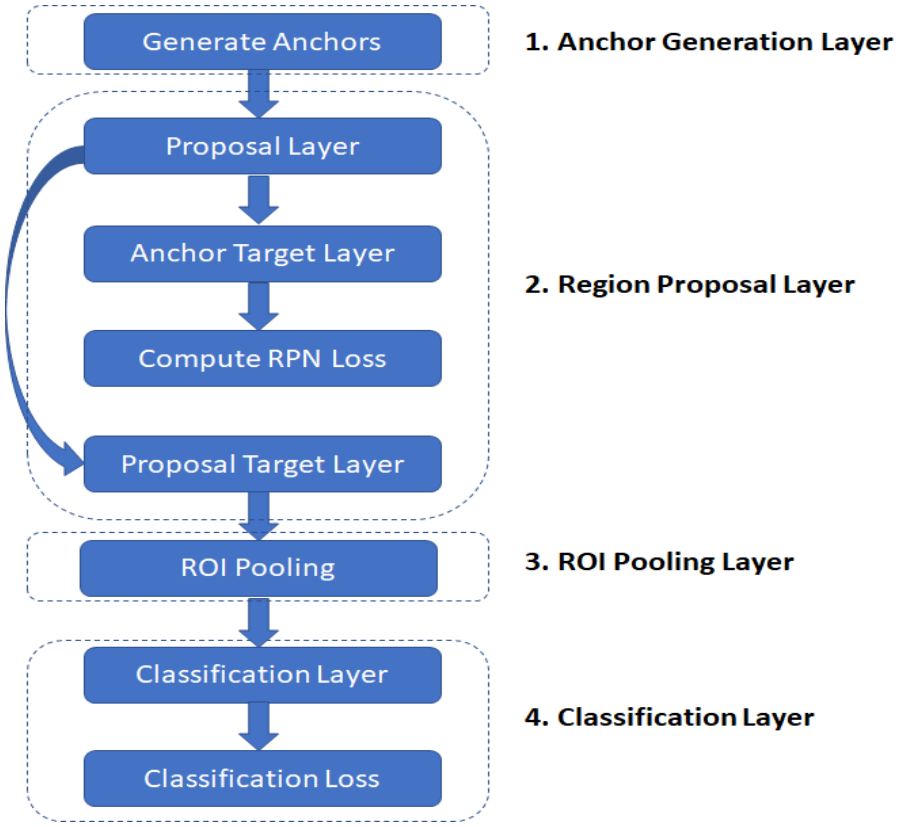
Why Faster RCNN?

- Faster R-CNN is a single-stage model that is trained end-to-end.

- Region proposal network (RPN) which is a fully convolutional network that generates proposals with various scales and aspect ratios
- Introduced the concept of anchor boxes
- Convolutional computations are shared across the RPN and the Fast R-CNN. This reduces the computational time
- Architecture of Faster R-CNN:
- RPN: For generating region proposals
- Fast R-CNN: For detecting objects in the proposed regions
- Drawback:
  - One drawback of Faster R-CNN is that the RPN is trained where all anchors in the mini-batch, of size 256, are extracted from a single image. Because all samples from a single image may be correlated (i.e. their features are similar), the network may take a lot of time until reaching convergence.

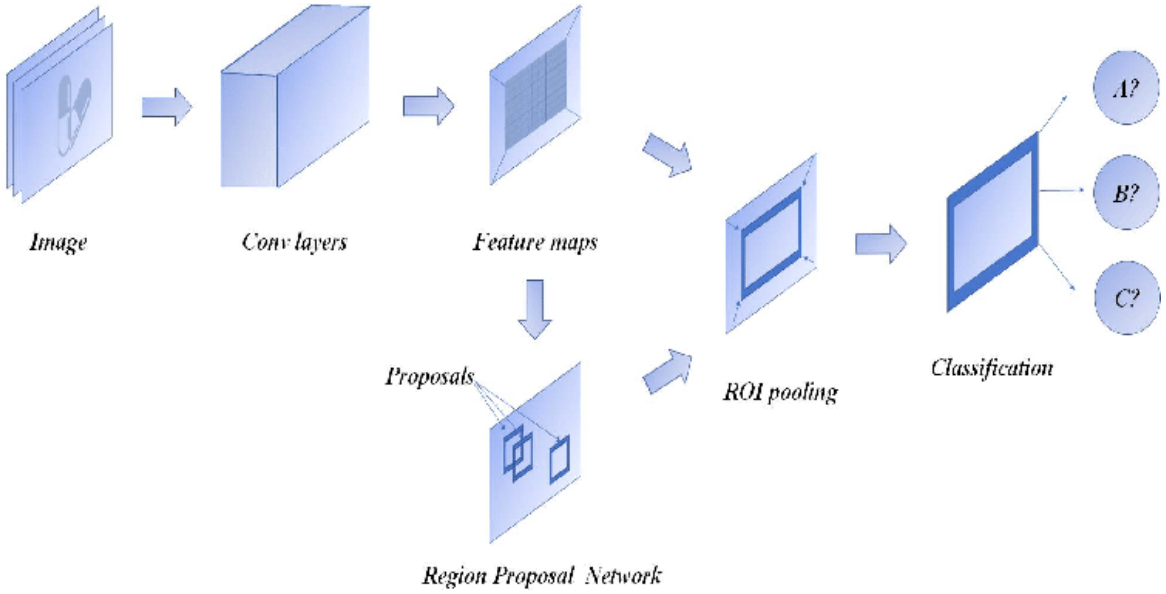
Model & Training Details

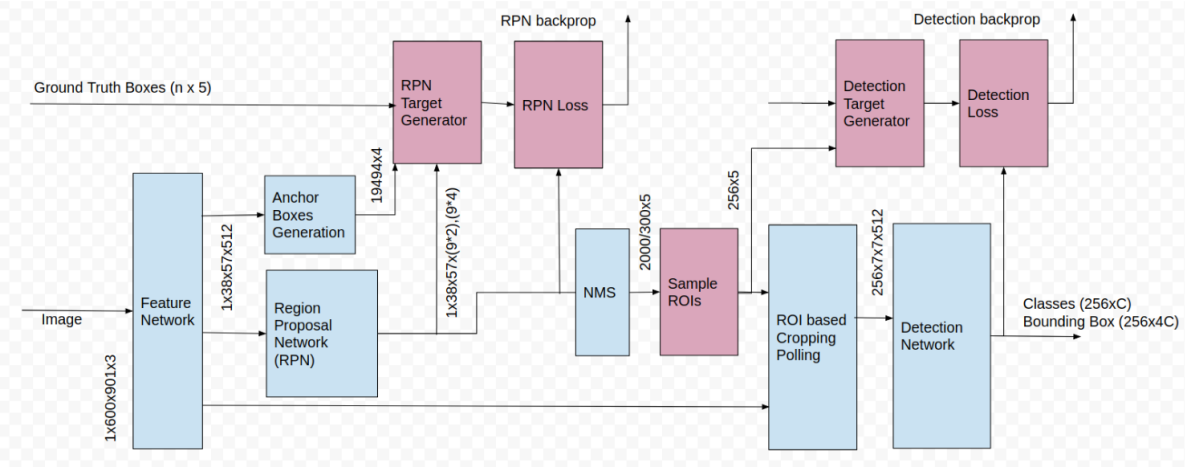
- Architecture details
  - Base network – Resnet50
  - All Layers kept trainable with low learning rate



- Anchors and bounding box suggested, green color box is the actual box provided.

– Deeper understanding of Architecture





- We have tried base network as VGG16 as well, but it has much higher memory foot print and resource consumption compared to resnet50. Hence stayed with resnet50.
- Transfer Learning
  - Pre trained Resnet50 model
  - resnet50_weights_tf_dim_ordering_tf_kernels.h5
- Image size
  - Based on EDA, Maximum images are having size of 640x480. High training time with image size 600 hence reduced.
  - 300
- Parameters Details
  - Total params: 28,342,195
  - Trainable params: 28,235,955
  - Non-trainable params: 106,240
- Important points
  - Training with augmented images improved accuracy
  - Average IOU has reached more than 75%. We can tune the overlap threshold to capture bboxes with higher probability as well.

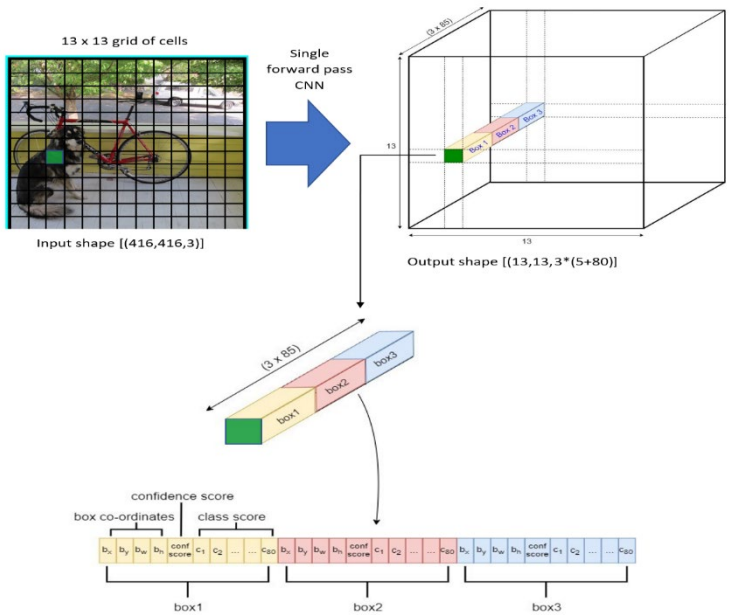
Result

```
00652.jpg
Elapsed time = 4.835828542709351
[('Car', 99.9584972858429)]
```

YOLO

YOLO and its advantage over other CNN

- Yolo is an algorithm that uses convolutional neural networks for object detection.
- Most methods the model to an image at multiple
- Locations and scales. High scoring regions of the image are considered detections. Yolo, on the other hand, applies a single neural network to the full image.
- The network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.
- **In comparison to recognition algorithms, a detection algorithm does not only predict class labels, but detects locations of objects as well**
- Batch normalization and fixed padding
  - Yolo uses convolution with fixed padding, which means that padding is defined only by the size of the kernel.
  - Almost every convolutional layer in Yolo has batch normalization after it. It helps the model train faster and reduces variance between units (and total variance as well).
- YOLO in Keras:



- YOLOV2 Vs YOLOV3:

	YOLO	**YOLO v2**	**YOLO v3**
Input size	224 x 224	448 x 448	
Framework	Darknet trained on ImageNet—1,000.	Darknet-19 19 convolution layers and 5 max pool layers.	Darknet-53 53 convolutional layers. For detection, 53 more layers are added, giving a total of 106 layers.
Small size detection	It cannot find small images.	Better than YOLO at detecting small images.	Better than YOLO v2 at small image detection.
		Uses anchor boxes.	Uses a residual block.

YOLOV2

Type	Filters	Size	Output
Convolutional	32	3 x 3	224 x 224
Maxpool		2 x 2 / 2	112 x 112
Convolutional	64	3 x 3	112 x 112
Maxpool		2 x 2 / 2	56 x 56
Convolutional	128	3 x 3	56 x 56
Convolutional	64	1 x 1	56 x 56
Convolutional	128	3 X 3	56 X 56
Maxpool		2 x 2 / 2	28 X 28
Convolutional	256	3 x 3	28 X 28
Convolutional	128	1 x 1	28 X 28
Convolutional	256	3 X 3	28 X 28
Maxpool		2 x 2 / 2	14 X 14
Convolutional	512	3 x 3	14 X 14
Convolutional	256	1 x 1	14 X 14
Convolutional	512	3 X 3	14 X 14
		2 x 2 / 2	7 X 7
Convolutional	1024	3 x 3	7 X 7
Convolutional	512	1 x 1	7 X 7
Convolutional	1024	3 X 3	7 X 7
Convolutional	512	1 x 1	7 X 7
Convolutional	1024	3 X 3	7 X 7
Convolutional	1000	1 X 1	7 X 7
Avgpool		Global	1000

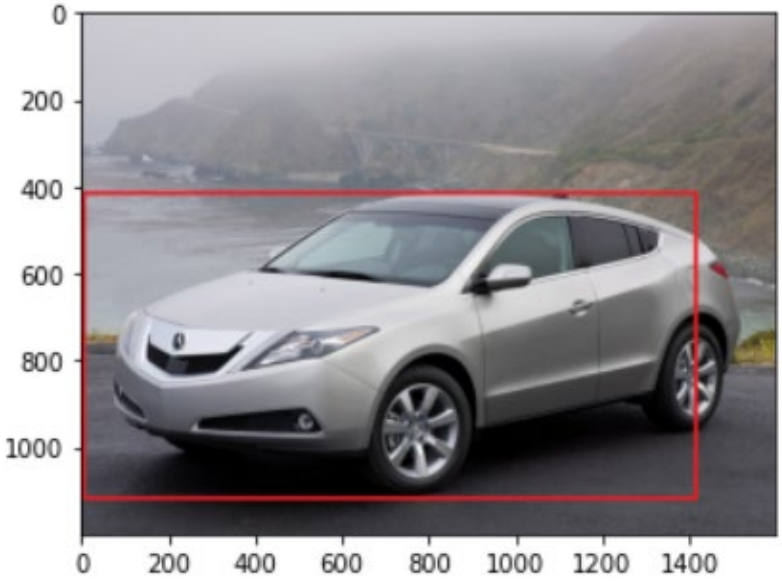
YOLOV3

	Type	Filters	Size	Output
	Convolutional	32	3 x 3	256 x 256
	Convolutional	64	3 x 3 / 2	128 x 128
1 x	Convolutional	32	1 x 1	
	Convolutional	64	3 X 3	
	Residual			128 X 128
	Convolutional	128	3 x 3 / 2	64 x 64
2 x	Convolutional	64	1 x 1	
	Convolutional	128	3 x 3	
	Residual			64 X 64
	Convolutional	256	3 x 3 / 2	32 x 32
8 x	Convolutional	128	1 x 1	
	Convolutional	256	3 x 3	
	Residual			32 x 32
	Convolutional	512	3 x 3 / 2	32 x 32
8 x	Convolutional	256	1 x 1	
	Convolutional	512	3 x 3	
	Residual			16 x 16
	Convolutional	1024	3 x 3 / 2	32 x 32
4 x	Convolutional	512	1 x 1	
	Convolutional	1024	3 x 3	
	Residual			8 x 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Result

```
# summarize what we found
for i in range(len(v_boxes)):
    print(v_labels[i], v_scores[i])

draw_boxes(image_path, v_boxes, v_labels, v_scores)
```



Challenges faced in this model

- We had issue in using the pre-trained model (darknet) for Yolo due to version mismatch in collab (as it always have the latest version of tensorflow and its child packages). Hence developed the model without pre-trained layers and all the layers are trained.
- Yolo has its own pre-trained weights with own common labels that works for multiple smaller objects with overlapping boxes.
- Though accuracy stands at 67%, the time of training was several hours for generating the bounding boxes at each threshold and labels are limited to coco dataset for pre-trained weights.
- It is a challenging model to implement from scratch, as it requires the development of many customized model elements for training and for prediction

Consolidated Model Results

Image Classification

Model	Train Accuracy	Train Loss	Val / Test Acc	Val / Test Loss
ResNet50	0.529	1.831	0.472	2.341
CNN	0.3774	2.3558	0.0020 / 0.0058	12.9033
VGG16	0.295	3.031	0.383	2.692

Object Detection

Model	Train Accuracy	Train Loss	Val / Test Acc	Val / Test Loss
ResNet50	0.7639	278.0834	0.7662	274.9186
MobileNet	0.8815	25.9123	0.8019	56.1673
VGG16	0.004	0.01	0.004	0.10
Yolo	0.768		0.579	

Model	Classification accuracy for bboxes from RPN	Loss RPN Classifier	Loss RPN Regression	Loss Detector Classifier	Loss Detector Regression	Test Set mAP	Test Set IOU
Faster RCNN	0.903	0.295	0.106	0.239	0.175	0.99	0.752

Final Model Selection

Based on our model accuracies we proceeded with Resent50 for image classification and Faster RCNN for image localization.

UI Preparation

Once your model is created and fine tuned to your desired extent, now you want to try it out in real world any time. One way is to keep it handy over colab or kaggle, but is it really efficient use of your time to upload the image, change paths and predict. You need to look for ease out the whole process as a one click approach.

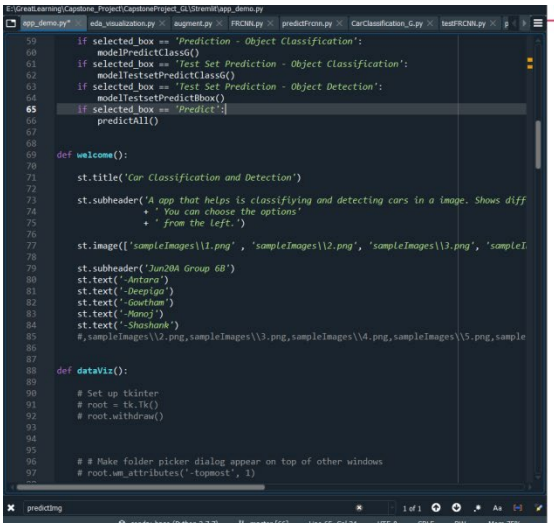
Best way to do is go for a UI which you can access anywhere for model prediction or trigger your training with new data sets, retraining and many other actions.

We have used Streamlit for our UI creation.

Streamlit

Why pick Streamlit

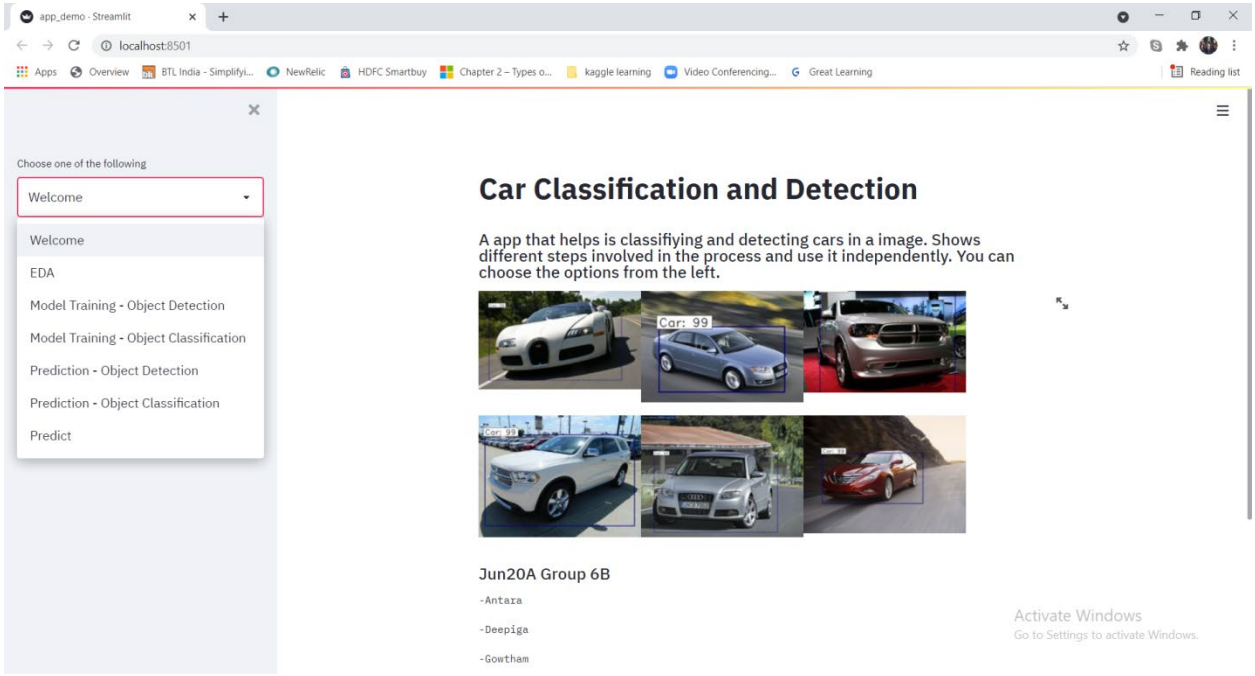
- Open Source, Fast, iterative, and interactive development loop
- Good for basic read only data viewer
- Quick to build highly interactive web applications around data & machine learning models
- One command needed to start all `streamlit run app_demo.py`
- But this does not means it is free of issues, being new and open source has its own limitations
  - Biggest issue is that nearly all applications need some form of state
  - It's based on the model of re-running entire application after anything changes is just fundamentally opposed to many kinds of basic user interaction. Need to plan UI pages keeping this in mind



UI Structure & Details

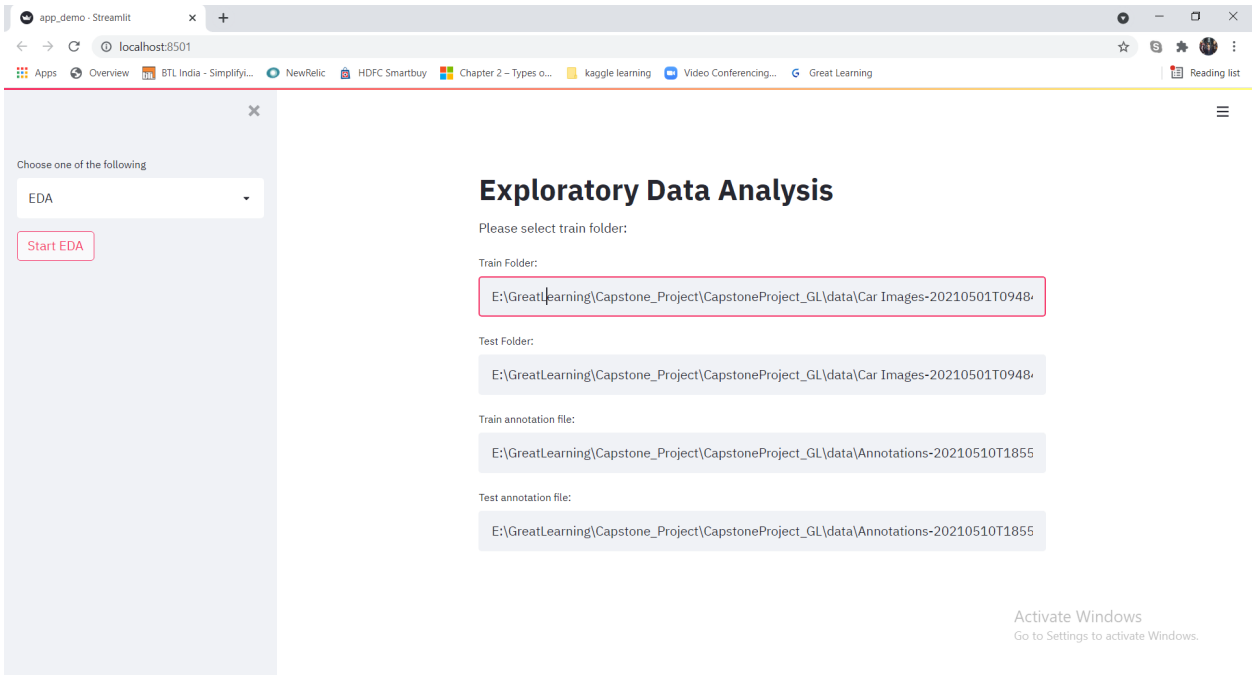
Home Page

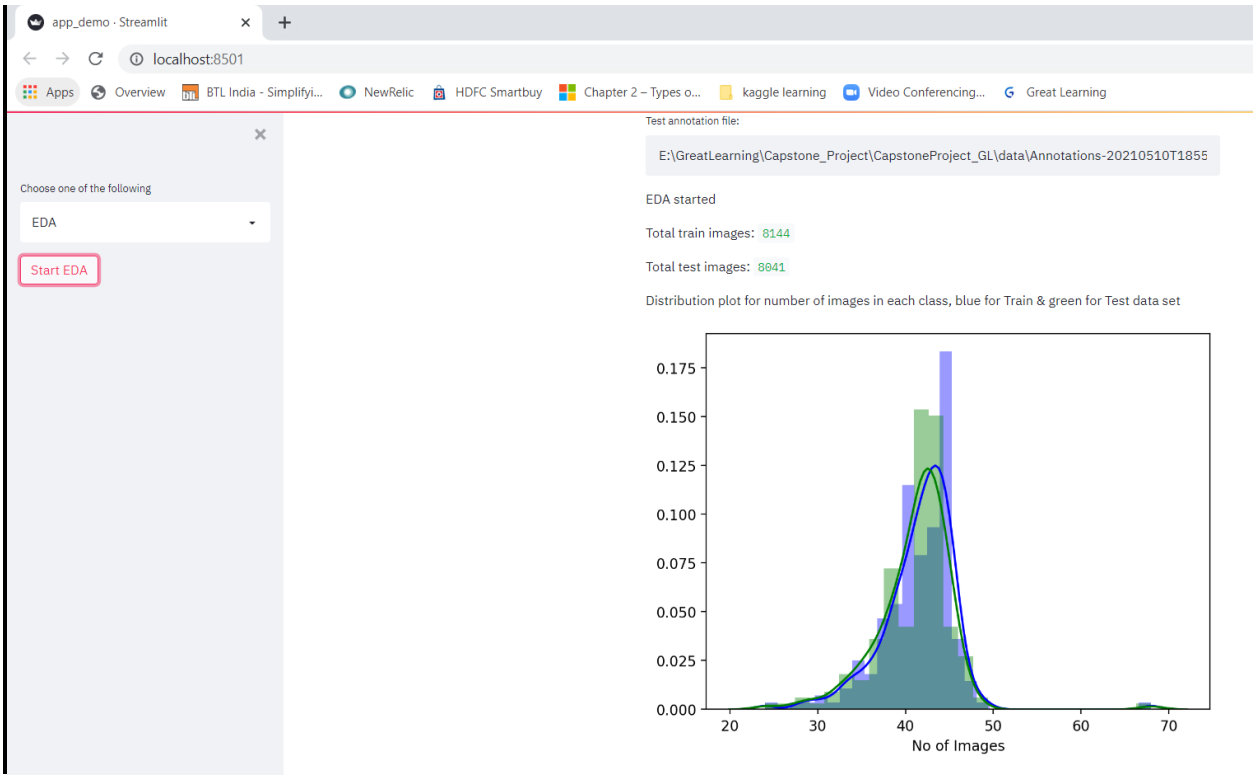
- Home page consists of welcome screen and left side drop down contains the navigation to other sections/modules

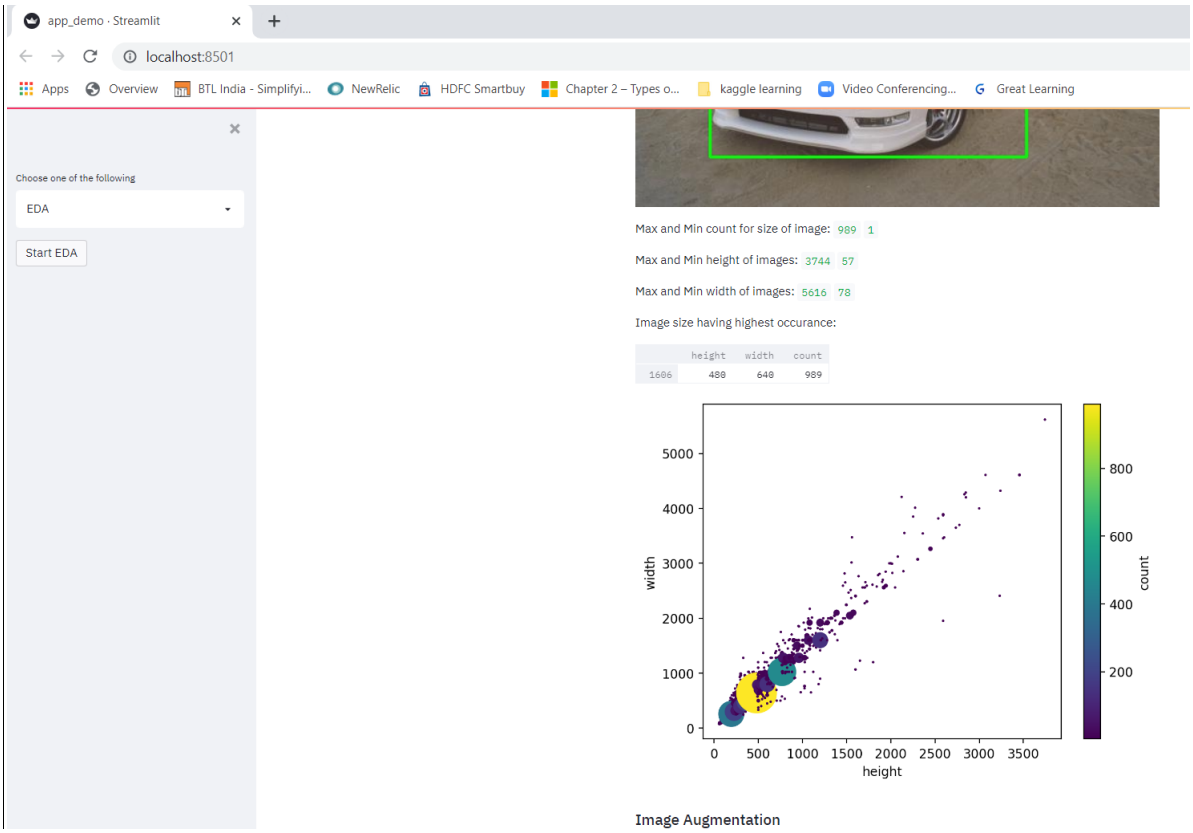


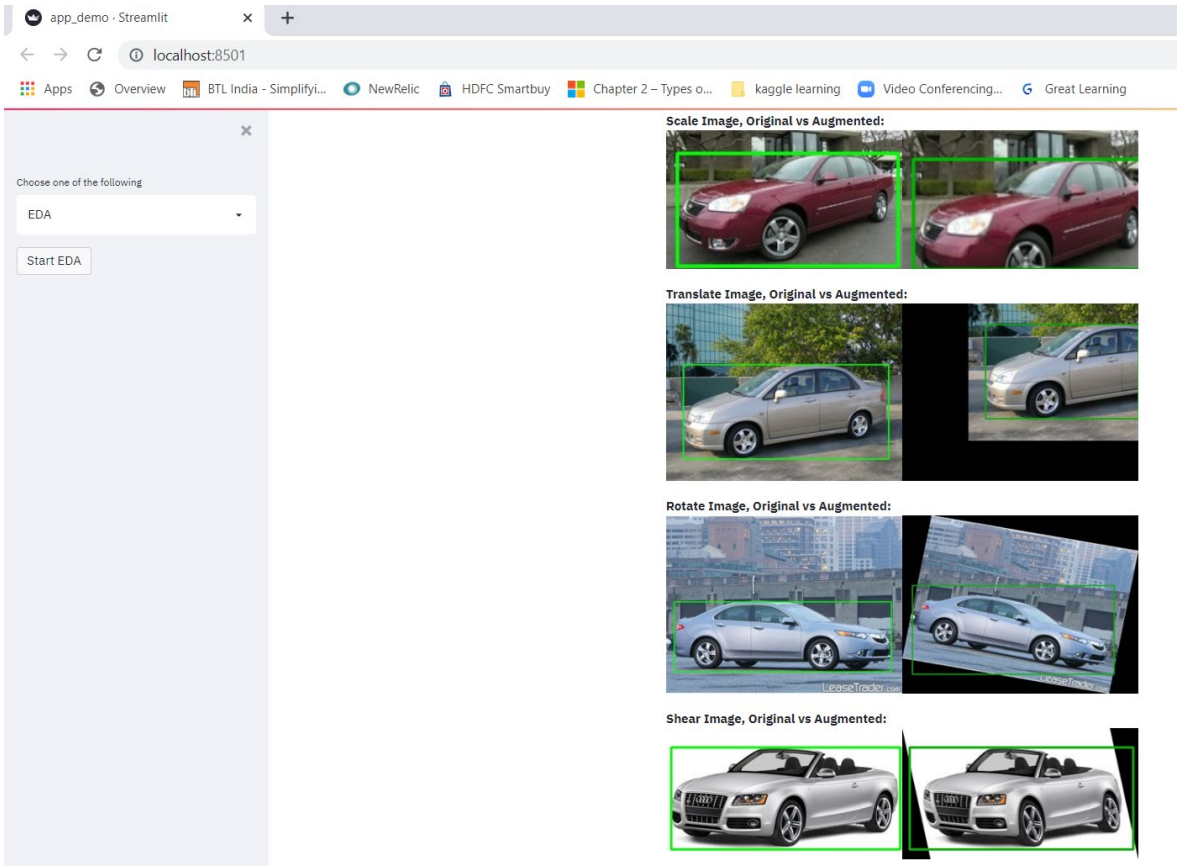
EDA Page

- We have to fill the location of the train folder, test folder, corresponding annotation file details.
- Once we have filled all data we can press 'Start EDA' button to start the process and visualize our data.



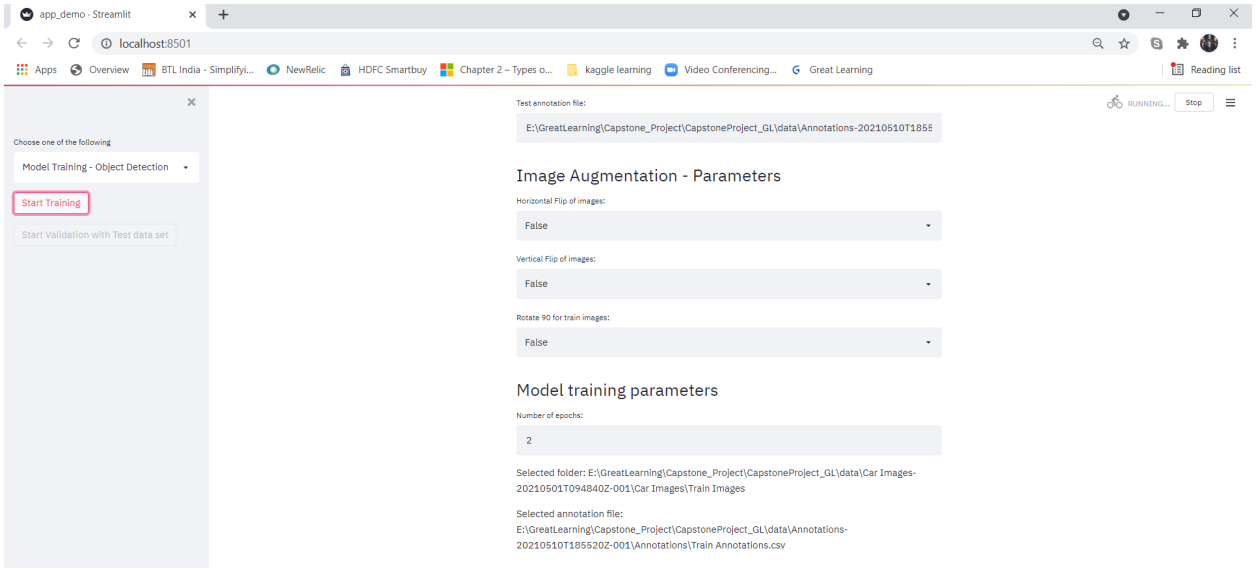


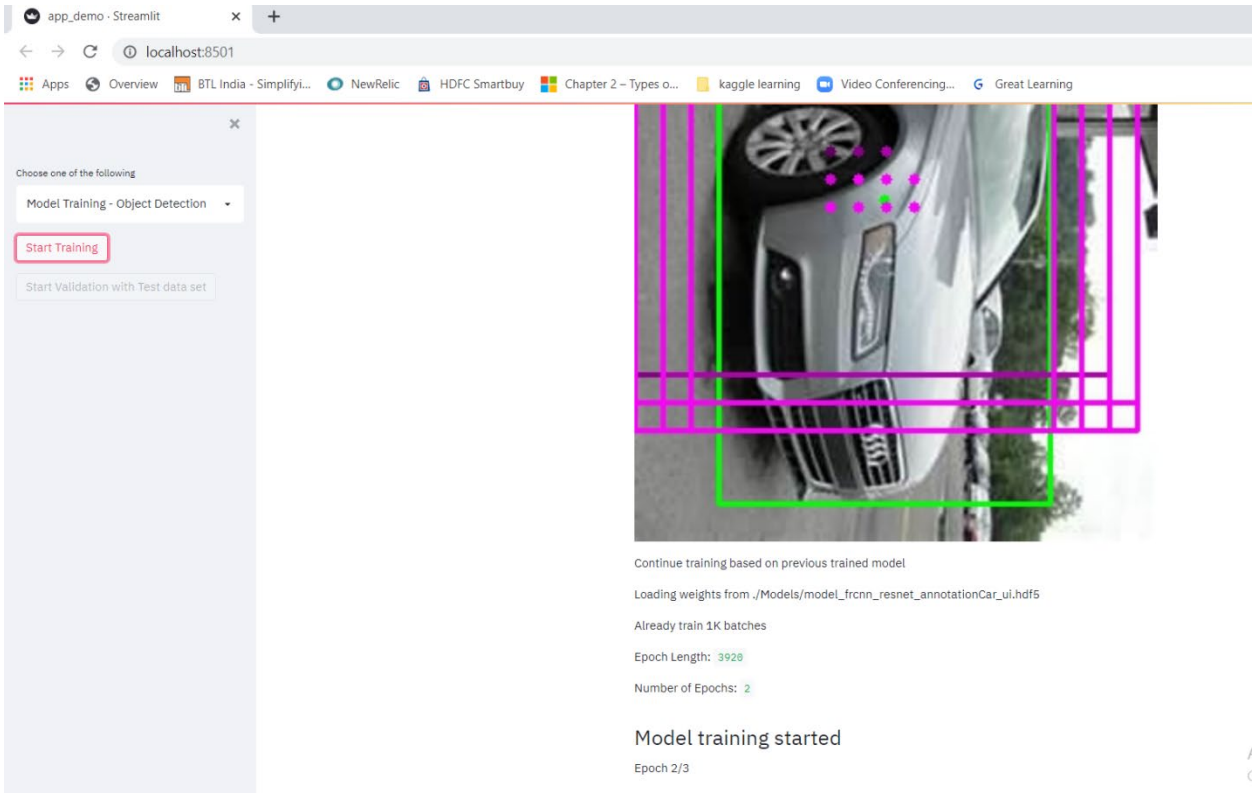

app_demo · Streamlit
localhost:8501
Apps
Overview
BTL India - Simplifyi...
NewRelic
HDFC Smartbuy
Chapter 2 – Types o...
kaggle learning
Video Conferencing...
Great Learning
Choose one of the following
EDA
Start EDA
Max and Min count for size of image: 989 1
Max and Min height of images: 3744 57
Max and Min width of images: 5616 78
Image size having highest occurance:
height width count
1606 480 640 989
width
height
count
Image Augmentation

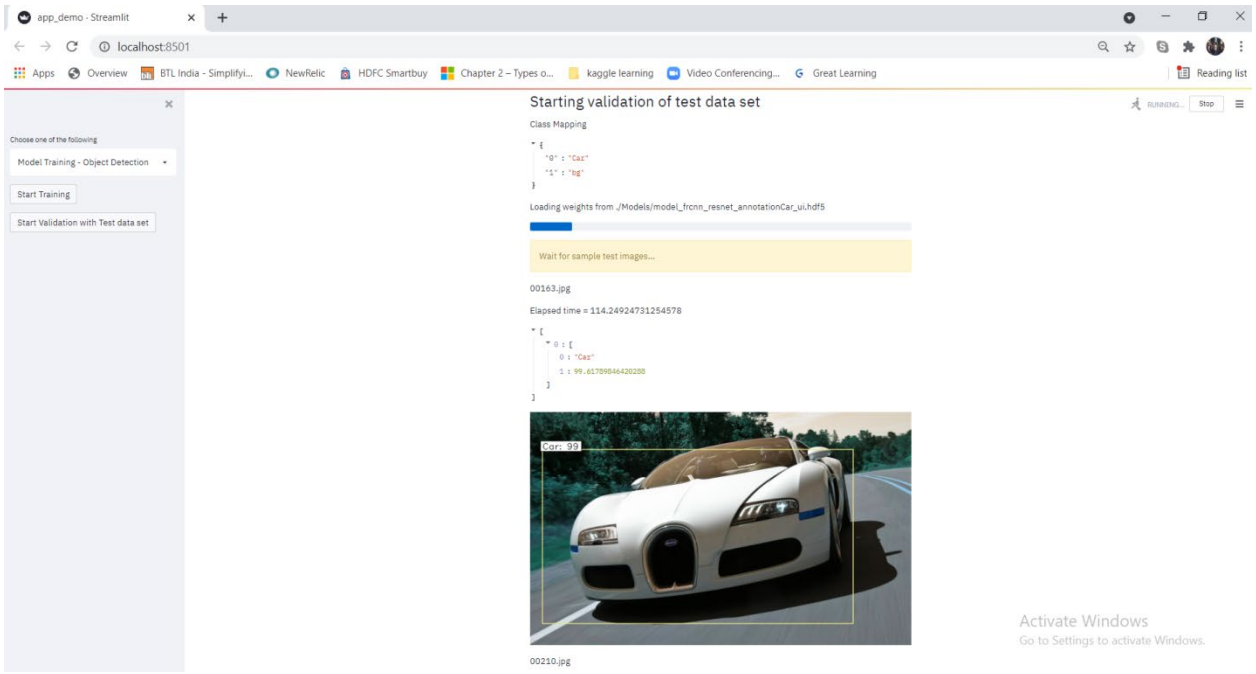

app_demo · Streamlit
localhost:8501
Apps
Overview
BTL India - Simplifyi...
NewRelic
HDFC Smartbuy
Chapter 2 – Types o...
kaggle learning
Video Conferencing...
Great Learning
Choose one of the following
EDA
Start EDA
Scale Image, Original vs Augmented:
Translate Image, Original vs Augmented:
Rotate Image, Original vs Augmented:
Shear Image, Original vs Augmented:

Model Training – Object Detection Page

- We can use this page to train a new model with new or old data set
- It is possible to provide multiple variables to give more freedom to user to interact to model and have more diverse operations while training. We kept the page simplistic and with necessary few parameters only to contact with main model for training.
- We need to fill the paths for our folders and files along with few image augmentation parameters that we want to use during training.
- Once the data is filled user can click on 'Start Training' button which will start the process.
- Post training is completed, user can click on 'Start Validation with Test data set' button, this will trigger validation of few random images and test validation set.

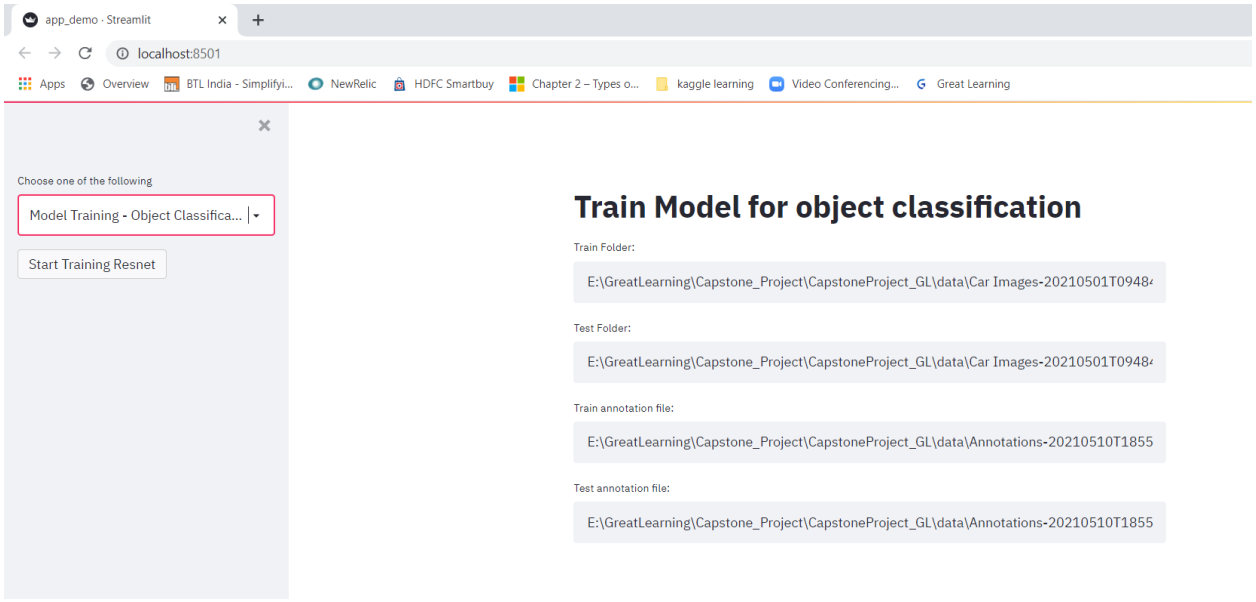


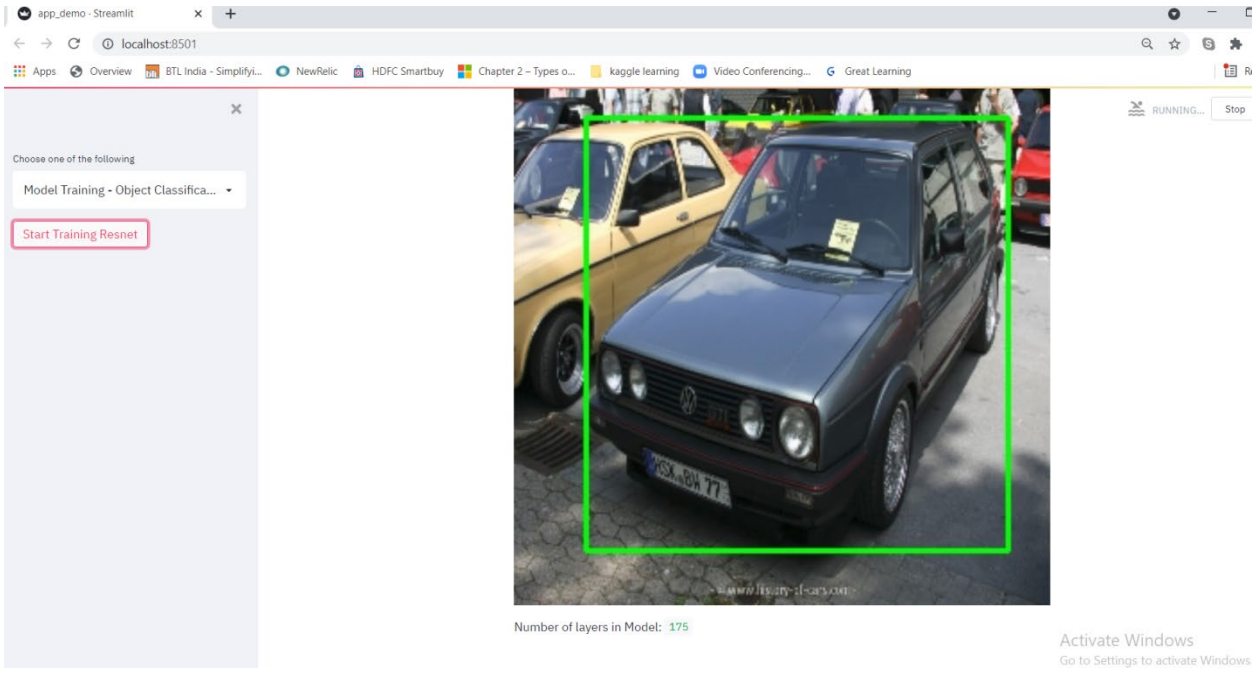

app_demo · Streamlit
localhost:8501
Apps
Overview
BTL India - Simplifyi...
NewRelic
HDFC Smartbuy
Chapter 2 – Types o...
kaggle learning
Video Conferencing...
Great Learning
Choose one of the following
Model Training - Object Detection
Start Training
Start Validation with Test data set
Continue training based on previous trained model
Loading weights from ./Models/model_frcnn_resnet_annotationCar_ui.hdf5
Already train 1K batches
Epoch Length: 3920
Number of Epochs: 2
Model training started
Epoch 2/3


app_demo · Streamlit
localhost:8501
Apps
Overview
BTL India - Simplifyi...
NewRelic
HDFC Smartbuy
Chapter 2 – Types o...
kaggle learning
Video Conferencing...
Great Learning
Reading list
Choose one of the following
Model Training - Object Detection
Start Training
Start Validation with Test data set
Starting validation of test data set
RUNNING...
Stop
Class Mapping
{
"0" : "Car"
"1" : "bg"
}
Loading weights from ./Models/model_frcnn_resnet_annotationCar_ui.hdf5
Wait for sample test images...
00163.jpg
Elapsed time = 114.24924731254578
[
0 : [
0 : "Car"
1 : 99.61709546420288
]
]
Car: 99
00210.jpg
Activate Windows
Go to Settings to activate Windows.

Model Training – Object Classification Page

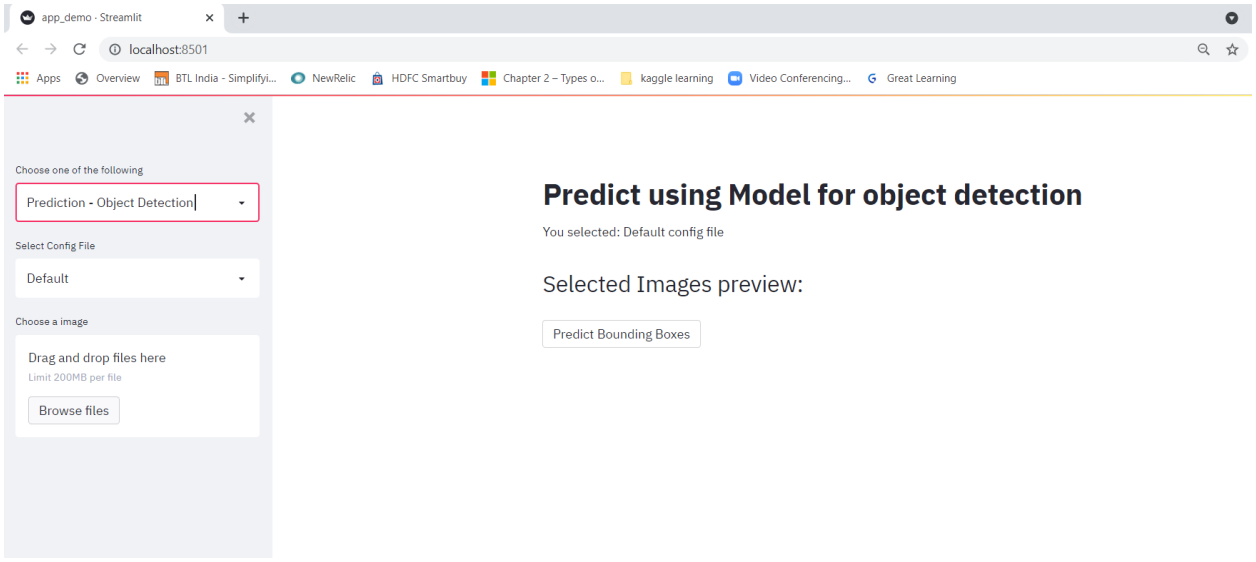
- User can fill mandatory paths for train & test details and click on start button to start training process for classification model.

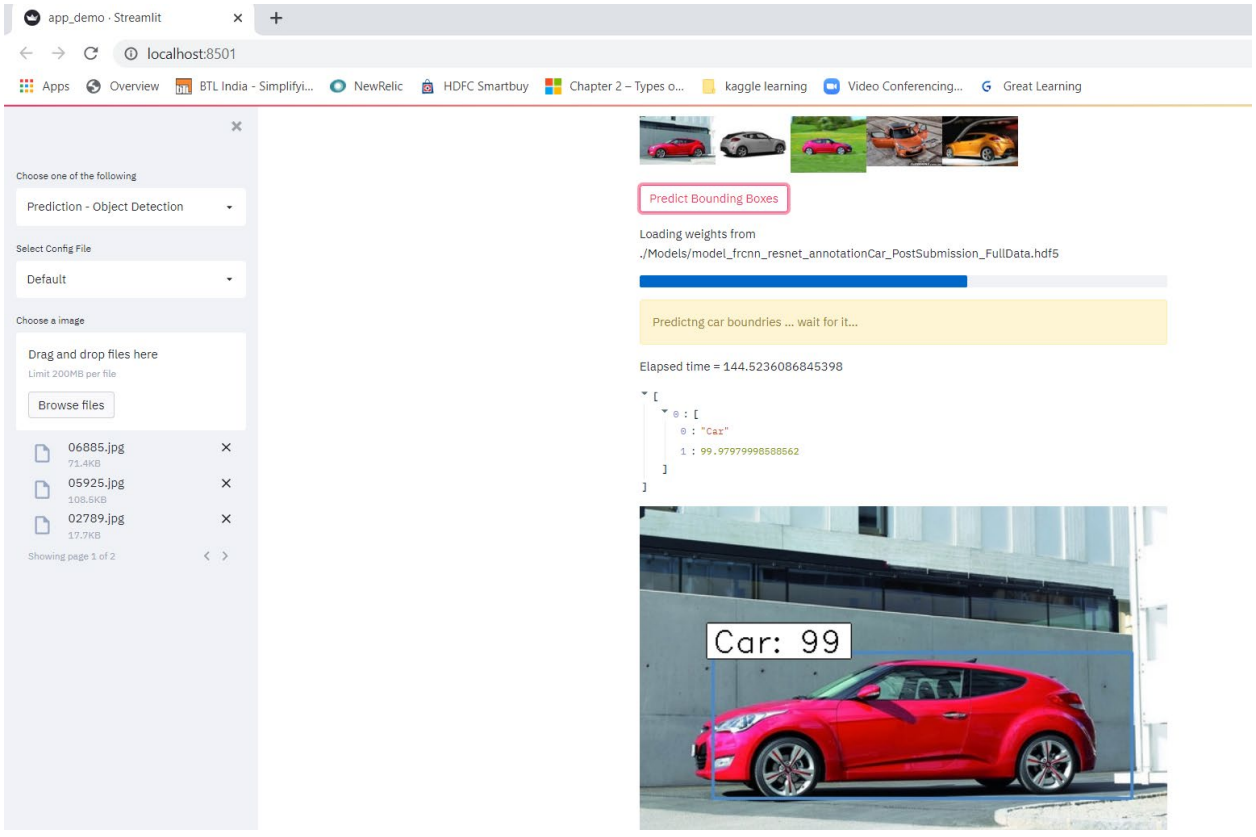




Prediction Object Detection Page

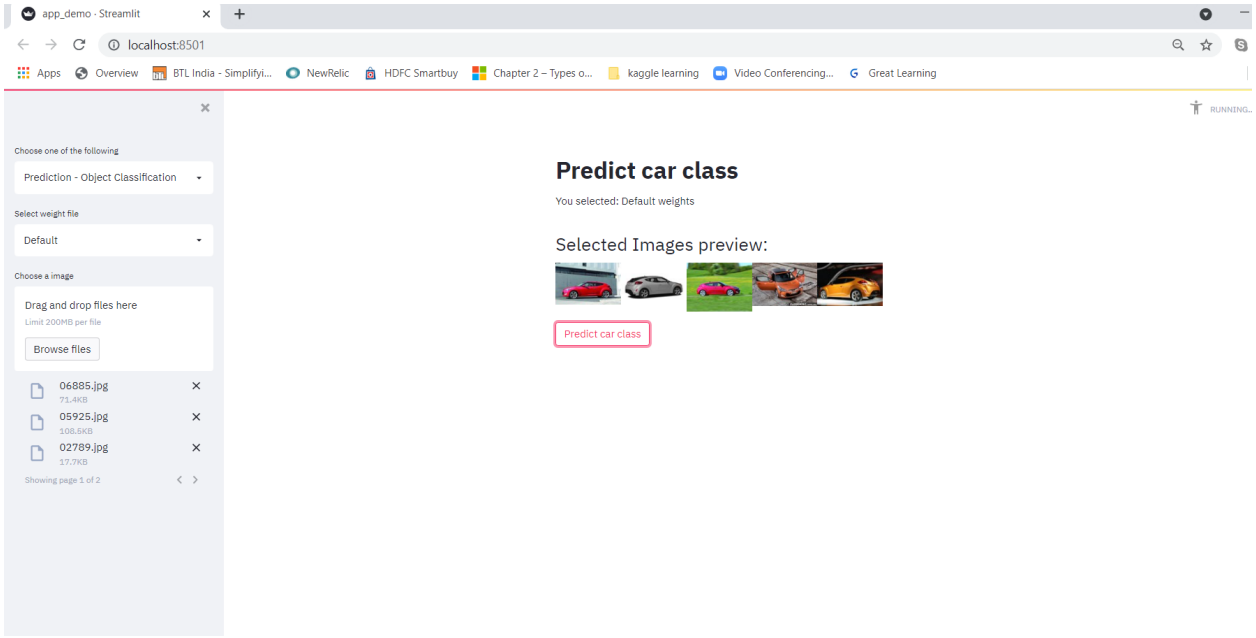
- On prediction page, user can browse multiple images and choose from option provided for model selection and predict the bounding boxes.

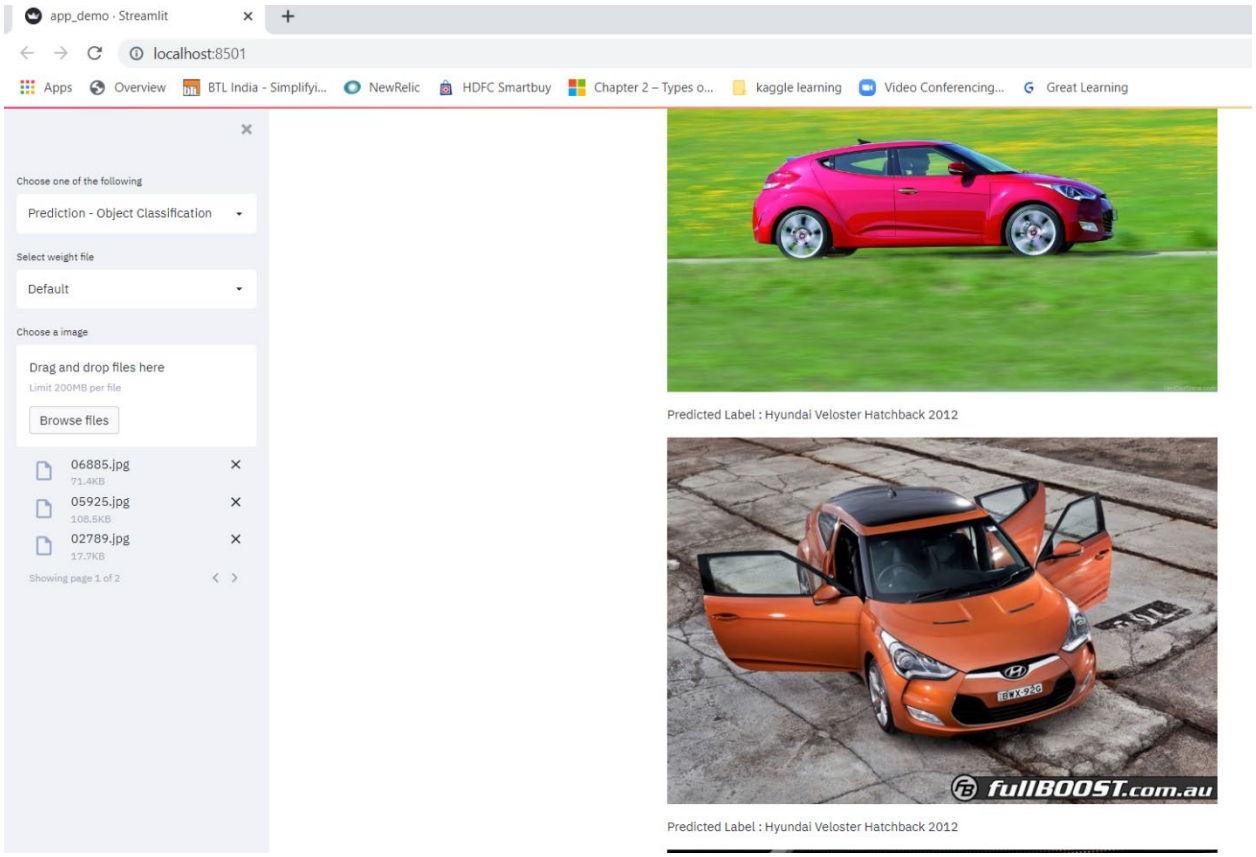




Prediction Object Classification Page

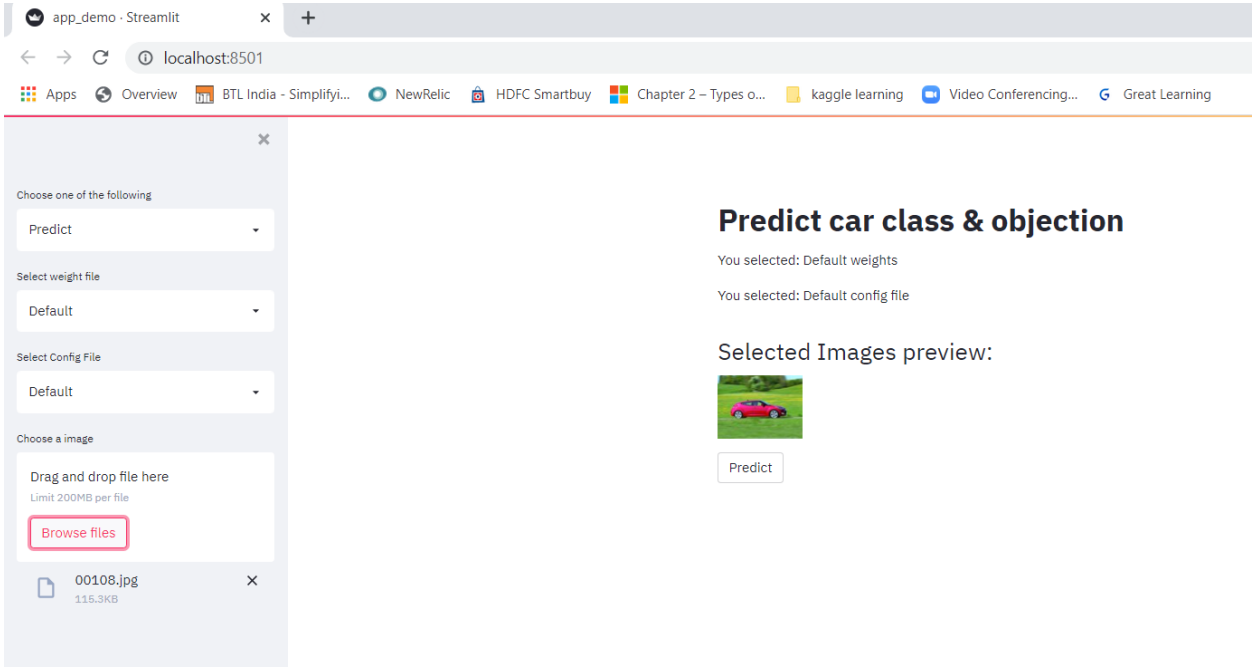
- On prediction page, user can browse multiple images and choose from option provided for model selection and predict the class of car.

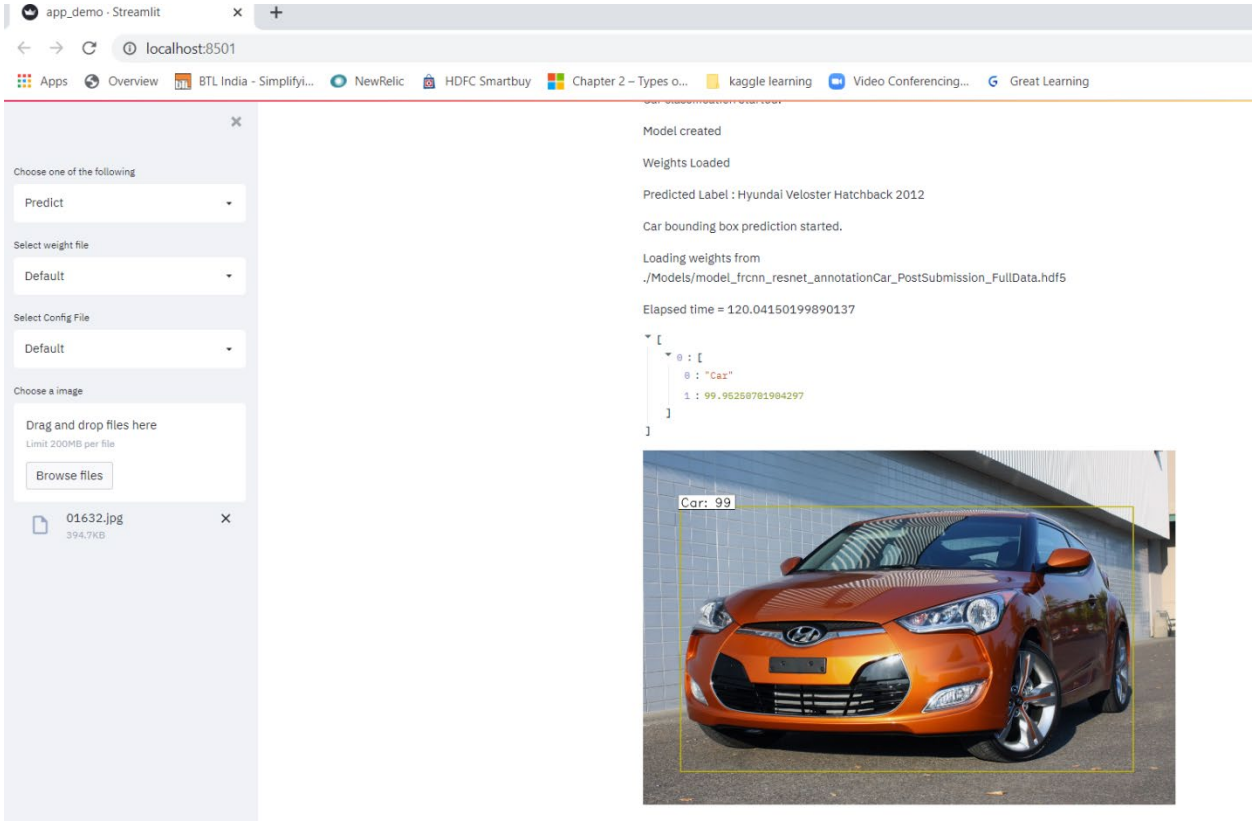




Prediction Page

- Single page to predict both bounding box and classification details as well.





Challenges Faced

- Issues faced during classification with usage of colab/kaggle notebooks
    - Colab or similar notebooks are very highly updated with latest codes and python codes are highly sensitive to its version compatibility
    - So, even if we take a working copy from our local, sometime we do have to spent hours to get it compatible with Colab and execute
    - Colab also has a issue with time bound and if the system remains idle it may get disconnected after some time

- Issues faced during Object detection
    - For these models, support or documents present are out dated and python and related libraries have moved a lot ahead
    - We have to carefully migrate existing code and update the parameters, functions and many related files as well to get it in working condition.

- Issues faced during UI
    - For UI, we have used a open source library, it has few basic features missing to make a more interactive UI pages.
    - We have to be very careful on the limitation and work around those limitations to design the pages.

Future Upgrade Plan

- With current evolving space of data science things will become obsolete very soon. To catch-up with current technology and tools we will be improving our models and UI along with it.

- Current UI has multiple limitations due to our design and due to tools side as well. This status quo can change any time with new data discovered, technologies appearing, updates coming and many other factors.
- Future Model Upgrade plans:
    - Try out with Pytorch
    - Try out new models to see how accuracies are getting affected

- Future plans for UI:
    - Deploy it in a server and use it on real time across web (we can do that now as well by opening the local port to internet)
    - More options for have control on model training