Great Learning

AIML

CAPSTONE PROJECT

FINAL REPORT

COMPUTER VISION

OBJECT DETECTION - CAR

Group Members:

Yatharth Deoly

Donkanna Jagannath

Itishree Dash Senapati

Rushikesh Upadhyay

Table of Contents

Summary of problem statement, data and findings

Stanford car dataset has been provided to design car identification model.
This dataset comprises of different formats of input i.e.,
Car Images.zip(Contains all the images of car)
Annotations.zip(Contains bounding-box coordinates of car image)
Car names and make.csv(Contains 196 unique car names with their build year).

Dataset contains 16,185 images which is split into 8,144 training images and 8,041 testing images, where each class has been split in a 50.31% training and 49.68% testing.

On the basis of images need to train model which will classify any new test car image as one of the car names from 196 unique class.

Let's define each input dataset:

1. "Car names and make.csv":
    a. Sample of data present in given csv file

```
car_name_df = pd.read_csv('Car names and make.csv', names=['Car_Name_&_Make'])
car_name_df.head()
```

	Car_Name_&_Make
0	AM General Hummer SUV 2000
1	Acura RL Sedan 2012
2	Acura TL Sedan 2012
3	Acura TL Type-S 2008
4	Acura TSX Sedan 2012

    b. Dataset dimension:

```
car_name_df.shape
```

```
(196, 1)
```

2. "Annotations.zip":
    a. Extract all data from zip achieve, which contains 'Train Annotations.csv' and 'Test Annotation.csv' files.
    b. These csv file contains bounding box region for both train and test images.
    c. 'Train Annotations.csv', sample of data present in given csv file:

```
annot_train_df = pd.read_csv('Annotations/Train Annotations.csv', header=0)
annot_train_df.columns=['Image_Name','x1','y1','x2','y2','Image_Class']
annot_train_df.head()
```

	Image_Name	x1	y1	x2	y2	Image_Class
0	00001.jpg	39	116	569	375	14
1	00002.jpg	36	116	868	587	3
2	00003.jpg	85	109	601	381	91
3	00004.jpg	621	393	1484	1096	134
4	00005.jpg	14	36	133	99	106

    d. Dimension of train annotations dataset:

```
annot_train_df.shape
```

```
(8144, 6)
```

    e. 'Test Annotation.csv', sample of data present in given csv file:

```
annot_test_df = pd.read_csv('Annotations/Test Annotation.csv', header=0)
annot_test_df.columns=['Image_Name','x1','y1','x2','y2','Image_Class']
annot_test_df.head()
```

	Image_Name	x1	y1	x2	y2	Image_Class
0	00001.jpg	30	52	246	147	181
1	00002.jpg	100	19	576	203	103
2	00003.jpg	51	105	968	659	145
3	00004.jpg	67	84	581	407	187
4	00005.jpg	140	151	593	339	185

f. Dimension of test annotations dataset:

```
annot_test_df.shape
```

```
(8041, 6)
```

3. "Car Images.zip":
   a. Extract all data from zip achieve, which contains 'Train Images' and 'Test Images' folder. Images are in RGB channel.
   b. Sample train images:

Bentley Arnage Sedan 2009

BMW X6 SUV 2012

   c. Sample test images;

Tesla Model S Sedan 2012

Rolls-Royce Ghost Sedan 2012

Sample images with bounding boxes:

Train Dataset bounding box

Hyundai Azera Sedan 2012

Ford F-150 Regular Cab 2007

Test Dataset bounding box

Hyundai Azera Sedan 2012

Ford F-150 Regular Cab 2007

Summary of the Approach to EDA and Pre-processing

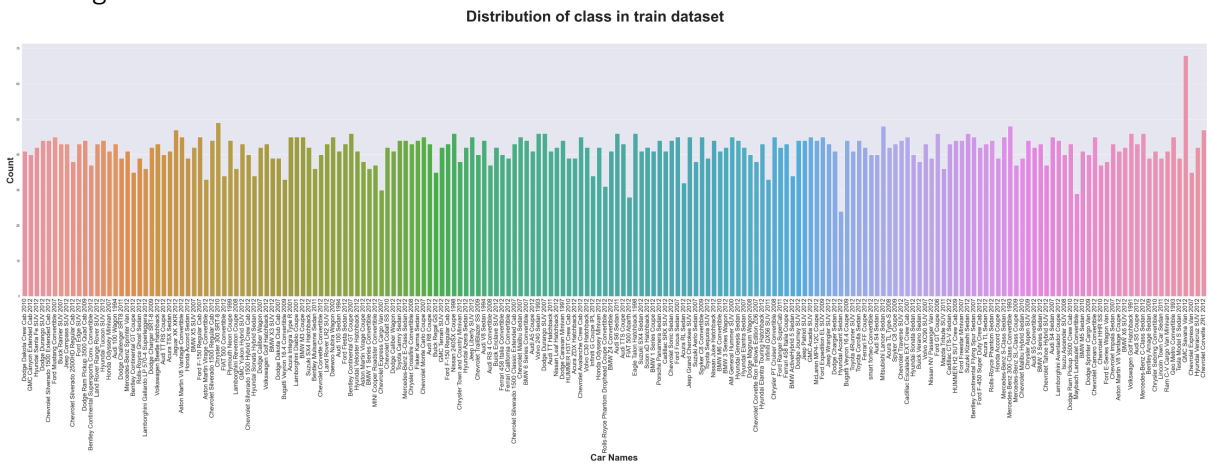
- As images provided are having different aspect ratio or height and width, so just to minimise the irrelevant data, have cropped the images using the boundary-box and extra margin of 15 spaces.
- As images were provided, so extracted them to matrix format.
- Unified all images to same height(224) and width(224) and then normalized pixels.

```
def convert(img_name, car_image_list):
    # Converting image to array
    cv_img=cv2.imread(img_name)

    # Unifying all images to same height and width and then normalizing pixels
    car_image_list.append(cv2.resize(cv_img, (224,224), interpolation = cv2.INTER_AREA).astype('float32')/255.0)
```

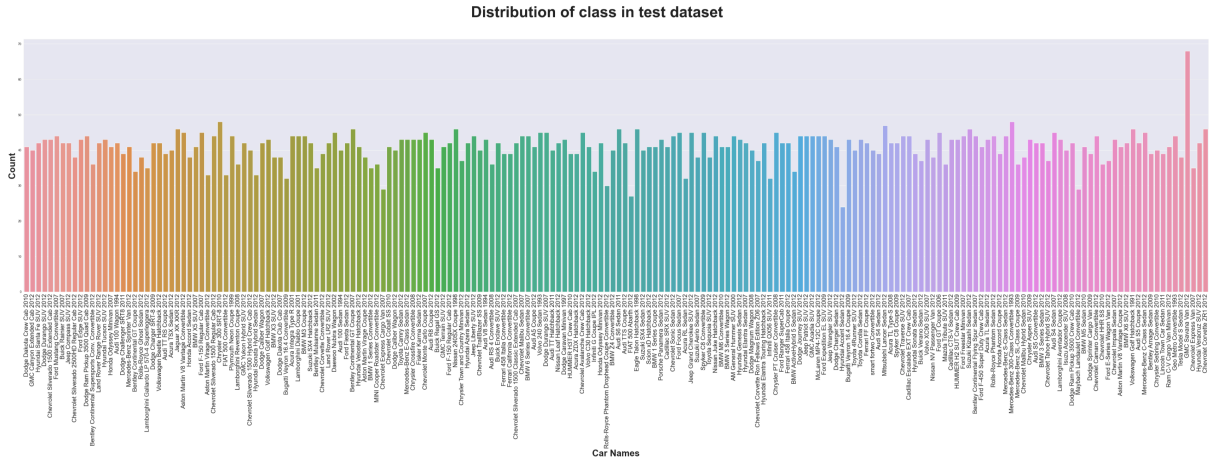
- Encoded target using labelEncoder after which categorised using keras package.

```
# Encoding labels of the images
y_train=annot_train_car_images_df['Image_Class']
y_train=le.fit_transform(y_train)
y_train=np_utils.to_categorical(y_train)
```

- Car name distribution:
  - Training set distribution:



  - Testing set distribution:



Deciding Models and Model Building

As the provided problem statement requirement is to identify car model on the basis of input image. Currently set of images have been provided on the basis of which we will train our model so that it can be used as car identification model.
So this type of model training comes Computer Vision where we will be making models from scratch and on the basis of that we will use other models.

In this milestone we will be using light models as training models can take ample time and because of resource constraint also, will be going for 2 shallow models and 1 deep model.

All models by default will be using 'Adam' as optimizer with call-backs functions as 'EarlyStopping', 'ModelCheckpoint' and ReduceLROnPlateau'

```
def create_callbacks(model):
    checkpoint=ModelCheckpoint(model+"_weights.h5",monitor='val_accuracy',
                               save_weights_only=True, mode='max',verbose=1)
    early_stop = EarlyStopping(patience=8, min_delta=1e-4)
    reduce_lr=ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=2,min_lr=0.00001,model='auto')

    return [checkpoint,reduce_lr,early_stop]
```

As models were taking ample time so have pickle the model for future usage.

Types model used are:

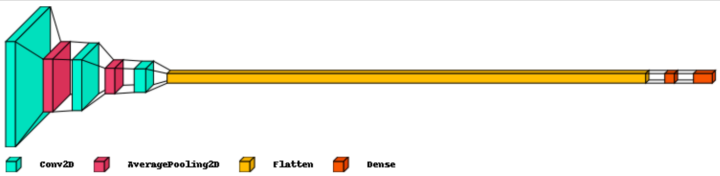
1. CNN with Architecture as : 3 Conv + 2 Avg pooling + 1 FC layer + 1 FC-SoftMax layer
    a. Model summarised:

```
## Looking into our base model
base_model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 220, 220, 6)       456

average_pooling2d (AverageP  (None, 110, 110, 6)       0
ooling2D)

conv2d_1 (Conv2D)            (None, 106, 106, 16)      2416

average_pooling2d_1 (Averag  (None, 53, 53, 16)        0
ePooling2D)

conv2d_2 (Conv2D)            (None, 49, 49, 120)       48120

flatten (Flatten)            (None, 288120)            0

dense (Dense)                (None, 84)                24202164

dense_1 (Dense)              (None, 196)               16660

=================================================================
Total params: 24,269,816
Trainable params: 24,269,816
Non-trainable params: 0
```

    b. Model summarised using visual keras package:

```
visualkeras_view(base_model_1)
```



    c. Model has been trained with 20 epochs and 50 batch size.
    d. Training Accuracy: 0.8409 with loss: 0.7045.
    e. Testing Accuracy: 0.0271 with loss: 9.8769.
    f. This model didn't worked well, will be checking for other models.

2. CNN with Architecture as : 5 Conv + 3 Max Pool + 2 FC layers + 1 FC-SoftMax layer
    a. Model summarised:

```
## Looking into our base model
base_model_2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 54, 54, 96)        34944

 batch_normalization (BatchN  (None, 54, 54, 96)       384
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 26, 26, 96)       0
 )

 conv2d_1 (Conv2D)           (None, 26, 26, 256)       614656

 batch_normalization_1 (Batc  (None, 26, 26, 256)      1024
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 384)       885120

 batch_normalization_2 (Batc  (None, 12, 12, 384)      1536
 hNormalization)

 conv2d_3 (Conv2D)           (None, 12, 12, 384)       1327488

 batch_normalization_3 (Batc  (None, 12, 12, 384)      1536
 hNormalization)

 conv2d_4 (Conv2D)           (None, 12, 12, 256)       884992

 batch_normalization_4 (Batc  (None, 12, 12, 256)      1024
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 5, 5, 256)        0
 2D)

 flatten (Flatten)           (None, 6400)              0

 dense (Dense)               (None, 4096)              26218496

 dropout (Dropout)           (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 196)               803012

=================================================================
Total params: 47,555,524
Trainable params: 47,552,772
Non-trainable params: 2,752
```

    b. Model summarised using visual keras package:

```
visualkeras_view(base_model_2)
```

    c. Model has been trained with 50 epochs and 50 batch size.
    d. Training Accuracy: 0.6720 with loss: 1.6347.
    e. Testing Accuracy: 0.1970 with loss: 3.6971.
    f. This model didn't worked well, will be checking for other models.

3. CNN with Architecture as : ResNet50 deep layer
    a. Model summarised, only last part attached, as this model is a deep layer model.

```
## Looking into our base model
base_model_3.summary()
```

```
 batch_normalization_52 (BatchN  (None, 7, 7, 2048)  8192        ['conv2d_52[0][0]']
 ormalization)

 add_15 (Add)                   (None, 7, 7, 2048)   0           ['batch_normalization_52[0][0]',
                                                                  'activation_45[0][0]']

 activation_48 (Activation)     (None, 7, 7, 2048)   0           ['add_15[0][0]']

 average_pooling2d (AveragePool  (None, 4, 4, 2048)  0           ['activation_48[0][0]']
 ing2D)

 flatten (Flatten)              (None, 32768)        0           ['average_pooling2d[0][0]']

 dense (Dense)                  (None, 196)          6422724     ['flatten[0][0]']

==================================================================================================
Total params: 30,010,436
Trainable params: 29,957,316
Non-trainable params: 53,120
```

    b. Model has been trained with 20 epochs and 50 batch size.
    c. Training Accuracy: 0.6720 with loss: 1.6347.
    d. Testing Accuracy: 0.1970 with loss: 3.6971.
    e. This model didn't worked well, will be checking for other models.

How to improve your model performance

Let’s compare scores for models trained from scratch.

```
pd.DataFrame(model_list,columns=['Model','Train_Loss','Test_Loss','Train_Accuracy','Test_Accuracy','Precision','Recall','F1 Score']).sort_values(by=['Recall','F1 Score'],
 ascending=False)
```

	Model	Train_Loss	Test_Loss	Train_Accuracy	Test_Accuracy	Precision	Recall	F1 Score
1	base_model_2	1.634652	3.697144	0.672028	0.196990	0.196990	0.196990	0.196990
2	base_model_3	3.095142	10.225157	0.997667	0.127223	0.127223	0.127223	0.127223
0	base_model_1	0.704541	9.876921	0.840864	0.027111	0.027111	0.027111	0.027111

Concluding what all things we can do to improve our model performance:

- Neural networks rely on loads of good training data to learn patterns from dataset. So we can apply image augmentation to increase dataset size by rotating, zooming, cropping and more pattern.
- As our model seems to be overfitting for some case, we can lower the learning rate so that at lower epochs model don’t achieve overfitting.
- Reshuffling the training data in between epochs can help model to understand dataset in different manner for every epochs.
- As we are developing model from scratch because of which training is taking much time. So for this we can use transfer learning and can use pre-trained models with pre-trained weights, just by changing some layers. We can use models like, EfficientNet-B4, MobileNet, ResNet-152 and more.
- We can use K-Fold Cross-Validation, which will divide the images into K parts of equal size and then train the model K number times with different training & validation set.

Overview of the final process

- In milestone-1, we have achieved model training with some EDA, data pre-processing and data visualisation. In this milestone-2 we will focus more over improvising models and using different transfer learning mechanism.
- As we have less data, we will be using image augmentation technique to increase dataset size by rotating, zooming, cropping, flips and more patterns.
- Once we have good amount of training dataset, we will be doing hyper-parameter tuning for the previous best model. Hyper-parameter tuning will include increase in dense layer with Dropout and BatchNormalization, increasing learning_rate with different optimizers, changing epochs, steps_per_epoch and more.
- Next step we will be training model using transfer learning mechanism where we will use ‘MobileNetV2’ as base model with weights of 'ImageNet'. After some changes over base model, we will train our model.
- So basically till now we were just classifying the images, now we will train models for defining boundary boxes with image classifier.
- To know more better for bbox and image classification, we will be using existing model i.e., ‘faster_rcnn_nas_coco_2018_01_28’ which is pretrained with coco dataset. So using this pre-trained model with our dataset and will check for results.
- Using transfer learning on ‘MobileNetV2’ we will train a model which will be resulting boundary box and class label. So for that, we will construct pre-model layers with some new modelling layer. As this model will be providing 2 details so accuracy check will also improvise.
- As all above models will be pickled or saved for future use, so we will make an flask and react application where user can upload image and will get result with boundary box and class label printed on uploaded image.

Step-by-step walk through the solution

i. Image Augmentation

As neural networks rely on loads of good training data to learn patterns from dataset. So we applying ImageDataGenerator to get different set of images for train and test dataset that can be used for further model training.

```
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input,
                                 rotation_range=20,
                                 shear_range = 0.2,
                                 width_shift_range=0.2,
                                 height_shift_range=0.1,
                                 zoom_range=0.2,
                                 horizontal_flip=True,
                                 vertical_flip=True,
                                 brightness_range=[0.1,1.0]
                                 )

test_datagen=ImageDataGenerator(preprocessing_function=preprocess_input,
                                 rotation_range=20,
                                 shear_range = 0.2,
                                 width_shift_range=0.2,
                                 height_shift_range=0.1,
                                 zoom_range=0.2,
                                 horizontal_flip=True,
                                 vertical_flip=True,
                                 brightness_range=[0.1,1.0]
                                 )
```

```
train_generator=train_datagen.flow_from_directory(
        directory="Car Images/Train Images Cropped/",
        batch_size=64,
        seed=random_state,
        target_size=(224,224),
        class_mode = 'categorical',
        shuffle=True)


test_generator=test_datagen.flow_from_directory(
    directory="Car Images/Test Images Cropped/",
    batch_size=64,
    seed=random_state,
    target_size=(224,224),
    class_mode = 'categorical',
    shuffle=True)
```

```
Found 8144 images belonging to 196 classes.
Found 8041 images belonging to 196 classes.
```

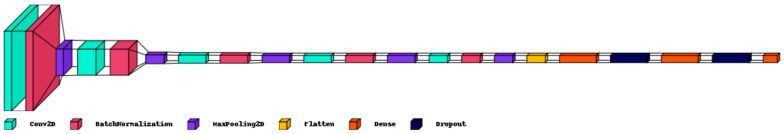
ii. Tuning old scratch model

CNN with Architecture as : 5 Conv + 5 Max Pool + 2 FC layers + 1 FC-SoftMax layer

a. Model Summary:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 96)      2688

batch_normalization (BatchN  (None, 222, 222, 96)      384
ormalization)

max_pooling2d (MaxPooling2D  (None, 74, 74, 96)        0
)

conv2d_1 (Conv2D)            (None, 72, 72, 256)       221440

batch_normalization_1 (Batc  (None, 72, 72, 256)       1024
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 24, 24, 256)       0
2D)

conv2d_2 (Conv2D)            (None, 22, 22, 384)       885120

batch_normalization_2 (Batc  (None, 22, 22, 384)       1536
hNormalization)

max_pooling2d_2 (MaxPooling  (None, 11, 11, 384)       0
2D)

conv2d_3 (Conv2D)            (None, 9, 9, 384)         1327488

batch_normalization_3 (Batc  (None, 9, 9, 384)         1536
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 4, 4, 384)         0
2D)

conv2d_4 (Conv2D)            (None, 2, 2, 256)         884992

batch_normalization_4 (Batc  (None, 2, 2, 256)         1024
hNormalization)

max_pooling2d_4 (MaxPooling  (None, 1, 1, 256)         0
2D)

flatten (Flatten)            (None, 256)               0

dense (Dense)                (None, 512)               131584

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: 3,822,020
Trainable params: 3,819,268
Non-trainable params: 2,752
```

b. Model summarised using visual keras package:



c. Model has been trained with 50 epochs, 100 steps_per_epoch, and 50 validation_steps with fit_generator.
d. Training Accuracy: 0.0323 with loss: 5.0460.
e. Testing Accuracy: 0.0298 with loss: 5.0862.
f. This model tuning didn't worked well, will be checking for other models.

iii. Initializing MobileNet CNN with Architecture using transfer learning

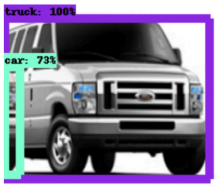
a. Base model as MobileNetV2 with input_shape as (224,224,3) and weights as 'ImageNet'.
b. Defining user model over the base model with starting 70 layers as not trainable and adding dense layer with 196 as SoftMax output.
c. Model summary trimmed:

```
Conv_1 (Conv2D)                (None, 7, 7, 1280)   409600      ['block_16_project_BN[0][0]']

Conv_1_bn (BatchNormalization)  (None, 7, 7, 1280)  5120        ['Conv_1[0][0]']

out_relu (ReLU)                (None, 7, 7, 1280)   0           ['Conv_1_bn[0][0]']

global_average_pooling2d (Glob  (None, 1280)        0           ['out_relu[0][0]']
alAveragePooling2D)

dropout (Dropout)              (None, 1280)         0           ['global_average_pooling2d[0][0]'
                                                                ]

dense (Dense)                  (None, 196)          251076      ['dropout[0][0]']

==================================================================================================
Total params: 2,509,060
Trainable params: 2,344,260
Non-trainable params: 164,800
__________________________________________________________________________________________________
```

d. Training Accuracy: 0.9815 with loss: 0.1280.
e. Testing Accuracy: 0.7929 with loss: 0.7583.
f. This model have performed well and can be further more tune but its already taking 11h 4m 6s to run this.

iv. Using pre-trained models of coco 'faster_rcnn_nas_coco_2018_01_28'

a. This exercise is just to check how pre-trained model shows bbox for different set of images.
b. Model has been downloaded from this site and have used with car dataset.
c. Train Image prediction using coco pre-trained set:



d. Test Image prediction using coco pre-trained set:

e. Boundary box and labelling understanding has been done via this model, now will train a faster rcnn model which will provide both bbox and class label.

Model evaluation

Faster-RCNN model has been picked as final model that will be used as future prediction of class labels and boundary boxes of user provided images.

- Training dataset have 8144 train image that are categorised into 196 different car categories. So using 'imgaug' one of image augmenter, we will be creating another set of training images which will be clubbed with original training dataset images.

```
aug_bbs_xy.shape,train_images_df.shape
```

```
((8144, 6), (8144, 6))
```

After clubbing both set into one final training dataset.

```
# After merging original and augmented dataset
final_train_df.shape
```

```
(16288, 6)
```

Visualising images from final dataset:

```
Class Name -> Aston Martin V8 Vantage Convertible 2012
```

```
Class Name -> Chrysler Town and Country Minivan 2012
```

Final dataset will be pre-processed for modelling. Pre-processing will include scaling, resizing, bbox updating and after that converting images to array format.

```
# Scaling final dataset
trainImagePaths=[]
trainBboxes =[]
trainLabels=[]

trainData= np.zeros((len(final_train_df), 224, 224,3), dtype=np.float32)

for i in range(len(final_train_df)):
    row_obj=final_train_df.iloc[i]
    fname=row_obj[0]
    label=row_obj[1]
    bbx1=row_obj[2]
    bbx2=row_obj[3]
    bbx3=row_obj[4]
    bbx4=row_obj[5]

    #appending values to list
    trainImagePaths.append(fname)
    trainLabels.append(label)

    ## operations on images and bounding boxes
    org_img=cv2.imread(fname)
    (h,w)=org_img.shape[:2]
    startX=float(bbx1)/w
    startY=float(bbx2)/h
    endX=float(bbx3)/w
    endY=float(bbx4)/h
    trainBboxes.append((startX,startY,endX,endY))

    trainData[i]=cv2.resize(org_img, dsize=(224, 224), interpolation=cv2.INTER_AREA)


trainData=np.array(trainData, dtype='float32')/255.0
trainLabels=np.array(trainLabels)
trainBboxes=np.array(trainBboxes, dtype='float32')
trainImagePaths=np.array(trainImagePaths)

np.save('Car Images/trainData.npy', trainData)
np.save('Car Images/trainLabels.npy', trainLabels)
np.save('Car Images/trainBboxes.npy', trainBboxes)
np.save('Car Images/trainImagePaths.npy', trainImagePaths)
```

On hot encoding will be performed over labels using 'LabelBinarizer'.

```
## performing one hot encoding on the labels
trainLabels_lb=lb.fit_transform(trainLabels)
```

Printing shapes of all train dataset.

```
trainData.shape, trainBboxes.shape, trainLabels_lb.shape
```

```
((16288, 224, 224, 3), (16288, 4), (16288, 196))
```

- Testing dataset have 8041 test image that are categorised into 196 different car categories.
  Test dataset will be pre-processed for modelling. Pre-processing will include scaling, resizing, bbox updating and after that converting images to array format.

```
# Scaling test dataset
testImagePaths=[]
testBboxes =[]
testLabels=[]

testData= np.zeros((len(annot_test_df), 224, 224,3), dtype=np.float32)

test_folder=glob.glob('Car Images/Test Images/*/*')
for j in range(len(test_folder)):
    index=test_file_names.index(test_folder[j].split('/')[-1])

    (x1,y1,x2,y2)=test_bboxes[index]
    test_filename=test_folder[j].split('/')[-1]
    classname=test_folder[j].split('/')[-2]
    test_file_path=os.path.join('Car Images/Test Images/'+classname+'/'+test_filename)

    testImagePaths.append(test_file_path)
    testLabels.append(classname)

    # Resizing the coordinates
    originalImage=cv2.imread(test_file_path)
    (h,w)=originalImage.shape[:2]
    startX=float(x1)/w
    startY=float(y1)/h
    endX=float(x2)/w
    endY=float(y2)/h
    testBboxes.append((startX,startY,endX,endY))
    testData[j] = cv2.resize(originalImage, dsize=(224, 224),interpolation=cv2.INTER_AREA)

testData=np.array(testData, dtype='float32')/255.0
testLabels=np.array(testLabels)
testBboxes=np.array(testBboxes, dtype='float32')
testImagePaths=np.array(testImagePaths)

np.save('Car Images/testData.npy', testData)
np.save('Car Images/testLabels.npy', testLabels)
np.save('Car Images/testBboxes.npy', testBboxes)
np.save('Car Images/testImagePaths.npy', testImagePaths)
```

  On hot encoding will be performed over labels using ‘LabelBinarizer’.

```
## performing one hot encoding on the labels
testLabels_lb=lb.fit_transform(testLabels)
```

  Printing shapes of all test dataset.

```
testData.shape, testBboxes.shape, testLabels_lb.shape
```

```
((8041, 224, 224, 3), (8041, 4), (8041, 196))
```

- As we will be using transfer learning with base model as MobileNetV2 with input_shape as (224,224,3) and weights as ‘ImageNet’, with first 70 layers as non-trainable.

```
backend.clear_session()
tf.random.set_seed(random_state)

mobile_net_model_new = MobileNetV2(input_shape=(224,224,3), include_top=False, weights='imagenet')

for layer in mobile_net_model_new.layers[:70]:
    layer.trainable=False
for layer in mobile_net_model_new.layers[70:]:
    layer.trainable=True

flatten=Flatten()(mobile_net_model_new.output)
```

- Our model will be classifying bbox and class label, so will be plugging some extra dense layer to pre-existing model.

```
# construct a fully-connected layer header to output the predicted bounding box coordinates
bboxHead = Dense(128, activation="relu")(flatten)
bboxHead = Dense(64, activation="relu")(bboxHead)
bboxHead = Dense(32, activation="relu")(bboxHead)
bboxHead = Dense(4, activation="sigmoid",name="bounding_box")(bboxHead)

# construct a second fully-connected layer head, this one to predict the class label
softmaxHead = Dense(512, activation="relu")(flatten)
softmaxHead = Dense(256, activation="relu")(softmaxHead)
softmaxHead = Dense(128, activation="relu")(softmaxHead)
softmaxHead = Dense(196, activation="softmax",name="class_label")(softmaxHead)
```

- As our model will predicting 2 things, i.e., bbox and labels, which will be numerical and categorical format, so our losses will contains both ‘mean_squared_error‘ and ‘categorical_crossentropy’.

```
losses = {
    "class_label": categorical_crossentropy,
    "bounding_box": "mean_squared_error",
}
```

- Adam optimizer will be used with learning_rate as 0.001, metrics as ‘accuracy’.

```
# put together our model which accept an input image and then output bounding box coordinates and a class label
model_mobile_net_bb = Model(inputs=mobile_net_model_new.input, outputs=(bboxHead, softmaxHead))

adam = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, decay=0.001)

model_mobile_net_bb.compile(optimizer=adam, loss= losses, metrics=['accuracy'],loss_weights=lossWeights)
```

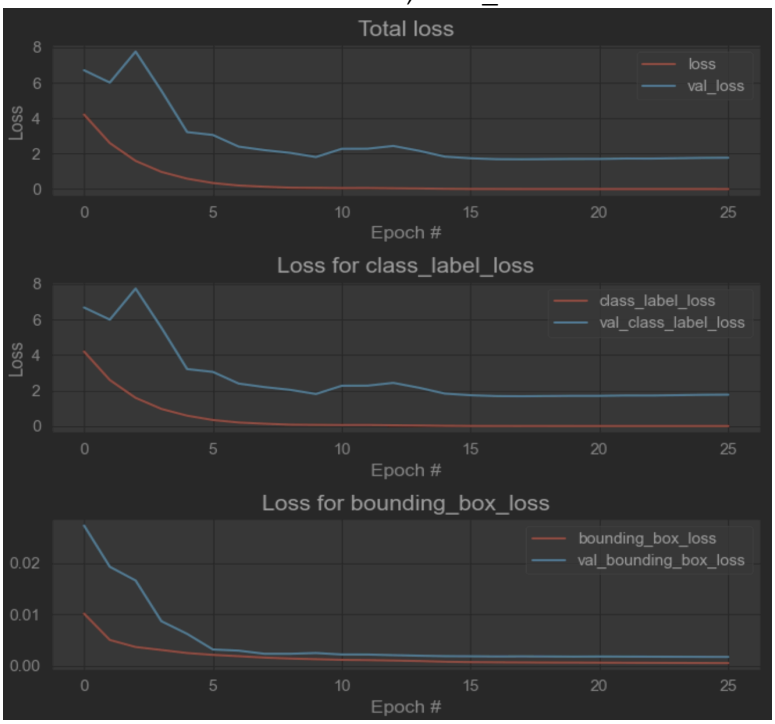
- Model summary trimmed:

```
 dense_3 (Dense)                (None, 512)          32113152    ['flatten[0][0]']

 dense_1 (Dense)                (None, 64)           8256        ['dense[0][0]']

 dense_4 (Dense)                (None, 256)          131328      ['dense_3[0][0]']

 dense_2 (Dense)                (None, 32)           2080        ['dense_1[0][0]']

 dense_5 (Dense)                (None, 128)          32896       ['dense_4[0][0]']

 bounding_box (Dense)           (None, 4)            132         ['dense_2[0][0]']

 class_label (Dense)            (None, 196)          25284       ['dense_5[0][0]']

==================================================================================================
Total params: 42,599,400
Trainable params: 42,434,600
Non-trainable params: 164,800
```

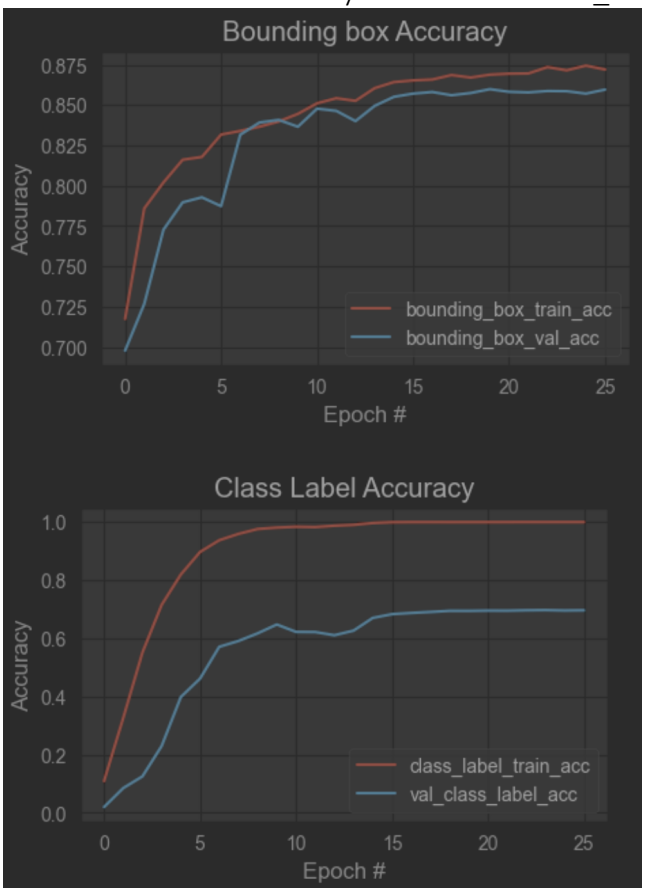
- Model training has taken 6h43m24s, where fit was provided with epochs as 50, batch_size as 50 but due to early stop, at 26 epochs only it got terminated.

```
# Fit the model
model_mobile_net_bb_history = model_mobile_net_bb.fit(trainData, trainTargets, validation_data=(testData, testTargets), epochs=50,
 batch_size=50, verbose=1, callbacks=create_callbacks('model_mobile_net_bb'))
Executed at 2023.06.23 00:06:45 in 6h 43m 24s
326/326 [==============================] - ETA: 0s - loss: 0.0033 - bounding_box_loss: 5.2044e-04 - class_label_loss: 0.0028 -
 bounding_box_accuracy: 0.8744 - class_label_accuracy: 0.9988
Epoch 25: saving model to model_mobile_net_bb_weights.h5
326/326 [==============================] - 855s 3s/step - loss: 0.0033 - bounding_box_loss: 5.2044e-04 - class_label_loss: 0.0028 -
 bounding_box_accuracy: 0.8744 - class_label_accuracy: 0.9988 - val_loss: 1.7631 - val_bounding_box_loss: 0.0017 -
val_class_label_loss: 1.7613 - val_bounding_box_accuracy: 0.8571 - val_class_label_accuracy: 0.6951 - lr: 1.0000e-04
Epoch 26/50
326/326 [==============================] - ETA: 0s - loss: 0.0029 - bounding_box_loss: 5.1107e-04 - class_label_loss: 0.0024 -
bounding_box_accuracy: 0.8721 - class_label_accuracy: 0.9988
Epoch 26: saving model to model_mobile_net_bb_weights.h5
326/326 [==============================] - 862s 3s/step - loss: 0.0029 - bounding_box_loss: 5.1107e-04 - class_label_loss: 0.0024 -
 bounding_box_accuracy: 0.8721 - class_label_accuracy: 0.9988 - val_loss: 1.7712 - val_bounding_box_loss: 0.0017 -
val_class_label_loss: 1.7695 - val_bounding_box_accuracy: 0.8596 - val_class_label_accuracy: 0.6958 - lr: 1.0000e-04
```

- Training Accuracy: bounding_box-> 0.8721 and class_label-> 0.9988.
- Testing Accuracy: bounding_box-> 0.8596 and class_label -> 0.6958.
- Model has been saved using pickle and keras save.
- Now let’s visualise loss for bbox, class_label and total loss.



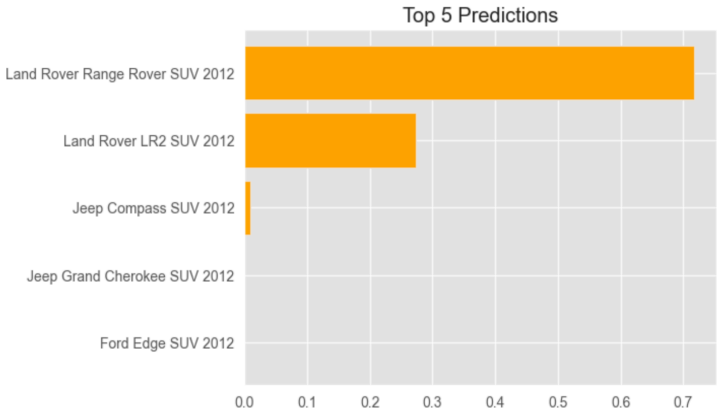
- Now let’s visualise accuracy for bbox and class_label.



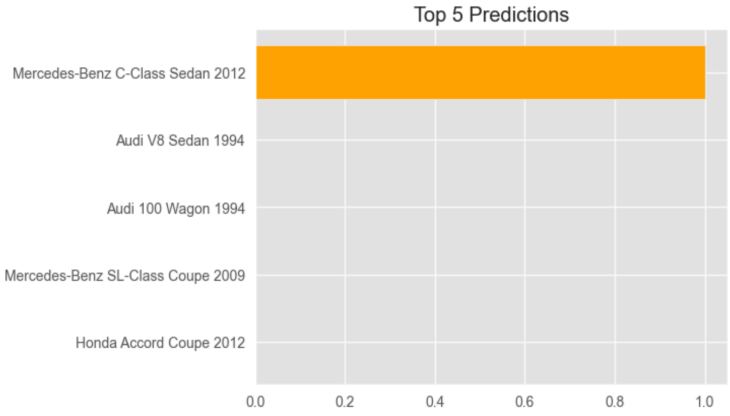
- Created a predict function which will load image from path, then resize, scale and convert to array. This function will use save model to predict bbox and label. This function will print image with bbox and label name on image and also plot a graph for total top 5 predictions.

```
def predict(image_path, model):
    img1 = load_img(image_path, target_size=(224, 224))
    img1 = img_to_array(img1) / 255.0
    img1 = np.expand_dims(img1, axis=0)

    car_name=image_path.split('/')[-2]

    (box_pred, label_pred) = model.predict(img1)
    (x1, y1, x2, y2) = box_pred[0]

    i = np.argmax(label_pred, axis=1)
    label = lb.classes_[i][0]

    image2 = cv2.imread(image_path)
    (h, w) = image2.shape[:2]

    # scale the predicted bounding box coordinates based on the image dimensions
    x1 = int(x1 * w)
    y1 = int(y1 * h)
    x2 = int(x2 * w)
    y2 = int(y2 * h)

    # draw the predicted bounding box and class label on the image
    y = y1 - 10 if y1 - 10 > 10 else y1 + 10

    cv2.rectangle(image2, (x1, y1), (x2, y2), (255, 165, 0), 2)

    cv2.putText(image2, label, (x1, y), cv2.FONT_HERSHEY_TRIPLEX, 0.65, (0, 255, 0), 2)

    top5_predictions = np.argsort(label_pred, axis=1)[:,-5:]

    names = []
    for i in range(1,6):
      names += [lb.classes_[top5_predictions[0][-i]]]

    probs = np.sort(label_pred)[:,-5:][0][::-1]

    # Plotting test image and predicted probabilites
    f, ax = plt.subplots(2,figsize = (6,10))

    ax[0].imshow(image2[...,::-1]),ax[0].axis('off'),ax[0].grid(None)
    ax[0].set_title(car_name)

    y_names = np.arange(len(names))
    ax[1].barh(y_names, probs/probs.sum(), color='orange')
    ax[1].set_yticks(y_names)
    ax[1].set_yticklabels(names)
    ax[1].invert_yaxis()
    ax[1].set_title('Top 5 Predictions')

    plt.show()
```

- Visualising image predictions:
  - 'Car Images/Test Images/Land Rover LR2 SUV 2012/01116.jpg'



  - 'Car Images/Test Images/Mercedes-Benz C-Class Sedan 2012/01127.jpg'



Python Flask and React Application

As model has been saved using keras save model and pickle, so here we have created a backend API platform which will take image as input and provide the bbox and label over image and label as text also.

Loading models and label encoder class.

```
# Load the bb_model, class_model and label_binarizer
bb_model = load_model('model_mobile_net_bb.h5', compile=False)
lb = pickle.load(open('lb.pkl', 'rb'))
class_model = load_model('mobile_net_model.h5', compile=False)
```

Currently there are 2 endpoints in FLASK:

- '/predict_label' POST endpoint:
  This is a POST endpoint which will take image as an input and will return predicted label for the image as JSON format.

```
@app.route('/predict_label', methods=['POST'])
def predict_label():
    path = os.path.join(app.config['UPLOAD_FOLDER'], 'uploaded_img.jpg')
    uploaded_img = request.files['file']
    uploaded_img.save(path)

    return jsonify(label=predict_label_class(path))
```

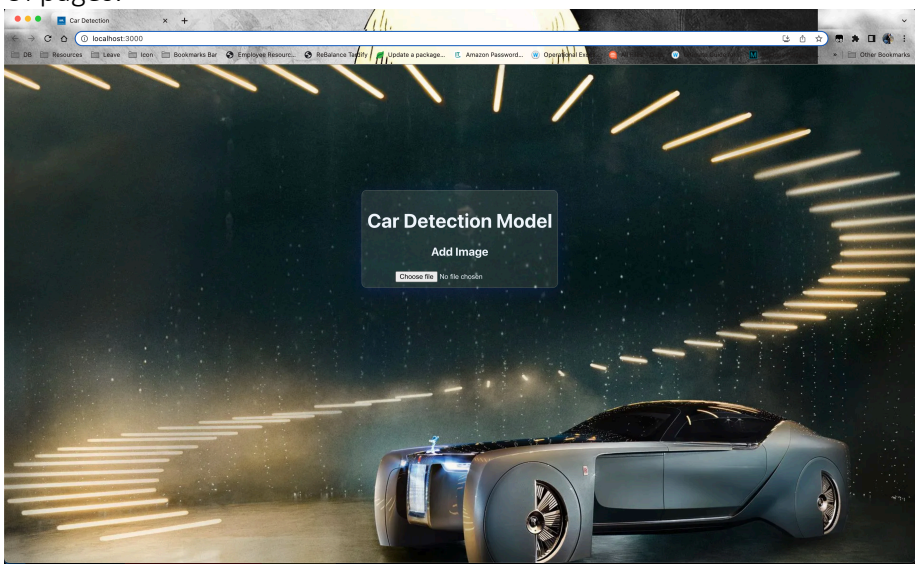
- '/predict_bbl' POST endpoint:
  This is a POST endpoint which will take image as an input and will return edited image which will be having bbox and label over image.

```
@app.route('/predict_bbl', methods=['POST'])
def predict_bbox_label():
    path = os.path.join(app.config['UPLOAD_FOLDER'], 'uploaded_img.jpg')
    uploaded_img = request.files['file']
    uploaded_img.save(path)

    # Make prediction using model loaded from disk as per the data.
    predict_bbox_class(path)

    return send_file(path, mimetype='image/jpg')
```

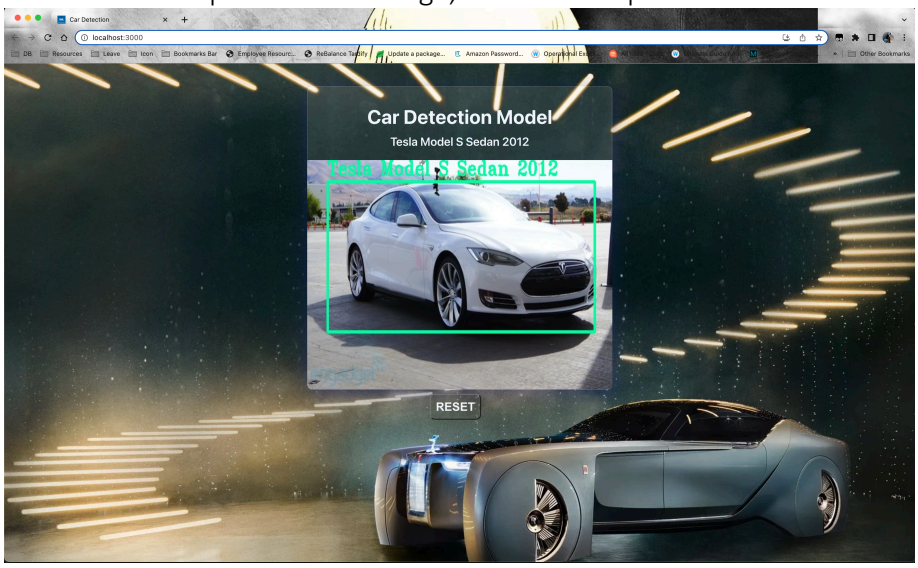
React application has been created which will be taking image from user and after executing and calling endpoints will be showing result to the user in UI.
Endpoint calls from react application:

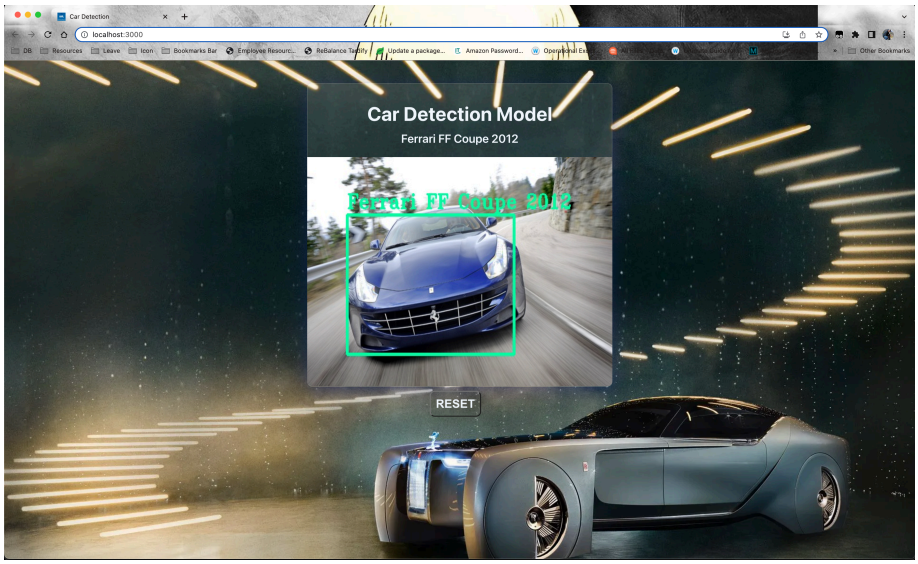
```
async function handleChange(e) {
  const data = new FormData();
  data.append("file", e.target.files[0]);

  await fetch("http://localhost:5000/predict_label", {
    method: "POST",
    body: data,
  }).then((response) => {
    response.json().then((text) => {
      setLabel(text.label);
    });
  });

  await fetch("http://localhost:5000/predict_bbl", {
    method: "POST",
    body: data,
  }).then((response) => {
    response.blob().then((imageBlob) => {
      setFile(URL.createObjectURL(imageBlob));
    });
  });
}
```

UI pages:



Once user has uploaded the image, it will shows predicted details in image.





Comparison to benchmark

In this model processing we have 2 models which have provided good accuracy but can be much better after some iterations and all.

- Model-1 with transfer learning
  In this model we got good training accuracy and testing accuracy as compare to other simple models. So after some iteration have increased test accuracy from 59% to 79%, which can still be increased by tuning some params. But this model already took 11h4m6s to get trained.
- Model-2 with Faster RCNN with transfer learning
  In this model we also got good training accuracy and testing accuracy for both boundary box and class labels. After many iteration testing accuracy has been improved drastically from 20-25% to 69%, which still can be increased by tuning some params.

So after few adjustment have stopped model training as its taking much ample time to get executed.
Still we can make improvements with using higher GPU/TPU which can train model much faster where we can adjust, learning_rate, epochs, batch_size, layers and many more checks.

Closing Reflections

In this process of car identification model using Computer Vision has provided many insights, learning, reporting and point to think and where to start.
While training model there were many limitation occurred because of which model accuracy was less for test dataset. In term of limitations, the main concern was the resource for training the model was not that much feasible. As we already know training model takes much ample time and because of which it was bit tedious task to make improvement via tuning some parameters or training with different pre-trained models.

In future via this dataset we can have many different tasks like, identifying vehicle number so that it can be used for some informative purpose, or checking cars via camera is the number plate is GOVT. approved or not and many other test cases can be achieved from this.

As we know training CNN models needs high GPUs/TPUs, higher dataset just example single image needs to be rotated, use different angles, different background images, several different lighting conditions and more.

So in future while making model training we need to make a good set of trainset, best resource capacity like Sagemaker, GoogleColab paid GPUs and other cloud providers. Once we are good with resources we can fine tune our models hyperparameter easily with changing different type of criteria of the specified used model.