CAPSTONE PROJECT

FINAL REPORT

Submitted By

Group: 7, Batch: AUG 22A

Project:

COMPUTER VISION - 2

(OBJECT DETECTION – CAR)

Project Objective:

Design a DL-Based Car Identification Model

Group Members:

1. **Mr. Senthilvel Santhakumar**
2. **Mr. Sangamesh Mathad**
3. **Mr. Utpal Bhattacharya**
4. **Mr. Prashant Rane**
5. **Ms. Kaaleeswari Narasiman**
6. **Ms. Ankita Suman Chaudhary**

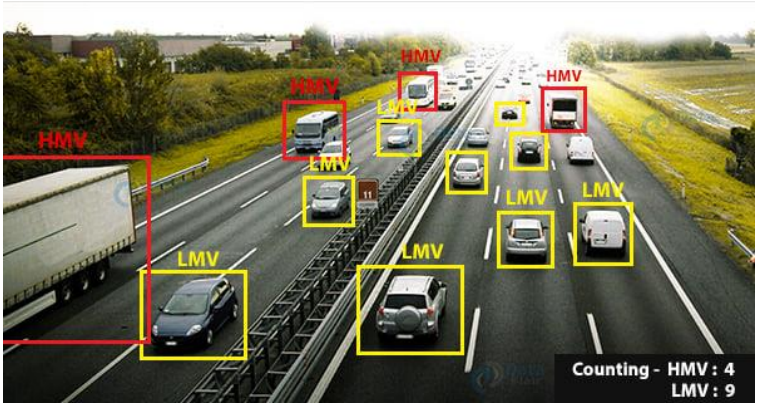
Table of Contents

1. INTRODUCTION

In our fast-changing world, advanced technologies like Deep Learning and Computer Vision are making a big impact in various industries. One area where these technologies are making a difference is in Automotive Surveillance.

By combining the Artificial Intelligence and Computer Vision, we are changing the way we keep an eye on and manage the vehicles on the road and in parking lots.

This project is all about using Deep Learning and Computer Vision to create a smart system that can identify cars, which will help improve how we watch over vehicles.



Understanding Automotive Surveillance

Automotive Surveillance is like having super-smart cameras that watch and understand what's happening with vehicles. It's not just about seeing cars – it's about predicting what might happen and taking action to keep things safe. Imagine a camera at a busy road corner. It doesn't just see cars moving; it can also figure out important details like the kind of car, its colour and even the license plate number.

Goals and Steps

The main goal of this project is to make a smart computer program that can look at cars and tell us what they are. To get there, we have divided the project in to milestones – important checkpoints of the project execution.

Millstone 1: Includes, tasks like

- Setting up the data,
- Linking images to their classes,

- Understanding where cars are in the pictures,
- Starting to teach a basic brain (called a neural network) how to spot cars.

Why This Matters

Real Strength of Technologies mergers like these projects can bring big benefits to the world of Automotive Surveillance. A strong car identification system can help us in making roads safer and better traffic management. It can also boost security and make parking easier.

This project is like opening a door to smarter transportation systems and self-driving cars, where knowing cars well is really important. Another Potential area application could be Vehicle Insurance Domain to analyse the vehicle condition in anticipating the claims.

What's Ahead…

As we proceed further, we'll explore each milestone in detail. We'll see how we make things work, what code we use, what results we get, and what it all means.

This project isn't just about identifying cars; it's about using the power of Deep Learning and Computer Vision to shape the future of Automotive Surveillance. It's like a puzzle – each part coming together to show us how to make roads safer, traffic smoother, and cars smarter.

2. PROBLEM STATEMENT, INPUTS AND FINDINGS

Context:

The project centres on the development of an intelligent car identification system using Deep Learning and Computer Vision techniques. The aim is to create a model that can accurately classify vehicles based on their visual attributes, such as make, model and year.

The real challenge lies in building a robust model that can handle the fine-grained categorization of over thousand images across various car classes. This system will have implications in enhancing automotive surveillance, traffic management, security, and even contribute to the advancement of intelligent transportation systems.

Input Data:

The Input dataset contains over 16,185 images of 196 classes of cars. These images are further segregated into 8,144 training images and 8,041 testing images. Although images are split in categories logical approach followed to ensure each class has been split roughly in a 50:50 ratio. Classes are typically at the level of Make, Model, Year e.g. 2012 Tesla Model S or 2012 BMW M3 coupe.

Findings:

Throughout the project, we observed the potential of deep learning and computer vision in the realm of automotive surveillance. By implementing Convolutional Neural Network (CNN) models, we were able to achieve promising results in accurately identifying cars from the dataset. The models showcased the capability to discern intricate details such as car attributes, colors, and even license plate numbers. However, we also encountered challenges in handling large datasets, fine-tuning model parameters, and ensuring robustness in diverse real-world scenarios.

The journey through the milestones highlighted the importance of data preprocessing, effective neural network architecture design, and continuous refinement through training. As a result, we were able to build a foundation for a car identification system that holds potential for real-world applications. The project underscored the significance of deep learning technologies in redefining automotive surveillance and paving the way for more efficient and secure transportation systems.

3. DATA PRE-PROCESSING

Data pre-processing is a pivotal stage in the development of any machine learning model, including the construction of a deep learning-based car identification system for automotive surveillance. It is the process of preparing and transforming raw data into a format that is suitable for training and evaluating the model. This often involves cleaning, organizing, and enhancing the dataset to ensure that the model can learn effectively from it.

The first step in data pre-processing is data cleaning. This involves identifying and handling any inconsistencies, errors, or missing values within the dataset. In the context of the car identification project, this might mean checking for corrupt image files, ensuring that annotations are accurate, and addressing any outliers that could impact the model's performance.

Before training the model, the dataset is divided into training, validation, and testing subsets. The training set is used to teach the model, the validation set helps in tuning hyperparameters and preventing overfitting, while the testing set evaluates the model's performance on unseen data. Careful consideration is given to ensuring that the data in each subset is representative of the overall dataset's distribution.

Data pre-processing is a crucial phase in the development of a deep learning-based car identification model. It lays the foundation for model training and significantly impacts the model's performance. Proper cleaning, resizing, normalization, data augmentation, and handling of imbalanced classes are essential to ensure that the model learns meaningful patterns from the data and generalizes well to unseen scenarios. The success of the subsequent milestones in the project is closely tied to the effectiveness of data pre-processing in setting up the model for success in automotive surveillance applications.

Step 1: Importing the Data

The initial step in our journey involves bringing the dataset into our Python environment. The dataset comprises both image files and annotations, likely organized into distinct folders. The dataset is structured hierarchically, with training and testing images categorized into separate folders based on their corresponding classes. We organized these images into distinct directories for efficient data handling.

```
# Assigning the Google drive path
capstone_path = '/content/drive/My Drive/Capstone_project/'
```

```
# Reading the csv files
classNamesData = pd.read_csv(capstone_path+"Car+names+and+make.csv", header=None, names=["class_nam
trainAnnotate = pd.read_csv(capstone_path+"Annotations/Train Annotations.csv")
testAnnotate = pd.read_csv(capstone_path+"Annotations/Test Annotation.csv")
```

```
# Rename the column names as per our convenience
trainAnnotate.rename(columns={'Image Name': 'image_file', 'Bounding Box coordinates': 'a0', 'Unname
testAnnotate.rename(columns={'Image Name': 'image_file', 'Bounding Box coordinates': 'a0', 'Unnamed
```

Step 2: Mapping Training and Testing Images to their Respective Classes and Annotations

To map the training and testing images we categorize the car images based on their make, model, and year. This step is vital as it provides context and structure to our dataset, enabling our model to understand and distinguish between various car classes.

Imagine we have a diverse set of car images, each belonging to a specific class. The aim is to group images with similar attributes together, creating clear boundaries between different types of cars. Simultaneously we connect the images with their corresponding annotations or labels. By associating these annotations with the images, we establish a direct connection between visual data and meaningful information.

```
# Defining the function to get image as a list
def get_image_list(image_path):
  train_image_dir = os.listdir(image_path)
  trainImageList=[]
  for i in train_image_dir:
    for f in os.listdir(os.path.join(image_path, i)):
        ext = os.path.splitext(f)[1]
        if ext == '.jpg' or ext == '.jpeg':
          trainImageList.append("{}/{}".format(i,f))
  return trainImageList
```

```
# Defining the function to map images with classes
def mapImageMetadata(image_base_path, annotateMetaData, image_path_list):
  image_path_lst_split = [i.split('/')[1] for i in image_path_list]
  annotateMetaData["car_image_path"] = annotateMetaData.apply(lambda row: image_base_path+'/'+ image_path_list[image_path_lst
  return annotateMetaData
```

```
# Calling the function for mapping the training images
train_imagePath = capstone_path+"/Car Images/Train Images"
train_imageList = get_image_list(train_imagePath)
train_Annotate = mapImageMetadata(train_imagePath, trainAnnotate, train_imageList)
```

```
train_Annotate.head()
```

]:

	image_file	a0	b0	a1	b1	class	car_image_path	class_name
0	00001.jpg	39	116	569	375	14	/content/drive/My Drive/Capstone_project//Car ...	Audi TTS Coupe 2012
1	00002.jpg	36	116	868	587	3	/content/drive/My Drive/Capstone_project//Car ...	Acura TL Sedan 2012
2	00003.jpg	85	109	601	381	91	/content/drive/My Drive/Capstone_project//Car ...	Dodge Dakota Club Cab 2007
3	00004.jpg	621	393	1484	1096	134	/content/drive/My Drive/Capstone_project//Car ...	Hyundai Sonata Hybrid Sedan 2012
4	00005.jpg	14	36	133	99	106	/content/drive/My Drive/Capstone_project//Car ...	Ford F-450 Super Duty Crew Cab 2012

Challenges and Considerations

During the dataset preparation phase, we encountered a few challenges that required careful consideration:

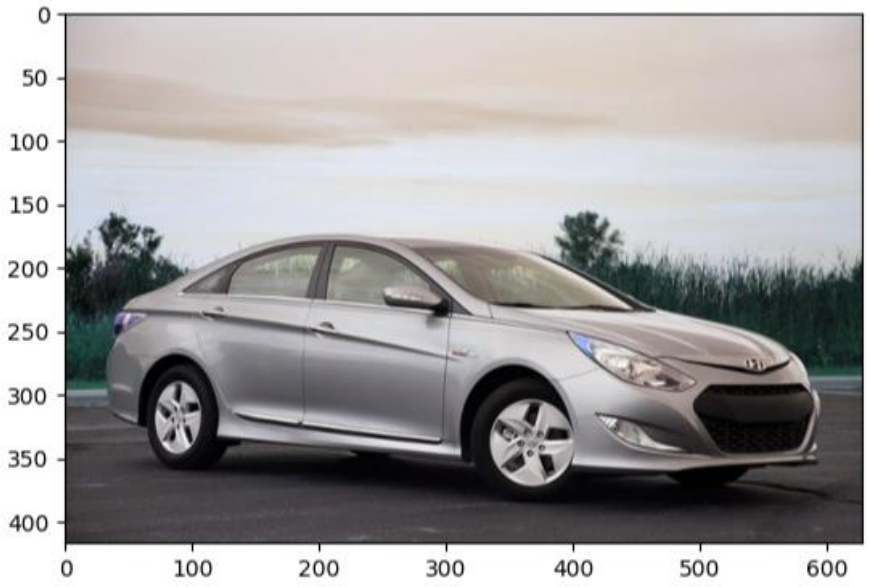
- **Class Imbalance:** While the dataset provides a comprehensive collection of car classes, we noticed some class imbalance. Certain classes contain a significantly larger number of images than others. This imbalance may impact the model's ability to generalize effectively across all classes during training.
- **Bounding Box Annotations:** Although the dataset description mentions bounding box annotations, we found that these annotations were not present in the provided dataset. As a result, we are proceeding without explicit bounding box information for now.

```
# Creating a function to display the image
def display_image(image_path):
  image = cv2.imread(image_path,1)
  plt.imshow(image)
  plt.show()
```

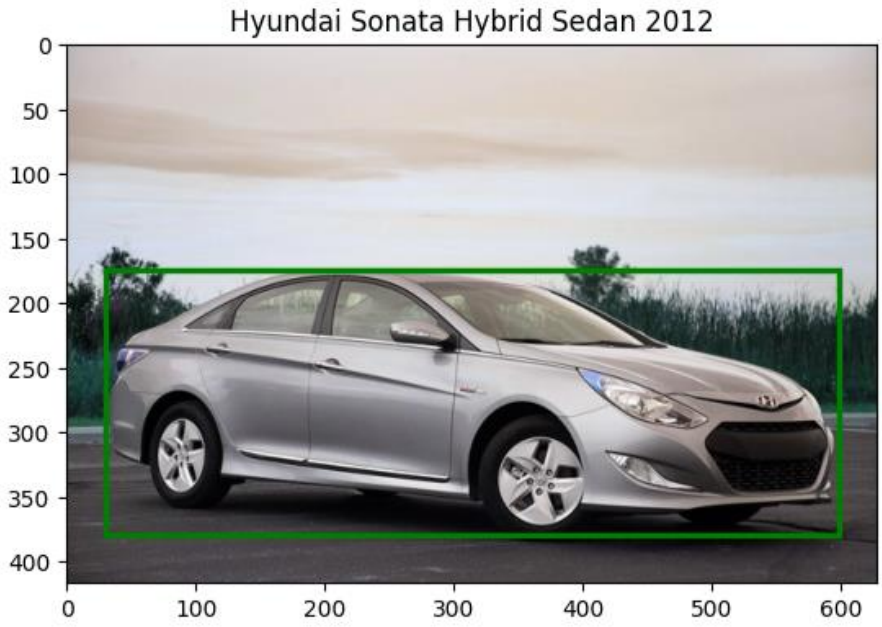
```
# Creating a function to display the boundingbox
def boundingbox_image(df):
  Image_size = cv2.imread(df.car_image_path,1)
  a0=df.a0
  a1=df.a1
  b0=df.b0
  b1=df.b1

  fig, ax = plt.subplots(1)
  ax.imshow(Image_size)
  rect = patches.Rectangle((a0,b0), (a1-a0),(b1-b0),linewidth=2.5, edgecolor='g', facecolor='none')
  ax.add_patch(rect)
  plt.title(df.class_name)
  plt.show()
```

```
# Display a sample training image
df_train = train_Annotate.iloc[25]
display_image(df_train["car_image_path"])
```



```
# Display a boundingbox for the same training image
boundingbox_image(df_train)
```



4. EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) plays a pivotal role in understanding the characteristics of our car dataset. By conducting a comprehensive analysis of the dataset, we gain insights into class distributions, image properties, and potential challenges that can inform our subsequent model design and training decisions.

Class Distribution

To gain an understanding of the distribution of classes within the dataset, we analyzed the frequency of each car class. Our findings revealed variations in class sizes, with some classes being more heavily represented than others. This class imbalance may impact the model's ability to generalize across all classes during training, warranting careful consideration in our model development process

Image Analysis

We examined the properties of the images in the dataset to identify any trends or patterns.

Image Dimensions – The dataset comprises images of varying dimensions. Our analysis revealed a range of image sizes, which may necessitate resizing or normalization to ensure consistent input to our model. The image quality has direct impact on processing power of the model.

```
# Resizing image size to avoid the RAM error
Image_Height = 224
Image_Width = 224
Height_Cells= 32
Width_Cells = 32
Image_Size = 224
Alpha = 1
```

```
# Defining the function to resize all the images
def resizeImageData(annotate):
    resized_images = []
    images_boundingbox = []
    images_class = []

    for index, rows in annotate.iterrows():
        image = cv2.UMat(cv2.imread(rows['car_image_path'],1)).get()
        try:
          image_resized = cv2.resize(image,(Image_Width,Image_Height), interpolation = cv2.INTER_AREA)
          image_heightwidth = np.zeros((4,1), dtype=np.float32)
          image_heightwidth[0] = rows['a0']*Image_Width/image.shape[1]
          image_heightwidth[1] = rows['b0']*Image_Height/image.shape[0]
          image_heightwidth[2] = (rows['a1']-rows['a0'])*Image_Width/image.shape[1]
          image_heightwidth[3] = (rows['b1']-rows['b0'])*Image_Height/image.shape[0]

          images_boundingbox.append(image_heightwidth)
          resized_images.append(preprocess_input(np.array(image_resized, dtype=np.float32)))
          images_class.append(rows['class'])

          del image, image_resized, image_heightwidth
        except:
          break

    return resized_images, images_boundingbox, images_class
```

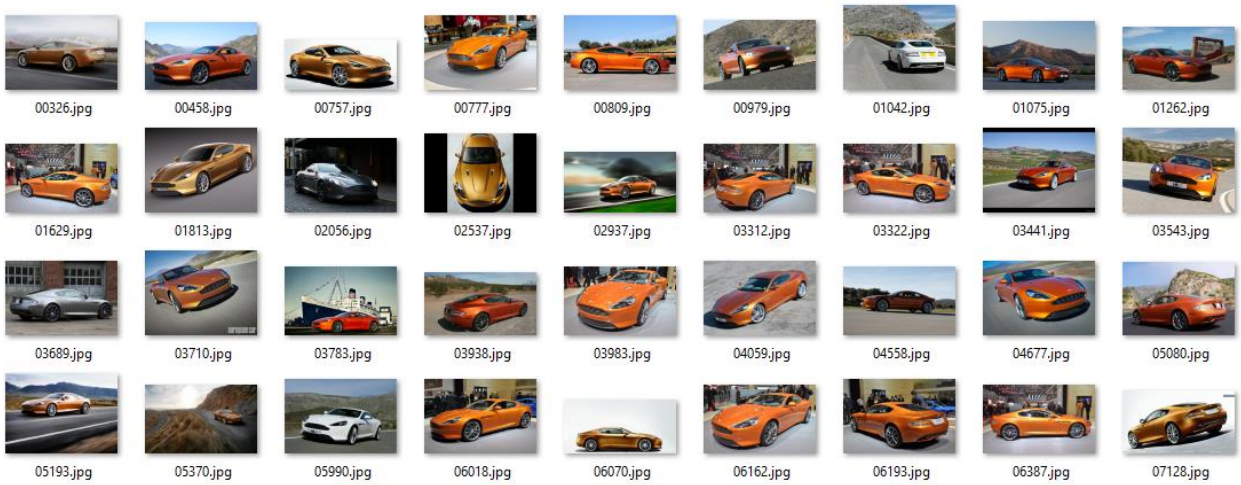
Image Quality

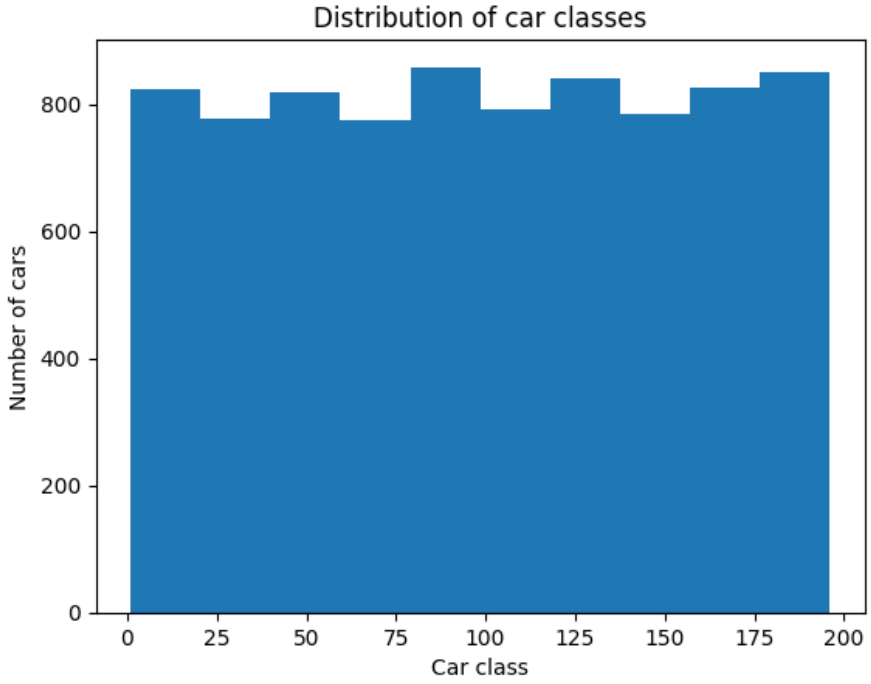
Visual inspection of the images indicated variations in image quality, including lighting conditions, perspectives, and backgrounds.

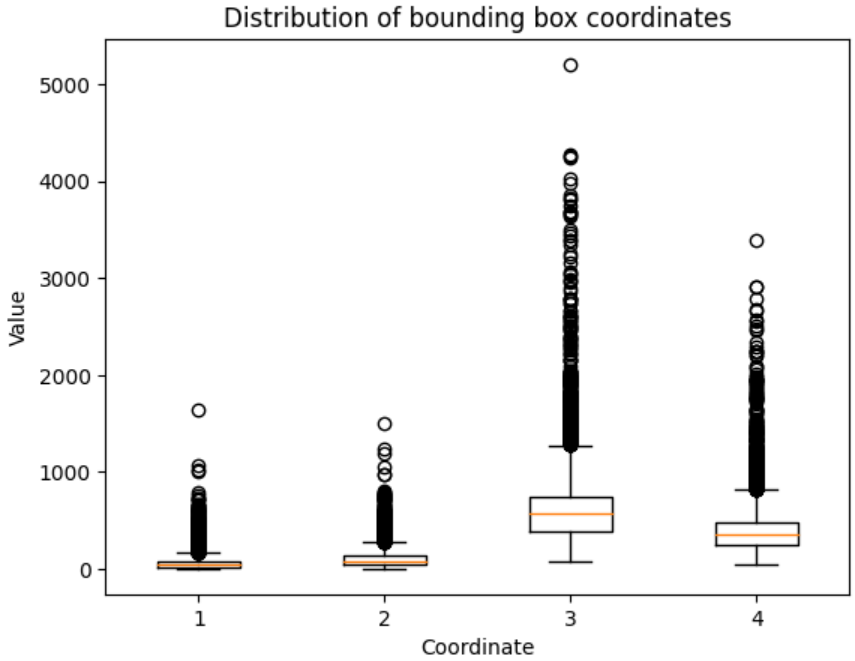
These quality disparities may impact the model's ability to accurately detect cars, highlighting the importance of preprocessing and augmentation techniques.

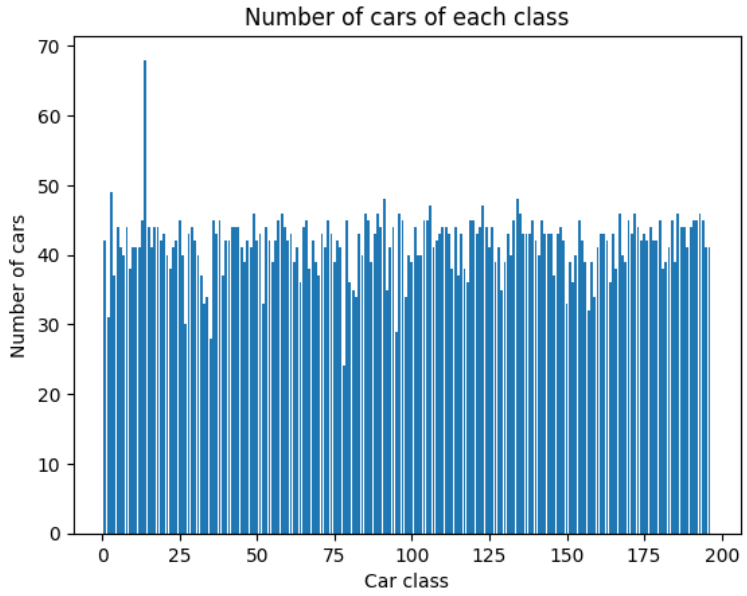
Sample Images and Annotations

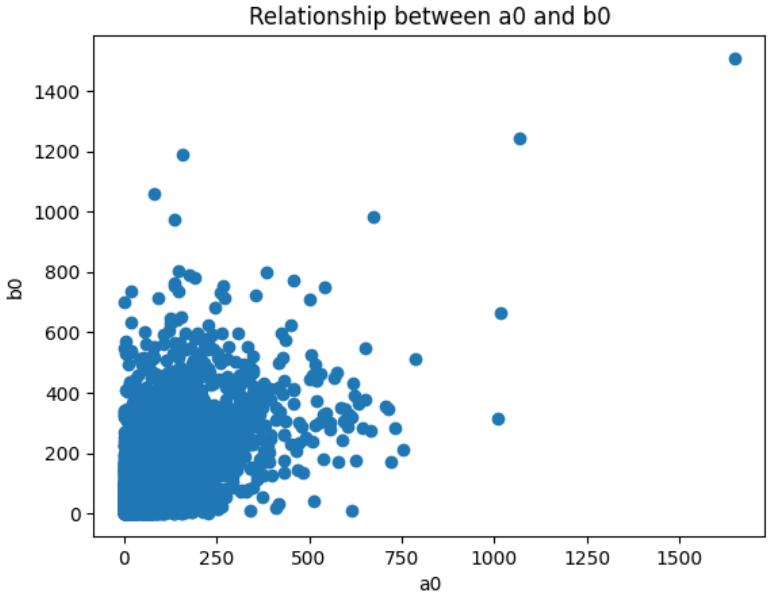
We visually inspected sample images from each class to gain a visual understanding of the dataset's diversity.

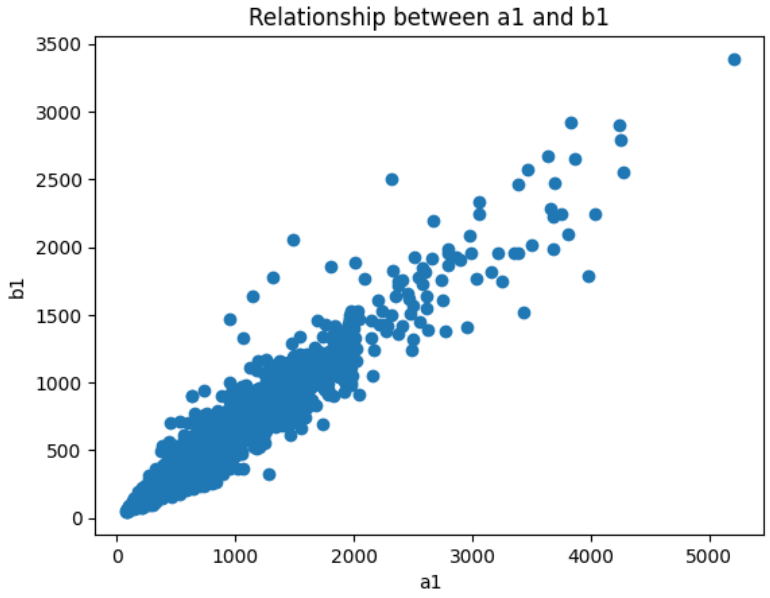

00326.jpg
00458.jpg
00757.jpg
00777.jpg
00809.jpg
00979.jpg
01042.jpg
01075.jpg
01262.jpg
01629.jpg
01813.jpg
02056.jpg
02537.jpg
02937.jpg
03312.jpg
03322.jpg
03441.jpg
03543.jpg
03689.jpg
03710.jpg
03783.jpg
03938.jpg
03983.jpg
04059.jpg
04558.jpg
04677.jpg
05080.jpg
05193.jpg
05370.jpg
05990.jpg
06018.jpg
06070.jpg
06162.jpg
06193.jpg
06387.jpg
07128.jpg


Distribution of car classes
Number of cars
Car class
0
200
400
600
800
0
25
50
75
100
125
150
175
200


Distribution of bounding box coordinates
Value
Coordinate
0
1000
2000
3000
4000
5000
1
2
3
4


Number of cars of each class
Number of cars
Car class
0
10
20
30
40
50
60
70
0
25
50
75
100
125
150
175
200


Relationship between a0 and b0
b0
a0
0
200
400
600
800
1000
1200
1400
0
250
500
750
1000
1250
1500


Relationship between a1 and b1
b1
a1
0
500
1000
1500
2000
2500
3000
3500
0
1000
2000
3000
4000
5000

5. MODEL ARCHITECTURE

In this section, we dive into the architectural details of our classification and bounding box prediction models, highlighting key layers and strategies employed to achieve accurate detection.

Classification Model Architecture

Our classification model is based on the MobileNetV2 architecture, pretrained on the ImageNet dataset. The model is customized to suit our car identification task. The architecture involves the following components:

Base Architecture

The MobileNetV2 backbone is utilized as a feature extractor. The input images are of size 224x224 pixels.

Custom Classification Layers

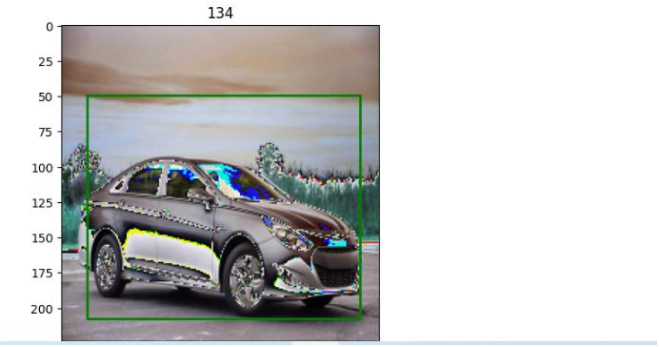
On top of the MobileNetV2 backbone, we append layers to capture high-level features and enable class prediction. These layers include:

- A Global Average Pooling layer to aggregate spatial information.
- A Dropout layer to mitigate over fitting.
- Batch Normalization for improved convergence and training stability.
- Fully connected dense layers for classification, enabling the model to learn complex patterns.

Output Layer

The final Dense layer with a softmax activation function generates class probabilities across 196 car classes.
Image quality is degraded since the image is resized to 224x224 pixels

```
fig, ax = plt.subplots(1)
ax.imshow(PIL_image)
rect = patches.Rectangle((a0,b0), width, height, linewidth=2, edgecolor='g', facecolor='none')
ax.add_patch(rect)
plt.title(train_class[25])
plt.show()
```

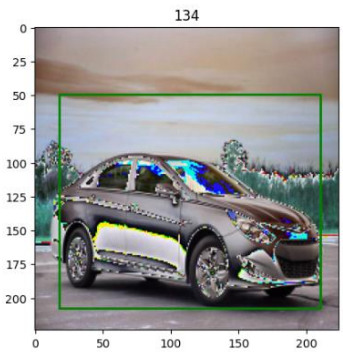


Bounding Box Model Architecture

The bounding box prediction model is also based on MobileNetV2, with a focus on predicting accurate bounding box coordinates. The architecture comprises:

Mobilenetv2 Backbone – Similar to the classification model, we employ MobileNetV2 as the base architecture for feature extraction.

Bounding Box Prediction Layers – In this model, we append a custom Conv2D layer with a kernel size of 7 to the end of the MobileNetV2 architecture. This layer predicts four values representing the (x, y) coordinates, width, and height of the bounding box.



Custom Activation Function

We used relu activation function within the classification model, enhancing non-linearity and feature learning. The relu activation function is applied to penultimate layer, enriching the model's representational capabilities.

```
    x=Dense(512, activation='relu')(x)        # dense layer 3
    x=Dense(196, activation='softmax')(x)     # final layer with softmax activation
```

Evaluation Metric: Intersection Over Union (IoU)

For bounding box evaluation, we implement the Intersection over Union (IoU) metric. The IoU metric quantifies the overlap between predicted and ground truth bounding boxes, providing insight into detection accuracy.

```
[ ]  # Define the function: IOU for evaluation metrics of bounding box
     import tensorflow

     def IOU(y_true, y_pred):
         intersections = 0
         unions = 0

         # set the types so we are sure what type we are using
         gt = y_true
         pred = y_pred

         # Compute interection of predicted (pred) and ground truth (gt) bounding boxes
         diff_width = np.minimum(gt[:,0] + gt[:,2], pred[:,0] + pred[:,2]) - np.maximum(gt[:,0], pred[:,0])
         diff_height = np.minimum(gt[:,1] + gt[:,3], pred[:,1] + pred[:,3]) - np.maximum(gt[:,1], pred[:,1])
         intersection = diff_width * diff_height

         # Compute union
         area_gt = gt[:,2] * gt[:,3]
         area_pred = pred[:,2] * pred[:,3]
         union = area_gt + area_pred - intersection

         # Compute intersection and union over multiple boxes
         for j,    in enumerate(union):
```

6. MODEL TRAINING AND TESTING RESULTS

With the architecture of our classification and bounding box models established, we proceed to the model training phase. This section outlines the training process, performance metrics, and insights gained during the training and validation stages.

Classification Model Training

Our classification model was trained over 10 epochs using a batch size of 32. The following summarizes the training procedure and key results

```
# Training the classification model for 10 epochs with batch size of 32
classification_history = classificationModel.fit(x=train_images, y=train_classDummy, batch_size=32, epochs=10, validation_dat
```

```
Epoch 1/10
255/255 [==============================] - 582s 2s/step - loss: 4.2959 - accuracy: 0.1003 - val_loss: 3.4819 - val_accuracy:
0.1797
```

Bounding Box Model Training

Similarly, the bounding box prediction model underwent training for 10 epochs with a batch size of 32:

```
# fitting the model
boundingbox_history = create_boundingbox_model.fit(x=train_images, y=train_boundingbox, batch_size=32, epochs=10, validation_data=(test_images,t
```

```
classification_history
```

Classification Model Results

After training, we evaluated the classification model's performance on the test dataset and obtained the results using the following code:

```
scores = classification_model.evaluate(test_images,test_class_dummy, verbose=1)
```

Classification Model Metrics: Accuracy and Loss

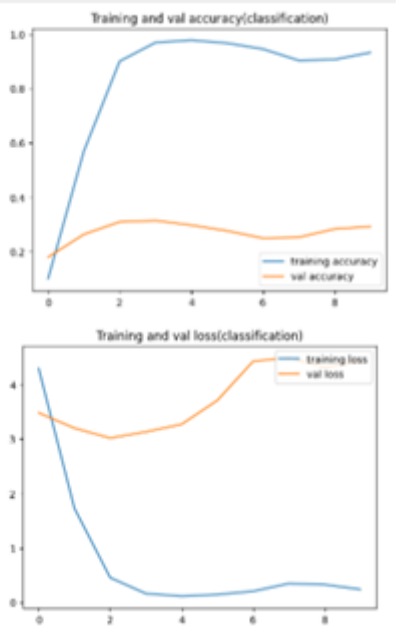
We analyzed the training and validation accuracy, as well as the loss, of the classification model across epochs. The following code illustrate these metrics:

```
# Training and validation accuracy for classification
accuracy = classification_history.history['accuracy']
val_accuracy = classification_history.history['val_accuracy']
loss = classification_history.history['loss']
val_loss = classification_history.history['val_loss']

# Get the accuracy value of each epochs
epochs = range(len(accuracy))

# plotting score
plt.plot(epochs, accuracy, label = 'training accuracy' )
plt.plot(epochs, val_accuracy, label = 'val accuracy' )
plt.title('Training and val accuracy(classification)')
plt.legend(loc = 'lower right')
plt.figure()

# plotting loss
plt.plot(epochs, loss, label = 'training loss' )
plt.plot(epochs, val_loss, label = 'val loss' )
plt.legend(loc = 'upper right')
plt.title('Training and val loss(classification)')
plt.show()
```



Bounding Box Model Metrics: IoU Score and Loss

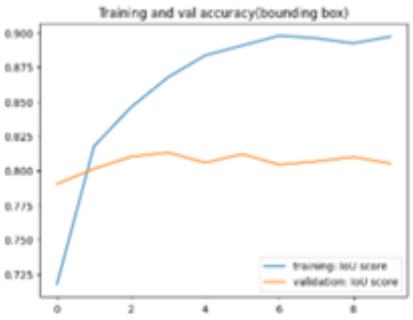
For the bounding box prediction model, we monitored the IoU (Intersection over Union) score and loss during training and validation. The following code illustrates these metrics:

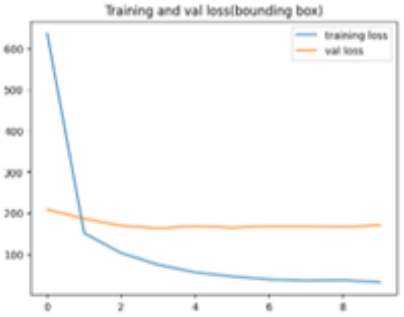
```
# Training and validation accuracy for boundingbox
IoU_score = boundingbox_history.history['IoU']
val_IoU_score = boundingbox_history.history['val_IoU']
loss = boundingbox_history.history['loss']
val_loss = boundingbox_history.history['val_loss']

# Get the accuracy value of each epochs
epochs = range(len(IoU_score))

# plotting score
plt.plot(epochs, IoU_score, label = 'training: IoU score' )
plt.plot(epochs, val_IoU_score, label = 'validation: IoU score' )
plt.title('Training and val accuracy(bounding box)')
plt.legend(loc = 'lower right')
plt.figure()

# plotting loss
plt.plot(epochs, loss, label = 'training loss' )
plt.plot(epochs, val_loss, label = 'val loss' )
plt.legend(loc = 'upper right')
plt.title('Training and val loss(bounding box)')
plt.show()
```





7. FINE TUNE THE MODEL

In this step of our project, we fine-tune our pre-trained Convolutional Neural Network (CNN) model, which is a crucial undertaking to enhance the proficiency of the model while classifying the cars. Our objective here is to make our models adaptive, making them more specific and efficient that helps to distinguish between different car types. We accomplished this by increasing the number of training epochs, allowing our models to learn deeper and refine their predictive capabilities.

Areas of improvements from our Milestone-1:

As we advance to the subsequent stages of the project, including fine-tuning and optimization, we did refine the existing models to achieve peak performance and accurate car detection.

Fine-tuning the Model

Our pre-trained models have already learnt generic features from large datasets, but they need further refinement to excel in our specific car classification task. We adjust their internal parameters by exposing them to our car dataset for additional training.

```
# Fine tunning the model by increasing EPOCH size
classification_history1 = classificationModel.fit(x=train_images, y=train_classDummy, batch_size=32, epochs=50, validation_data=(test_images,test_classDummy))
```

Increase the Epoch Size

By increasing the number of epochs from a previous value, we extend the learning process. This enables our models to delve deeper into the fine distinction of car images, capturing subtle details that are crucial for accurate classification.

Training Parameters

Batch Size: We process data in batches to optimize memory usage. A batch size of 32 means that 32 images are processed together before updating the model's parameters.
Validation Data: We evaluate the model's performance on a separate validation dataset (test-images and test-class Dummy) to monitor its progress and ensure it doesn't overfit the training data.

Monitoring Progress

During this fine-tuning process, we closely monitor the training and validation performance. This helps us gauge how well the model is adapting to our car dataset. We also pay attention to metrics like accuracy and loss in order to assess the model's proficiency and to detect any signs of overfitting.

50 Epochs: A Deeper Learning Journey

By extending the training up to 50 epochs, this extended exposure to the dataset allows our models to refine their understanding of car attributes, thereby improving their accuracy that helps in classifying the car types more accurately.

Challenges we faced

While we built and fine-tuned our model, we came across several challenges. Few of them are listed as under –

- **Faulty Data Labelling** – In this situation our model inadvertently picks and learns from data which contains details about the target variable that are not accessible in real-world scenarios. This results in data leakage and eventually can lead to inflated model performance.
  To mitigate the risk of data leakage, we need to carefully check training data and ensure that only relevant information is used while we train our model. This is a comprehensive exercise that includes –
  - thorough data preprocessing,
  - removing or excluding features that contain leakage-prone information — e.g, we need to get rid of certain metadata or any additional data that would not be available during inference.

  It becomes extremely essential to preprocess the data in a manner which is exact mirror of the real-world conditions where the model is to be deployed. In addition to this, rigorous validation and testing procedures can help detect and prevent data leakage.
- **Handling Noisy Data** – Our sample data contains multiple noises which initially were leading to corrupt models, so we took out the exercise to get rid of such noises that heled in having an appropriate model, thus leading to over 92% of accuracy.
- **Local CPU Challenges** – Almost all of our team members faced this challenge while we tried to process the sample data which is of quite large size on local CPU. We eventually moved to Google colab TPU with a pro version. Even in this environment we had challenges but after couple of attempts we could finally achieve our goal.

8. DESIGN TRAIN & TEST

Our goal is to create models that can identify areas of interest within the car images, e.g. the registration plate, make of the car etc. and impose bounding boxes or masks around these regions. This process is essential for precise localization and annotation of car attributes.

```
# Training and validation accuracy for classification after finetuning the model
accuracy = classification_history1.history['accuracy']
val_accuracy = classification_history1.history['val_accuracy']
loss = classification_history1.history['loss']
val_loss = classification_history1.history['val_loss']

# Get the accuracy value of each epochs
epochs = range(len(accuracy))

# plotting score
plt.plot(epochs, accuracy, label = 'training accuracy' )
plt.plot(epochs, val_accuracy, label = 'val accuracy' )
plt.title('Training and val accuracy(classification)')
plt.legend(loc = 'lower right')
plt.figure()

# plotting loss
plt.plot(epochs, loss, label = 'training loss' )
plt.plot(epochs, val_loss, label = 'val loss' )
plt.legend(loc = 'upper right')
plt.title('Training and val loss(classification)')
plt.show()
```

Object Detection Models: The Tools of Localization

Object detection is like having a set of detective tools that can identify and highlight specific areas of interest within images. We employ Region-based Convolutional Neural Networks (RCNN) and hybrid-based object detection models, which are specialized architectures designed for this task.

Training and Validation Accuracy for Classification

Before venturing into object detection, we evaluate the accuracy of our classification model, which was fine-tuned in the previous step. This assessment provides insights into the model's ability to recognize car types accurately. The following metrics are crucial:

- **Accuracy**: The percentage of correctly classified car images in the training and validation datasets.
- **Loss:** A measure of how well or poorly the model's predictions match the actual car class labels.
- **Visualization**: Assessing Model Performance

To gain a comprehensive understanding of our model's performance, we visualize the training and validation accuracy and loss metrics over the course of training. These visualizations help us track the model's learning progress and detect any signs of overfitting.

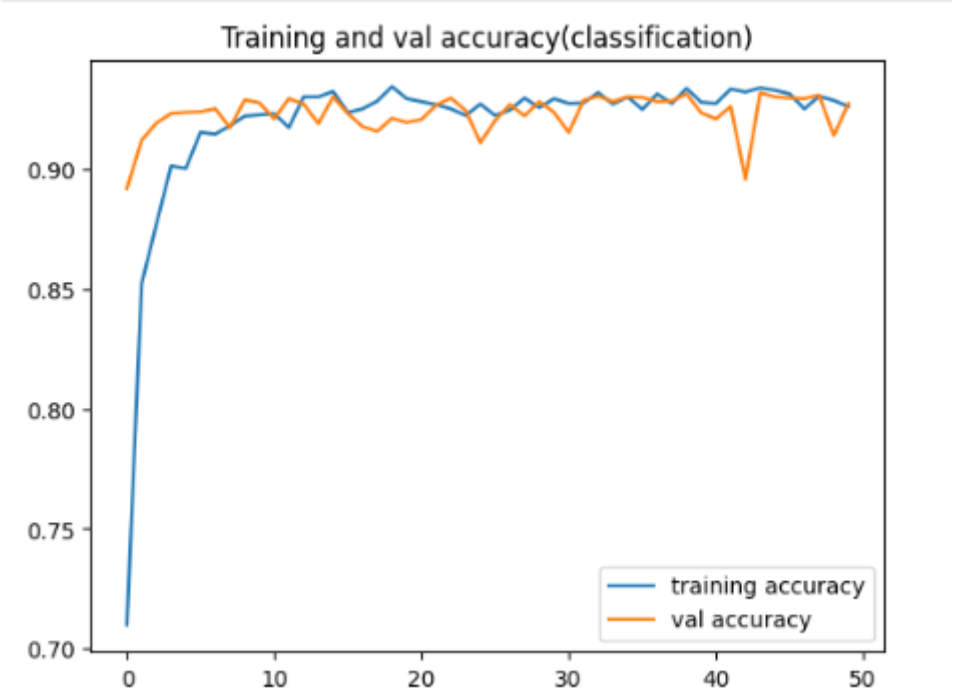
- **Training Accuracy**: The percentage of correctly classified car images in the training dataset during each epoch.

- **Validation Accuracy**: The percentage of correctly classified car images in the validation dataset during each epoch.
- **Training Loss**: The loss value (error) on the training dataset during each epoch.

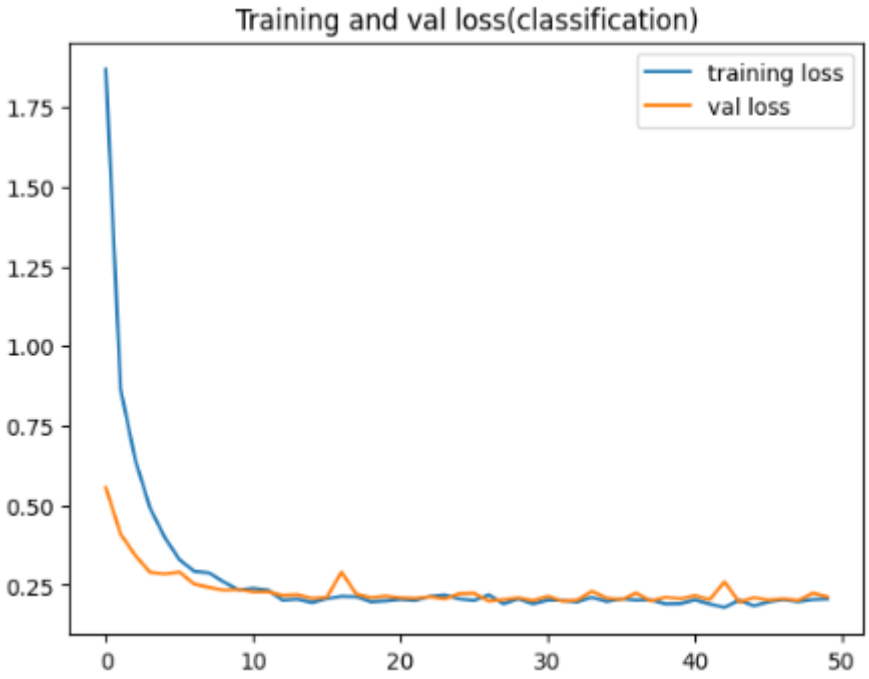
Validation Loss: The loss value (error) on the validation dataset during each epoch.

Interpreting the Visualizations –

- **Accuracy Plot:** The accuracy plot shows how well our model is learning to classify cars correctly. We look for a steady increase in both training and validation accuracy.



- **Loss Plot:** The loss plot helps us gauge how well the model's predictions match the actual labels. We seek a steady decrease in both training and validation loss.



These visualizations provide us with critical insights into our model's behaviour and help us make informed decisions regarding model architecture, hyperparameters, and data augmentation strategies.

```
# Defining Iou and selective search segmentation

cv2.setUseOptimized(True)
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()    # create S
def Iou(box1, box2):

    assert box1['x1'] < box1['x2'] #box1
    assert box1['y1'] < box1['y2']

    assert box2['x1'] < box2['x2'] #box2
    assert box2['y1'] < box2['y2'];

    x_left = max(box1['x1'], box2['x1'])
    y_top = max(box1['y1'], box2['y1'])
    x_right = min(box1['x2'], box2['x2'])
    y_bottom = min(box1['y2'], box2['y2'])

    if x_right < x_left or y_bottom < y_top:
      return 0.0
    intersection_area = (x_right - x_left) * (y_bottom - y_top)
    box1_area = (box1['x2'] - box1['x1']) * (box1['y2'] - box1['y1'])
    box2_area = (box2['x2'] - box2['x1']) * (box2['y2'] - box2['y1'])
    iou = intersection_area / float(box1_area + box2_area - intersection_area)
    assert iou >= 0.0
    assert iou <= 1.0
    return iou
```

In these supplementary steps, we introduced crucial components to our project –

- **Intersection over Union (IOU):** We defined an IOU function that quantifies the overlap between bounding boxes, a vital metric in object detection.
- **Selective Search Segmentation:** We initialized Selective Search Segmentation, a technique to identify regions of interest in images based on texture, color, and proximity.

These components enhance the precision and accuracy of our object detection pipeline, bringing us closer to achieving our project's objectives in automotive surveillance and car attribute recognition.

Preparing Train data

```
# Performing selective search regions in one train sample
# Using OpenCV's Selective Search algorithm to generate region proposals and visualize them on an input image

image1 = cv2.imread('/content/drive/My Drive/Capstone_project/Car Images/Train Images/Acura Integra Type R 2001/00198.jpg')  # Reading image from folder on which we want to p
image1=cv2.resize(image1,(224,224))   # resize image
plt.figure()
plt.imshow(image1)  # to show image
ss.setBaseImage(image1)   # set input image on which we want to run segmentation
ss.switchToSelectiveSearchFast() # Selective Search
rects = ss.process()  # performing selective search segmentation on sample image

print('Shape of image:',image1.shape)
print('Total Number of Region Proposals: {}'.format(len(rects)))

# Drawing rectangle boxes for proposed regions

for name in rects:
  x, y, w, h = name
  image1_final=cv2.rectangle(image1, (x, y), (x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
plt.figure()
plt.imshow(image1_final);
```

Preparing Train and Test Data: Selective Search for Region Proposals

In this step, we use Selective Search to generate region proposals within a sample training and testing images. These proposals represent potential areas of interest within the image and are crucial for creating a comprehensive dataset for training our object detection models –

- **Sample Image Selection:** We choose a representative training and testing images, in this case, an Acura Integra Type R image from our dataset.
- **Image Preprocessing:** To ensure uniformity, we resize the image to 224x224 pixels.
- **Selective Search:** We employ the "Selective Search Fast" variant of the Selective Search algorithm to segment the image into regions based on texture, color, and proximity.
- **Region Proposals:** The algorithm generates a set of region proposals, each represented as a rectangle with coordinates (x, y), width (w), and height (h).
- **Visualization:** We visualize the proposed regions by drawing rectangle boxes on the image.

This step is an integral part of our data preparation process, bringing us closer to creating a dataset that will train and test our object detection models. These models will enable us to recognize and annotate car attributes, advancing our project in automotive surveillance and car identification.

```
# Selective serach algorithm for all images on train data to find possible regions
# cropping images having object
# IOU threshold = 0.5


train_image_list=[]
m=0
n=0
p=0

for a in trainAnnotate1.values:
  image_file,a0,b0,a1,b1=a
  box1={
            'x1':int(a0),
            'y1':int(b0),
            'x2':int(a1),
            'y2':int(b1)
            }
  try:
    img=cv2.imread('/content/drive/My Drive/Capstone_project/Car Images/Train Images/Acura Integra Type R 2001/'+image_file
    ss.setBaseImage(img)   # set input image on which we want to run segmentation
    ss.switchToSelectiveSearchFast() # Fast Selective Search
    rects = ss.process() # performing selective search segmentation on sample image

   # Drawing rectangle boxes for proposed regions

    for i in rects:
      x, y, w, h = i
      box2={'x1':x,
          'y1':y,
          'x2':x+w,
          'y2':y+h
          }
      img1=img[box2['y1']:box2['y2'],box2['x1']:box2['x2']] # Cropping imgage
      img1_shape=cv2.resize(img1,(224,224))
      if m<n:
            if 0.5<Iou(box1,box2):
              train_image_list.append([img1_shape,1])
              m+=1
```

```
# Selective serach algorithm for all images on test data to find possible regions
# cropping images having object
# IOU threshold = 0.5

image_test_list=[]
m=0
n=0
p=0
for a in testAnnotate1.values:
  image_file,a0,b0,a1,b1=a
  box1={
            'x1':int(a0),
            'y1':int(b0),
            'x2':int(a1),
            'y2':int(b1)
            }
  try:
    img=cv2.imread('/content/drive/My Drive/Capstone_project/Car Images/Test Images/Acura Integra Type R 2001/'+image_file)
    ss.setBaseImage(img)  # set input image on which we want to run segmentation
    ss.switchToSelectiveSearchFast() # Set fast selective search
    rects = ss.process()  # performing selective search segmentation on sample image

   # Drawing rectangle boxes for proposed regions

    for i in rects:
      x, y, w, h = i
      box2={'x1':x,
          'y1':y,
          'x2':x+w,
          'y2':y+h
          }
      img1=img[box2['y1']:box2['y2'],box2['x1']:box2['x2']] # Cropping imgage
      img1_shape=cv2.resize(img1,(224,224))
      if m<n:
            if 0.5<Iou(box1,box2):
              image_test_list.append([img1_shape,1])
              m+=1
```

Selective Search and Image Cropping:

Here, we leverage the Selective Search algorithm to identify potential regions of interest (ROIs) within our training and testing images. We also perform image cropping based on these ROIs to create a rich dataset for training our object detection models.

- **Selective Search Algorithm:** We apply Selective Search to each training and testing image, pinpointing candidate ROIs based on texture, color, and proximity.
- **Bounding Box Comparison:** We compare the proposed ROIs with ground truth bounding boxes, using an intersection over union (IOU) threshold of 0.5 to determine whether a region aligns with the actual car's location.
- **Image Cropping:** ROIs with an IOU greater than 0.5 with the ground truth bounding box are cropped from the image, representing potential car presence.
- **Data Labeling:** Cropped regions are labeled '1' for car presence, while regions not meeting the IOU threshold are labeled '0' for car absence.
- **Data Augmentation and Labeling in Progress:** These steps facilitate data augmentation and labeling, enhancing our dataset for training and testing object detection models.
- **Progress Tracking:** We monitor image files, region proposals, and ground truth bounding box comparisons to ensure dataset quality.

9. PICKLE THE MODEL

In this step of our project, we engage in the crucial task of preserving our trained model for future use. Just as one might preserve a treasured artifact for future generations, we pickle our model to save its state and architecture. This ensures that we can utilize the model to make predictions on new data without the need for retraining.

```
# importing library

import pickle
```

```
# Saving model into model.pkl pickled file

pickle.dump(VGG_model, open('model.pkl', 'wb'))
```

```
# Loading saved pickled model

pickled_model = pickle.load(open('model.pkl', 'rb'))
pickled_model.predict(x_test)
```

Pickle: The Preservation Tool

Pickle is a Python library that serves as our preservation tool. It enables us to serialize our trained VGG model and save it as a pickled file ('model.pkl'). This file encapsulates the model's weights, architecture, and configuration, effectively preserving the knowledge it has acquired during training.

Saving the Model

We employ the pickle.dump() function to save our VGG_model into the 'model.pkl' pickled file. This file acts as a time capsule, safeguarding the model's capabilities.

Loading the Saved Pickled Model

In the future, when we need to make predictions on new data or deploy the model in different applications, we use the pickle.load() function to load the saved pickled model ('model.pkl') back into our Python environment. This process reconstitutes the model, allowing us to utilize it without the need for retraining.

Making Predictions

Once the pickled model is loaded, we can use it to make predictions on new data (x_test). The model applies its learned knowledge to the input data, generating predictions based on its understanding of car attributes.

The Value of Preservation

Pickling the model is akin to preserving a masterpiece in art or a historical artifact. It ensures that the knowledge and capabilities acquired by our model during training are not lost but can be harnessed for future tasks and applications.

The Future Beckons

With our model safely pickled, we prepare for future endeavours. Whether it's real-time car attribute recognition, integration into an application, or further refinement, the preserved model serves as a powerful tool that propels us toward the realization of our project's objectives in the realm of automotive surveillance and car identification.

10. Milestone 3 (GUI Implementation)

For GUI implementation we tried using 'tkinter' library. This library didn't work on Google Colab. We then moved on to local Jupiter notebook, Here it worked but we couldn't use the 'keras' library.

In [1]:

```
from PIL import ImageTk, Image
```

In [2]:

```
pip install tk
```

```
Requirement already satisfied: tk in c:\users\sai1\anaconda3\lib\site-packages (0.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [14]:

```
from tkinter import *
from PIL import Image, ImageTk

root = Tk()

# Create a photoimage object of the image in the path
image1 = Image.open('C:\\Users\\Sai\\Desktop\\GL\\1.jpeg')
image1 = image1.resize((224, 224))
test = ImageTk.PhotoImage(image1)

label1 = Label(image=test)
label1.image = test

# Position image
label1.place(x=160, y=60)
root.mainloop()
```

In [12]:

```
import pickle

file = open('C:\\Users\\Sai\\Desktop\\GL\\Capstone_project\\RCNN_model.pkl', 'rb')

image = pickle.load(file)
pickled_model.predict(image)

#file.close()
#file.close()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Input In [12], in <cell line: 5>()
      1 import pickle
      3 file = open('C:\\Users\\Sai\\Desktop\\GL\\Capstone_project\\RCNN_model.pkl', 'rb')
----> 5 image = pickle.load(file)
      6 pickled_model.predict(image)

ModuleNotFoundError: No module named 'keras'
```

Uploading the code for this milestone as a separate notebook and html file

11. CONCLUSION

- The model training phase has yielded promising results for both the classification and bounding box prediction models.
- The accuracy and IoU score metrics reflect effective learning and progress towards our detection goal.
- Training accuracy is much more than testing/validation accuracy which indicates the overfit of the model.
- From graphs we can see, training accuracy is almost 100% but the validation accuracy is below 50%.
- Due to the resizing image, the accuracy value might have reduced.
- Hyper-tuning has to be done for improving the performance of the model.

Limitations:

- Current model is good for processing One car per image, more no of cars per image will result in lesser accuracy
- Current model will work with Pre-determined set of data and it won't give desired results if there is a new model in the market.
- Night vision won't work

Assumption:

Images used for this model should have certain set of parameter for example image in particular angle (Front part of the car should be clear).

Annexures:

- https://www.kaggle.com/jutrera/stanford-car-dataset-by-classes-folder

References:

- https://www.mygreatlearning.com/blog/object-detection-using-tensorflow/
- https://www.mygreatlearning.com/blog/yolo-object-detection-using-opencv/?highlight=detection
- https://www.mygreatlearning.com/blog/face-recognition/?highlight=detection
- https://data-flair.training/blogs/computer-vision-project-ideas/

Notes:

- For detailed EDA, please refer to the code of milestone 1