In [1]:
```
import os
import numpy as np
import cv2

from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adam
import tensorflow as tf

import random
import matplotlib.pyplot as plt
```

In [2]:
```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\dell\tfenv\lib\site-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in c:\users\dell\tfenv\lib\site-packages (from opencv-python) (2.1.3)
```

In [3]:
```
!pip install opencv-python-headless
```

```
Requirement already satisfied: opencv-python-headless in c:\users\dell\tfenv\lib\site-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in c:\users\dell\tfenv\lib\site-packages (from opencv-python-headless) (2.1.3)
```

In [6]:
```
#define the directories of the dataset
train_dir = r'C:\Users\dell\Downloads\archive\car\train'
```

In [7]:
```
#preprocessing
 def load_images_and_bboxes(img_dir, label_dir, img_size = (224,224)):
     images = []
     bboxes = []

     for filename in os.listdir(img_dir):
         if filename.endswith(" .jpg") or filename endswith(" .png"):
             #load_images
             img_path = os.path.join(img_dir, filename)
             img = cv2.imread(img_path)
             img = img.resize(img, img_size)
             img = img.astype("float32") / 255.0 #0,1
             images.append(img)
```

```
            #Load label bounding box if present
            label_path = os.path.join(label_dir, filename.replace('.jpg','.txt').replace('.png','.txt'))
            if os.path.exists(label_path):
                with open(label_path, 'r') as f:
                    label = f.read().strip().split()
                    if len(label) >= 5
                        bbox = [float(val) for val in label[1:5]]
                        bboxes.append(bbox)
                    else:
                        bboxes.append([0,0,0,0])


                else:
                   bboxes.append([0,0,0,0])
    return np.array(images), np.array(bboxes)

X, y = load_images_and_bboxes(os.path.join(train_dir, 'images'), os.path.join(train_dir, 'labels'))
```

```
  Cell In[7], line 2
    def load_images_and_bboxes(img_dir, label_dir, img_size = (224,224)):
    ^
IndentationError: unexpected indent
```

In [8]:

```
import os
import cv2
import numpy as np

# Define the directory of the dataset
train_dir = r'C:\Users\dell\Downloads\archive\car\train'

# Preprocessing function
def load_images_and_bboxes(img_dir, label_dir, img_size=(224, 224)):
    images = []
    bboxes = []

    for filename in os.listdir(img_dir):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            # Load image
```

```
            img_path = os.path.join(img_dir, filename)
            img = cv2.imread(img_path)
            img = cv2.resize(img, img_size)  # Corrected from img.resize to cv2.resize
            img = img.astype("float32") / 255.0
            images.append(img)

            # Load corresponding label (bounding box)
            label_filename = filename.replace('.jpg', '.txt').replace('.png', '.txt')
            label_path = os.path.join(label_dir, label_filename)

            if os.path.exists(label_path):
                with open(label_path, 'r') as f:
                    label = f.read().strip().split()
                    if len(label) >= 5:
                        bbox = [float(val) for val in label[1:5]]  # Usually [x_center, y_center, width, height]
                        bboxes.append(bbox)
                    else:
                        bboxes.append([0, 0, 0, 0])
            else:
                bboxes.append([0, 0, 0, 0])

    return np.array(images), np.array(bboxes)

# Usage
X, y = load_images_and_bboxes(
    os.path.join(train_dir, 'images'),
    os.path.join(train_dir, 'labels')
)
```

In [9]:
```
X[0]
```

Out[9]:
```
array([[[0.60784316, 0.37254903, 0.2627451 ],
        [0.6039216 , 0.36862746, 0.25490198],
        [0.6       , 0.3647059 , 0.2509804 ],
        ...,
        [0.28627452, 0.2901961 , 0.30980393],
        [0.32156864, 0.39607844, 0.43529412],
        [0.2627451 , 0.38039216, 0.42745098]],

       [[0.6117647 , 0.3764706 , 0.2627451 ],
        [0.60784316, 0.37254903, 0.25882354],
        [0.6       , 0.3647059 , 0.2509804 ],
        ...,
        [0.21960784, 0.22745098, 0.23921569],
        [0.23137255, 0.29411766, 0.31764707],
        [0.23921569, 0.34509805, 0.3764706 ]],

       [[0.6117647 , 0.38039216, 0.2627451 ],
        [0.60784316, 0.37254903, 0.25882354],
        [0.6039216 , 0.36862746, 0.25490198],
        ...,
        [0.25882354, 0.2627451 , 0.2627451 ],
        [0.3647059 , 0.40784314, 0.4117647 ],
        [0.27058825, 0.34117648, 0.34901962]],

       ...,

       [[0.14901961, 0.23921569, 0.29803923],
        [0.15294118, 0.24313726, 0.3019608 ],
        [0.15686275, 0.25882354, 0.3137255 ],
        ...,
        [0.15294118, 0.23921569, 0.28627452],
        [0.15294118, 0.24705882, 0.29411766],
        [0.15294118, 0.24705882, 0.29411766]],

       [[0.15294118, 0.24313726, 0.3019608 ],
        [0.20784314, 0.29411766, 0.3529412 ],
        [0.26666668, 0.36862746, 0.42352942],
        ...,
        [0.16470589, 0.2509804 , 0.29803923],
        [0.13725491, 0.23137255, 0.2784314 ],
```

```
        [0.15686275, 0.24705882, 0.29411766]],

       [[0.11764706, 0.20392157, 0.26666668],
        [0.18039216, 0.26666668, 0.32941177],
        [0.14509805, 0.24705882, 0.3019608 ],
        ...,
        [0.16078432, 0.24705882, 0.29411766],
        [0.09803922, 0.19215687, 0.23921569],
        [0.08627451, 0.1882353 , 0.23529412]]], dtype=float32)
```

In [10]:
```
y
```

Out[10]:
```
array([[0.53365385, 0.31730769, 0.16947115, 0.31730769],
       [0.57692308, 0.36057692, 0.41586538, 0.27524038],
       [0.51802885, 0.48557692, 0.62259615, 0.45072115],
       ...,
       [0.66947115, 0.43629808, 0.27283654, 0.43870192],
       [0.64302885, 0.48677885, 0.58413462, 0.85336538],
       [0.50120192, 0.30649038, 0.65024038, 0.44591346]])
```

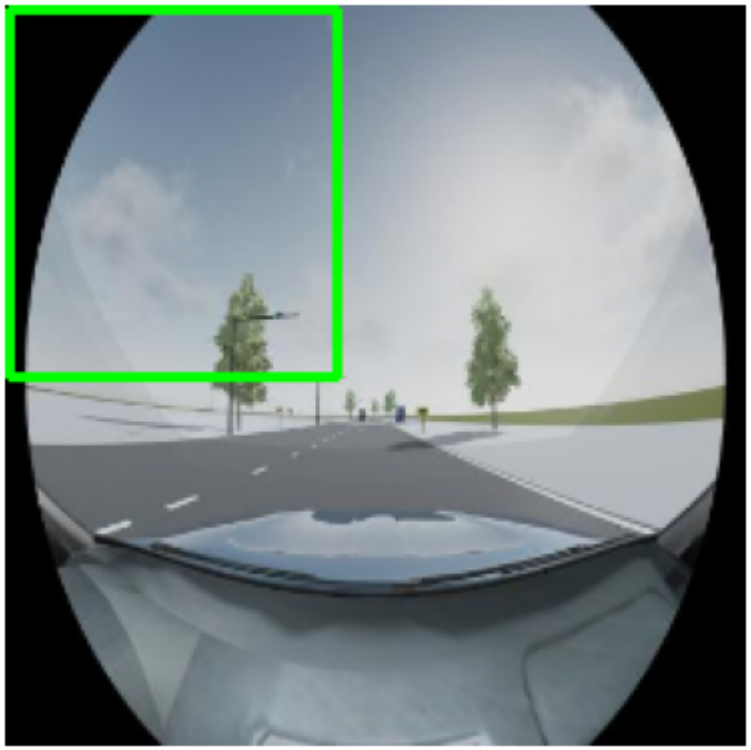
In [11]:
```
#Function to display a random image with its corresponding bounding box
def display_random_image_with_bbox(images, bboxes):
    #choose a random index
    random_index = random.randint(0, len(images) -1)

    #Get image and corresponding bounding box
    img = images[random_index]
    bbox = bboxes[random_index]

    #Convert from BGR to RGB makes easy for plotting
    image_rgb = cv2.cvtColor(img , cv2.COLOR_BGR2RGB)

    #convert bbox coordinates back to pixel values (relative to image size)
    img_height , img_width = img.shape[:2]
    x_min, y_min, x_max, y_max = bbox
    x_min = int(x_min * img_width)
    y_min = int(y_min * img_height)
    x_max = int(x_max * img_width)
    y_max = int(y_max * img_height)

    #Draw the bounding box
```

```
    image_with_bbox = cv2.rectangle(image_rgb,
                                    (x_min, y_min),
                                     (x_max , y_max),
                                     (0 ,255, 0), 2) # green bounding box

    plt.imshow(image_with_bbox)
    plt.axis('off')
    plt.show()

display_random_image_with_bbox(X, y)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..2
55.0].
```

In [12]:
```
display_random_image_with_bbox(X, y)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..2
55.0].
```



In [13]:
```
display_random_image_with_bbox(X, y)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..2
55.0].
```

In [14]:
```
#test train split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

In [15]:
```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adam

# Load pretrained ResNet50 without top classification layer
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False  # Freeze base model weights

# Build custom model
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1024, activation='relu'),
```

```
    layers.Dense(4, activation='sigmoid')  # Adjust output units for your problem
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=1e-4),
              loss='mean_squared_error',
              metrics=['accuracy'])

# Fit the model to training data
history = model.fit(X_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(X_test, y_test))
```

```
Epoch 1/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 265s 3s/step - accuracy: 0.3421 - loss: 0.0479 - val_accuracy: 0.3909 - val_loss: 0.0339
Epoch 2/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 205s 2s/step - accuracy: 0.4407 - loss: 0.0337 - val_accuracy: 0.4235 - val_loss: 0.0285
Epoch 3/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 206s 2s/step - accuracy: 0.4684 - loss: 0.0272 - val_accuracy: 0.4164 - val_loss: 0.0268
Epoch 4/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 212s 2s/step - accuracy: 0.4783 - loss: 0.0248 - val_accuracy: 0.4547 - val_loss: 0.0223
Epoch 5/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 207s 2s/step - accuracy: 0.5328 - loss: 0.0217 - val_accuracy: 0.5071 - val_loss: 0.0196
Epoch 6/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 209s 2s/step - accuracy: 0.5408 - loss: 0.0202 - val_accuracy: 0.5623 - val_loss: 0.0180
Epoch 7/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 197s 2s/step - accuracy: 0.5644 - loss: 0.0193 - val_accuracy: 0.5850 - val_loss: 0.0176
Epoch 8/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 198s 2s/step - accuracy: 0.5798 - loss: 0.0175 - val_accuracy: 0.5652 - val_loss: 0.0163
Epoch 9/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 197s 2s/step - accuracy: 0.5616 - loss: 0.0168 - val_accuracy: 0.4490 - val_loss: 0.0168
Epoch 10/10
89/89 ━━━━━━━━━━━━━━━━━━━━ 205s 2s/step - accuracy: 0.5683 - loss: 0.0165 - val_accuracy: 0.5878 - val_loss: 0.0150
```

In [16]:
```
#Evaluation
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_acc: .2f}")
print(f"Test loss : {test_loss: .2f}")
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 40s 2s/step - accuracy: 0.5980 - loss: 0.0145
Test accuracy:  0.59
Test loss :  0.01
```

In [17]:

```python
import cv2
import numpy as np
import os

def predict_and_visualize(model, image_path):
    # Read the image
    image = cv2.imread(image_path)

    # Check if the image is loaded properly
    if image is None:
        print(f"Failed to load image from: {image_path}")
        return

    # Store the original image shape
    original_height, original_width, _ = image.shape
    print(f"Original image shape: {image.shape}")

    # Resize the image to the model input size (224x224)
    image_resized = cv2.resize(image, (224, 224))

    # Normalize the image
    image_normalized = image_resized / 255.0

    # Add batch dimension
    image_expanded = np.expand_dims(image_normalized, axis=0)

    # Make prediction
    predicted_bbox = model.predict(image_expanded)
    predicted_bbox = predicted_bbox[0]  # Get first prediction from batch

    # Print predicted bounding box
    print(f"Predicted bounding box (normalized): {predicted_bbox}")

    # Convert normalized bbox to original image size
    x_min = int(predicted_bbox[0] * original_width)
    y_min = int(predicted_bbox[1] * original_height)
    x_max = int(predicted_bbox[2] * original_width)
```

```
    y_max = int(predicted_bbox[3] * original_height)
    print(f"Bounding box coordinates on the original image: ({x_min}, {y_min}), ({x_max}, {y_max})")

    # Draw bounding box on the original image
    image_with_bbox = image.copy()
    cv2.rectangle(image_with_bbox, (x_min, y_min), (x_max, y_max), (0, 255, 0), 2)

    # Display image (or save it if running in a headless environment)
    cv2.imshow("Predicted Bounding Box", image_with_bbox)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example usage (make sure the model is already loaded)
image_path = r"C:\Users\dell\Downloads\archive\car\train\images\road72_png.rf.175dec4085120f452e4f6955c441b603.jpg"
predict_and_visualize(model, image_path)
```

```
Original image shape: (416, 416, 3)
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step
Predicted bounding box (normalized): [0.54570556 0.4008032  0.22758491 0.28965944]
Bounding box coordinates on the original image: (227, 166), (94, 120)
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
Cell In[17], line 52
     50 # Example usage (make sure the model is already loaded)
     51 image_path = r"C:\Users\dell\Downloads\archive\car\train\images\road72_png.rf.175dec4085120f452e4f6955c441b603.jpg"
---> 52 predict_and_visualize(model, image_path)

Cell In[17], line 46, in predict_and_visualize(model, image_path)
     43 cv2.rectangle(image_with_bbox, (x_min, y_min), (x_max, y_max), (0, 255, 0), 2)
     45 # Display image (or save it if running in a headless environment)
---> 46 cv2.imshow("Predicted Bounding Box", image_with_bbox)
     47 cv2.waitKey(0)
     48 cv2.destroyAllWindows()

error: OpenCV(4.11.0) D:\a\opencv-python\opencv-python\opencv\modules\highgui\src\window.cpp:1301: error: (-2:Unspecified erro
r) The function is not implemented. Rebuild the library with Windows, GTK+ 2.x or Cocoa support. If you are on Ubuntu or Debia
n, install libgtk2.0-dev and pkg-config, then re-run cmake or configure script in function 'cvShowImage'
```

In [18]:
```
def predict_and_visualize(model, image_path):
    image = cv2.imread(image_path)
```

```
    # Store the original image shape
    original_height, original_width, _ = image.shape
    print(f"Original image shape: {image.shape}")

    # Resize the image to the model input size (224x224)
    image_resized = cv2.resize(image, (224, 224))
    image_normalized = image_resized / 255.0
    image_expanded = np.expand_dims(image_normalized, axis=0)  # Add batch dimension

    # Predict bounding box
    predicted_bbox = model.predict(image_expanded)
    predicted_bbox = predicted_bbox[0]
    print(f"Predicted bounding box (normalized): {predicted_bbox}")

# ✅ Call the function **outside** the function definition
image_path = r"C:\Users\dell\Downloads\archive\car\train\images\road72_png.rf.175dec4085120f452e4f6955c441b603.jpg"
predict_and_visualize(model, image_path)
```

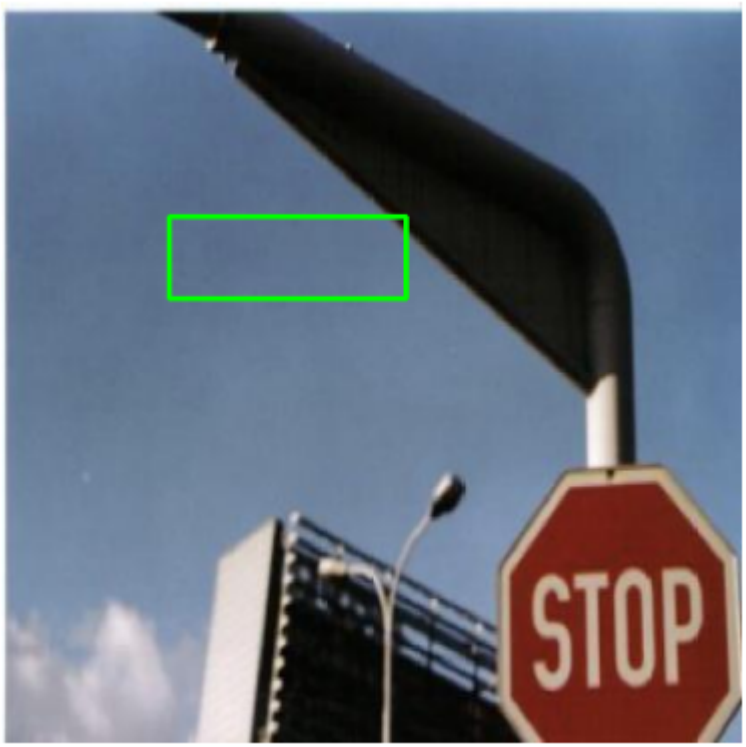
```
Original image shape: (416, 416, 3)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 233ms/step
Predicted bounding box (normalized): [0.54570556 0.4008032  0.22758491 0.28965944]
```

In [22]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def predict_and_visualize(model, image_path):
    image = cv2.imread(image_path)

    # Store the original image shape
    original_height, original_width, _ = image.shape
    print(f"Original image shape: {image.shape}")

    # Resize the image to the model input size (224x224)
    image_resized = cv2.resize(image, (224, 224))
    image_normalized = image_resized / 255.0
    image_expanded = np.expand_dims(image_normalized, axis=0)  # Add batch dimension

    # Predict bounding box
    predicted_bbox = model.predict(image_expanded)
    predicted_bbox = predicted_bbox[0]
```

```
    print(f"Predicted bounding box (normalized): {predicted_bbox}")

    # Extract normalized coordinates
    x_min, y_min, x_max, y_max = predicted_bbox

    # Scale to original image dimensions
    x_min = int(x_min * original_width)
    y_min = int(y_min * original_height)
    x_max = int(x_max * original_width)
    y_max = int(y_max * original_height)

    # Clamp coordinates to image bounds
    x_min = max(0, x_min)
    y_min = max(0, y_min)
    x_max = min(original_width, x_max)  # ✅ fixed here
    y_max = min(original_height, y_max)

    print(f"Bounding box coordinates on the original image: ({x_min}, {y_min}) , ({x_max}, {y_max})")

    # Draw the bounding box on the original image
    image_with_bbox = cv2.rectangle(image.copy(),
                                    (x_min, y_min),
                                    (x_max, y_max),
                                    (0, 255, 0), 2)  # Green bounding box

    plt.imshow(cv2.cvtColor(image_with_bbox, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

# ✅ Call the function
image_path = r"C:\Users\dell\Downloads\archive\car\train\images\road72_png.rf.175dec4085120f452e4f6955c441b603.jpg"
predict_and_visualize(model, image_path)
```

```
Original image shape: (416, 416, 3)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 228ms/step
Predicted bounding box (normalized): [0.54570556 0.4008032  0.22758491 0.28965944]
Bounding box coordinates on the original image: (227, 166) , (94, 120)
```



In [23]:
```
image_path = r"C:\Users\dell\Downloads\archive\car\train\images\road79_png.rf.45c47a11851b6f864bfc4d1a5df7db98.jpg"
predict_and_visualize(model, image_path)
```

```
Original image shape: (416, 416, 3)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 253ms/step
Predicted bounding box (normalized): [0.5186181  0.39110598 0.45651066 0.5528475 ]
Bounding box coordinates on the original image: (215, 162) , (189, 229)
```

In [24]:
```
image_path =  r"C:\Users\dell\Downloads\archive\car\train\images\road746_png.rf.16a262dc32378a4529131ba5aed1e815.jpg"
predict_and_visualize(model, image_path)
```

```
Original image shape: (416, 416, 3)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 250ms/step
Predicted bounding box (normalized): [0.4627753  0.4008373  0.22281986 0.23306286]
Bounding box coordinates on the original image: (192, 166) , (92, 96)
```

In [ ]: