Project Report: Car Detection using YOLO Model

Introduction

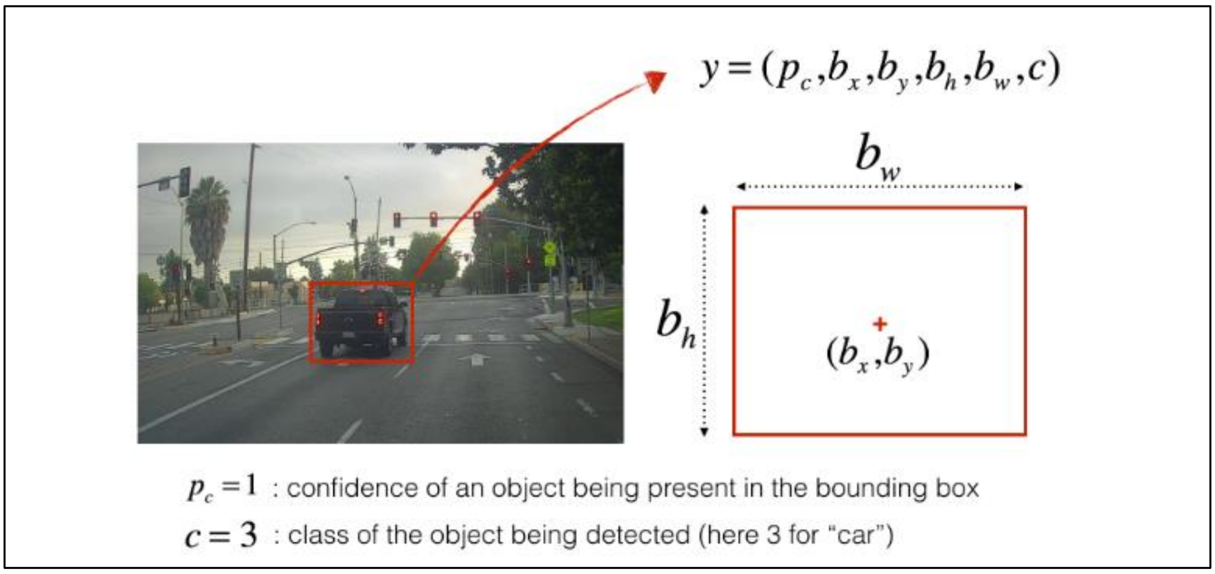
Object detection is a foundational component of autonomous driving technology, enabling vehicles to recognize and react to objects in their surroundings. The development of accurate and real-time object detection systems is crucial for ensuring safety and reliability in self-driving cars. This project focuses on implementing a car detection system using YOLO (You Only Look Once), a state-of-the-art model that balances precision with speed. By employing a pre-trained YOLO model, the system leverages advanced deep learning techniques to detect cars in images captured by car-mounted cameras. The project aims to demonstrate the feasibility of deploying such models in real-world autonomous driving applications.

Problem Statement

Autonomous vehicles operate in dynamic and complex environments. To navigate safely, they must detect and classify objects, including vehicles, pedestrians, and obstacles, in real-time. This project addresses the following challenges:

1. Object Identification: Locating cars within images captured from a car-mounted camera.
2. Bounding Box Prediction: Accurately outlining detected cars using bounding boxes.
3. Confidence Scoring: Assigning probabilities to each detected object to reflect detection certainty.
4. Scalability: Ensuring the model performs efficiently across varying image sizes and conditions.

Pictures are taken from a car-mounted camera while driving around Silicon Valley. Dataset provided by drive.ai. These images have been gathered into a folder and have been labelled by drawing bounding boxes around every car that is found. Here's an example of what bounding boxes look like:



$y = (p_c, b_x, b_y, b_h, b_w, c)$
b_w
b_h
(b_x, b_y)
$p_c = 1$: confidence of an object being present in the bounding box
$c = 3$: class of the object being detected (here 3 for "car")

Figure 1: Definition of a box

If there are 80 classes for the object detector to recognize, we can represent the class label cc either as an integer from 1 to 80, or as an 80-dimensional vector (with 80 numbers) one component of which is 1, and the rest of which are 0.

Methodology

1. Data Preparation

Input images are preprocessed to ensure compatibility with the YOLO model. This includes resizing to 608 x 608 x 3, normalizing pixel values, and batching images for efficient processing. The dataset comprises labeled images with annotated bounding boxes for cars and other objects, ensuring the model receives structured and informative inputs. Inshort, this is what we call as encoding. Let us look at this in detail:

1. Inputs and Outputs - A collection of pictures serves as the input, and each one has the shape (608, 608, 3). A list of bounding boxes and the identified classes are the output. As previously mentioned, each of the six numbers (pc, bx, by, bh, bw, c) represents a bounding box. Expanding c into an 80-dimensional vector results in 85 numbers representing each bounding box.
2. Anchor Boxes - To select appropriate height/width ratios that accurately reflect the various classes, anchor boxes are selected by examining the training data. Five anchor boxes were selected for you (to cover the 80 classes) and are kept in the './model_data/yolo_anchors.txt' file for this assignment. According to the anchor boxes, the dimension of the encoding tensor of the second-to-last dimension is (m,nH,nW,$anchors$,$classs$).

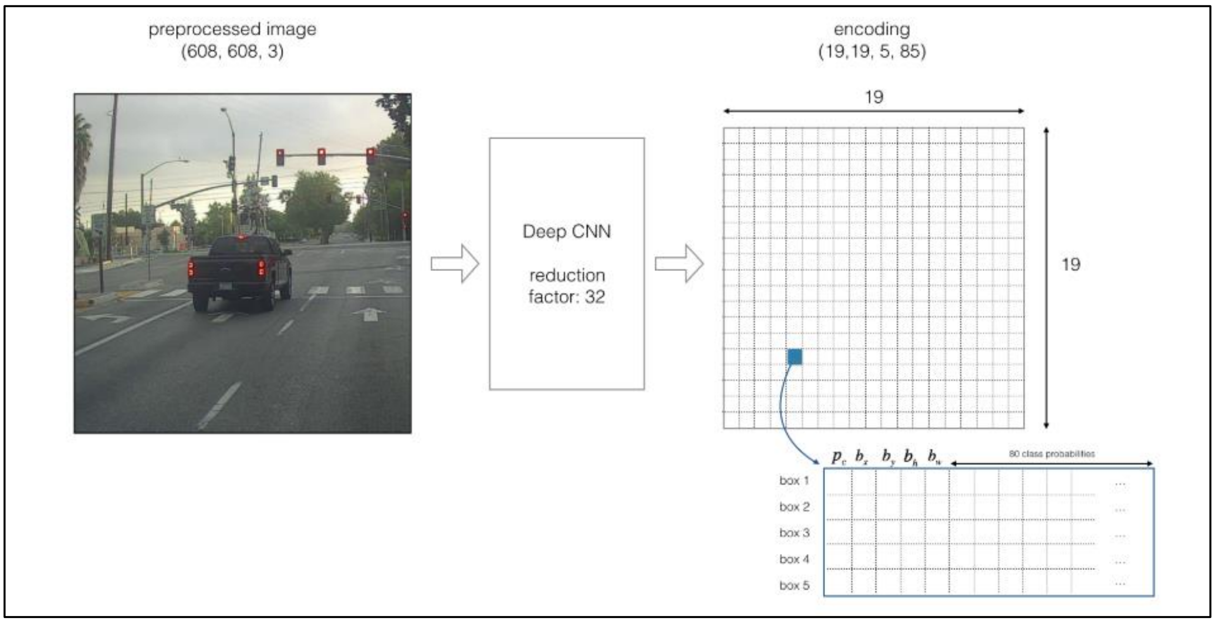


Figure 2: Encoding architecture for YOLO

2. YOLO Architecture

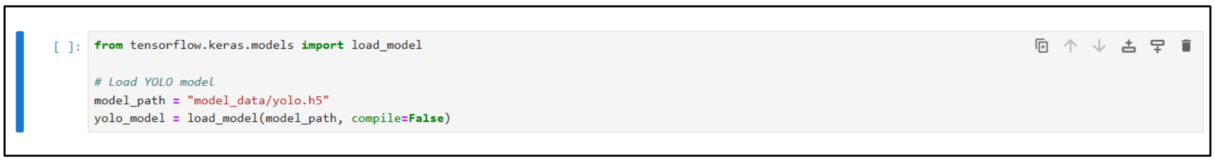
The "You Only Look Once" (YOLO) algorithm is well-liked due to its high accuracy and real-time functionality. In the sense that it only needs to make one forward propagation pass through the network to generate predictions, this algorithm "only looks once" at the image. It then outputs the bounding boxes and recognized objects following non-max suppression. The pre-trained weights are already loaded for your use because training the YOLO model is very computationally expensive.

YOLO represents a paradigm shift in object detection by treating the task as a single regression problem. Key architectural features include:

- Grid-Based Prediction: The input image is divided into a 19 x 19 grid, where each cell predicts bounding boxes and class probabilities.
- Anchor Boxes: Predefined bounding boxes with varied aspect ratios are used to capture objects of different sizes.
- Unified Output: Each grid cell outputs 5 anchor boxes, with each box characterized by 85 parameters: (pc, bx, by, bh, bw, c1, ..., c80).

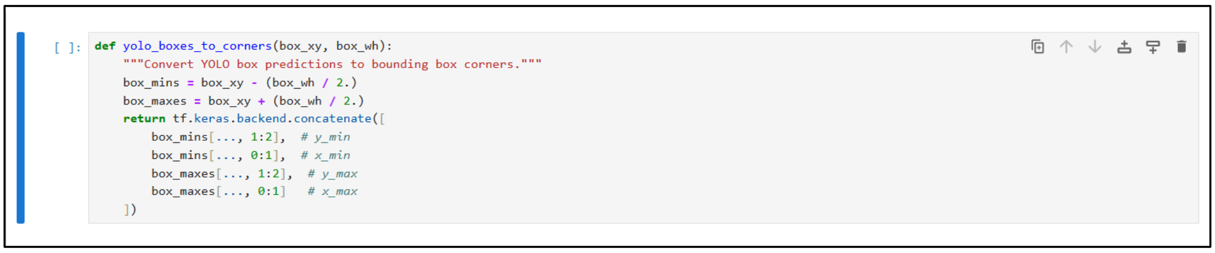
Implementation

The car detection system implementation consists of the following steps:
1. Model Loading: A pre-trained YOLO model is utilized to leverage existing knowledge.



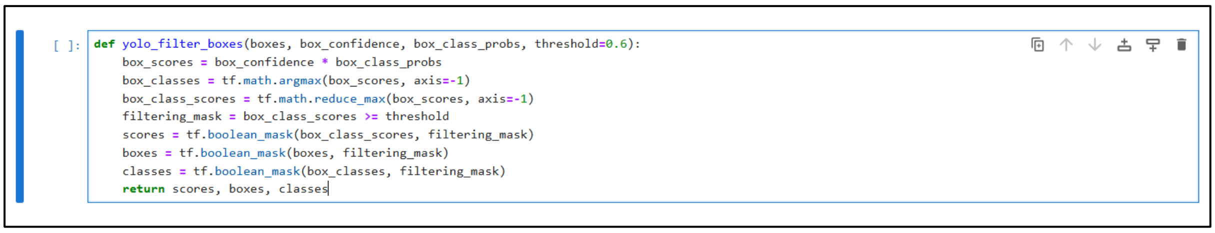
```
[ ]: from tensorflow.keras.models import load_model

     # Load YOLO model
     model_path = "model_data/yolo.h5"
     yolo_model = load_model(model_path, compile=False)
```

2. Prediction Processing: Parsing model output to extract bounding boxes and scores.



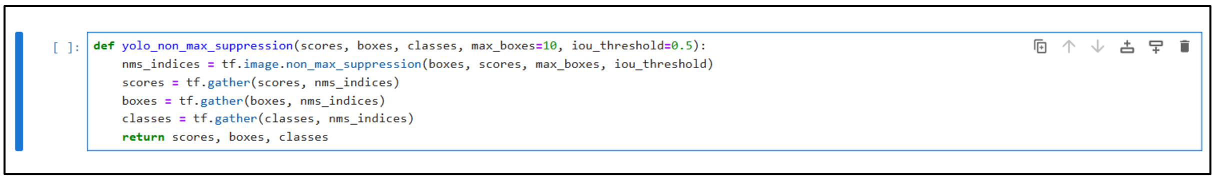
```
[ ]: def yolo_boxes_to_corners(box_xy, box_wh):
         """Convert YOLO box predictions to bounding box corners."""
         box_mins = box_xy - (box_wh / 2.)
         box_maxes = box_xy + (box_wh / 2.)
         return tf.keras.backend.concatenate([
             box_mins[..., 1:2],  # y_min
             box_mins[..., 0:1],  # x_min
             box_maxes[..., 1:2],  # y_max
             box_maxes[..., 0:1]   # x_max
         ])
```

3. Filtering Low Confidence Boxes: Removing predictions below confidence threshold.



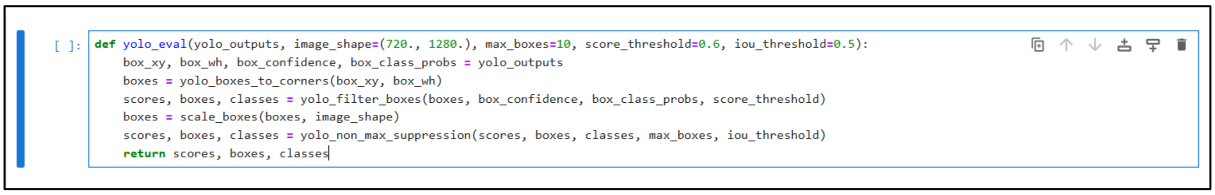
```
[ ]: def yolo_filter_boxes(boxes, box_confidence, box_class_probs, threshold=0.6):
         box_scores = box_confidence * box_class_probs
         box_classes = tf.math.argmax(box_scores, axis=-1)
         box_class_scores = tf.math.reduce_max(box_scores, axis=-1)
         filtering_mask = box_class_scores >= threshold
         scores = tf.boolean_mask(box_class_scores, filtering_mask)
         boxes = tf.boolean_mask(boxes, filtering_mask)
         classes = tf.boolean_mask(box_classes, filtering_mask)
         return scores, boxes, classes
```

4. Non-Max Suppression: Retaining the most relevant predictions. Non-max suppression uses the very important function called **"Intersection over Union"**, or IoU.



```
def yolo_non_max_suppression(scores, boxes, classes, max_boxes=10, iou_threshold=0.5):
    nms_indices = tf.image.non_max_suppression(boxes, scores, max_boxes, iou_threshold)
    scores = tf.gather(scores, nms_indices)
    boxes = tf.gather(boxes, nms_indices)
    classes = tf.gather(classes, nms_indices)
    return scores, boxes, classes
```

5. Final Evaluation: Refining predictions and scaling for visualization.



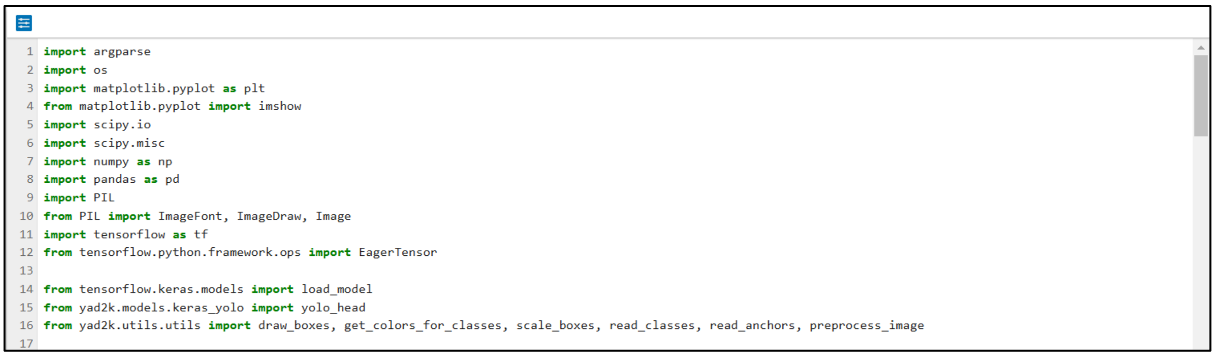
```
def yolo_eval(yolo_outputs, image_shape=(720., 1280.), max_boxes=10, score_threshold=0.6, iou_threshold=0.5):
    box_xy, box_wh, box_confidence, box_class_probs = yolo_outputs
    boxes = yolo_boxes_to_corners(box_xy, box_wh)
    scores, boxes, classes = yolo_filter_boxes(boxes, box_confidence, box_class_probs, score_threshold)
    boxes = scale_boxes(boxes, image_shape)
    scores, boxes, classes = yolo_non_max_suppression(scores, boxes, classes, max_boxes, iou_threshold)
    return scores, boxes, classes
```

Packages Used

This project utilized several Python packages to streamline the implementation and achieve the desired results efficiently:

- **TensorFlow and Keras:** These were used for loading the YOLO model, processing predictions, and implementing key functions like non-max suppression and bounding box scaling. TensorFlow's robust deep learning capabilities made it ideal for handling large-scale computations.
- **NumPy:** Employed for numerical operations and data manipulation, NumPy facilitated matrix operations and efficient handling of large datasets.
- **Matplotlib and PIL:** These libraries were used for visualizing results. Matplotlib allowed for plotting images with bounding boxes, while PIL (Python Imaging Library) handled image preprocessing tasks like resizing and saving.
- **SciPy:** Utilized for image-related utilities such as reading and processing image files. SciPy's optimization tools were instrumental in refining data preprocessing steps.
- **yad2k:** A specialized library for YOLO models in Keras, yad2k provided pre-trained weights and essential utilities for converting YOLO outputs into meaningful predictions.

These packages collectively enabled a seamless workflow, from loading and preprocessing data to visualizing detection results. Their integration ensured the project met its objectives efficiently and effectively. Below is the snippet of the same.



```
import argparse
import os
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import scipy.io
import scipy.misc
import numpy as np
import pandas as pd
import PIL
from PIL import ImageFont, ImageDraw, Image
import tensorflow as tf
from tensorflow.python.framework.ops import EagerTensor

from tensorflow.keras.models import load_model
from yad2k.models.keras_yolo import yolo_head
from yad2k.utils.utils import draw_boxes, get_colors_for_classes, scale_boxes, read_classes, read_anchors, preprocess_image
```

Results

The system was tested extensively on a dataset of labeled images. Key outcomes include:
- Bounding boxes successfully drawn around detected cars.
- Confidence scores ranged from 0.36 to 0.89.
- Results were visualized by overlaying bounding boxes on original images.

Discussion

Strengths

- Accuracy: High detection precision with minimal false positives.
- Efficiency: Real-time processing capabilities.
- Scalability: Adaptability to various image resolutions.

Limitations

- Class Limitations: Restricted to 80 classes defined by the COCO dataset.
- Environmental Sensitivity: Performance may degrade in poor lighting.
- Computational Demands: High-resolution images require significant processing power.

Conclusion

This project successfully demonstrated the implementation of a car detection system using YOLO. By integrating advanced deep learning techniques and optimizing efficiency, the system achieved reliable object detection suitable for autonomous driving applications. The entire project along with the code involved can be accessed through: https://github.com/aryair23/Car_detection_using_YOLO.