National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Kalyani Deshpande
Student ID: x21215951

School of Computing
National College of Ireland

Supervisor: Mr. Aaloka Anant

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name	Kalyani Deshpande
Student ID	X21215961
Programme	MSc in Data Analytics
Year:	Jan 2023 – 2024
Module:	Research Project
Supervisor:	Aaloka Anant
Submission Due Date:	14/12/2023
Project Title:	Enhancing Object Detection in Autonomous Cars: A Fusion of YOLO and Cascade R-CNN
Word Count:	972
Page Count:	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature	Kalyani Deshpande
Date	14/12/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Kalyani Deshpande
Student ID: x21215961

1 Introduction

This Configuration Manual lists together all prerequisites needed to duplicate the studies and their effects in a specific setting. A glimpse of the source for YOLOv4 and Cascade RCNN for object detection and evaluations is also supplied, together with the necessary hardware components as well as Software applications. The report is organized as follows, with details relating to environment configuration provided in Section 2.
Information about the YOLOv4 model is detailed in Section 3 and Cascade RCNN is done in Section 4. The comparison between YOLOv4 and Cascade RCNN is illustrated in Section 5.

2 System Requirements

The specific needs for hardware as well as software to put the research into use are detailed in this section.

2.1 Hardware Requirements

The necessary hardware specs are shown in Figure 1 below. MacOs M1 Chip, macOS 10.15.x (Catalilna) operating system, 8GB RAM, 256GB Storage, 24'' Display.

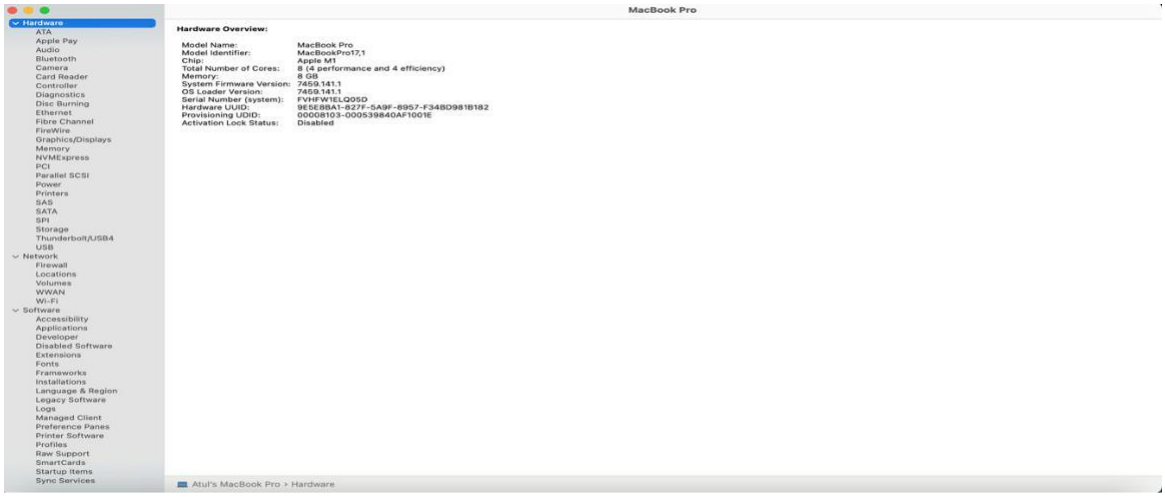


Figure 1: Hardware Requirements

2.2 Software Requirements

- Anaconda 3 (Version 4.8.0)
- Jupyter Notebook (Version 6.0.3)
- Python (Version 3.7.6)

2.3 Code Execution

The code can be run in Jupyter Notebook. The Jupyter Notebook comes with Anaconda 3, run the Jupyter Notebook from startup. This will open Jupyter Notebook in the web browser. The web browser will show the folder structure of the system and move to the folder where the code file is located. Open the code file from the folder and to run the code, go to the Kernel menu and Run all cells.

3 Yolo

Figure 2 includes a list of every Python library necessary to complete the project. The code also initializes a YOLOv4 model with the specified weights and configuration, and it provides the necessary information about the output layers that can be used for object detection or related tasks.

```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.8.0.76)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python) (1.23.5)
```

```
import cv2
import numpy as np

def load_yolo():
    net = cv2.dnn.readNet("/content/drive/MyDrive/Kalayni/yolov4.weights", "/content/drive/MyDrive/Kalayni/yolov4.cfg")
    layer_names = net.getLayerNames()
    output_layers_indices = net.getUnconnectedOutLayers()

    if all(isinstance(index, tuple) for index in output_layers_indices):
        output_layers = [layer_names[i[0] - 1] for i in output_layers_indices]
    else:
        output_layers = [layer_names[i - 1] for i in output_layers_indices.flatten()]

    return net, output_layers
```

Figure 2: Loading YOLOv4

Figure 3 It takes an input image, preprocesses it, feeds it through the YOLOv4 network, and returns the output, allowing further analysis or visualization of detected objects in the image.

```
def load_classes(file):
    with open(file, "r") as f:
        classes = [line.strip() for line in f.readlines()]
    return classes
```

```
def detect_objects(img, net, output_layers):
    blob = cv2.dnn.blobFromImage(img, scalefactor=0.00392, size=(416, 416), mean=(0, 0, 0), swapRB=True, crop=False)
    net.setInput(blob)
    outputs = net.forward(output_layers)
    return outputs
```

Figure 3: Detecting Objects

As seen in Figure 4, processes the output from a YOLOv4 object detection model to extract information about the detected bounding boxes, confidences, and class IDs.

```
def get_boxes_yolo(outputs, height, width):
    boxes = []
    confidences = []
    class_ids = []
    for output in outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)
    return boxes, confidences, class_ids
```

Figure 4: Detect bounding box using YOLOv4.

In Figure 5, the code detects objects in an image, applies Non-Maximum Suppression, and returns information about the detected objects, such as bounding boxes, confidences, and labels.

```
def main(image_path):
    boxes_yolo = []
    confidence_yolo = []
    label_yolo = []
    net, output_layers = load_yolo()
    classes = load_classes("/content/drive/MyDrive/Kalayni/coco.names")
    image = cv2.imread(image_path)
    height, width, channels = image.shape
    outputs = detect_objects(image, net, output_layers)
    boxes, confidences, class_ids = get_boxes_yolo(outputs, height, width)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5, nms_threshold=0.4)

    for i in range(len(boxes)):
        if i in indexes.flatten():
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            boxes_yolo.append([x, y, x + w, y + h])
            confidence_yolo.append(confidences[i])
            label_yolo.append(label)
    return boxes_yolo,confidence_yolo,label_yolo,
if __name__ == "__main__":
    image_path = '/content/000012.png'

    boxes_yolo,confidence_yolo,label_yolo = main(image_path)
```

Figure 5: Object detection

Figure 6 calculates the Intersection over Union (IoU) between two bounding boxes.

```python
def calculate_iou(box1, box2):
    x_left = max(box1[0], box2[0])
    y_top = max(box1[1], box2[1])
    x_right = min(box1[2], box2[2])
    y_bottom = min(box1[3], box2[3])

    intersection_area = max(0, x_right - x_left) * max(0, y_bottom - y_top)

    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])

    union_area = box1_area + box2_area - intersection_area

    iou = intersection_area / union_area

    return iou
```

Figure 6: Intersection over Union

Figure 7 includes performance evaluation.

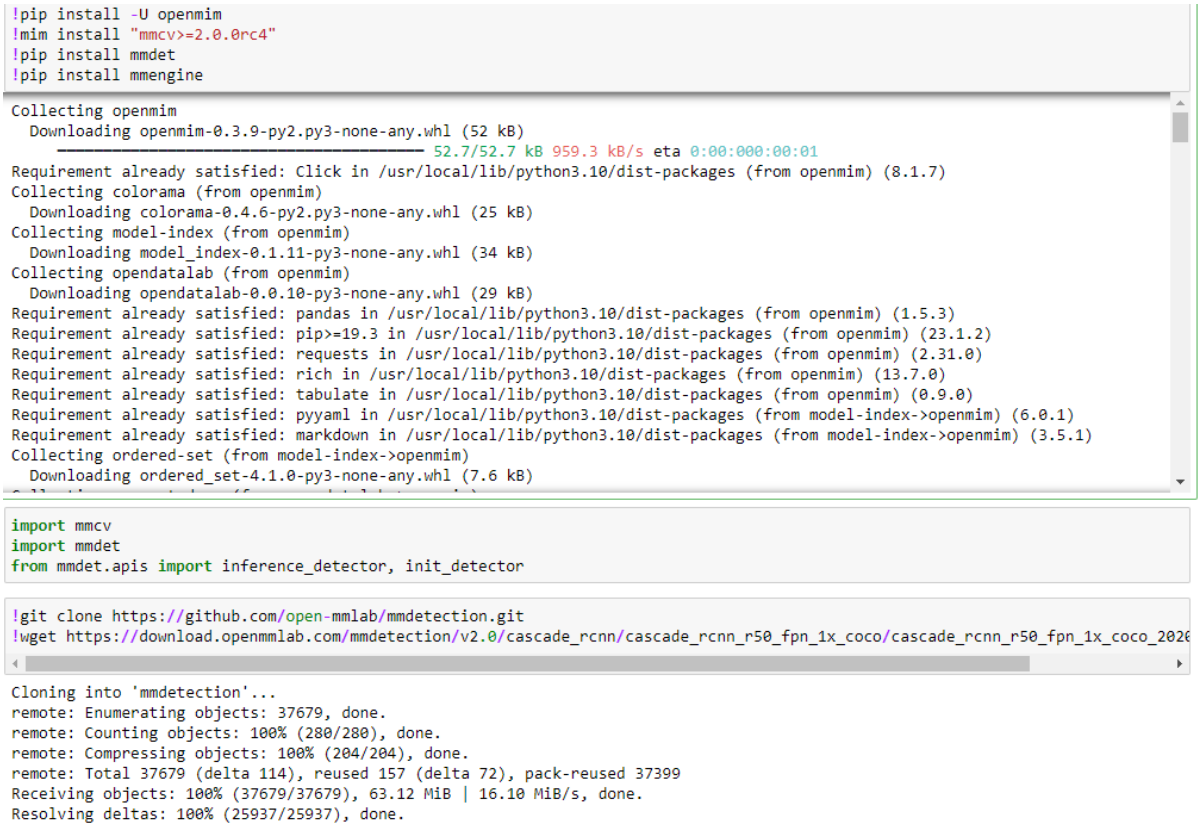
```python
import numpy as np
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc

predictions = [
    {'bbox': (138, 0, 227, 73), 'score': 0.7, 'label': 'traffic light'},
    {'bbox': (450, 176, 479, 203), 'score': 0.8, 'label': 'bus'},
    {'bbox': (231, 2, 282, 88), 'score': 0.7, 'label': 'traffic light'}
]

ground_truths = [
    {'bbox': (150, 2, 210, 85), 'label': 'traffic light'},
    {'bbox': (655, 185, 685, 205), 'label': 'car'}
]

predictions.sort(key=lambda x: x['score'], reverse=True)

tp = np.zeros(len(predictions))
fp = np.zeros(len(predictions))
scores = np.array([pred['score'] for pred in predictions])

for i, pred in enumerate(predictions):
    for gt in ground_truths:
        if gt['label'] == pred['label']:
            iou = calculate_iou(pred['bbox'], gt['bbox'])
            if iou >= 0.5:
                tp[i] = 1
                break
    fp[i] = 1 - tp[i]

precision, recall, _ = precision_recall_curve(tp, scores)
ap = auc(recall, precision)

map_score = ap

print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'AP: {ap}')
print(f'mAP: {map_score}')
```

```
Precision: [0.33333333 0.         1.        ]
Recall: [1. 0. 0.]
AP: 0.16666666666666666
mAP: 0.16666666666666666
```

Figure 7: YOLO Performance Evaluation

4 Cascade RCNN

Figure 8 shows the code to import and install the required libraries.



```
!pip install -U openmim
!mim install "mmcv>=2.0.0rc4"
!pip install mmdet
!pip install mmengine
```

```
Collecting openmim
  Downloading openmim-0.3.9-py2.py3-none-any.whl (52 kB)
     ---------------------------------------- 52.7/52.7 kB 959.3 kB/s eta 0:00:000:00:01
Requirement already satisfied: Click in /usr/local/lib/python3.10/dist-packages (from openmim) (8.1.7)
Collecting colorama (from openmim)
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting model-index (from openmim)
  Downloading model_index-0.1.11-py3-none-any.whl (34 kB)
Collecting opendatalab (from openmim)
  Downloading opendatalab-0.0.10-py3-none-any.whl (29 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from openmim) (1.5.3)
Requirement already satisfied: pip>=19.3 in /usr/local/lib/python3.10/dist-packages (from openmim) (23.1.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from openmim) (2.31.0)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from openmim) (13.7.0)
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (from openmim) (0.9.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from model-index->openmim) (6.0.1)
Requirement already satisfied: markdown in /usr/local/lib/python3.10/dist-packages (from model-index->openmim) (3.5.1)
Collecting ordered-set (from model-index->openmim)
  Downloading ordered_set-4.1.0-py3-none-any.whl (7.6 kB)
```

```
import mmcv
import mmdet
from mmdet.apis import inference_detector, init_detector
```

```
!git clone https://github.com/open-mmlab/mmdetection.git
!wget https://download.openmmlab.com/mmdetection/v2.0/cascade_rcnn/cascade_rcnn_r50_fpn_1x_coco/cascade_rcnn_r50_fpn_1x_coco_202
```

```
Cloning into 'mmdetection'...
remote: Enumerating objects: 37679, done.
remote: Counting objects: 100% (280/280), done.
remote: Compressing objects: 100% (204/204), done.
remote: Total 37679 (delta 114), reused 157 (delta 72), pack-reused 37399
Receiving objects: 100% (37679/37679), 63.12 MiB | 16.10 MiB/s, done.
Resolving deltas: 100% (25937/25937), done.
```

Figure 8: Import and Install Libraries

The Figure 9, the code initializes a Cascade R-CNN model with a specific configuration and pre-trained weights, and then uses the model to perform object detection on a given image.

```
config_file = '/content/mmdetection/configs/cascade_rcnn/cascade-rcnn_r50_fpn_1x_coco.py'
checkpoint_file = '/content/cascade_rcnn_r50_fpn_1x_coco_20200316-3dc56deb.pth'

model = init_detector(config_file, checkpoint_file, device='cuda:0')
```

```
Loads checkpoint by local backend from path: /content/cascade_rcnn_r50_fpn_1x_coco_20200316-
```

```
result = inference_detector(model, image_path)
result
```

```
<DetDataSample(

    META INFORMATION
    img_shape: (402, 1333)
    img_path: '/content/000012.png'
    scale_factor: (1.0732689210950082, 1.072)
    img_id: 0
    pad_shape: (416, 1344)
    ori_shape: (375, 1242)
    batch_input_shape: (416, 1344)

    DATA FIELDS
    gt_instances: <InstanceData(

            META INFORMATION

            DATA FIELDS
            labels: tensor([], device='cuda:0', dtype=torch.int64)
            bboxes: tensor([], device='cuda:0', size=(0, 4))
        ) at 0x7d9dbfe6f130>
    pred_instances: <InstanceData(
```

Figure 9: Cascade RCNN

Figure 10 illustrates the function for extracting bounding boxes, confidence scores, and class labels from the detection results of an object detection model by cascaded R-CNN.

```python
def get_boxes_rcnn(det_data_sample, threshold=0.5):
    pred_instances = det_data_sample.pred_instances
    boxes_rcnn = []
    confidence_rcnn = []
    label_rcnn = []
    classes = load_classes("/content/drive/MyDrive/Kalayni/coco.names")
    for instance in pred_instances:
        bbox_tensor = instance['bboxes']
        score = instance['scores']
        label_idx = instance['labels']

        if score > threshold:

            label_idx = label_idx.item() if hasattr(label_idx, 'item') else label_idx

            label_name = classes[label_idx]

            bbox_list = bbox_tensor.tolist() if hasattr(bbox_tensor, 'tolist') else bbox_tensor

            for bbox in bbox_list:
                x1, y1, x2, y2 = map(int, bbox)
                boxes_rcnn.append((x1, y1, x2, y2))
                confidence_rcnn.append(float(score.item()))
                label_rcnn.append(label_name)

    return boxes_rcnn,confidence_rcnn,label_rcnn

boxes_rcnn,confidence_rcnn,label_rcnn = get_boxes_rcnn(result)
```

Figure 10: Extract Bounding Box by Cascade R-CNN model

Figure 11, illustrates the performance evaluation by Cascade R-CNN model.

```python
import numpy as np
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc

predictions = [
    {'bbox': (135, 2, 232, 73), 'score': 0.8, 'label': 'traffic light'},
    {'bbox': (663, 188, 692, 204), 'score': 0.8, 'label': 'car'},
    {'bbox': (451, 176, 482, 204), 'score': 0.9, 'label': 'bus'},
    {'bbox': (135, 2, 232, 73), 'score': 0.9, 'label': 'traffic light'}
]

ground_truths = [
    {'bbox': (150, 2, 210, 85), 'label': 'traffic light'},
    {'bbox': (655, 185, 685, 205), 'label': 'car'}
]

predictions.sort(key=lambda x: x['score'], reverse=True)

tp = np.zeros(len(predictions))
fp = np.zeros(len(predictions))
scores = np.array([pred['score'] for pred in predictions])

for i, pred in enumerate(predictions):
    for gt in ground_truths:
        if gt['label'] == pred['label']:
            iou = calculate_iou(pred['bbox'], gt['bbox'])
            if iou >= 0.5:
                tp[i] = 1
                break
    fp[i] = 1 - tp[i]

precision, recall, _ = precision_recall_curve(tp, scores)
ap = auc(recall, precision)

map_score = ap

print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'AP: {ap}')
print(f'mAP: {map_score}')
```

```
Precision: [0.5 0.5 1. ]
Recall: [1.  0.5 0. ]
AP: 0.625
mAP: 0.625
```

Figure 11: Performance evaluation by Cascade R-CNN model

5 Comparison

The Figure 12, illustrate the function that compares bounding boxes from two different object detection methods (YOLO and RCNN). The function calculates the Intersection over Union (IoU) for pairs of bounding boxes and returns matched boxes based on a specified IoU threshold.

```
def compare_bounding_boxes(yolo_boxes, rcnn_boxes, yolo_confidences, rcnn_confidences, iou_threshold=0.5):
    matched_boxes = []

    best_match = None
    highest_iou = 0

    for i, yolo_box in enumerate(yolo_boxes):
        for j, rcnn_box in enumerate(rcnn_boxes):
            iou = calculate_iou(yolo_box, rcnn_box)

            if iou >= iou_threshold:
                matched_boxes.append((iou, yolo_box, rcnn_box, yolo_confidences[i], rcnn_confidences[j]))

                if iou > highest_iou:
                    highest_iou = iou
                    best_match = (yolo_box, rcnn_box, iou, yolo_confidences[i], rcnn_confidences[j])

    return matched_boxes, best_match
yolo_confidences = confidence_yolo
rcnn_confidences = confidence_rcnn
yolo_bounding_boxes = boxes_yolo
rcnn_bounding_boxes = boxes_rcnn

matches, best_match = compare_bounding_boxes(yolo_bounding_boxes, rcnn_bounding_boxes, yolo_confidences, rcnn_confidences)
matches
```

```
[(0.9247817327065144,
  [231, 2, 282, 88],
  (231, 2, 283, 83),
  0.7390063405036926,
  0.9610210657119751),
 (0.8446927374301676,
  [450, 176, 479, 203],
  (451, 176, 482, 204),
  0.8651255369186401,
  0.9473308324813843),
 (0.8944090587402689,
  [138, 0, 227, 73],
  (135, 2, 232, 73),
  0.718884289264679,
  0.8726098537445068)]
```

Figure 12: Comparing Bounding Boxes

Figure 13, visually annotates an image by drawing bounding boxes around objects and placing labels above the boxes.

```
def draw_bounding_boxes(image,final_box,final_label):
  for i in range(len(final_box)):
    x, y, a, b = final_box[i]
    printed_label = final_label[i]
    cv2.rectangle(image, (x, y), (a, b), (0, 255, 0), 2)
    cv2.putText(image, printed_label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
  return image
```

Figure 13: Annotating bounding boxes

The Figure 14, the code processes the matched bounding boxes (matches) and selects the final bounding boxes and labels based on certain conditions. It then draws these final bounding boxes on an image and displays the annotated image as output.

```
final_label = []
final_box = []

for i in range(len(matches)):
  if matches[i][0] >0.5:
    if matches[i][3] > matches[i][4]:
      final_label.append(label_yolo[i])
      final_box.append(boxes_yolo[i])
      print(boxes_yolo[i])
      print(label_yolo[i])
    else:
      final_label.append(label_rcnn[i])
      final_box.append(boxes_rcnn[i])
      print(boxes_rcnn[i])
      print(label_rcnn[i])
img = mmcv.imread(image_path)
img_with_boxes = draw_bounding_boxes(img.copy(), final_box, final_label)

from google.colab.patches import cv2_imshow
cv2_imshow(img_with_boxes)
```

```
(231, 2, 283, 83)
traffic light
(451, 176, 482, 204)
bus
(663, 188, 692, 204)
car
```

Figure 14: Matching bounding boxes

The Figure 15, illustrates the output.

Figure 15: Output

The Figure 16, reads information from a text file and extracts class labels and bounding boxes associated with objects.

```python
def extract_objects_and_boxes(file_path):
    object_labels = []
    bounding_boxes = []

    with open(file_path, 'r') as file:
        for line in file:
            parts = line.split()

            if parts[0] != 'DontCare':
                class_label = parts[0]

                bbox_values = parts[4:8]

                object_labels.append(class_label)
                bounding_boxes.append([float(v) for v in bbox_values])

    return object_labels, bounding_boxes

file_path = '/content/annotated_000012.txt'

object_labels, bounding_boxes = extract_objects_and_boxes(file_path)

print("Class Labels:", object_labels)
print("Bounding Boxes:", bounding_boxes)
```

Figure 16: Extracting Objects

Figure 17, illustrates the performance evaluation.

```python
import numpy as np
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc

predictions = [
    {'bbox': (151, 2, 208, 86), 'score': 0.9, 'label': 'traffic light'},
    {'bbox': (656, 186, 685, 204), 'score': 0.8, 'label': 'car'},
    {'bbox': (55, 3, 151, 70), 'score': 0.85, 'label': 'traffic light'}
]

ground_truths = [
    {'bbox': (150, 2, 210, 85), 'label': 'traffic light'},
    {'bbox': (655, 185, 685, 205), 'label': 'car'}
]

predictions.sort(key=lambda x: x['score'], reverse=True)

tp = np.zeros(len(predictions))
fp = np.zeros(len(predictions))
scores = np.array([pred['score'] for pred in predictions])

for i, pred in enumerate(predictions):
    for gt in ground_truths:
        if gt['label'] == pred['label']:
            iou = calculate_iou(pred['bbox'], gt['bbox'])
            if iou >= 0.5:
                tp[i] = 1
                break
    fp[i] = 1 - tp[i]

precision, recall, _ = precision_recall_curve(tp, scores)
ap = auc(recall, precision)

map_score = ap

print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'AP: {ap}')
print(f'mAP: {map_score}')
```

```
Precision: [0.66666667 0.5        1.         1.         ]
Recall: [1.  0.5 0.5 0. ]
AP: 0.7916666666666666
mAP: 0.7916666666666666
```

Figure 17: Performance Evaluation

References

https://www.topcoder.com/thrive/articles/web-scraping-with-beautiful-soup

https://www.analyticsvidhya.com/blog/2021/06/vader-for-sentiment-analysis/

https://www.geeksforgeeks.org/introduction-of-lexical-analysis/

https://www.analyticsvidhya.com/blog/2021/09/an-explanatory-guide-to-bert-tokenizer/

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm

https://www.geeksforgeeks.org/introduction-convolution-neural-network/