Object Detection

Alan Casey and Alex Racapé

October 2023

1 Introduction

In this lab, we tested the Faster R-CNN and YOLO models for Object Detection. This is a central problem in computer vision that is concerned with not only discerning the class of an object, but also discerning where the objects are located in an image. The primary objective of our experiment was to both qualitatively and quantitatively evaluate the detection performance of the Faster R-CNN and YOLO models, which are two prominent neural network architectures for this task. Leveraging these popular neural network models, we conducted studies to evaluate their overall performance based on their object and boundary box classifications, as well as their total inference times.

2 Methodology

Our studies aimed to investigate the performance of the Faster R-CNN and YOLO models on a curated set of images and a larger testing dataset. We used the official PyTorch release of Faster R-CNN with a Resnet backbone and PyTorch hub's implementation of YOLOv5.

To initiate our study, we selected five images showcasing a variety of scenes and objects to assess potential variations between the two models. The images we chose can be found in the appendix.

There are a few pictures with cats and people as the subjects, and we chose an image of a shed with numerous objects. We also chose an image with some cats and cars mixed in. The 5 images contain a range of scene and object complexities, and we used them to make qualitative observations about the models and their different results. We ran all 5 images on both models, and plotted the images with the resulting bounding boxes and class predictions.

Afterwards, moving beyond qualitative observations, we shifted towards a quantitative analysis, using a kaggle data set with images of buses and trucks to test detection accuracy and inference time. The dataset includes images from everyday scenes with buses in trucks in the foreground and background. To simplify our analysis, we focused on a smaller random subset of the data. We randomly selected 50 images that included only one labeled bus or truck. We made this decision to simplify our code, so we would only need to check our predictions against a single ground truth. This analysis of both Faster R-CNN and YOLOv5 models provides us with some numeric data to compare with our previous qualitative judgements.

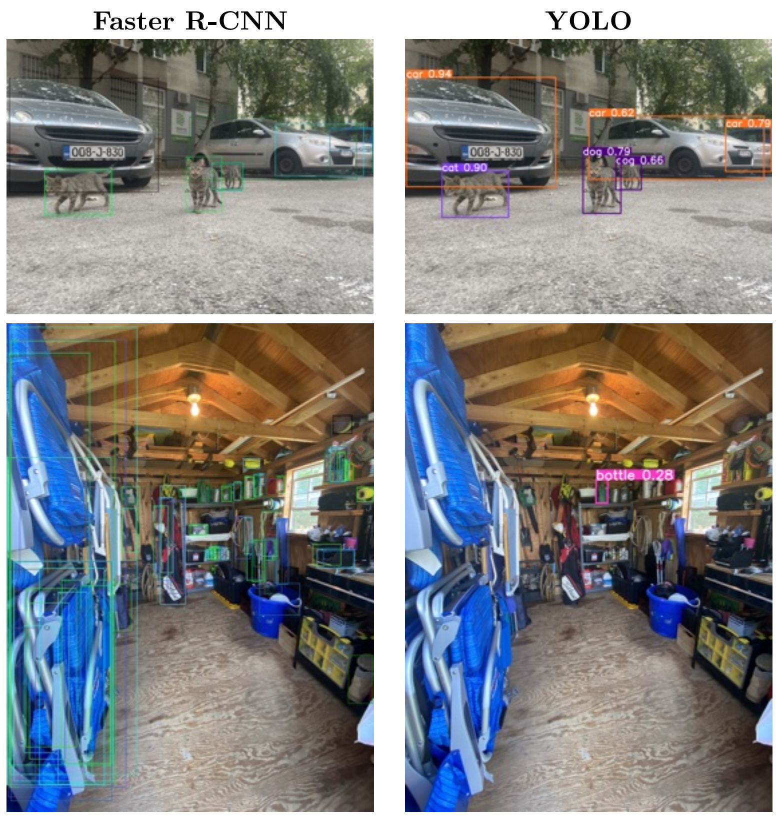
We measured the models' detection accuracy using the models' classification confidence levels and IoU (Intersection over Union). We established different threshold values for confidence levels (30% confidence) as well as for IoU (40%), which determined whether an inference was successful or not. We chose these thresholds based on observations from our first experiment where most correct observations were greater than 30% confidence, and detections below this were mostly wrong. In our analysis, we looked at the detections outputted by the model that were greater than our confidence threshold. Then we calculated the accuracy by looking at how many of these confident detections were adequately overlapping with a truck's true bounding box. A detection could be wrong if there was no bus in the scene or the bounding box was too far from our true bounding box (IoU below our threshold). This logic can be found in the Lines 189 and 276 of our code in the appendix. In order to calculate the IoU between two bounding boxes, we wrote a function called *compute-IoU* (Line 123) that computes the overlap between two bounding boxes—specifically the training box and the predicted box—to determine the accuracy of the object's bounding box. We measured the inference time by adding a timer to our code. After we calculated both the confidence levels and IoU for all images (using both models), as well as the inference time, we proceeded to compute the overall detection accuracy, and compared the results for both models in a following table. The quantitative analysis code can be found on line 148 for the Faster-RCNN model and 241 for the YoloV5 model.

3 Results and Discussion

3.1 Qualitative Observations

For our qualitative analysis, we observed the results of running 5 of our images through both models. We saw distinct characteristics for these two models. The Faster R-CNN model outputted many more proposed detections, and the model appeared to be more confident in its predictions, often producing predictions with higher confidence scores. One reason for this could be that the Faster R-CNN model has a lower confidence threshold for its predictions than YOLOv5, or maybe the region proposal network is able to find better proposals that lead to higher confidences. Faster R-CNN displayed a wider range of accurate and erroneous predictions. Moreover, we observed a considerable degree of overlapping boundary boxes in the results generated by Faster R-CNN.

This confidence from Faster R-CNN, however, did not necessarily correlate with the accuracy of the predictions. For example, in the image of Alex's shed, it was 99% sure that a sander was a clock. All of our results can be seen in the Appendix. In contrast, YOLOv5 exhibited a more reserved approach, generating fewer bounding boxes and predictions with lower confidence scores,



leading to fewer false positives. However, in one of the images with the cats lying on the clay roof it did not make any prediction at all, and it misclassified cats as dogs as illustrated in the figure.

Furthermore, it is noteworthy that both models better able to detect objects on the road, especially with respect to cars and pedestrians. These detection may be explained by the fact that both Faster R-CNN and YOLOv5 have been designed and optimized for tasks closely related to autonomous driving, making them particularly adept at recognizing objects typically encountered in such scenarios.

3.2 Quantitative Observations

Table 1: Model Performance Comparison

Model	Detection Accuracy	Inference Time
Faster R-CNN	75.0%	6m 39s
YOLO	87.0%	18s

In our experiment with bus images, the YOLO model outperformed faster r-cnn in both speed and detection accuracy. We expected YOLO to be faster because of its fully convolutional architecture, but we were surprised by the magnitude of this difference. One contributing factor could be that we had to look through more proposed detections from the faster r-cnn model when calculating detection accuracy. Additionally, there are some differences in the way data is loaded for each models' input, though each image is read in both cases. In terms of detection accuracy, YOLO performed better once again with an accuracy of 87%. For each model we displayed the images that were incorrectly detected, and they were understandably difficult. Both models struggled with images of unique looking busses, and both had instances where it labeled a truck as a bus. There were two particular images that led to failures in both models, and they included a bus-like car in the background and a large obstructing object with a bus peeking out from behind.

In some research online we found that faster r-cnn is considered by some to be slower but more accurate. Perhaps this is because of its convolutional method for proposing regions of interest that is less restrictive than the grid used by YOLO. This reasoning fits with our qualitative findings where YOLO seemed less accurate with less confident detections, but when calculating detection accuracy for the buses, faster r-cnn was actually less accurate. While it is unclear why, this could be due to the nature of the dataset since buses are generally large rectangular objects that might be better suited for YOLO.

4 Conclusion

In this study, we conducted a comprehensive evaluation of the Faster R-CNN and YOLO object detection models, both qualitatively and quantitatively. Object detection is a pivotal task in computer vision, as it involves not only recognizing object classes but also accurately localizing objects within images. Our qualitative analysis of five diverse images revealed distinct characteristics of these models. Faster R-CNN demonstrated a high prediction rate and a tendency to provide a lot of bounding boxes with high confidence scores. However, this confidence did not always translate to accurate predictions, leading to a considerable number of false positives and overlapping bounding boxes. In contrast, YOLOv5 exhibited a more cautious approach with fewer bounding boxes and lower confidence scores, resulting in a higher precision rate.

Our quantitative results revealed that YOLOv5 significantly outperformed Faster R-CNN in terms of both speed and detection accuracy. YOLOv5's superior speed can be primarily attributed to its architectural design, which avoids the computationally intensive fully connected layers used by Faster R-CNN. This advantage is particularly pronounced when dealing with a substantial number of proposed detections, highlighting YOLOv5's efficiency in processing images.

Overall, our study underscores the significance of both qualitative and quantitative assessments when selecting object detection models for specific applications. Furthermore, the results highlight the importance of considering the intended purpose and training strategies of these models in understanding their performance variations, which may vary depending on the specific requirements of the task at hand.

5 Appendix

```
# -*- coding: utf-8 -*-
"""lab5.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/11fMwQaswak4hwoPZji-eUxGjmZu5NhXf

> Indented block Yoooo
"""

import pandas as pd
import torch
import torch.nn as nn
from torch.optim import Adam
from torchsummary import summary
import numpy as np
import time
import os
import random
from google.colab import drive

# Get Faster R-CNN models
from torchvision.models.detection import fasterrcnn_resnet50_fpn, FasterRCNN_ResNet50_FPN_Weights

# Imports for Data
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import transforms, Resize, ToTensor
from torchvision.transforms.functional import to_pil_image, convert_image_dtype
from torchvision.io import read_image
from torchvision.utils import draw_bounding_boxes

# Set up drive storage and device
drive.mount('/content/drive')
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(device)

# Get models set up
classes = FasterRCNN_ResNet50_FPN_Weights.DEFAULT.meta["categories"]
fast_rcnn = fasterrcnn_resnet50_fpn(weights=FasterRCNN_ResNet50_FPN_Weights.DEFAULT)
fast_rcnn.eval()
```

```

yolo = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

# Experiment 1: Testing models on random images

# Images
imgs = ["alan.jpg", "cats2.jpg", "cats.jpg", "shed.jpg", "market.jpg"]
img_tensors = []
for path in imgs:
    image = read_image(path)
    image = image / 255.0  # Normalise the image to [0, 1]
    img_tensors.append(image)

# Inference
fast_rcnn_results = fast_rcnn(img_tensors)
yolo_results = yolo(imgs)

# Display Results with bounding boxes and captions
def show_bb(img, boxes, captions):
    img = (img * 255).to(torch.uint8)
    img = draw_bounding_boxes(img, boxes, captions, width=1)
    img = img.detach()
    img = to_pil_image(img)
    display(img)

# Display the results for each image from faster r-cnn
yolo_results.print()
for i, result in enumerate(fast_rcnn_results):

    # Get captions, labels, confidences
    confs = result["scores"]
    labels = result["labels"]
    captions = []
    for label, conf in zip(labels, confs):
        captions.append(f"{classes[label]} {(conf * 100).round()}")

    # Display
    boxes = result["boxes"]
    show_bb(img_tensors[i], boxes, captions)

# Display the YOLO results
yolo_results.show()

# Experiment 2: Testing models on bus dataset

# Get Data for R-CNN
class BusDataset(Dataset):
    def __init__(self, data_root):
        self.root = data_root
        self.image_root = data_root + "/images/images"
```

```python

        # Read CSV data from path
        data = pd.read_csv(data_root + "/subset.csv")

        # Set up x and y from CSV stuff
        x = data["ImageID"]
        y = data[["LabelName", "XMin", "XMax", "YMin", "YMax"]]
        self.x, self.y = x, y

    def __getitem__(self, ix):

        # Retrieve image using ID
        img_id = self.x.iloc[ix]
        path = self.image_root + f"/{img_id}.jpg"
        img = read_image(path)
        img = img / 255.0
        label = self.y.iloc[ix].values
        return img, label

    def __len__(self):
        return len(self.x)

test_dataset = BusDataset("drive/MyDrive/Data/bus_data")

def area_from_points(min_p, max_p):
    """Calculates the area of a square defined by its min and max point"""
    width = max_p[0] - min_p[0]
    height = max_p[1] - min_p[1]
    return width * height


def compute_IoU(box1, box2):
    """Computes Intersection over Union for two bounding boxes"""

    min1, max1 = (box1[0], box1[2]), (box1[1], box1[3])
    min2, max2 = (box2[0], box2[2]), (box2[1], box2[3])

    i_min = max(min1, min2)
    i_max = min(max1, max2)
    if i_min[0] > i_max[0] or i_min[1] > i_max[1]:
        return 0
    intersection = area_from_points(i_min, i_max)

    area1 = area_from_points(min1, max1)
    area2 = area_from_points(min2, max2)
    union = area1 + area2 - intersection
    return intersection / union

def convert_boxes(boxes):
    """Converts bounding boxes from xxyy to xyxy format"""
```

```python
    new_boxes = []
    for box in boxes:
        new_box = [box[0], box[2], box[1], box[3]]
        new_boxes.append(new_box)
    return new_boxes

def evaluate_fast_rcnn(model, dataset):
    correct = 0
    total = 0
    confidence_thresh = .3
    iou_thresh = .4

    # Set the model to evaluation mode
    model.eval()

    # Disable gradient calculation
    start = time.time()
    for i in range(50):
        print(f"STARTING IMAGE {i}...")

        # Get image, label, and prediction
        img, label = dataset[i]
        prediction = model([img])[0]
        true_class = label[0]
        true_box = label[1:5]
        height, width = img.size()[1], img.size()[2]

        # Find which predicted boxes are buses, with conf > thresh
        bus_indexes = []
        pred_classes = [classes[label] for label in prediction["labels"]]
        for j in range(len(pred_classes)):
            if pred_classes[j] == "bus" and prediction["scores"][j] > confidence_thresh:
                bus_indexes.append(j)

        # Adjust predicted bounding box to size of image
        true_box = [val * width if idx < 2 else val * height for idx, val in enumerate(true_box)]

        # For each confident detection, check if it was correct
        for bus_index in bus_indexes:

            # Get our predicted bounding box for a bus
            predict_box = prediction["boxes"][bus_index].tolist()
            predict_box = convert_boxes([predict_box])[0]

            # Compute predicted IoU with true bounding box
            print(true_box, predict_box)
            iou = compute_IoU(true_box, predict_box)
            if iou > iou_thresh and true_class == "Bus":
                correct += 1

```

```python
            # Display our wrong guess, if there is no bus
            elif true_class != "Bus":
                boxes = convert_boxes([predict_box])
                show_bb(img, torch.tensor(boxes), ["Predicted Bus"])

            # Display the bad guess, if we are confident and missed
            elif true_class == "Bus" and iou < iou_thresh:
                print([true_box, predict_box])
                boxes = convert_boxes([true_box, predict_box])
                print(boxes)
                show_bb(img, torch.tensor(boxes), ["True Bus", "Predicted Bus"])

            print(f"Predicted Box: {predict_box}, IoU: {iou}")
            total += 1

    # Calculate accuracy
    accuracy = (correct / total) * 100.0
    testing_time = time.time() - start
    return accuracy, testing_time

# Run through our tests
evaluate_fast_rcnn(fast_rcnn, test_dataset)

class YoloDataset(Dataset):
    def __init__(self, data_root):
        self.root = data_root
        self.image_root = data_root + "/images/images"

        # Read CSV data from path
        data = pd.read_csv(data_root + "/subset.csv")

        # Set up x and y from CSV stuff
        x = data["ImageID"]
        y = data[["LabelName", "XMin", "XMax", "YMin", "YMax"]]
        self.x, self.y = x, y

    def __getitem__(self, ix):

        # Retrieve image using ID
        img_id = self.x.iloc[ix]
        path = self.image_root + f"/{img_id}.jpg"
        img_size = read_image(path).size()
        label = self.y.iloc[ix].values
        return path, (img_size[1], img_size[2]), label

    def __len__(self):
        return len(self.x)


def evaluate_yoloV5(model, dataset):
```

```
accurate_predictions = 0
total_bus_predictions = 0
confidence_thresh = 0.3
iou_thresh = 0.4

start = time.time()
for i in range(50):

    img, size, true_label = dataset[i]
    height, width = size

    # Get the true class and true box dimensions
    true_class = true_label[0]
    true_box = true_label[1:5]
    # Adjust predicted bounding box to size of image
    true_box = [val * width if idx < 2 else val * height for idx, val in enumerate(true_box)]

    # Compute and store the results for the yolo model
    results = model(img)
    label_table = results.pandas().xyxy[0]

    """
        Count the total number of true predictions
        A true prediction consists of:
            1) A confident "bus" when the image contains a bus, with
            2) An accurate bounding box
    """
    for index, row in label_table.iterrows():
        # Only loop through the predictions that are a confident "bus"
        if row["name"] == "bus" and row["confidence"] > confidence_thresh:
            total_bus_predictions += 1
            pred_box = [row["xmin"], row["xmax"], row["ymin"], row["ymax"]]
            iou = compute_IoU(pred_box, true_box)

            if iou > iou_thresh and true_class == "Bus":
                # Increase accuracy if IoU passes our threshold
                accurate_predictions += 1
            elif true_class != "Bus":
                # Display the incorrect bus prediction box
                draw_box = convert_boxes([pred_box])
                img = read_image(img)
                show_bb(img, torch.tensor(draw_box), ["Predicted Bus"])
            else:
                # Display the true bus bounding box and our incorrect bus prediction box
                draw_boxes = convert_boxes([true_box, pred_box])
                img = read_image(img)
                show_bb(img, torch.tensor(draw_boxes), ["True Bus", "Predicted Bus"])
```

```python

    accuracy = (accurate_predictions / total_bus_predictions) * 100.0
    testing_time = time.time() - start
    return accuracy, testing_time

    # What the results format should look like:
    #      xmin    ymin    xmax   ymax  confidence  class    name
    # 0  749.50   43.50  1148.0  704.5    0.874023      0  person
    # 2  114.75  195.75  1095.0  708.0    0.624512      0  person
    # 3  986.00  304.00  1028.0  420.0    0.286865     27     tie

yolo_dataset = YoloDataset("drive/MyDrive/Data/bus_data")
evaluate_yoloV5(yolo, yolo_dataset)

# Create a list of images we were able to download
downloaded_files = os.listdir("drive/MyDrive/Data/bus_data/images/images")

# Create a new DataFrame with only the downloaded images (12,308)
original_df = pd.read_csv("drive/MyDrive/Data/bus_data/df.csv")
original_df['ImageID'] = original_df['ImageID'] + '.jpg'  # Add file extension so we can compare
downloaded_df = original_df[original_df['ImageID'].isin(downloaded_files)]
downloaded_df['ImageID'] = downloaded_df['ImageID'].str.rstrip('.jpg')
downloaded_df = downloaded_df.drop_duplicates(subset=['ImageID'], keep=False)

# Filter down to a subset
size_of_subset = 50
print(downloaded_df["ImageID"].nunique())
subset = random.sample(downloaded_df['ImageID'].tolist(), size_of_subset)
subset_df = downloaded_df[downloaded_df["ImageID"].isin(subset)]
print(subset_df)

# Download to drive
downloaded_df.to_csv('drive/MyDrive/Data/bus_data/subset.csv', index=False)
```

Faster R-CNN | **YOLOd**