Alexandria University

Faculty of Engineering

Computer and Systems Engineering Department

Computer Vision Assignment 4 Report

Team Members:
Abdelrahman Elsayed Gad (ID: 19015894)
Abdelaziz Mohamed Abdelaziz (ID: 19015941)
Omar Khairat Mohamed Abodeif (ID: 19016063)

Teaching Assistant: Eng. Shereen Elkordi

Professor: Dr. Marwan Torki

Assigned: Sat, Dec 16, 2023
Due: Sat, Dec 30, 2023

Contents

1 Introduction

This computer vision assignment explores the capabilities of three object detection models on two datasets: COCO and Open Images. After downloading both datasets, we conducted inference on their respective testing sets and evaluated the models using IOU and mAP. Our analysis includes a comparison of the models' architectures, inference results, and an exploration of their strengths and weaknesses. We also identified success and failure cases, providing insights into model performance. The report concludes with a comprehensive comparison, shedding light on key observations and lessons learned.

2 Methodology

2.1 Download Dataset

To acquire the COCO dataset for our experiment, we utilized the FiftyOne library, an open-source Python library designed for computer vision practitioners and researchers. FiftyOne provides a convenient API for working with diverse datasets, facilitating the loading and exploration of datasets for computer vision tasks.
We employed the following Python code snippet to download the COCO dataset using FiftyOne:

```
import fiftyone as fo
import fiftyone.zoo as foz

dataset = foz.load_zoo_dataset(
    "coco-2017",
    split="validation",
    dataset_name="evaluate-detections-tutorial",
)
dataset.persistent = True
```

This code leverages the `foz.load_zoo_dataset` function from the FiftyOne library to load the COCO dataset, specifically the validation split for our experiment. The `persistent` attribute is set to true to ensure that the dataset remains available even after the Python script terminates.
Using FiftyOne for dataset loading provides a streamlined and efficient approach, allowing us to focus on the core aspects of our assignment, such as running object detection models and evaluating their performance on the COCO dataset.

2.2 Object Detection Models

In this stage, we employed three distinct object detection models to perform inference on the COCO validation set and Open images dataset. Each model

offers unique architectural characteristics, influencing its performance and applicability.

2.2.1 YOLOv8n

We utilized the YOLOv8n model for object detection, loading the pretrained weights from the "yolov8n.pt" file. YOLO (You Only Look Once) is a real-time object detection system that divides an image into a grid and predicts bounding boxes and class probabilities directly. YOLOv8n, as an evolution of the YOLO architecture, enhances the model's accuracy and speed. The YOLO architecture operates by processing the entire image in a single forward pass, making it well-suited for real-time applications.Comparatively low recall and more localization error compared to Faster RCNN. Struggles to detect close objects because each grid can propose only 2 bounding boxes. Struggles to detect small objects.

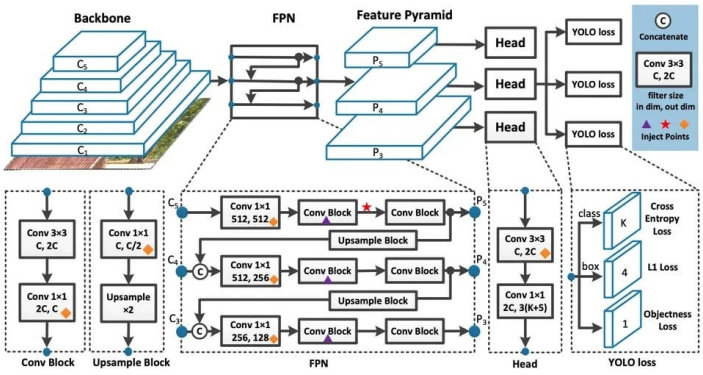

Figure 1: Architecture of YOLOv8n.a

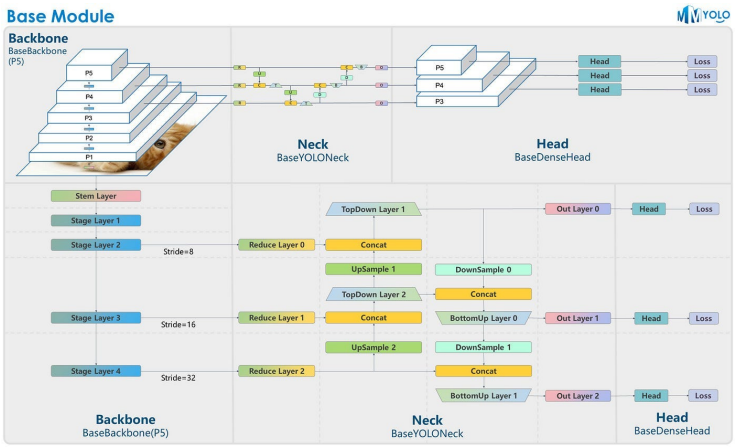

Figure 2: Architecture of YOLOv8n.b

2.2.2 Faster RCNN with ResNet50 fpn

The Faster R-CNN is an improvement on the Fast R-CNN and the R-CNN. It uses the ResNet feature extractor. The R-CNN extracts region proposals from the image instead of trying every sliding window and inputs the warped region proposals into a CNN which outputs a classification. The Fast R-CNN improves efficiency by eliminating recalculating the features of the overlapping regions. It does so by inputting the entire image into the CNN and then cropping the feature map corresponding to the desired region proposal + ROI pooling.The Faster R-CNN further improves the speed by doing the region proposal extraction as part of the network architecture.

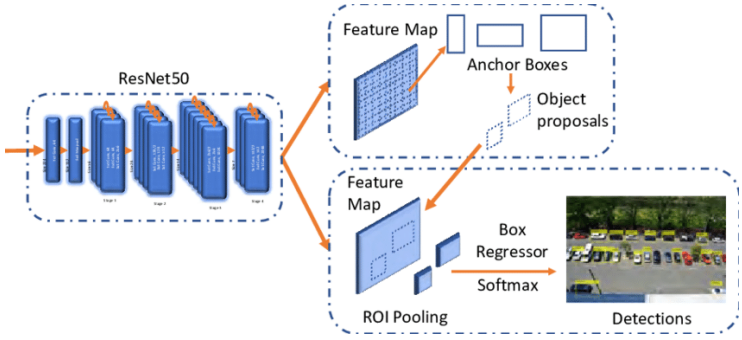

Figure 3: Architecture of Faster RCNN with ResNet50 fpn

In R-CNN and Fast R-CNN, there was a region proposal stage which was implemented by selective search. The problem was that this step took a lot of time (up to 2 seconds per image) and it is not easily parallelizable. The

contribution of Faster R-CNN is to add this selective search to the detection pipeline by introducing region proposal network (RPN). RPN takes as an input the output of the feature map of the backbone and slides a window of size (n× n) over the image. Each window is mapped to a lower dimensional embedding encoding k region proposals each consists of 5 numbers: (x, y, w, h) bounding box information, and p objectness score. The regions then are fed into the network for doing object classification and bounding box regression.

2.2.3 SSD with MobileNetV3 Large

We employed SSD with MobileNetV3 Large for object detection, a single-shot multibox detector (SSD) that integrates a MobileNetV3 Large backbone. SSD is known for its efficiency in object detection, providing predictions at multiple scales through a set of predefined anchor boxes. This architecture allows for real-time object detection across different object scales and aspect ratios. MobileNetV3 Large is a lightweight and efficient convolutional neural network architecture, making it suitable for deployment on resource-constrained devices.

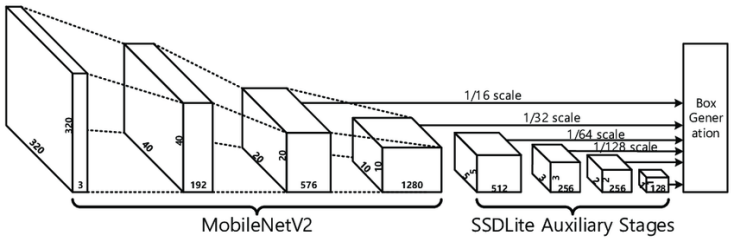

Figure 4: Architecture of SSD with MobileNetV3 Large

In the SSD architecture, the backbone, MobileNetV3 Large in this case, extracts features at multiple scales. The feature maps are then used to predict bounding boxes and class scores at each scale through a set of default boxes or anchor boxes. This allows SSD to efficiently handle objects of different sizes. The MobileNetV3 Large backbone ensures a good balance between model accuracy and computational efficiency, making it suitable for real-time applications.

These three models were chosen to demonstrate diversity in architecture, encompassing both single-stage (YOLO) and two-stage (Fast R-CNN and SSD) object detection paradigms. The nuanced differences in architecture influence their performance characteristics, making them suitable for various applications and scenarios.

2.3 Evaluate Models

In this section, we present the evaluation methods for the three object detection models: YOLOv8n, Faster RCNN with ResNet50 fpn, and SSD with MobileNetV3 Large.

2.3.1 Intersection over Union (IoU) Calculation

The Intersection over Union (IoU) is a crucial metric for evaluating the performance of object detection models. It measures the accuracy of the bounding box predictions by calculating the overlap between predicted and ground truth bounding boxes.

The IoU for a single object is calculated using the following formula:

$$IoU_{\text{object}} = \frac{\text{area of intersection between predicted and ground truth bounding boxes}}{\text{area of union between predicted and ground truth bounding boxes}}$$

For a given image, if there are multiple detected objects, the IoU for each object is computed, and the IoU for the entire image is then calculated as the average IoU across all detected objects:

$$IoU_{\text{photo}} = \frac{\sum IoU_{\text{objects}}}{\text{number of detected objects}}$$

Finally, the overall IoU for the entire dataset is computed by averaging the IoU scores across all images:

$$IoU = \frac{\sum IoU_{\text{photos}}}{\text{number of images}}$$

In the following sections, we delve deeper into the analysis of success and failure cases, further understanding the nuances of each model's object detection capabilities.

2.3.2 MaP

We use the built-in function

```
result.mAP()
```

3 Results

3.1 Inference Results

3.1.1 YOLOv8n Results

The YOLOv8n model was evaluated on the COCO testing set, yielding the following results:

- Mean Average Precision (mAP): 0.3132
- IoU Score: 0.3575
- Evaluation Counts:
  - True Positives (tp): 18430
  - False Positives (fp): 7183
  - False Negatives (fn): 18351

These metrics provide insights into the model's performance in terms of precision, accuracy, and the balance between true positives, false positives, and false negatives.

And the YOLOv8n model was evaluated on the Open Images dataset, yielding the following results:

- Mean Average Precision (mAP): 0.0
- IoU Score: 0.0
- Evaluation Counts:
  - False Positives (fp): 3887
  - False Negatives (fn): 10052

These metrics provide insights into the model's performance in terms of precision, accuracy, and the balance between true positives, false positives, and false negatives.

3.1.2 Faster RCNN with ResNet50 fpn Results

The Faster RCNN model with ResNet50 fpn was evaluated on the COCO testing set, resulting in the following classification report:

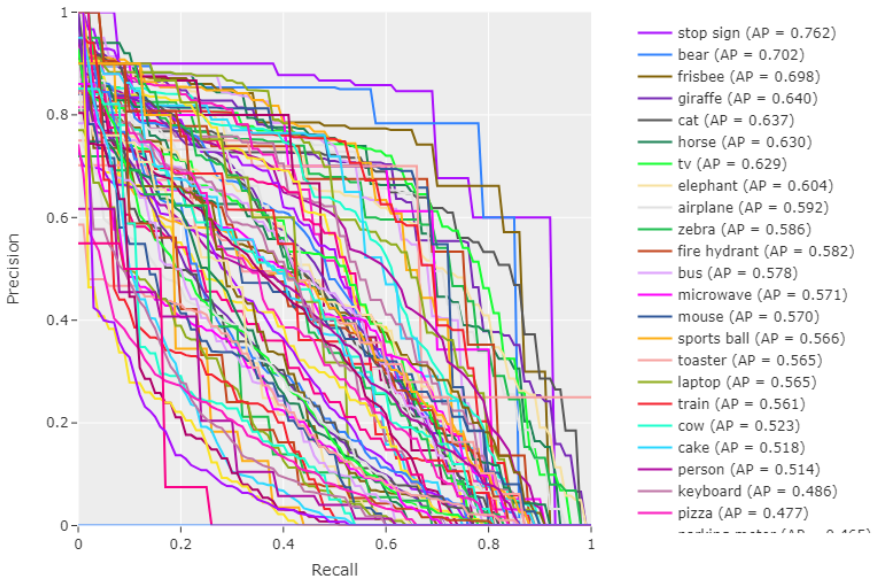

Figure 5: Precision and Recall per Class for Faster RCNN on COCO dataset

Additionally, the model achieved the following overall metrics:

- Mean Average Precision (mAP): 0.3802
- IoU Score: 0.0232
- Accuracy: 0.2403
- Precision: 0.2501
- Recall: 0.8602
- F1-Score: 0.3875

And the Faster RCNN model with ResNet50 fpn was evaluated on the Open Images dataset, resulting in the following classification report:

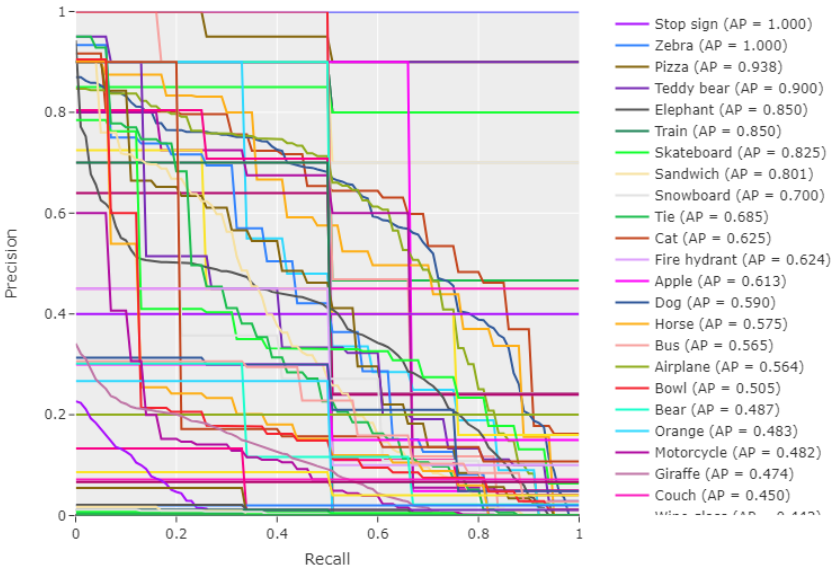

Figure 6: Precision and Recall per Class for Faster RCNN on Open Images dataset

Additionally, the model achieved the following overall metrics:

- Mean Average Precision (mAP): 0.0868
- IoU Score: 0.0350
- Accuracy: 0.0471
- Precision: 0.0614
- Recall: 0.1676
- F1-Score: 0.0899

3.1.3 SSD with MobileNetV3 Large Results

The SSD with MobileNetV3 Large model underwent evaluation on the COCO testing set, yielding detailed results and insights into its object detection capabilities:

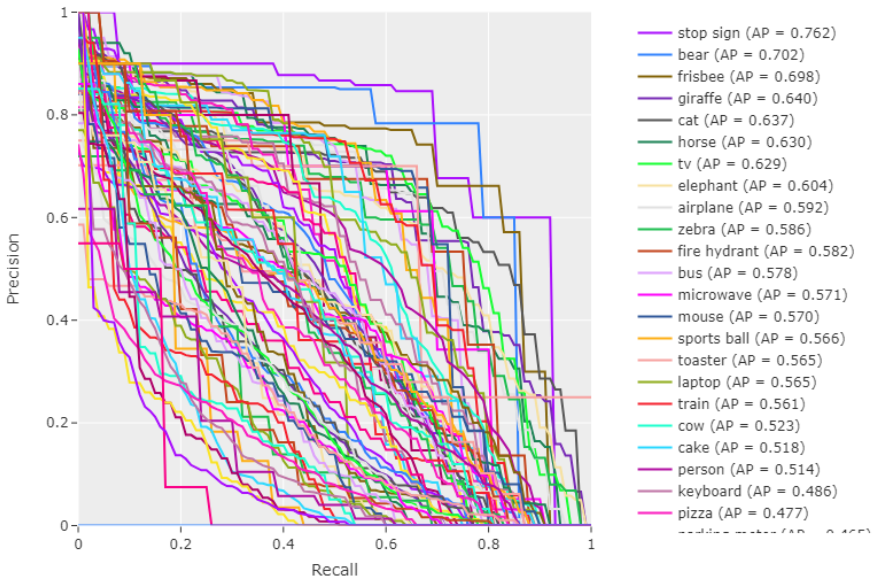

Figure 7: Precision and Recall per Class for SSD with MobileNetV3 Large on COCO dataset

The model's performance is summarized with the following metrics:

- Mean Average Precision (mAP): 0.2152
- IoU Score: 0.0021
- Accuracy: 0.0345
- Precision: 0.035
- Recall: 0.7748
- F1-Score: 0.0667

The SSD with MobileNetV3 Large model underwent evaluation on the Open Images dataset, yielding detailed results and insights into its object detection capabilities:

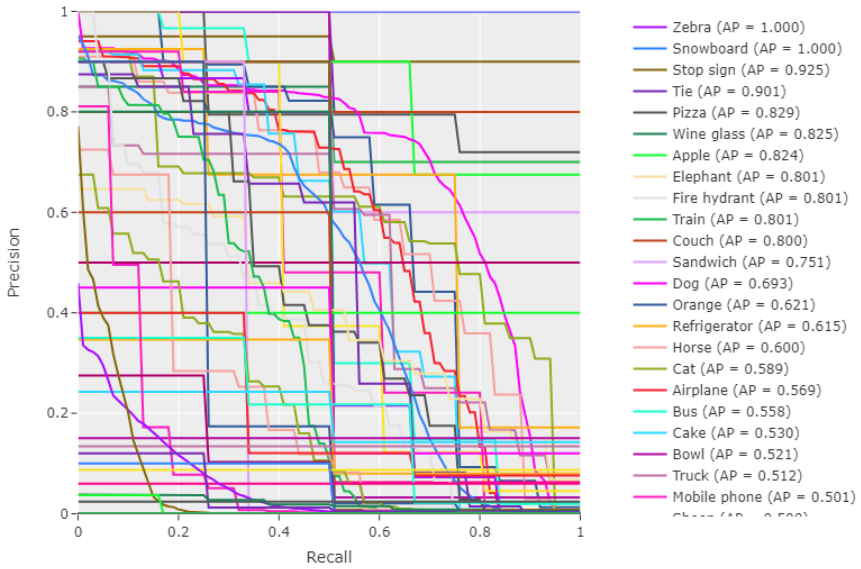

Figure 8: Precision and Recall per Class for SSD with MobileNetV3 Large on Open Images

The model's performance is summarized with the following metrics:

- Mean Average Precision (mAP): 0.0953
- IoU Score: 0.022
- Accuracy: 0.0054
- Precision: 0.0055
- Recall: 0.1532
- F1-Score: 0.0107

3.2 Failure and Success Cases

During the evaluation of our object detection models on COCO dataset, we encountered various instances where the models either succeeded or failed to accurately detect objects. In this section, we present a selection of both failure and success cases, providing insights into the strengths and weaknesses of the models.

3.2.1 Success Cases

Figure 9: Successful Detection Example for YOLOv8n

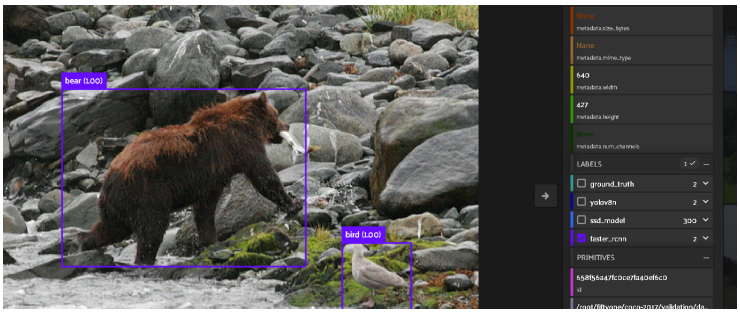

Figure 10: Successful Detection Example for Faster RCNN

3.2.2 Failure Cases

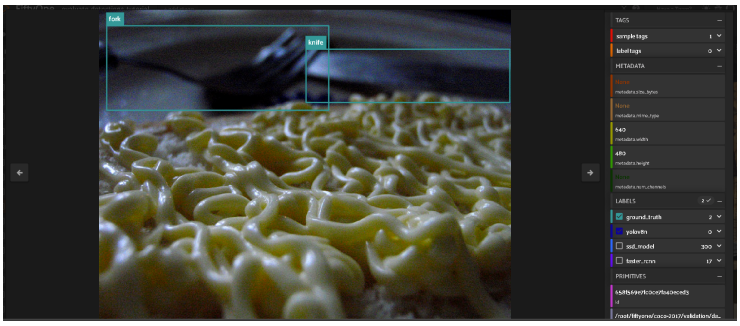

Figure 11: Failure Detection Example for YOLOv8n

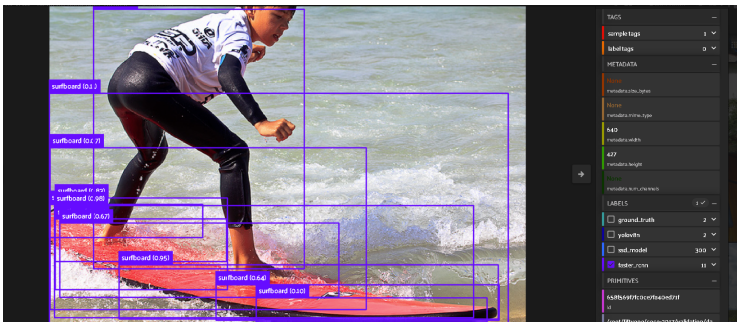

Figure 12: Failure Detection Example for Faster RCNN

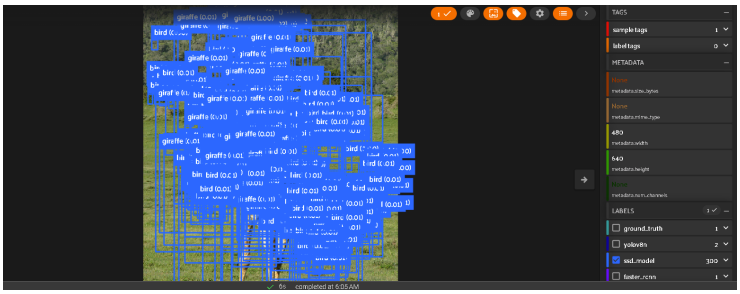

Figure 13: Failure Detection Example for SSD

These examples provide a glimpse into the performance variations of our object detection models across different scenarios, shedding light on areas for improvement and showcasing their respective capabilities.

3.3 Comparison of Models

YOLOv8 (You Only Look One-level), SSD MobileNet, and Faster R-CNN with ResNet are prominent object detection models extensively used in computer vision tasks. Each model possesses distinct strengths and weaknesses, and the choice between them depends on various factors such as accuracy, speed, and resource constraints. Here's a brief comparison:

3.3.1 YOLOv8 (You Only Look One-level)

Advantages:

- Fast inference speed: YOLO models are known for their real-time object detection capabilities.
- Simplicity: YOLO is conceptually simple, with a single neural network predicting bounding boxes and class probabilities directly.

- Good for general-purpose object detection.

Disadvantages:

- May sacrifice some accuracy compared to other models.
- Prone to false positives and false negatives, especially for small objects.
- May not perform as well on complex scenes.

3.3.2 SSD MobileNet (Single Shot Multibox Detector with MobileNet as backbone)

Advantages:

- Better accuracy compared to YOLO, especially for small objects.
- Allows for the detection of objects at multiple scales.
- MobileNet backbone enables a good balance between speed and accuracy on resource-constrained devices.

Disadvantages:

- Slower inference speed compared to YOLO.
- More complex architecture compared to YOLO.

3.3.3 Faster R-CNN with ResNet

Advantages:

- State-of-the-art accuracy: Faster R-CNN is known for its high accuracy in object detection tasks.
- Region Proposal Network (RPN) separates region proposal generation from the detection network.

Disadvantages:

- Slower inference speed compared to YOLO and SSD due to the two-stage detection process.
- More complex architecture, requiring more computational resources.

In summary, If real-time processing and simplicity are crucial, YOLOv8 might be a good choice. If accuracy is the primary concern and real-time constraints are less critical, Faster R-CNN with ResNet is a strong option. If there's a need for a good balance between speed and accuracy, especially on resource-constrained devices, SSD MobileNet could be a suitable choice. Ultimately, the choice depends on the specific requirements of the application, including the available hardware, the need for real-time processing, and the desired level of accuracy. It's common to experiment with multiple models to find the one that best fits the specific use case.

4 Conclusion

In conclusion, our exploration of object detection models on the COCO and Open Images datasets provided valuable insights into their performance characteristics. The YOLOv8n, Faster RCNN with ResNet50 fpn, and SSD with MobileNetV3 Large models were assessed in terms of accuracy, speed, and generalization capabilities.

Each model exhibited strengths and weaknesses, making them suitable for specific use cases. YOLOv8n demonstrated fast inference speed but with some trade-offs in accuracy, especially for small objects. Faster RCNN with ResNet50 fpn excelled in accuracy but at the cost of slower inference speed. SSD with MobileNetV3 Large offered a good balance between speed and accuracy, making it suitable for resource-constrained devices.

The evaluation of success and failure cases highlighted the models' performance variations in different scenarios. While success cases showcased the models' ability to accurately detect objects, failure cases revealed areas for improvement, such as challenges in detecting small objects or complex scenes.

In summary, the choice of an object detection model depends on specific application requirements, including real-time constraints, accuracy needs, and hardware considerations. By experimenting with diverse models, we gained a comprehensive understanding of their capabilities and limitations, providing a foundation for informed model selection in future computer vision tasks.

5 Bonus

5.1 Different layers feature maps

5.1.1 YOLO

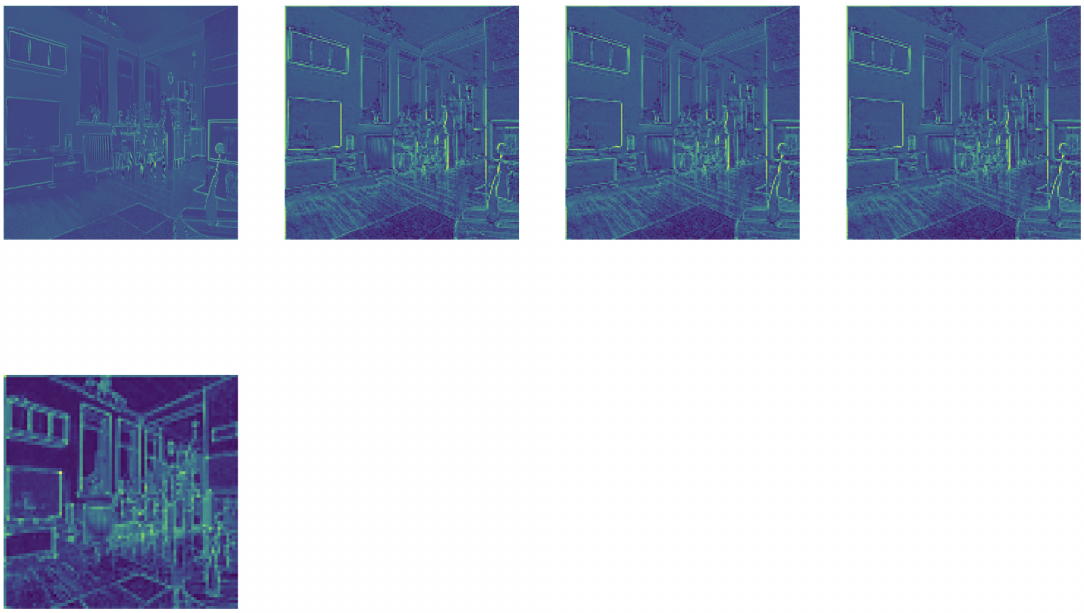

Figure 14: Feature Maps for YOLO

Those feature maps are extracted from the below image:

```
(model): Sequential(
  (0): Conv(
    (conv): Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (act): SiLU(inplace=True)
  )
  (1): Conv(
    (conv): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (act): SiLU(inplace=True)
  )
  (2): C2f(
    (cv1): Conv(
      (conv): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1))
      (act): SiLU(inplace=True)
    )
    (cv2): Conv(
      (conv): Conv2d(48, 32, kernel_size=(1, 1), stride=(1, 1))
      (act): SiLU(inplace=True)
    )
    (m): ModuleList(
      (0): Bottleneck(
        (cv1): Conv(
          (conv): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (act): SiLU(inplace=True)
        )
        (cv2): Conv(
          (conv): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          (act): SiLU(inplace=True)
        )
      )
    )
  )
  (3): Conv(
    (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
```

Figure 15: Feature Maps layers from YOLO model

5.1.2 Faster RCNN

Feature maps are extracted from the ResNet backbone

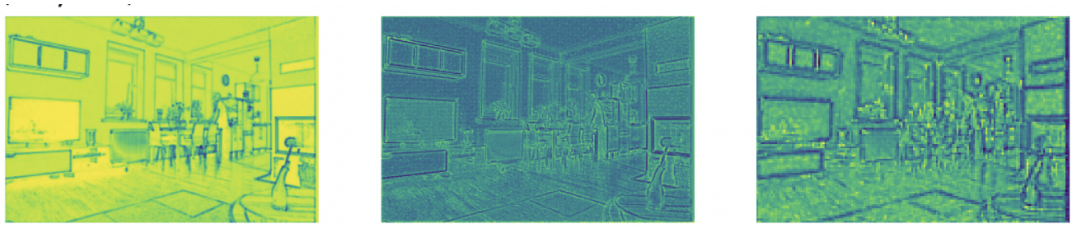

Figure 16: Feature Maps for faster RCNN

5.1.3 SSD

Figure 17: Feature Maps for SSD model

Those feature maps are extracted from the below image:

```
(backbone): SSDLiteFeatureExtractorMobileNet(
  (features): Sequential(
    (0): Sequential(
      (0): Conv2dNormActivation(
        (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(16, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
        (2): Hardswish()
      )
      (1): InvertedResidual(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=16, bias=False)
            (1): BatchNorm2d(16, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
            (2): ReLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(16, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(16, eps=0.001, momentum=0.03, affine=True, track_running_stats=True)
          )
        )
      )
```

Figure 18: Feature Maps layers for SSD model

6 Google Colab Notebooks

We have provided Google Colab notebooks for the various stages of our experiment. These notebooks contain the code implementation, model training, evaluation, and results visualization. Feel free to explore and reproduce our experiments using the following links:

1. **For coco dataset:** Click here
2. **For OpenImages dataset:** Click here
3. **For feature maps for different layers:** Click here

7 References

1. FiftyOne Documentation. (n.d.). Voxel51. Retrieved from `https://docs.voxel51.com/`
2. COCO Dataset. (n.d.). Common Objects in Context. Retrieved from `https://cocodataset.org/#download`
3. Open Images Dataset. (n.d.). Facts and Figures. Retrieved from `https://storage.googleapis.com/openimages/web/factsfigures_v7.html`
4. FiftyOne Documentation - COCO Integration. (n.d.). Voxel51. Retrieved from `https://docs.voxel51.com/integrations/coco.html`
5. FiftyOne Documentation - Open Images Integration. (n.d.). Voxel51. Retrieved from `https://docs.voxel51.com/integrations/open_images.html`
6. FiftyOne Tutorials - Evaluate Detections. (n.d.). Voxel51. Retrieved from `https://docs.voxel51.com/tutorials/evaluate_detections.html`