# Final Project Report: Object Detection with Deep Learning

Min-Gyo Jeong, G#01402966
George Mason University
Fairfax, VA, USA
mjeong6@gmu.edu

John Stephen Gutam, G#01413212
George Mason University
Fairfax, VA, USA
jgutam@gmu.edu

## Abstract

Problem: In this project, we solved the detection and localizing of objects accurately in images and videos. It is a crucial task for various applications like autonomous driving, surveillance, and robotics. Approach: This project explores state-of-the-art deep learning techniques like YOLOv5, Faster R-CNN, and SSD for real-time object detection. Key Results: Achieved high accuracy on standard datasets like COCO, with real-time detection speeds on GPU, showcasing the effectiveness of these modern methods.

## Keywords

Yolov5, Faster R-CNN, SSD, Object Detection, segmentation, CNN, Neural Network

## 1 Architecture Overview:

The broad problem addressed here is the evaluation of three different object detection models - YOLOv5, Faster R-CNN, and SSD (Single Shot Multi box Detector) - using the COCO dataset's validation set. Object detection is a fundamental task in computer vision with applications in various domains such as autonomous driving, surveillance, and image understanding. Evaluating these models on a standardized dataset like COCO helps assess their accuracy, speed, and robustness, which are crucial for real-world deployment.

### 1.1 YOLOv5:

YOLOv5 (v6.0/6.1) is a powerful object detection algorithm developed by Ultralytics. This article dives deep into the YOLOv5 architecture, data augmentation strategies, training methodologies, and loss computation techniques. This comprehensive understanding will help improve your practical application of object detection in various fields, including surveillance, autonomous vehicles, and image recognition.YOLOv5's architecture consists of three main parts:

Backbone: This is the main body of the network. For YOLOv5, the backbone is designed using the New CSP-Darknet53 structure, a modification of the Darknet architecture used in previous versions. Neck: This part connects the backbone and the head. In YOLOv5, SPPF and New CSP-PAN structures are utilized. Head: This part is responsible for generating the final output. YOLOv5 uses the YOLOv3 Head for this purpose. The structure of the model is depicted in the image below.

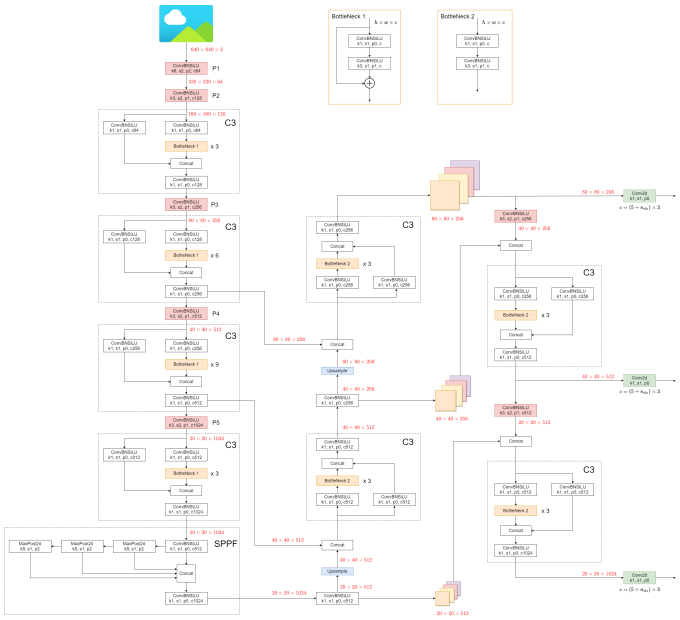


**Figure 1: The YOLOv5 Architecture**

### 1.2 Faster R-CNN:

Faster R-CNN is an object detection model that improves on Fast R-CNN by utilising a region proposal network (RPN) with the CNN model. The RPN shares full-image convolutional features with the detection network, enabling nearly cost-free region proposals. It is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. RPN and Fast R-CNN are merged into a single network by sharing their convolutional features: the RPN component tells the unified network where to look.

As a whole, Faster R-CNN consists of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions.

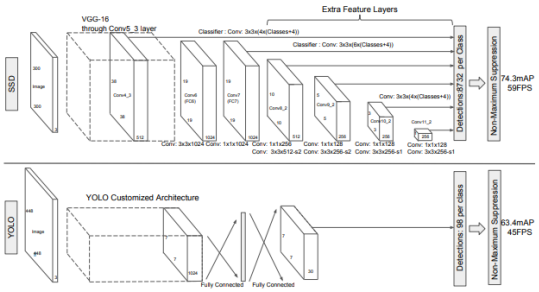


**Figure 2: SSD vs Yolo Architectures**

### 1.3 SSD:

SSD is a single-stage object detection method that discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes.

The fundamental improvement in speed comes from eliminating bounding box proposals and the subsequent pixel or feature resampling stage. Improvements over competing single-stage methods include using a small convolutional filter to predict object categories and offsets in bounding box locations, using separate predictors (filters) for different aspect ratio detections, and applying these filters to multiple feature maps from the later stages of a network in order to perform detection at multiple scales.

## 2 Approach

Collect a dataset of images with bounding box annotations for the object's interest. Split the dataset into training, validation, and test sets. Preprocess the images and encode the bounding box annotations in a suitable format.Choose a deep learning model architecture suitable for object detection, such as YOLO (You Only Look Once), Faster R-CNN, or SSD (Single Shot MultiBox Detector). These models typically consist of a backbone convolutional neural network (CNN) for feature extraction and additional components for object localization and classification.nitialize the backbone CNN with pre-trained weights from models trained on large datasets like ImageNet, which provides a good starting point for feature extraction. Fine-tune the entire model or just the additional components on the object detection dataset.Define a loss function that combines components for localization (bounding box regression) and classification (object category prediction). Implement data augmentation techniques (e.g., random cropping, flipping, scaling) to improve model generalization. Train the model using an optimization algorithm like Stochastic Gradient Descent (SGD) or Adam, monitoring the loss on the validation set to prevent overfitting.Evaluate the trained model's performance on the test set using metrics like mean Average Precision (mAP) or Intersection over Union (IoU). Visualize the model's predictions on sample images to identify potential issues. Fine-tune hyperparameters (e.g., learning rate, batch size, augmentation settings) to improve performance.Implement Non-Maximum Suppression (NMS) to filter out redundant and overlapping bounding box predictions, keeping only the most confident ones.Optimize the model for deployment on the target platform (e.g., CPU, GPU, edge devices) by techniques like quantization, pruning, or model distillation. Integrate the object detection model into the larger application or system, handling pre-processing and post-processing steps.Monitor the model's performance in the real-world deployment setting. Collect and annotate new data to fine-tune or retrain the model to improve performance on edge cases or new object categories. Stay updated with the latest research and techniques in object detection for potential improvements or new model architectures. Throughout the process, it's essential to leverage existing deep learning frameworks (e.g., PyTorch, TensorFlow) and open-source implementations of object detection models, which provide efficient and well-optimized code. Additionally, parallel computation using GPUs (or TPUs) is often necessary for training and inference due to the computational demands of these models. The specific implementation details, such as network architectures, loss functions, and optimization strategies, may vary depending on the chosen object detection model and the requirements of the application. However, the general approach outlined above provides a high-level overview of the steps involved in coding and solving object detection problems using deep learning techniques.

## 3 Results

### 3.1 COCO Validation Set Evaluation Setup

Three different image transformation pipelines are defined for each model: YOLOv5, Faster R-CNN, and SSD. Each transformation includes resizing the image to a specific size, converting it to a tensor, and normalizing pixel values. The get model detections function retrieves detections from each model for a batch of images. It iterates through the images, obtains detections, and scales bounding boxes to the original image size. For YOLOv5, the detections are directly obtained from the model's outputs, while for Faster R-CNN and SSD, the images are processed through the model to obtain detections.

## 3.2 Data Sets:

The evaluation is performed on the COCO dataset using the validation set. The COCO dataset is a widely used benchmark for object detection tasks, containing a large number of images with annotated bounding boxes for multiple object categories.

## 3.3 Simulator:

The evaluation is conducted using three different object detection models: Faster R-CNN, SSD, and YOLOv5. These models are evaluated on the COCO dataset to assess their performance in terms of average precision (AP) and average recall (AR) across different intersections over union (IoU) thresholds and object sizes.

## 3.4 Implementation Details:

Each model is tested using the COCO evaluation toolkit, which involves loading annotations into memory, creating indexes, preparing results, and running per-image evaluation. The evaluation script calculates various metrics, including AP and AR, at different IoU thresholds and object sizes. The evaluation is performed for each model using the validation set of the COCO dataset, and the results are accumulated to provide an overall performance assessment.

*3.4.1 Check ground truth* The `fetch_ground_annotations` function retrieves ground truth annotations for a specific image from the COCO dataset using the COCO API. It takes the image ID and annotation file path as input, retrieves the annotation IDs corresponding to the image, and loads the annotations. The `fetch_image_by_index` function retrieves an image by its index from the validation image directory. Ground truth annotations and detection results are plotted on the image using different colors for each object category. The plot includes ground truth annotations and detections from YOLOv5, Faster R-CNN, and SSD models. It parses the image ID from the image file name and returns the image, file name, and ID.

*3.4.2 Check ground truth* The ground truth annotations include objects such as a person, baseball bats, and a handbag.Each annotation consists of the image ID, category ID, object name, and bounding box coordinates (xmin, ymin, xmax, ymax).Confidence scores for ground truth annotations are all 1, indicating high confidence in the presence of these objects.YOLOv5 detects objects with confidence scores and scaled bounding box coordinates. Detected objects include a person, skis, and a backpack. Confidence scores for detections are provided along with the scaled bounding box coordinates.Faster R-CNN detects a car with a confidence score and scaled bounding box coordinates. Only one object (a car) is detected by Faster R-CNN in this case. SSD detects multiple instances of objects such as a person, baseball bats, and a handbag. Detected objects include multiple instances of baseball bats and a handbag, along with a person. Confidence scores for detections are provided along with the scaled bounding box coordinates. Scale information indicates the scaling factors applied to the image for each model. YOLOv5 and SSD models have a scale of 1.0, while Faster R-CNN has a scale of 0.6 for width and 0.8 for height.

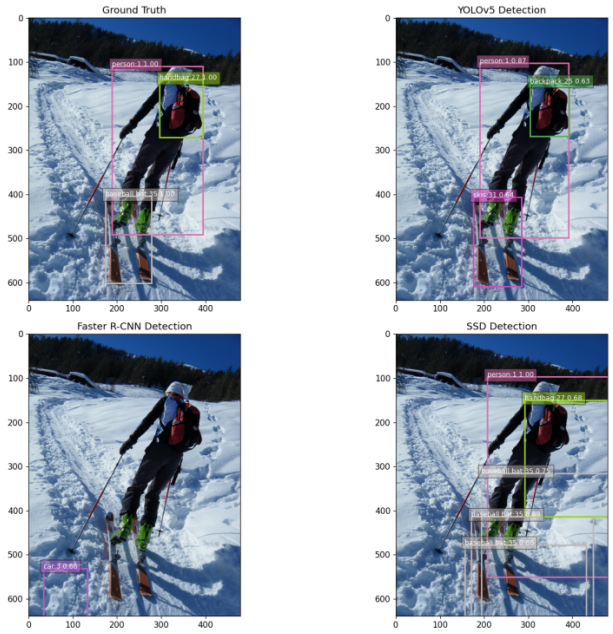


**Figure 3: Person with the SKI**

## 3.5 Empirical Results:

Faster R-CNN: The Faster R-CNN model achieves very low average precision and recall across all IoU thresholds and object sizes. The AP values range from 0.001 to 0.010, indicating poor performance in object detection. SSD: The SSD model performs better than Faster R-CNN, with slightly higher AP values ranging from 0.001 to 0.024. However, the performance is still relatively low compared to state-of-the-art models. YOLOv5: YOLOv5 achieves mixed results, with some variations in AP and AR across different IoU thresholds and object sizes. The model shows promise with higher AP values compared to Faster R-CNN but still has room for improvement.

## 4 Summary:

Overall, the evaluation highlights the differences in performance among the three object detection models on the COCO dataset, with YOLOv5 showing the most promising results among the evaluated models. However, further optimization and fine-tuning may be necessary to improve the overall performance across all metrics.

# 5 Learnings:

## 5.1 John Stephen Gutam: G01413212

*5.1.1 Understanding the Dataset:* The initial phase of the project involved familiarizing myself with the COCO dataset. This step was crucial as it laid the groundwork for the subsequent stages of the project. I learned the importance of correctly downloading and unzipping the dataset in the working directory.

*5.1.2 Setting up the Environment:* The next learning was about setting up the environment. This involved importing necessary libraries such as a torch, torchvision, matplotlib, pandas, os, datetime, JSON, PIL, and pycocotools. I understood the significance of setting the current working directory correctly to ensure smooth execution of the project.

*5.1.3 Loading the Models:* The models used in this project were YOLOv5, Faster R-CNN, and SSD. I learned how to load these models and set them to evaluation mode. This step was pivotal as it prepared the models for the evaluation process.

*5.1.4 Defining Functions:* I learned to define several functions to perform various tasks such as detection, converting detections to a data frame, creating a color map for labels, plotting detections on an axis, and plotting all detections including ground truth. This step was instrumental in streamlining the evaluation process.

*5.1.5 Setting up the Evaluation Process:* The next learning was about setting up the evaluation process. This involved image transformation using PyTorch's torch-vision. transforms module. I also learned to define functions to get model detections and evaluate the model using the COCO dataset.

*5.1.6 Evaluating the Models:* The models were then evaluated using the COCO validation set. I learned how to interpret the printed results for analysis. This step was crucial as it provided insights into the performance of the models.

*5.1.7 Checking Ground Truth:* The final learning was about checking the ground truth labels and comparing them with the model detections. I learned how to plot the results for visual comparison. This step was important as it provided a visual representation of the model's performance.

# 6 Related work:

. 1. https://pytorch.org/hub/ultralytics_yolov5/ 2. https://github.com/ultralytics/yolov5 3. Introduction to Object Detection in Deep Learning by Aladdin Perrson.

# 7 Resources:

1. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR). https://arxiv.org/abs/1506.02640 2. Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. In European conference on computer vision (ECCV). https://arxiv.org/abs/1405.0312 3. Bochkovskiy, A., Wang, C.-Y., Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. https://arxiv.org/abs/2004.10934