OBJECT DETECTION IN DIGITAL IMAGES: A COMPARATIVE STUDY OF YOLO AND FASTER R-CNN

Lovro Akmačić, Filip Buljan, Lucia Crvelin, Tomislav Čupić, Lorena Švenjak
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia
{lovro.akmacic, filip.buljan, lucia.crvelin, tomislav.cupic, lorena.svenjak}@fer.hr

***Abstract*—This report details a digital image processing and analysis project focused on object detection. We explore and implement state-of-the-art deep learning models, specifically YOLO (You Only Look Once) and Faster R-CNN (Region-based Convolutional Neural Network), for their efficacy in accurately identifying and localizing objects within images. The project aims to compare the performance, advantages, and limitations of these two prominent architectures in a practical application. We will discuss the theoretical underpinnings of each method, their practical implementation, and the insights gained from their application to a relevant dataset.**

I. DESCRIPTION OF AREA AND ACTIVITY

Digital image processing and analysis is a multidisciplinary field at the intersection of computer science, engineering, and mathematics, focusing on methods for manipulating and analyzing digital images. Its applications are vast, ranging from medical imaging and remote sensing to autonomous vehicles and security systems.

Our project delves into the sub-area of object detection, a fundamental task in computer vision that involves identifying instances of semantic objects of a certain class (e.g., humans, cars, animals) in digital images or videos and localizing them by drawing bounding boxes around them. The primary activity of this project involves the application and comparative analysis of two prominent deep learning-based object detection frameworks: YOLO and Faster R-CNN.

II. OVERVIEW OF METHODS USED

This section will provide a detailed technical overview of the two primary object detection methodologies employed in this project: YOLO and Faster R-CNN.

A. YOLO (You Only Look Once)

YOLO is a single-shot detector known for its speed and real-time processing capabilities. Unlike traditional object detection systems that separate the object detection pipeline into distinct stages (e.g., region proposal, classification, non-maximum suppression), YOLO processes the entire image in a single pass. It divides the image into a grid and simultaneously predicts bounding boxes and class probabilities for each grid cell. We will discuss its architectural components, including the backbone network, detection heads, and the loss function used for training [1]. Different versions of YOLO (e.g., YOLOv3 [2], YOLOv4 [3]) will be briefly mentioned, highlighting their key improvements.

B. Faster R-CNN (Region-based Convolutional Neural Network)

Faster R-CNN is a two-stage object detector that significantly improved upon its predecessors (R-CNN and Fast R-CNN) by introducing the Region Proposal Network (RPN) [4]. The RPN is a fully convolutional network that simultaneously predicts object bounds and objectiveness scores at each position. The proposed regions are then fed into the Fast R-CNN detection network for classification and bounding box regression refinement. We will elaborate on its architecture, including the shared convolutional features, RPN operation, RoI (Region of Interest) pooling, and the classification and regression heads.

III. IMPLEMENTATION OF YOLO FOR CAR DETECTION

Our YOLO implementation focuses on efficiently detecting and counting cars in images. The core logic revolves around leveraging a **pre-trained YOLOv8 model** from the `ultralytics` framework. This choice significantly streamlines the development process by utilizing a model already trained on a vast dataset (COCO), thus benefiting from **transfer learning** and eliminating the need for extensive custom training.

The implementation follows a clear workflow:

1) **Model Initialization**: A `YOLO` object is instantiated, loading the pre-trained weights (e.g., `yolov8l.pt`). This prepares the deep learning model for inference.
2) **Inference and Result Parsing**: For each input image, the model performs a single forward pass, predicting all potential objects. The results, which include **bounding box coordinates, class IDs, and confidence scores**, are then extracted.
3) **Car-Specific Filtering**: A crucial step involves filtering these raw detections. Since our goal is car detection,

we specifically select detections where the `class_id` corresponds to 'car' (which is class ID 2 in the COCO dataset). This ensures that only relevant objects are processed further.

4) **Visualization and Counting**: For each confirmed car detection, its bounding box is drawn on the image, along with a label indicating its class and confidence score. A running count of detected cars is maintained and displayed on the image. This visual feedback is essential for immediate verification of the model's performance.
5) **Automated Batch Processing**: The system is designed to handle entire datasets. It iterates through specified input directories (e.g., `train`, `val`, `test` subsets), automatically loading each image, applying the detection logic, and saving the annotated output to a designated folder.

In essence, the implementation provides a robust and automated pipeline for car detection, focusing on clear, efficient use of a powerful pre-trained model combined with standard image processing techniques for visualization and data handling.

IV. IMPLEMENTATION OF FASTER R-CNN: PRE-TRAINED VS. FINE-TUNED

Our project implemented Faster R-CNN using two distinct strategies to detect cars: one leveraging a **generically pre-trained model** and another employing a **fine-tuned model**. Both methods use the `torchvision` library for model handling.

A. Pre-trained Faster R-CNN

This approach directly utilizes a Faster R-CNN model pre-trained on the **COCO dataset**. The model comes with extensive knowledge of 80 different object categories, including cars.

- **Logic**: The model is loaded with its original weights, allowing it to detect cars based on its general understanding from the diverse COCO training data. Car identification relies on matching detected objects to the 'car' class ID (class ID 3) within COCO's existing classifications.
- **Application**: This method is straightforward, requiring no additional training. It applies a universal object detector to our specific task.

B. Fine-tuning Faster R-CNN

Fine-tuning is a crucial technique for adapting a pre-trained deep learning model to a new, specific task or dataset, often with limited data. Instead of training a model from scratch, which is computationally expensive and requires vast amounts of data, fine-tuning leverages the robust features learned by a model on a large general dataset (like COCO) and adjusts them to the nuances of the target dataset.

Our implementation details the process of fine-tuning a Faster R-CNN model (specifically `fasterrcnn_resnet50_fpn`) to accurately detect only cars.

1) Fine-Tuning Process:

1) **Model Initialization and Customization**: We begin by loading a `fasterrcnn_resnet50_fpn` model with its original pre-trained weights from the COCO dataset. This provides a strong starting point with general object recognition capabilities. The crucial step for adaptation is modifying the model's final classification head. The original `box_predictor` (which predicts class scores for 80+ COCO categories) is replaced with a new `FastRCNNPredictor`. This new predictor is configured to output probabilities for only **two classes**: background and car (`NUM_CLASSES = 2`). This effectively re-purposes the model to focus exclusively on our target object.
2) **Dataset Preparation**: We use the `CocoDetection` dataset class from `torchvision` to load our car-specific images and their corresponding annotations. These annotations are in the COCO format, which includes bounding box coordinates and category IDs. A key aspect of data preparation is mapping the original COCO category ID for 'car' (which is 3) to a new local label (1) consistent with our two-class setup. This ensures the model correctly associates the car detections in our dataset with the new 'car' class in its adapted output layer. Images are transformed into PyTorch tensors, and a `DataLoader` is set up to efficiently batch and shuffle the training data. A custom `collate_fn` handles the varied number of objects per image in a batch.
3) **Training Configuration**: An Adam optimizer is selected to update the model's parameters during training. Importantly, only the parameters that require gradients (i.e., those that are trainable) are updated, which typically includes the newly added classification head and potentially some earlier layers if they are unfrozen. The training is configured for a specific number of epochs (`NUM_EPOCHS = 10`) and a learning rate (`LR = 1e-4`).
4) **Training Loop**: The model is set to training mode (`model.train()`). The training loop iterates through the `train_loader`, processing images in batches. For each batch, target annotations are processed: bounding box coordinates are converted to the expected format (`[x_min, y_min, x_max, y_max]`), and COCO category IDs are mapped to our custom 'car' label. Images without valid car annotations are handled to prevent errors. The model performs a forward pass (`loss_dict = model(images, formatted_targets)`), which calculates the loss values for region proposal, classification, and bounding box regression. The losses are summed, backpropagated (`losses.backward()`), and the optimizer updates the model's weights (`optimizer.step()`). Progress is monitored via a `tqdm` progress bar, showing the loss for each batch.

Through this fine-tuning process, the Faster R-CNN model transitions from a general-purpose object detector to a specialized and effective car detector, leveraging its pre-trained knowledge while adapting to the specific characteristics of our car dataset.

V. RESULTS

This section presents the results of our experiments with YOLO and Faster R-CNN on a chosen dataset.

A. Results: YOLOv8l

This section details the performance evaluation of the YOLOv8l model for car detection. The model's performance was assessed on a test dataset comprising 54 images with a total of 91 car instances.

The key performance metrics obtained are as follows:

- **Precision (P)**: The model achieved a precision of **0.919**. This indicates that 91.9% of the objects detected by the model were indeed cars, highlighting a very low rate of false positives.
- **Recall (R)**: The recall stands at **0.857**. This means that the model successfully identified 85.7% of all actual car instances present in the test set, demonstrating a good ability to find most of the relevant objects.
- **mAP@50**: The Mean Average Precision at an Intersection over Union (IoU) threshold of 0.50 is **0.904**. This high value signifies that the model is highly effective at detecting cars and localizing them with at least 50% overlap with the ground truth bounding boxes.
- **mAP50-95**: The Mean Average Precision averaged across IoU thresholds from 0.50 to 0.95 (with steps of 0.05) is **0.360**. This metric provides a comprehensive assessment of the model's performance across varying degrees of localization accuracy.

Analysis:

The YOLOv8l model demonstrates strong performance in car detection, particularly in terms of precision and recall. The high precision (0.919) indicates that when the model identifies an object as a car, it is very likely correct. The good recall (0.857) suggests that the model is effective at finding a large proportion of the cars present in the images.

The mAP@50 of 0.904 is exceptionally high, indicating that YOLOv8l is very reliable at detecting the presence of cars and providing reasonably accurate bounding boxes. This makes it suitable for applications where the primary concern is identifying whether an object is present and roughly where it is located.

However, the mAP50-95 value of 0.360, while respectable, is considerably lower than the mAP@50. This difference highlights that while the model is excellent at detecting cars with a loose localization requirement (IoU=0.50), its performance for very precise bounding box localization (IoU=0.75 and higher) is more challenging. This is a common characteristic of single-stage detectors like YOLO, which prioritize speed over extreme localization precision compared to two-stage detectors.

Overall, YOLOv8l proves to be a highly efficient and accurate solution for car detection, offering a strong balance between speed and performance for this specific task.

B. Faster R-CNN Results

1) Results: Faster R-CNN without Fine-Tuning: This section summarizes the performance of the Faster R-CNN model using only its **pre-trained COCO weights**, without fine-tuning for car detection.

- **Overall mAP**: The model achieved an overall Mean Average Precision (mAP) of **0.2439**.
- **mAP@50**: At an Intersection over Union (IoU) threshold of 0.50, the mAP was **0.7768**, indicating reasonable object detection and localization at a relaxed overlap criterion.
- **mAP@75**: For a stricter IoU of 0.75, the mAP significantly dropped to **0.0645**, showing limitations in precise bounding box localization.
- **Performance by Size**: The model performed notably better on **large objects (mAP: 0.4650)** compared to medium (mAP: 0.2457) and struggled with **small objects (mAP: -1.0000)**.
- **Average Recall (AR@100)**: The recall over 100 detections was **0.3575**.

Analysis: While the pre-trained Faster R-CNN can generally detect objects (high mAP@50), its performance for accurate localization (low mAP@75) and small object detection is limited. This is expected as the model was trained for general object recognition, not specialized car detection. Specific per-class mAP for 'car' was not available in this output.

2) Results: Fine-tuned Faster R-CNN: This section presents the performance evaluation of the Faster R-CNN model after it has undergone fine-tuning on a car-specific dataset. The results are compared against the baseline performance of the pre-trained model to highlight the impact of domain adaptation.

The key evaluation metrics are summarized below:

- **Overall mAP**: The overall Mean Average Precision (mAP) for the fine-tuned model is significantly higher at **0.6396**. This represents a substantial improvement compared to the 0.2439 achieved by the pre-trained model, indicating a much better balance of localization and classification accuracy.
- **mAP@50**: At an IoU threshold of 0.50, the mAP reached an impressive **0.9987**. This near-perfect score demonstrates that the fine-tuned model is highly effective at detecting cars with at least 50% overlap, signifying robust presence detection.
- **mAP@75**: For the stricter IoU threshold of 0.75, the mAP dramatically improved to **0.7541**. This is a critical improvement from the 0.0645 of the pre-trained model, indicating that fine-tuning has enabled the model to localize cars with significantly higher precision.
- **Performance by Object Size**:
  - **mAP for Medium objects**: The mAP for medium-sized objects is **0.6284**, showing strong performance.

- **mAP for Large objects**: For large objects, the mAP is **0.7378**, maintaining high accuracy.
- **mAP for Small objects**: The mAP for small objects remains at **-1.0000**, suggesting that even with fine-tuning, detecting very small instances of cars remains a challenge for this configuration.

- **Average Recall (AR)**:
  - The maximum Average Recall over 100 detections per image (MAR@100) is **0.6973**. This indicates a much-improved ability to retrieve ground truth cars compared to the pre-trained model's 0.3575.
  - Recall for large objects (0.7818) is higher than for medium objects (0.6823), consistent with general object detection trends.
- **Classes Detected**: The output 'classes: 1.0000' confirms that the model is now specifically focused on detecting a single target class, which is 'car', as intended by the fine-tuning process.

Analysis: The results unequivocally demonstrate the profound impact of fine-tuning on the Faster R-CNN model's performance for car detection. There is a substantial increase across all mAP metrics, particularly in mAP@75, which signifies a remarkable improvement in the precision of bounding box localization. The near-perfect mAP@50 further underscores the model's reliability in identifying the presence of cars. While the detection of small objects continues to be a limitation, the overall enhancement in accuracy and recall validates fine-tuning as an extremely effective strategy for adapting general-purpose object detectors to specific domain tasks. This specialization allows the model to leverage its pre-trained feature extraction capabilities while learning the subtle nuances required for highly accurate car detection.

VI. CONCLUSION

This project successfully explored and implemented state-of-the-art deep learning models, YOLO and Faster R-CNN, for the task of car detection and counting in digital images. Our comparative analysis provides valuable insights into the performance characteristics of these distinct architectures under various conditions, particularly highlighting the profound impact of fine-tuning.

The YOLOv8l model demonstrated strong performance in car detection, characterized by high Precision (0.919) and Recall (0.857), leading to an impressive mAP@50 of 0.904. This indicates YOLO's robust capability for real-time presence detection and reasonable localization. However, its mAP50-95 score of 0.360 suggests that while fast and generally accurate, its precise bounding box localization for stricter IoU thresholds is less refined compared to multi-stage detectors.

The Faster R-CNN implementation showcased a clear dichotomy based on its training regimen:

- When utilizing only its pre-trained COCO weights, Faster R-CNN exhibited an overall mAP of 0.2439 and a significantly lower mAP@75 of 0.0645. It struggled particularly with precise localization and the detection of small objects, underscoring the limitations of a generic model when applied to a specific domain without adaptation.
- Conversely, the fine-tuned Faster R-CNN model achieved a dramatic improvement across all metrics. Its overall mAP surged to 0.6396, with a near-perfect mAP@50 of 0.9987 and a highly impressive mAP@75 of 0.7541. This substantial gain directly demonstrates the immense power of transfer learning and fine-tuning in specializing a model for a particular task. While it still faced challenges with very small objects (mAP: -1.0000), its ability to precisely localize cars was vastly superior to both the generic Faster R-CNN and, notably, to YOLO's stricter localization performance.

In summary, for applications prioritizing high precision in object localization and classification accuracy, especially when data is available for domain-specific training, fine-tuning a model like Faster R-CNN proves to be exceptionally effective. For scenarios demanding extreme real-time processing and where slightly less precise localization is acceptable, YOLO offers a compelling and highly efficient solution.

A. Future Trends and Work

The field of object detection is continuously evolving, and several avenues could extend this project:

1) **Exploring Advanced Architectures**: Investigate newer object detection models, such as more recent YOLO versions (e.g., YOLOv9), or transformer-based architectures like DETR (Detection Transformer), which offer different trade-offs in terms of accuracy, speed, and complexity.
2) **Robustness to Challenging Conditions**: Enhance model performance under adverse conditions, including varying lighting, heavy occlusion, motion blur, and diverse weather conditions, through advanced data augmentation or specialized training techniques.
3) **Optimization for Edge Devices**: Quantize and optimize the models for deployment on resource-constrained embedded systems and edge devices (e.g., automotive platforms, drones), which is critical for real-world autonomous applications.
4) **Temporal Object Tracking**: Extend the capabilities from static image detection to multi-object tracking in video sequences, which involves associating detected objects across consecutive frames.
5) **Dataset Expansion and Diversity**: Further augment the training dataset with more diverse car types, angles, environments, and challenging scenarios to improve the model's generalization capabilities.
6) **Explainable AI (XAI)**: Explore methods to interpret why the models make certain detection decisions, increasing trust and facilitating debugging in critical applications.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE*

conference on computer vision and pattern recognition, pp. 779–788, 2016. [Online]. Available: https://arxiv.org/abs/1506.02640

[2] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[3] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimalflow and strong baseline for real-time object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015. [Online]. Available: https://arxiv.org/abs/1506.01497