Comparative Analysis of the Newest YOLO Models: v8, v7, v6, and NAS in Urban Vehicle Detection

Irina Getman
Bachelor of Software Engineering, Yoobee Colleges
Supervisory team: Dr. Mohammad Norouzifard

August 25, 2023

Abstract

Object detection, a crucial area within computer vision, has seen transformative advancements with the YOLO (You Only Look Once) series of algorithms. This research delves into the intricacies of the YOLOv6, YOLOv7, YOLOv8, and YOLO-NAS models, highlighting their architectural distinctions and performance benchmarks. A significant portion of our study is dedicated to understanding the impact of dataset size and quality on the performance of these models. We also discuss the configurations used during model training and the evaluation metrics derived from training the YOLOv8 and YOLOv7 models on our augmented dataset. Our results emphasize the critical role of a robust and high-quality dataset in achieving superior detection accuracy. Specifically, training the YOLOv8 model on an enhanced dataset showed a noticeable performance improvement. This observation is consistent with existing literature, further underscoring the importance of dataset quality in object detection endeavors. Our study offers insights into other YOLO variants, presenting a comprehensive view of the YOLO series' evolution and capabilities.
Keywords: Object detection, autonomous driving, YOLOv8, YOLO-NAS, computer vision, YOLOv6, YOLOv7.

0.1 Introduction

Autonomous driving technology has heralded a new era of research and innovation spanning several decades. The ultimate aspiration of this technology is to engineer vehicles that can navigate roads and traffic without human intervention. Achieving this capability is made possible by leveraging the power of advanced sensors and intricate artificial intelligence algorithms. Central to this technological marvel is the capacity to precisely detect vehicles, especially in urban environments with ever-changing and unpredictable dynamics.

Historically, the concept of autonomous driving has roots that trace back centuries. The Renaissance genius Leonardo da Vinci first envisioned the idea of a self-driving vehicle in the 1500s (Hiziroglu, 2020). However, the 20th century marked the commencement of more sophisticated experiments in this domain. Fast forward to recent times, the landscape of autonomous driving has evolved exponentially. Major car manufacturers heralded a new phase by introducing the Advanced Driving Assistance System (ADAS) in 2017. While these systems have had their share of challenges and unfortunate incidents (Levin and Wong, 2018), their potential to mitigate traffic-related fatalities is undeniable, as underscored by reports from Injury Facts (Facts, 2022).

A pillar in the success of autonomous driving systems is their ability to perceive their surroundings in real time. This real-time perception is pivotal for identifying objects, such as other vehicles, which informs the autonomous system's decision-making processes. However, achieving this in real-world scenarios, with challenges like fluctuating lighting conditions, occlusions, and rapid motion, is no mean feat. Researchers have proposed numerous algorithms to navigate these challenges, aiming to balance detection speed and accuracy.

A notable contender in this space is YOLO (You Only Look Once), a one-stage detector that processes images holistically, subsequently outputting bounding boxes and class labels for detected objects. Over time, YOLO has seen iterative improvements, culminating in this year's versions, YOLOv8 and YOLO-NAS. Its predecessors, introduced just a year before, YOLOv6 and YOLOv7, set benchmarks with their state-of-the-art performance.

This paper is structured to provide readers with a comprehensive understanding of the advancements in YOLO and its implications for autonomous driving. We commence with a "Literature Review" section, offering an in-depth analysis of the YOLO versions v6, v7, v8, and NAS. This sets the foundation for the subsequent "Methodology" section, where the processes of Data Collection, Model Training, and Evaluation Metrics employed are elucidated. The "Results" section showcases the culmination of our research efforts, where the insightful outcomes are displayed using both figures and tabular representations. Here, we delve into the Model Performance Analysis and the Computational Efficiency of the YOLO versions under study. Finally, the "Discussion" section synthesizes our findings, drawing connections to broader implications and potential future directions in autonomous driving technology.

0.1.1 Objective

The primary objective of this paper is to understand, evaluate, and compare the performance of the newest YOLO models: v8, v7, v6, and NAS. Building upon the foundation of the previous semester's work, which exclusively focused on YOLOv8, we aim to broaden our scope by incorporating the other YOLO versions. Facilitating a comprehensive discussion on their respective performances, strengths, and limitations becomes particularly relevant in the context of urban vehicle detection. Through this comparative analysis, we endeavor to provide a holistic understanding of the advancements and nuances of each YOLO iteration.

0.2 Literature Review

The realm of object detection has been a focal point of numerous research endeavors, with each study contributing to the collective understanding and advancement of the field. To ensure a comprehensive grasp of the existing literature and to identify gaps and opportunities for our research, we undertook an exhaustive review of the available studies, methodologies, and findings related to object detection and the YOLO series of algorithms. This literature review was conducted systematically, adhering to rigorous criteria to ensure the inclusion of relevant and impactful studies. The process and criteria adopted for selecting studies and the flow of information through the different phases of this review are detailed in the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) chart provided in the appendix. We invite readers to refer to the PRISMA chart for a visual representation of

our literature review process, which underscores the depth and breadth of our exploration. A thorough literature review was conducted to understand the intricacies and advancements in object detection. The PRISMA flow diagram depicts the systematic review process and the flow of information through different phases, as shown in Figure 1.

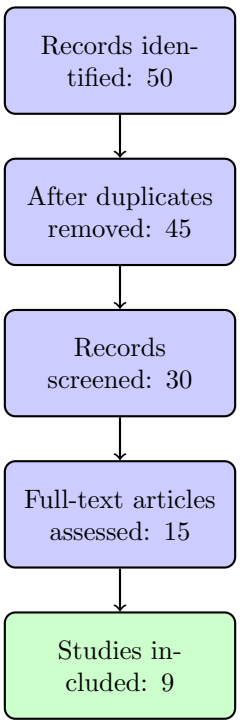


Figure 1: PRISMA Flow Diagram of the Literature Review Process

This section briefly overviews the YOLO algorithm, highlighting its key features and technical details. Initially, we will touch upon the earlier YOLO versions, from v1 to v5. Subsequently, we will delve into the technical features of YOLO versions 6, 7, 8, and NAS, accompanied by details of their respective architectures.

0.2.1 Key Features of YOLO

The YOLO (You Only Look Once) algorithm has gained prominence as an object detection technique owing to its distinctive methodology and efficiency. According to "A Review of Yolo Algorithm Developments", 2022, the YOLO algorithm possesses several distinctive characteristics that set it apart from other object detection methods:

- YOLO boasts a compact model size coupled with rapid computational speed.
- It directly predicts the position and category of the bounding box via the neural network.
- By utilizing the entire image for detection, YOLO can encode global information, thereby minimizing the error of misidentifying the background as an object.
- Its robust generalization capability enables it to learn highly generalized features, making it transferable to various domains.
- YOLO reformulates the challenge of target detection into a regression problem.

One of YOLO's primary strengths is its speed and efficiency, allowing for real-time object detection. This makes it apt for applications like autonomous driving and video surveillance. Nevertheless, there is room for enhancing YOLO's detection precision, especially regarding closely situated objects or those in groups.

0.2.2 Technical Overview of YOLO

The YOLO algorithm, simply put, operates as follows:

1. **Input Image:** The initial image is segmented into a grid of cells, with each cell accountable for detecting objects within its confines.
2. **Feature Extraction:** A convolutional neural network (CNN) is employed to extract features from the input image. This CNN, pre-trained on an extensive dataset, is adept at identifying various patterns and shapes in the image.
3. **Object Detection:** Every cell in the grid anticipates a fixed set of bounding boxes to localize the objects in the image. The algorithm predicts the likelihood of an object's presence for each box and the box's coordinates relative to the cell.
4. **Non-Maximum Suppression:** The algorithm implements non-maximum suppression to eliminate redundant detections and choose the most precise bounding boxes. This ensures each object is detected singularly and that the final results are accurate.
5. **Output:** YOLO's final output comprises a list of bounding boxes, each associated with its object class and probability.

0.2.3 Early Iterations of YOLO

The YOLO algorithm has undergone several iterations since its inception, each building upon the previous versions to enhance its performance, accuracy, and efficiency. This subsection provides a brief overview of the early versions of YOLO, tracing its evolution from the original YOLO algorithm introduced in 2015 to the YOLO-v5 version in 2020 (Hussain, 2023).

- **YOLO-v1 (2015):** The pioneering YOLO algorithm utilized a single-stage object detector that employed a solitary convolutional neural network (CNN) to predict bounding boxes and class probabilities directly from entire images. While it achieved real-time object detection with commendable accuracy, it faced challenges detecting smaller objects.
- **YOLO-v2 (2016):** Addressing the limitations of its predecessor, YOLO-v2 incorporated anchor boxes and a revamped network architecture to detect small objects better. Introducing batch normalization and enhanced training techniques resulted in superior accuracy and expedited training.
- **YOLO-v3 (2018):** YOLO-v3 brought further improvements in both accuracy and speed by integrating a feature pyramid network and a novel detection head. This version also introduced innovative data augmentation techniques and refined training methods.
- **YOLO-v4 (2020):** Introducing many new techniques, YOLO-v4 featured a new backbone network, spatial pyramid pooling, and a novel loss function. It set new standards by achieving state-of-the-art performance on various object detection benchmarks.
- **YOLO-v5 (2020):** YOLO-v5, a complete overhaul of the YOLO algorithm, was architected around a CSP (cross-stage partial) backbone network. It introduced several groundbreaking techniques, including self-attention and a new loss function. Designed with mobile and edge devices in mind, it achieved real-time object detection with impressive accuracy.

0.2.4 In-depth Analysis of Recent Iterations

0.2.5 YOLOv6

YOLOv6, also known as MT-YOLOv6, is a recent release by the Meituan Technical Team, showcasing an initial evaluation that surpasses previous open-source object detection frameworks. Developed by researchers at Meituan, a large e-commerce company in China, YOLOv6 builds on the YOLO architecture with several significant improvements. This model was introduced in the official paper "YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications" by Li et al., 2022:

- EfficientRep Backbone and Rep-PAN Neck: YOLOv6 introduces a redesigned backbone and neck with hardware considerations, enhancing the efficiency of feature extraction and processing.

- Decoupled Head: The head is decoupled in YOLOv6, separating the classification and box regression features, leading to increased model performance.
- Training Enhancements: The YOLOv6 repository includes new training techniques, such as anchor-free training, SimOTA tag assignment, and SIoU box regression loss.
- Benchmark Performance: YOLOv6 achieves higher mAP on the COCO dataset than YOLOv5, YOLOX, and PP-YOLOE, maintaining comparable inference speeds (Sirisha et al., 2023).

Architecture of YOLOv6

The architecture of YOLOv6 builds on the traditional YOLO model with specific enhancements:

1. Backbone: Utilizes the EfficientRep backbone for effective feature extraction.
2. Neck: Employs the Rep-PAN neck to process and combine features from the backbone.
3. Head: Features a decoupled head to separate classification and box regression features, improving performance.
4. Training Enhancements: Includes new training methods to optimize learning.

0.2.6 YOLOv7

YOLOv7, a predecessor to YOLOv8, is another significant iteration in the YOLO series. Though not as advanced as YOLOv8, it laid the groundwork for many innovations in the subsequent version. Some of its notable features and advancements include:

- Enhanced accuracy through innovative architectural changes and feature extraction mechanisms.
- Improved inference speeds suitable for real-time applications, though not as optimized as YOLOv8.
- Robustness in various scenarios, including challenging lighting conditions and occlusions.
- Scalability across different resolutions, ensuring adaptability to various hardware setups.
- Flexibility in model architecture allows customization based on specific use cases.

YOLOv7 was published in ArXiv in July 2022. It surpassed all known object detectors in speed and accuracy in the range of 5 FPS to 160 FPS. The paper "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors" detailed the architecture changes and a series of bag-of-freebies, which increased the accuracy without affecting the inference speed, only the training time (Wang et al., 2022).

The architecture changes of YOLOv7 include an Extended efficient layer aggregation network (E-ELAN) and implicit knowledge inspired in YOLOR. YOLOv7 achieved a 75% reduction in parameters and a 36% reduction in computation while simultaneously improving the average precision (AP) by 1.5% compared to YOLOv4. Compared to YOLOv4-tiny, YOLOv7-tiny reduced parameters and computation by 39% and 49%, respectively, while maintaining the same AP. Lastly, compared to YOLOR, YOLOv7 reduced the number of parameters and computation by 43% and 15%, respectively, along with a slight 0.4% increase in AP. Evaluated on MS COCO dataset test-dev 2017, YOLOv7-E6 achieved an AP of 55.9% and AP50 of 73.5% with an input size of 1280 pixels with a speed of 50 FPS on an NVIDIA V100 (Terven and Cordova-Esparza, 2023).

Architecture of YOLOv7

YOLOv7, like its successors, is built on a deep learning framework and features:

1. A backbone network designed for efficient feature extraction from input images.
2. A neck network that combines features and generates multi-resolution feature maps.
3. A head network predicts bounding boxes and class probabilities.
4. Various pooling mechanisms to process feature maps at different scales.
5. Aggregation networks combine features from different scales and generate a unified feature representation.
6. A detector layer that uses anchor boxes for predicting object locations and sizes.

0.2.7 YOLOv8

Introduced in January 2023 by Ultralytics—the developers behind YOLOv5—YOLOv8 represents the latest advancement in the YOLO series of object detection models. The model offers five scaled versions: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x. While its detection heads are reminiscent of YOLOv7, comprising five detection modules and a prediction layer, YOLOv8 has set new benchmarks in object detection and semantic segmentation, balancing speed with accuracy. Specifically, YOLOv8x, when evaluated on the MS COCO dataset test-dev 2017, registered an AP of 53.9% using a 640-pixel image size—surpassing the 50.7% AP of YOLOv5 for the same input. This was achieved at a remarkable speed of 280 FPS on an NVIDIA A100 with TensorRT. YOLOv8 is versatile, operable via the command line interface (CLI) or as a PIP package, and is complemented by integrations for labeling, training, and deployment (Terven and Cordova-Esparza, 2023).

The enhancements introduced in this iteration include (AI, 2023):

- Augmented accuracy through refined architecture and feature pyramid networks.
- Superior inference speeds, reaching over 500 FPS on a CPU without sacrificing accuracy.
- Resilience in challenging environments, such as low-light conditions and occlusions.
- Scalability across diverse input resolutions.
- Customizable model architecture and hyperparameters.

Architecture of YOLOv8 Building upon the legacy of the YOLO series and developed using PyTorch, YOLOv8 incorporates the following:

1. A modified CSPDarknet53 as its backbone for feature extraction.
2. A neck network, inspired by YOLOv5, to produce feature maps.
3. A head network is dedicated to predicting bounding boxes and class probabilities.
4. Spatial Pyramid Pooling (SPP) for handling feature maps.
5. Path Aggregation Network (PAN) for feature integration.
6. A detector layer employs anchor boxes for precise object prediction.

0.2.8 YOLO-NAS

Following the release of YOLOv8, the object detection community witnessed the introduction of YOLO-NAS, a state-of-the-art model developed by Deci AI. Designed to address the limitations of prior YOLO models, Deci AI constructs YOLO-NAS using AutoNAC, a Neural Architecture Search Engine. This model boasts superior speed and performance, marking a significant advancement in object detection by refining the balance between accuracy and latency.

In a recent paper titled "From YOLOv1 and Beyond", it is highlighted that YOLO-NAS is specifically designed to detect small objects, improve localization accuracy, and enhance the performance-per-compute ratio, making it suitable for real-time edge-device applications Terven and Cordova-Esparza, 2023. The paper also emphasizes the novelty of YOLO-NAS, with limited literature available due to its recent introduction.

- AutoNAC Integration: Deci's AutoNAC (Automated Neural Architecture Construction) generates YOLO-NAS's architecture, enhancing the performance of existing deep neural networks.
- RepVGG Utilization: YOLO-NAS employs RepVGG, a neural network architecture inspired by VGG. RepVGG is trained using a multi-branch architecture and is later converted to a single branch using re-parameterization, making YOLO-NAS optimal for production deployment.
- Quantization-Aware Training: YOLO-NAS incorporates QSP and QCI modules to minimize accuracy loss during post-training quantization.
- Hybrid Quantization: Unlike standard methods that quantize the entire model, YOLO-NAS's hybrid approach quantizes specific model sections, balancing latency and accuracy.
- Automatic Architecture Design: Leveraging Deci's proprietary NAS technology, AutoNAC, YOLO-NAS integrates foundational YOLO model architectures to produce an optimized model.

Architecture of YOLO-NAS

YOLO-NAS's architecture is a product of several innovative components:

1. Backbone: The AutoNAC-generated architecture achieves efficient feature extraction.
2. Neck: The model combines features and produces multi-resolution feature maps.
3. Head: YOLO-NAS predicts bounding boxes and class probabilities.
4. Quantization Modules: The model uses QSP and QCI for quantization-aware training. The YOLO-NASL architecture, the largest of the three architectures generated by AutoNAC, is designed to balance latency vs. throughput. It includes Quantization-aware modules QSP and QCI that combine re-parameterization for 8-bit quantization to minimize the accuracy loss during post-training quantization Terven and Cordova-Esparza, 2023.
5. Optimization Techniques: RepVGG is employed for post-training optimization, enhancing the model's generalization ability.

0.2.9 Other YOLO Models

The evolution of YOLO (You Only Look Once) models has witnessed the introduction of several variants, each designed to address specific challenges and improve object detection performance. In the paper "From YOLOv1 and Beyond," several models are discussed in detail, providing insights into their unique architectures and functionalities. As presented in the paper(Terven and Cordova-Esparza, 2023), these models are visualized in Figure 2.

- **YOLOR:** YOLOR is a real-time object detection system that merges the anchor-free YOLOv3 architecture with the anchor-based RetinaNet architecture. This combination aims to enhance both detection accuracy and processing speed.
- **YOLOX:** YOLOX introduces a high-performance object detection system that employs a novel anchor-free detection paradigm named "DETR." This approach has demonstrated state-of-the-art performance across various benchmarks.
- **DAMO-YOLO:** Developed by Alibaba Group, DAMO-YOLO is a real-time object detection system. It incorporates a neural architecture search (NAS) method to identify an efficient architecture automatically. The system also features a large neck, a small head, and an aligned OTA label assignment method, all contributing to improved detection accuracy.
- **PP-YOLO:** PP-YOLO utilizes a pyramid pooling module to capture multi-scale features, enhancing detection accuracy.
- **PP-YOLOv2:** An advancement over PP-YOLO, PP-YOLOv2 integrates a spatial attention module. This addition serves to amplify the feature representation, leading to better detection accuracy.
- **PP-YOLOE:** PP-YOLOE, an enhanced version of PP-YOLOv2, employs a dynamic head architecture. This design allows the system to adapt to varying object scales and aspect ratios, further refining detection accuracy.
- **YOLO with Transformers:** This variant of YOLO integrates a transformer-based architecture, enabling the capture of long-range dependencies to improve detection accuracy. Additionally, it introduces a new loss function named "GIoU-L1" to enhance localization accuracy.

0.3 Methodology

0.3.1 Project Methodology Overview

Before delving into the detailed methodologies employed in this research, it is essential to provide a holistic overview of the entire project's structure and approach. The following mindmap serves as a visual representation, capturing the core components of our research methodology. It encompasses the project's duration, the chosen project methodology, the specific YOLO models explored, and the tools

Version	Date	Anchor	Framework	Backbone	AP (%)
YOLO	2015	No	Darknet	Darknet24	*63.4*
YOLOv2	2016	Yes	Darknet	Darknet24	*63.4*
YOLOv3	2018	Yes	Darknet	Darknet53	36.2
YOLOv4	2020	Yes	Darknet	CSPDarknet53	43.5
YOLOv5	2020	Yes	Pytorch	YOLOv5CSPDarknet	55.8
PP-YOLO	2020	Yes	PaddlePaddle	ResNet50-vd	45.9
Scaled-YOLOv4	2021	Yes	Pytorch	CSPDarknet	56.0
PP-YOLOv2	2021	Yes	PaddlePaddle	ResNet101-vd	50.3
YOLOR	2021	Yes	Pytorch	CSPDarknet	55.4
YOLOX	2021	No	Pytorch	YOLOXCSPDarknet	51.2
PP-YOLOE	2022	No	PaddlePaddle	CSPRepResNet	54.7
YOLOv6	2022	No	Pytorch	EfficientRep	52.5
YOLOv7	2022	No	Pytorch	YOLOv7Backbone	56.8
DAMO-YOLO	2022	No	Pytorch	MAE-NAS	50.0
YOLOv8	2023	No	Pytorch	YOLOv8CSPDarknet	53.9
YOLO-NAS	2023	No	Pytorch	NAS	52.2

Figure 2: Summary of YOLO models as discussed in "From YOLOv1 and Beyond."

utilized for brainstorming and visualization. This mindmap offers a concise snapshot of the research's foundational elements, ensuring clarity and coherence as we progress through the subsequent sections.

As depicted in Figure 3, our approach is multifaceted, combining traditional research methods with cutting-edge techniques and tools. The subsequent sections will delve deeper into each of these components, elucidating the rationale behind our choices and their implications for the research outcomes.

0.3.2 Data Collection and Augmentation

Meticulous data collection forms the foundation of our research. In May 2023, we recorded a dataset on Symonds Street, Auckland CBD. We extracted 516 images from a 17-second video at 29 frames per second. The recording was specifically against sunbeams, simulating real-life driving experiences when blinded by intense lighting. This approach offers a realistic training environment for deep learning models addressing such scenarios.

Data Augmentation Methodology

Initially, the plan was to train the YOLOv8 model on the dataset of 516 images. To ascertain if augmentation techniques could enhance the model's performance, we employed Roboflow for various augmentations:

- Outputs per training example: 3
- Flip: Horizontal
- 90° Rotate: Clockwise, Counter-Clockwise
- Grayscale: Applied to 25% of images
- Saturation: Adjusted between -25% and +25%
- Brightness: Adjusted between -25% and +25%
- Exposure: Adjusted between -25% and +25%
- Bounding Box Adjustments:
  - Flip: Horizontal
  - Brightness: Adjusted between -25% and +25%
  - Exposure: Adjusted between -25% and +25%

Post-augmentation, the enhanced dataset (version 2) consists of 1300 images, providing a more comprehensive training set for the model.

Our comprehensive approach to data collection and augmentation aims to develop a robust vehicle tracking system optimized for real-world conditions.

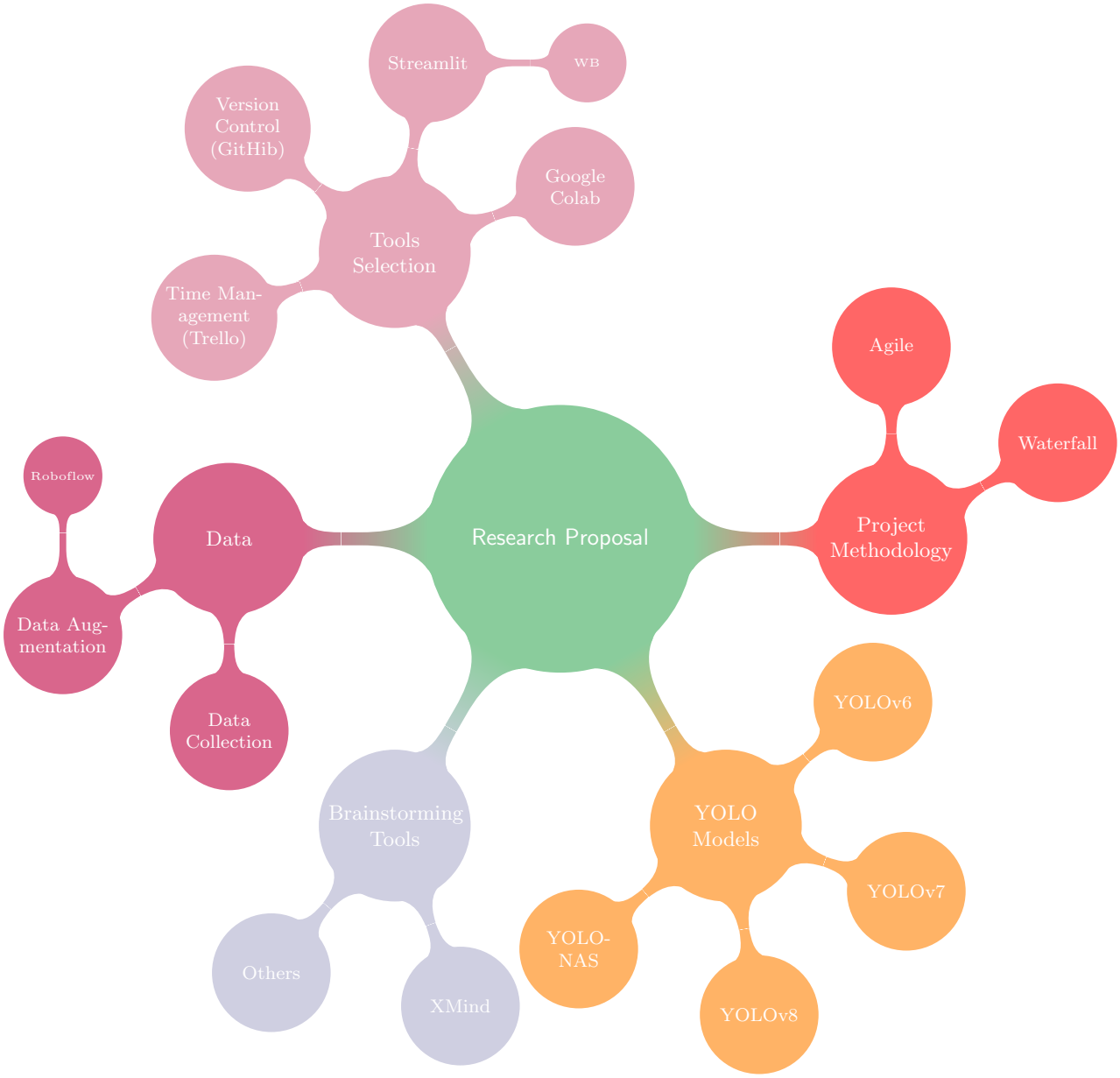


Figure 3: Mindmap of the Research Methodology

0.3.3 Data Labeling

In the realm of object detection, Grounding DINO emerges as a notable model. As delineated by Roboflow, Grounding DINO operates as a zero-shot object detection model. It is crafted by integrating a Transformer-based DINO detector with grounded pre-training. This unique combination enables the model to register remarkable performance metrics, notably achieving a 52.5 AP on the COCO detection zero-shot transfer benchmark. For our dataset's annotation, we followed the guidelines and insights presented in the Roboflow blog post *Enhance Image Annotation with Grounding DINO and SAM.*

Our labeling process also employed the SAM (Segment Anything Model) technique. SAM is dedicated to segmenting various entities within an image, offering a more detailed level of annotation. The synergy of Grounding DINO and SAM ensured thorough and precise labeling of our dataset, laying a solid foundation for practical model training.

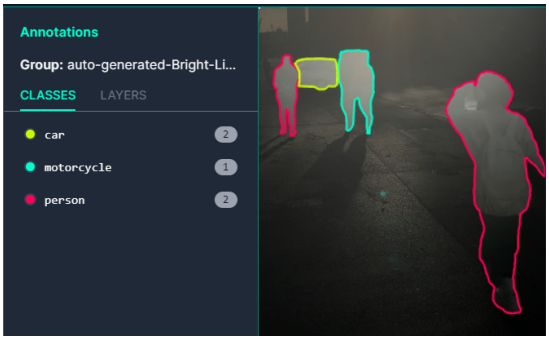
Even though the detection of objects using Grounding DINO wasn't 100% accurate (as shown in Figure 4), the primary objective was to expedite the annotation process for the 516 images. This acceleration was crucial, especially considering this is a solo-person project. The emphasis was on more than just achieving perfect accuracy but examining whether a more extensive dataset could enhance the model's performance, even if not entirely accurate. In a previous project last semester, I trained a

model using a dataset from the same video, and it consisted of only 36 images for training, including the applied augmentation techniques. The goal is to observe the performance difference when scaling up the dataset.

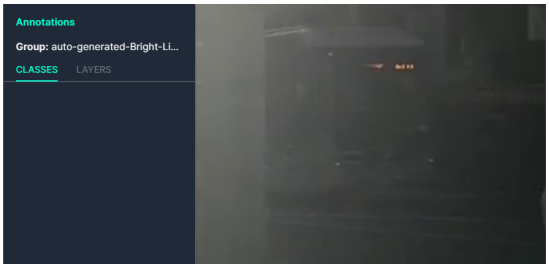
0.3.4 Naming Convention for Custom Models

To ensure clarity throughout the paper, we adopt the following naming conventions for our custom models:

- YOLOv8 (Original Dataset): **YOLOv8-OD**
- YOLOv8 (Enhanced Dataset): **YOLOv8-ED**
- YOLOv7: **YOLOv7-Base**



(a) Misclassification: A couple classified as a "motorcycle"



(b) Detection Error: Bus not detected in the image.

Figure 4: Illustration of detection anomalies: misclassification and missed detection.

0.3.5 Model Training

The training phase plays a pivotal role in determining the performance of deep learning models. Our methodology is tailored to be compatible with four models: YOLOv8, YOLOv7, YOLOv6, and YOLO-NAS.

Training Strategy: We will train each model using our custom dataset, which comprises images taken under challenging lighting conditions. This dataset is divided into training and validation subsets to ensure a diverse range of scenarios for both training and evaluation. Stochastic gradient descent with momentum will be the training algorithm of choice, and batch normalization will follow each convolutional layer to stabilize the activations.

Re-training (fine-tuning) YOLO Models:

Fine-tuning a YOLO model means adjusting a pre-trained model to suit a new dataset better. This approach is more resource-efficient than training a model from the ground up. The steps and strategy are as follows:

1. **Model Selection:** A YOLO model, best suited for our training needs, will be selected. The rationale behind this choice will be elaborated upon in the Results section.
2. **Configuration Adjustments:** The YOLO configuration file will be modified to accommodate the four classes in our dataset: 'car', 'person', 'motorcycle', and 'bus'. Other parameters, such as learning rate, batch size, and subdivisions, will be adjusted accordingly.
3. **Transfer Learning:** The model will be initialized with weights from either our previous training sessions or those provided by the original authors, excluding the final detection layer. This strategy ensures the model retains its feature extraction capabilities while adapting to the new dataset.
4. **Training:** The model will be trained on our dataset, with metrics like loss and average precision being closely monitored.

5. **Evaluation:** Post-training, the model's performance will be assessed on a validation set using metrics such as mAP (mean Average Precision).

Hardware and Software Configurations: All training sessions will be conducted on Google Colab in GPU mode, a platform that offers robust computational capabilities without any associated costs. We chose Google Colab for its superior GPU support, which is crucial for deep learning tasks. Due to its adaptability and comprehensive support, we will utilize the PyTorch deep learning framework for the YOLO series.

Hyperparameter Tuning and Optimization: Initial hyperparameter values will be sourced from the official YOLO documentation. To fine-tune the models, we will conduct a grid search over various hyperparameters, such as learning rates, weight decay values, and momentum rates. Early stopping will be implemented based on the validation loss to prevent overfitting.

Epoch Selection: The number of training epochs will be determined iteratively. After training the models, we will analyze their loss curves to decide if additional training is beneficial. This iterative method ensures model convergence and optimal performance.

Currently, the YOLOv8 and YOLOv7 models have completed training and initial evaluations. We plan to apply the same methodology to YOLOv6 and YOLO-NAS to maintain consistency in the training process across all models.

0.3.6 Evaluation Metrics

To evaluate the performance of our models, we will rely on several established metrics, including Precision, Recall, mAP (mean Average Precision), and the F1 score. These metrics are standard in object detection and will provide a comprehensive understanding of model accuracy and robustness.

Precision

Precision is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

where TP represents True Positives, and FP represents False Positives.

Recall

Recall, also known as Sensitivity or True Positive Rate, is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

where FN represents False Negatives.

mAP (mean Average Precision)

The average precision (AP) for a class is the average of precision values at different recall levels:

$$\text{AP} = \frac{1}{11} \sum_{r \in \{0.0, 0.1, \ldots, 1.0\}} p_{\text{interp}}(r)$$

where $p_{\text{interp}}(r)$ is the maximum precision for a given recall value r over all recall thresholds.

The mAP is then the mean of APs over all classes:

$$\text{mAP} = \frac{1}{C} \sum_{i=1}^{C} \text{AP}_i$$

where C is the number of classes and AP_i is the average precision for class i.

F1 score

The **F1 score** is a widely used metric for binary classification problems, especially when the classes are imbalanced. It is the harmonic mean of precision and recall, providing a balance between the two. Precision, denoted as P, is the ratio of correctly predicted positive observations to the total predicted positives. It is given by:

$$P = \frac{TP}{TP + FP}$$

where TP is the number of true positives and FP is the number of false positives. Recall, also known as sensitivity or true positive rate, denoted as R, is the ratio of correctly predicted positive observations to all the actual positives. It is defined as:

$$R = \frac{TP}{TP + FN}$$

where FN is the number of false negatives. The F1 score is then calculated as follows:

$$F1 = 2 \times \frac{P \times R}{P + R}$$

A perfect model would have an F1 score of 1, while a model with either precision or recall of zero will have an F1 score of 0. It is a robust metric to gauge the balance between precision and recall.

0.3.7 Deployment Plans

Our deployment strategy encompasses a blend of open-source and proprietary tools, each tailored to specific facets of the deployment process:

- **Google Colab:** A free cloud-based platform by Google, offering GPU runtime mode, which has been instrumental for training our deep learning models.
- **TensorFlow Serving:** An open-source tool by Google, tailored for serving TensorFlow-based machine learning models.
- **Docker:** Ensures consistent model runtime across different environments by containerizing the models and their dependencies.
- **NVIDIA Triton Inference Server:** Optimized for deploying AI models at scale, it supports TensorFlow, PyTorch, and other frameworks, leveraging GPU acceleration.
- **Streamlit:** An open-source framework for rapidly creating web applications, which will serve as the interface for our deployed YOLO models.
- **Roboflow:** Facilitates computer vision workflows, especially in data processing, augmentation, and labeling.
- **Weights and Biases (W&B):** A platform for tracking experiments, versioning datasets, and optimizing models, ensuring a streamlined training process and optimal model performance.

Continuous Integration and Continuous Deployment (CI/CD): To ensure seamless updates and improvements to the models, we'll establish a CI/CD pipeline using:

- **Jenkins:** An open-source automation server for reliable build and delivery.
- **GitLab CI/CD:** Integrated within GitLab, it automates scripts, ensuring continuous integration, delivery, and deployment.
- **Travis CI:** A cloud-based service that integrates with GitHub, automating the build and testing process.

Monitoring and Maintenance: For real-time performance tracking of the deployed models, we'll employ:

- **Prometheus:** An open-source toolkit for monitoring and alerting, collecting real-time metrics from the models.

- **Grafana:** Integrates with Prometheus, offering visualization dashboards for insights into model performance.
- **ELK Stack:** Comprising Elasticsearch, Logstash, and Kibana, it aids in structured logging and visualization.

For the YOLO models, dependencies include:

- TensorFlow or PyTorch (based on the YOLO implementation)
- CUDA and cuDNN for GPU acceleration
- OpenCV for image processing
- NumPy and Pandas for data manipulation
- Python libraries like `torchvision` for PyTorch-based YOLO.

Ensuring these dependencies are correctly installed and configured in the deployment environment is crucial.

0.3.8 Project Duration

The project is slated to span four months, commencing from July 2023 and culminating in November 2023. This timeframe has been meticulously planned to ensure that each phase of the project receives adequate attention and ample time for testing, feedback, and refinements.

Chosen Project Methodology

After carefully considering the project's requirements, scope, and objectives, the **Agile** methodology has been chosen as the most suitable approach for this project. The reasons for this choice are manifold:

- **Flexibility and Adaptability:** Agile allows for changes after the initial planning. As the project progresses, adjustments can be made in response to challenges encountered or new requirements that may arise.
- **Incremental Progress:** The project will be broken down into smaller, manageable units or sprints. A functional module or feature will be delivered at the end of each sprint, ensuring continuous progress and regular feedback.
- **Stakeholder Engagement:** Agile emphasizes regular communication with stakeholders (in our case, supervisors). This ensures that the project aligns with their expectations and requirements and that any changes or feedback can be promptly addressed.
- **Risk Management:** Regular reviews at the end of each sprint mean that potential issues or risks are identified and addressed early, reducing the impact on the project.
- **Quality Assurance:** Continuous testing and review ensure that the quality of the project is maintained throughout its duration.

While the Waterfall methodology was also considered, given its linear and phase-based approach, it was deemed less suitable for this project. The dynamic nature of the project's requirements and the need for regular feedback and adaptability made Agile a more fitting choice.

In conclusion, combining a well-defined project duration and the Agile methodology ensures the project's successful and timely completion, meeting all set objectives and expectations.

0.3.9 Project Milestones and Deliverables

In the course of this project, we've established a series of milestones and deliverables to ensure timely progress and to meet the project's objectives. The following outlines these milestones, their corresponding deliverables, and any dependencies:

0.3.10 Milestones

1. **Initial Design and Setup**
   - Deliverable: A preliminary design document outlining the project's scope and objectives.
   - Dependency: None.
2. **Training of YOLO Models**
   - Deliverable: Trained YOLOv6, YOLOv7, YOLOv8, and YOLO-NAS models.
   - Dependency: A curated and augmented dataset.
3. **Tool Selection and Setup**
   - Deliverable: A functional development environment with tools like Streamlit, Google Colab, and version control systems in place.
   - Dependency: Initial design and setup.
4. **Data Collection and Augmentation**
   - Deliverable: A comprehensive dataset ready for model training.
   - Dependency: None.
5. **Streamlit Development**
   - Deliverable: A working prototype of the Streamlit application.
   - Dependency: Training of YOLO models.
6. **Final Presentation and Documentation**
   - Deliverable: A comprehensive report and presentation detailing the project's outcomes.
   - Dependency: All previous milestones.

Dependencies

The project's success hinges on the timely completion of each milestone, as several milestones are dependent on the outputs of previous ones. For instance, the training of YOLO models is contingent on the availability of a curated dataset. Similarly, the development of the Streamlit application is dependent on the successful training of the YOLO models.

To manage these dependencies and ensure the project stays on track, we've employed tools like Trello for time management and GitHub for version control. The Gantt charts, as discussed earlier (refer to Figures 6, 7,8,9 in the Appendix), provides a visual representation of the project timeline, milestones, and dependencies.

0.4 Results

The results section explains in detail the performance and findings derived from the methodologies employed in this research. Through rigorous training and evaluation, Our goal was to determine the impact of different datasets on model performance, the efficiency of the models in terms of computational cost and training time, and the robustness of the models when tested under challenging conditions.

0.4.1 Model Performance Analysis

Here, we will focus on the models' accuracy, precision, recall, and other metrics.

Dataset Impact on Model Performance

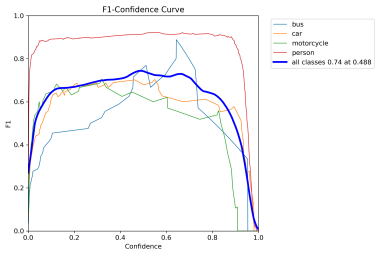
To understand the influence of the dataset on model performance, we conducted a comparative analysis of the YOLOv8 models:

- **Dataset Variations:** YOLOv8 was trained on different datasets: the original dataset comprising 516 images and the enhanced dataset with 1300 images. This comparison allowed us to gauge the effect of dataset size and diversity on detection accuracy.
- **Performance Insights:** Preliminary findings suggest that the model trained on the enhanced dataset exhibits superior performance compared to the one trained on the original dataset. This highlights the significance of a diverse and augmented dataset in crafting robust object detection models.

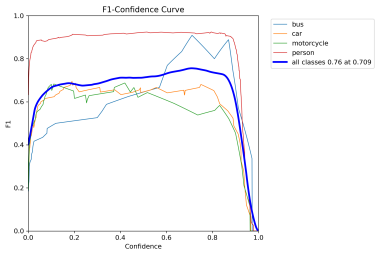
Additionally, our models were tested on a challenging dataset filled with brightly lit images. This dataset originates from a video shot on the same day as our training video but from a different perspective. Notably, it has fewer instances of the 'person' class compared to the primary dataset, where the 'person' class was predominant. Detailed results from this evaluation will be elaborated upon later in this section.

Training Visualization

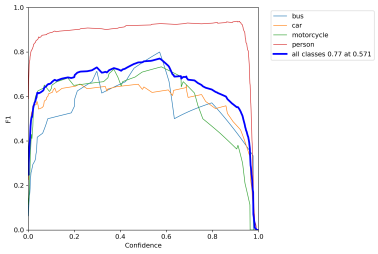
The **F1** score, a harmonic mean of precision and recall, is pivotal in evaluating model performance, particularly in scenarios with imbalanced datasets. Precision quantifies the number of correct positive predictions made, while recall assesses how many actual positives the model correctly identifies. By encapsulating both these metrics, the F1 ore offers a singular value that harmonizes the trade-off between precision and recall. In this study, we employ the F1 score to compare the performance of our models during the training phase, visualizing the F1 curve to understand the evolution of model performance over time. A higher F1 score signifies a more optimal balance between precision and recall, making it a cornerstone of our comparative analysis. A more in-depth discussion and elucidation of the F1 score was presented in the subsequent "Evaluation Metrics" section.



(a) YOLOv8-OD F1 Curve



(b) YOLOv8-ED F1 Curve



(c) YOLOv7-Base F1 Curve

Figure 5: Comparison of F1 curves during the training process for YOLOv8-OD, YOLOv8-ED, and YOLOv7-Base.

Model Selection and Evaluation

In this section, we delve deeper into the criteria and methodologies used for selecting the optimal model for our task. We will also discuss the evaluation metrics and techniques employed to assess the performance

of the fine-tuned model. During the project's initial phase, various YOLOv8 models were trained using different methods to determine the most effective approach. It was observed that models trained with pre-trained weights and loaded models produced identical results. In contrast, training a model from scratch using the '.yaml' configuration did not yield results as promising as the previous two methods. Table 1 summarizes the evaluation metrics for the different models:

Model	Epochs	Precision	Recall	mAP50	mAP50-95	Storage
YOLO('yolov8n.pt')	100	0.686	0.765	0.795	0.707	train2
YOLO('yolov8n.yaml')	100	0.773	0.711	0.753	0.566	train3
YOLO('yolov8n.yaml').load('yolov8n.pt')	100	0.686	0.765	0.795	0.707	train4
YOLO('yolov8n.yaml').load('train4/best.pt')	50	0.748	0.771	0.817	0.726	train5
YOLO('yolov8n.yaml').load('train5/best.pt')	50	0.788	0.718	0.805	0.702	train6

Table 1: Evaluation of different YOLOv8 models

The champion model was selected for further work and analysis based on the evaluation metrics.

Testing results on Challenging Bright Lighting Dataset

To further evaluate the robustness of our models, we tested them on a challenging dataset containing brightly lit images. This dataset was derived from another video recorded on the same day as the video used for our training phase. However, it was captured from a different angle and lacked many instances of the 'person' class, unlike the initial dataset where the 'person' class was overrepresented. As presented in Table 2, the results underscore the performance variations under these challenging conditions.

In model training and evaluation, striking the right balance is crucial. Balancing training time, memory consumption, and model performance is essential. While faster training times are desirable, they shouldn't come at the expense of reduced accuracy or excessive memory usage.

We have now evaluated the training time and computational cost for the YOLOv8 and YOLOv7 models. The same evaluation metrics and steps will be applied to the forthcoming YOLOv6 and YOLO-NAS models to ensure consistency in our analysis.

Table 2: Testing Results on Challenging Bright Lighting Dataset

Model	Precision	Recall	mAP50	mAP50-95
YOLOv8-OD	0.518	0.144	0.145	0.0474
YOLOv8-ED	0.957	0.133	0.196	0.0907
YOLOv8x (pre-trained)	0.000905	0.133	0.000963	0.000167

As shown in Table 2, the performance of the custom models, particularly 'YOLOv8-ED', outperforms the pre-trained YOLOv8x in specific metrics. This emphasizes the importance of custom training tailored to specific scenarios.

0.4.2 Computational Efficiency Analysis

This subsection will focus on the computational aspects of the models, including training time and memory consumption.

Training Time and Computational Cost

Training deep learning models, especially with extensive datasets, can be both time-intensive and resource-demanding. A model's efficiency is gauged by its accuracy, training speed, and computational demands.

Training Duration

The duration of training a model is pivotal, especially for iterative development. For our models:

- The YOLOv8-OD model took 0.365 h for 100 epochs and an additional 0.17 h for 50 epochs.
- The YOLOv8-ED model completed 150 epochs in 1.463 hours.
- The YOLOv7-Base finished 100 epochs in 2.281 hours.

Computational Cost and Memory Usage

Optimal memory consumption ensures that models can be trained on devices with limited resources, such as personal computers or free-tier platforms like Google Colab. The memory usage for our models was:

- The YOLOv8 models utilized 2.8-3 G of memory during training.
- The YOLOv7 model required 8.64 G of memory.

The results comprehensively explain the model's performance under various conditions and configurations. These findings set the stage for an in-depth discussion of this research's implications, potential applications, and future directions. The subsequent "Discussion" section will delve into these aspects, providing a holistic view of the study's significance and potential impact.

0.5 Discussion

The YOLO series has been instrumental in advancing object detection capabilities in computer vision. The series has consistently improved architecture and performance, starting with the original YOLO(You Only Look Once) model, which revolutionized the field by combining bounding box prediction and classification. By the time YOLOv8 and YOLO-NAS were introduced in 2023, the series had incorporated numerous advancements, leading to significant improvements in throughput and accuracy, especially compared to its predecessors.

In our study, we primarily examined the YOLOv8 and YOLOv7 models. Both models showcased exemplary performance, particularly under challenging lighting conditions, underscoring their robustness. Notably, YOLOv8 exhibited quicker training durations and was more computationally efficient. Yet, its performance metrics closely mirrored those of YOLOv7 despite the latter's higher memory consumption during training. This observation is consistent with the findings of Terven and Cordova-Esparza, 2023. Specifically, YOLOv7-E6 registered an AP of 55.9% using a 1280-pixel input size and operated at 50 FPS on an NVIDIA V100. In contrast, YOLOv8x achieved an AP of 53.9% with 640-pixel input size, reaching a speed of 280 FPS on an NVIDIA A100 with TensorRT.

0.5.1 Influence of Batch Size and Training Strategy

The batch size and training strategy are important when training a model. The batch size is how many samples we look at in each training step. A bigger batch size can help the model learn faster and better, but it also uses more computer memory. A smaller batch size uses less memory but might learn slower and not as well.

Our study used batch sizes 16 for YOLOv8 and 8 for YOLOv7. Even though YOLOv7 had less batch size, its memory usage was three times as much as the YOLOv8 model during training. This matches what "A Review of Yolo Algorithm Developments", 2022 found about the YOLO models.

The training strategy includes how we adjust the model during training, how fast we learn, and if we change the training data in any way. According to Hussain, 2023, YOLO-v2 added batch normalization to help the model learn better and faster. YOLO-v3 brought new ways to change the training data and improve training methods. These changes show that the batch size and training strategy can change how well YOLO works. However, more studies might be needed to determine how batch size and training strategy affect YOLO.

.

0.5.2 Dataset Size and Quality: Implications on Performance

Our study also delved into the influence of dataset size and quality on model performance. We trained the YOLOv8 model on an original dataset comprising 516 images and an enhanced dataset with 1300 images. The enhanced dataset, augmented using various techniques, yielded better results, underscoring the importance of dataset quality.

The dataset's size and quality can profoundly impact the performance of object detection algorithms like YOLO. As noted by "A Review of Yolo Algorithm Developments" in their study, a larger and more diverse dataset can aid the algorithm in learning more generalized features, thereby enhancing its accuracy. Conversely, a smaller and less varied dataset might lead the model to overfit, resulting in subpar generalization.

Furthermore, the quality of the dataset is paramount. Datasets with low-quality images or inaccurate annotations can affect the algorithm's performance detrimentally. For instance, low-resolution images or poor lighting conditions can impede the algorithm's ability to detect objects accurately. Similarly, erroneous or incomplete annotations can mislead the algorithm, causing it to learn incorrect features or miss detecting objects entirely "A Review of Yolo Algorithm Developments", 2022.

To bolster the YOLO algorithm's performance, utilizing a comprehensive and diverse dataset replete with high-caliber images and precise annotations is imperative. Additionally, leveraging data augmentation techniques can amplify the dataset's size and diversity, further enhancing the algorithm's generalization capabilities.

This observation is consistent with the findings of the mentioned paper, which emphasized the significance of high-quality datasets in achieving optimal model performance.

0.5.3 Future Directions

With its continuous advancements and adaptability, the YOLO series has emerged as a cornerstone in object detection, particularly in autonomous driving. While primarily centered on YOLOv8 and YOLOv7, our study offers a snapshot of these models' potential in real-world driving scenarios. However, our research endeavors do not end here. We aim to broaden our investigation to encompass YOLOv6 and YOLO-NAS. By maintaining a consistent methodology and evaluation metrics, we aspire to ensure a thorough comparison across all models. Such an expanded analysis is anticipated to furnish a more holistic understanding of the YOLO series. This, in turn, will be instrumental in aiding researchers and practitioners in the autonomous driving domain to pinpoint the most apt model tailored to their specific requirements.

In the next phase of our project, we plan to develop a web application that will serve as an interactive platform for users to visualize and compare the performance metrics of different YOLO models. This application will facilitate a more intuitive understanding of the models' capabilities, assist in model selection based on specific use cases, and provide real-time feedback on object detection results.

As the landscape of autonomous driving evolves, we believe that our research, complemented by the upcoming web application, will serve as a guiding light, illuminating the path for future explorations and applications in computer vision.

Bibliography

AI. (2023). Ultralytics — revolutionizing the world of vision ai. https://ultralytics.com/yolov8

Facts, I. (2022). Advanced driver assistance systems-data details [Accessed: 2023-07-25]. https://injuryfacts.nsc.org/motor-vehicle/occupant-protection/advanced-driver-assistance-systems/data-details/

Hiziroglu, A. B. (2020). The car: A brief history. In *Autonomous vehicles and the law: How each field is shaping the other* (pp. 1–10). Springer.

Hussain, M. (2023). Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, *11*(7), 677–677. https://doi.org/10.3390/machines11070677

Levin, S., & Wong, J. C. (2018). Self-driving uber kills arizona woman in first fatal crash involving pedestrian. Retrieved July 25, 2023, from https://www.theguardian.com/technology/2018/mar/19/uber-self-driving-car-kills-woman-arizona-temp

Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W., Li, Y., Zhang, B., Liang, Y., Zhou, L., Xu, X., Chu, X., Wei, X., & Wei, X. (2022). Yolov6: A single-stage object detection framework for industrial applications. *ArXiv.org*. https://arxiv.org/abs/2209.02976

A review of yolo algorithm developments [Scholarsportal.info]. (2022). https://journals.scholarsportal.info/details/18770509/v199icomplete/1066_aroyad.xml

Sirisha, U., Praveen, S., Srinivasu, P. N., Barsocchi, P., & Bhoi, A. K. (2023). Statistical analysis of design aspects of various yolo-based deep learning models for object detection. *International Journal of Computational Intelligence Systems*, *16*(1). https://doi.org/10.1007/s44196-023-00302-w

Terven, J., & Cordova-Esparza, D. (2023). A comprehensive review of yolo: From yolov1 and beyond. *ArXiv.org*. https://arxiv.org/abs/2304.00501

Wang, C.-Y., Bochkovskiy, A., & Liao, H.-Y. M. (2022). Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.

Appendix

Project Schedule

Detailed Gantt charts outlining the deployment strategy and timelines are provided below:
Please click here to view my Trello board.

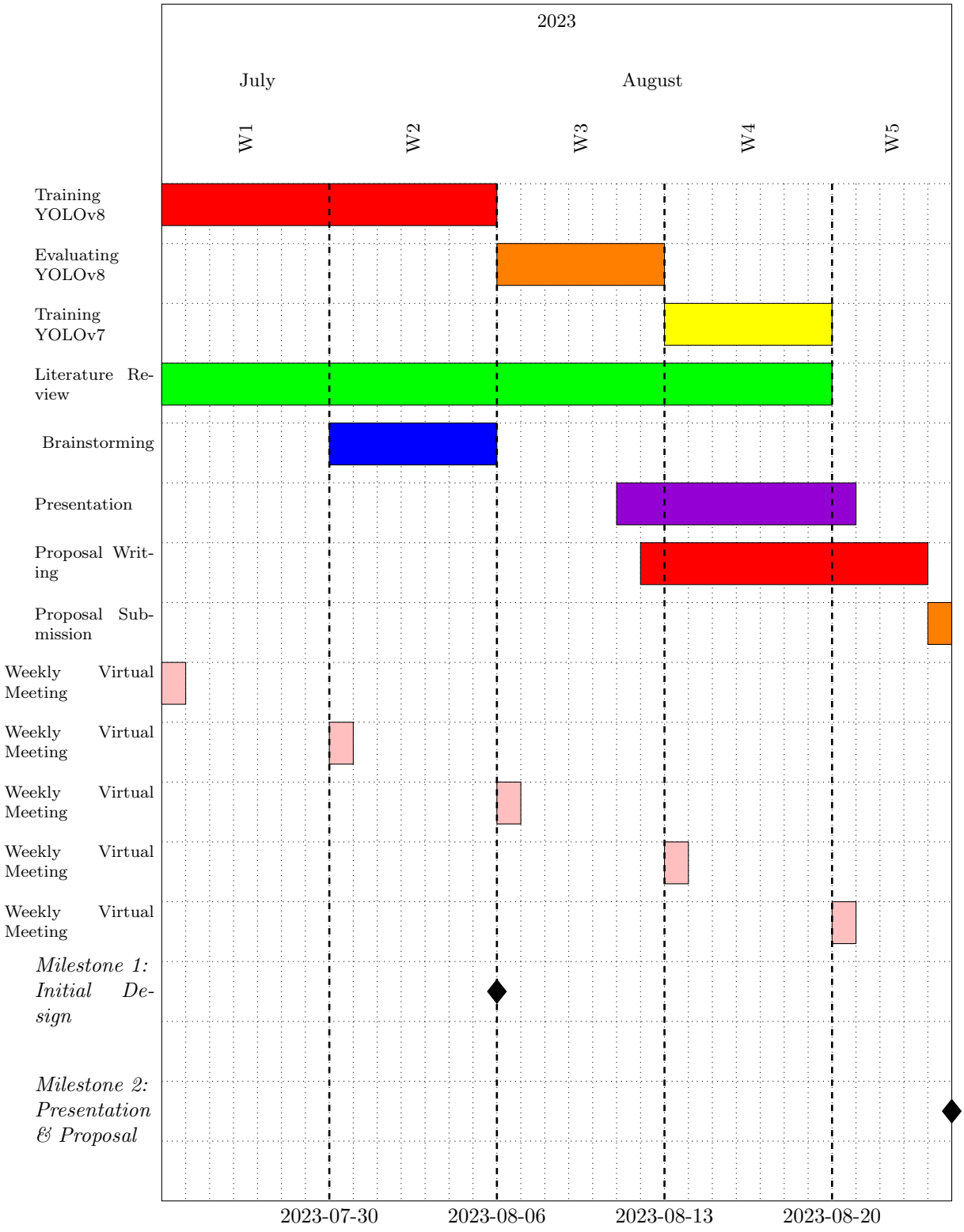


Figure 6: Gantt Chart for weeks 1 to 5

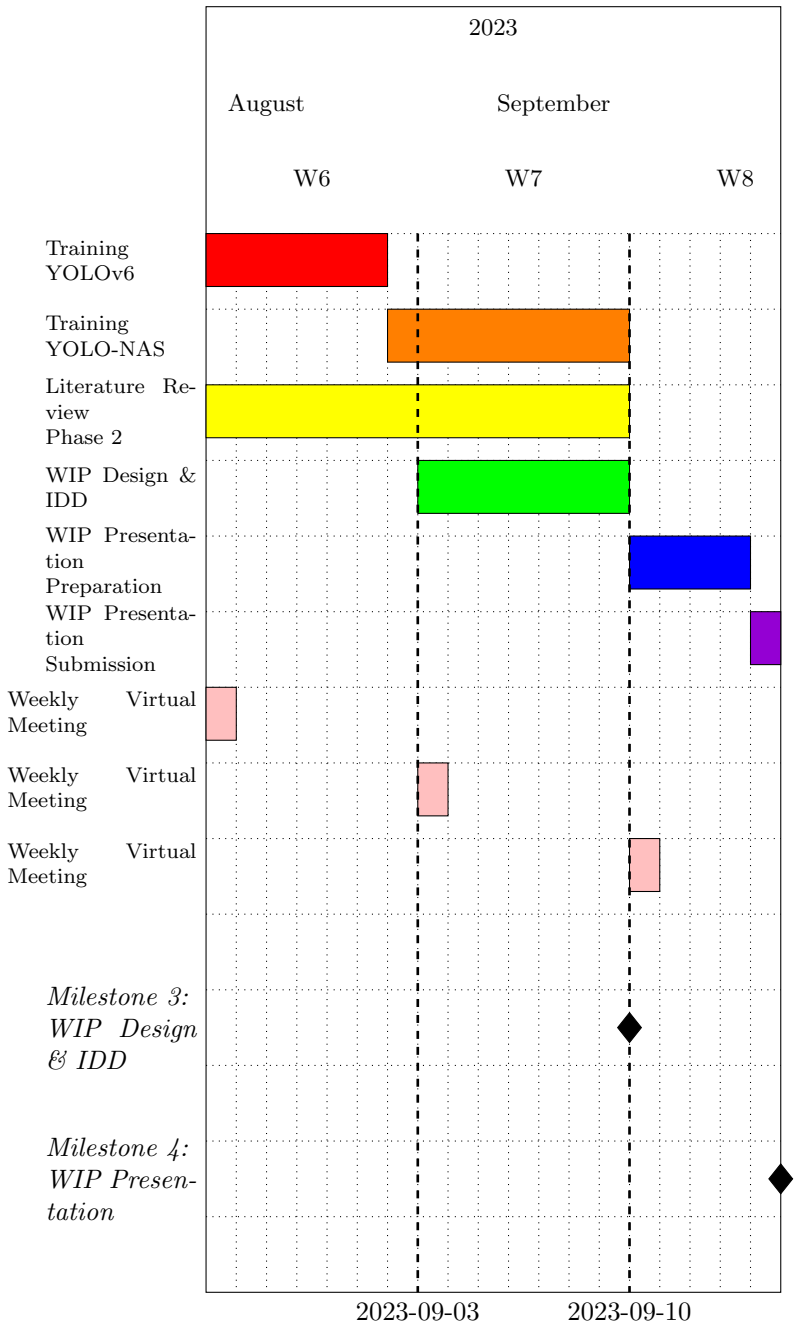


Figure 7: Gantt Chart for weeks 6 to 8

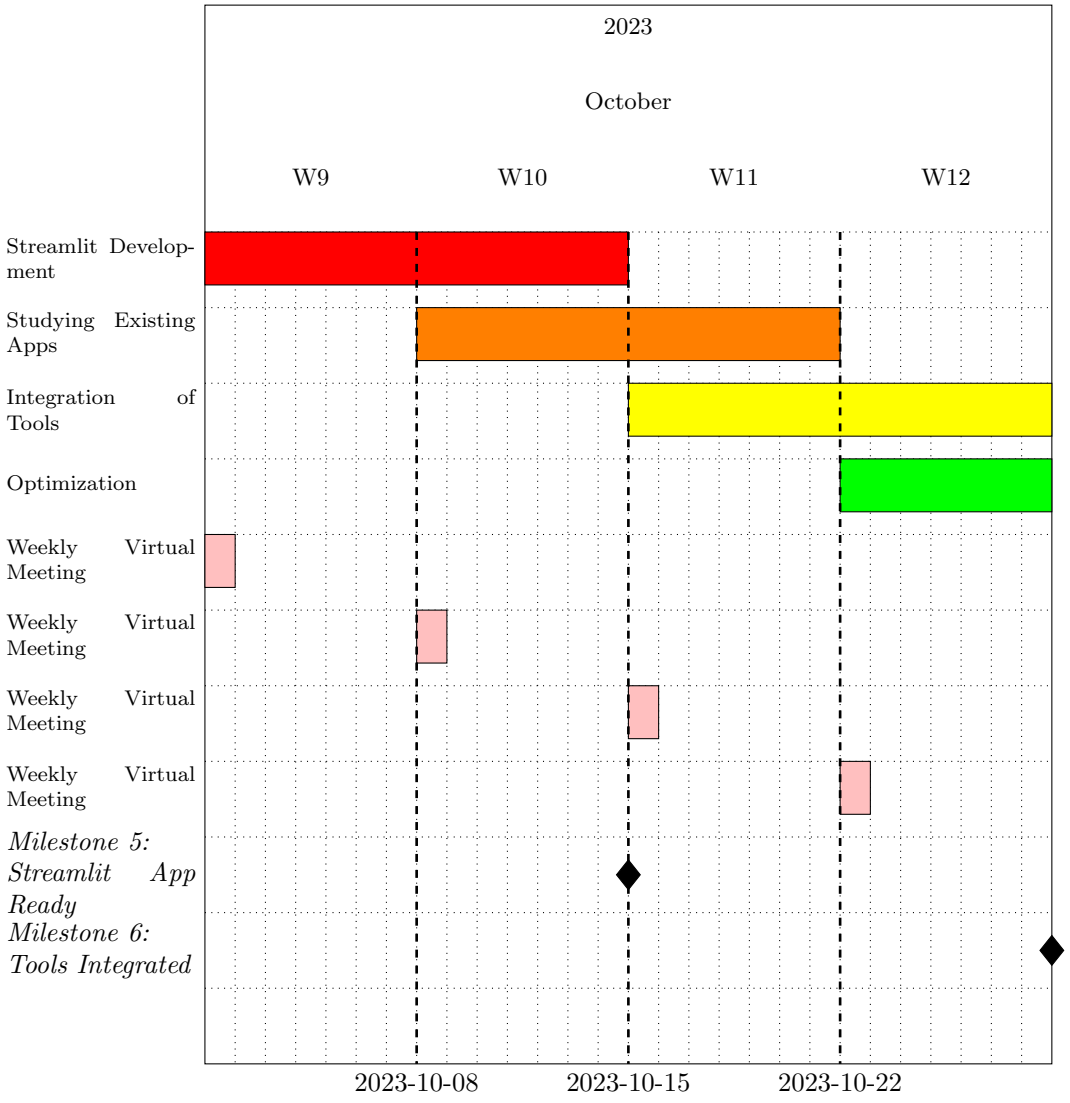


Figure 8: Gantt Chart for weeks 9 to 12

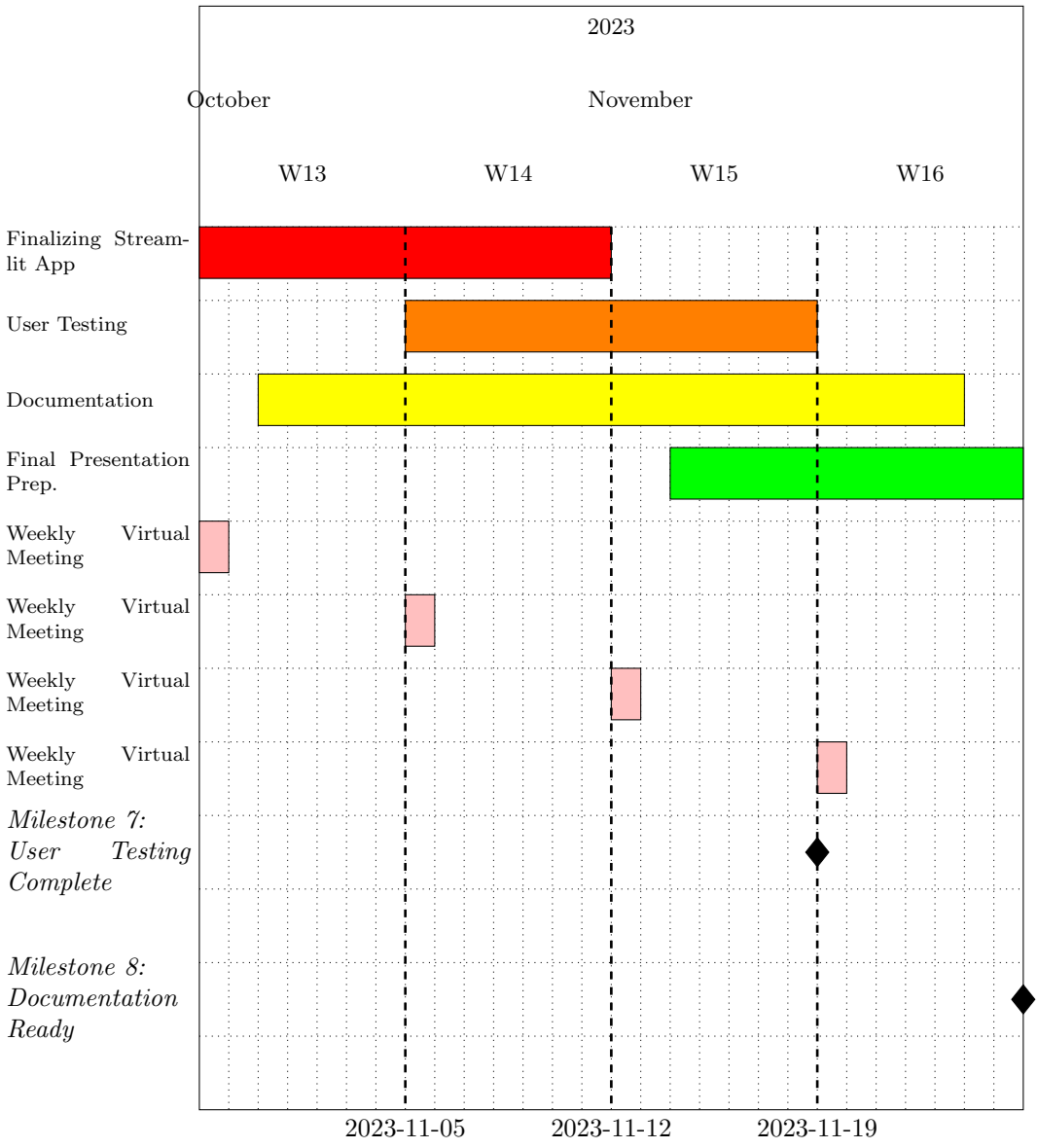


Figure 9: Gantt Chart for weeks 13 to 16