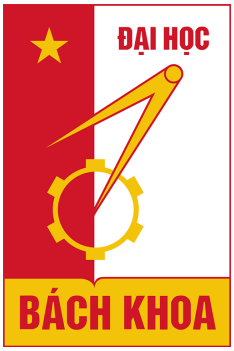
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



INTRODUCTION TO DEEP LEARNING

COURSE: IT3320E

A Journey Through YOLO Versions for Aircraft Detection in Satellite Imagery

Author
NGUYEN HAI DANG
NGUYEN TIEN DAT
PHAN MINH HOA
NGUYEN VO NGOC KHUE
NGUYEN HOANG QUAN

Mentor
NGUYEN HUNG SON
TRAN VIET TRUNG

December 2024

Contents

Abstract

Aircraft detection is an old problem, whose result could be beneficial for both civil and military organizations. With pictures provided by advanced satellite technology, the detection algorithms are also racing to give better and better result. This project aim to explore and evaluate YOLO versions using GDIT Aerial Airport Dataset. We are using several metrics but mainly mAP to compare these model. The result indicated that YOLOv8, with multiple technical solution applied, performed the best, emphasizing the importance of technical advancements in solving real-life problems. in object detection tasks.

1. Introduction

Research in aircraft detection from satellite imagery has advanced significantly, driven by improvements in remote sensing and machine learning techniques. Traditional detection systems like radar, Radio Frequency (RF) detection or Multilateration (MLAT) while effective, often suffer from limited coverage and weather interference. They also tend to give low-resolution information and require collaborative synchronization from multiple stations, which is much less effective than using satellite imagery. High-resolution satellite images now allow for detailed analysis of the Earth's surface, especially critical infrastructures like airports or military bases, which enables the automatic detection and classification of aircraft under various conditions. This capability is crucial for national security, where it enhances airspace monitoring, and for disaster safety management, providing rapid assessments in crisis situations, as well as the potential for real-time surveillance. Additionally, it contributes to environmental studies by tracking the impacts of aviation on sensitive ecosystems. As satellite technologies and algorithms continue to evolve, the precision and scalability of aircraft detection methods are expected to improve further, making them indispensable tools in both civil and military applications.

Recently, deep learning has emerged to become a very effective tool for applications in computer vision [9]. Deep convolutional neural networks have been proven very effective in extracting features from digital images, which is extremely useful in classification and detection [10]. A common approach is to combine machine learning techniques with the use of Synthetic-aperture Radar (SAR) to generate informative images of aircrafts in adverse environments [1], [2]. The works in [3] and [4] also utilized the use of pyramid networks to handle multiscale features in SAR imaging in aircraft detection. Another method involves using remote sensing images to capture data across different spectral bands and varying resolutions for inference in [5], [6], [7] and [8]. Especially in [5] and [6], a YOLO (You Only Look Once) model was leveraged and further improved specifically for aircraft detection. These researches have one thing in common is to utilize a machine learning model combined with image processing techniques or image capturing techniques to enhance the detection ability of that model. YOLO models, renowned for its real-time object detection capabilities, can be used as an effective tool for the task of analyzing extensive real-time satellite imagery. It has proven to be successful in multiple major fields regarding autonomous vehicles, tracking, detection and surveillance. The GDIT dataset, with its high-resolution images and diverse

aircraft annotations, has become a critical benchmark for evaluating these models. This dataset is particularly challenging due to the variety of aircraft types, orientations, and complex backgrounds present in the satellite images, which closely mirror real-world conditions.

Other models such as Faster R-CNN, known for its two-stage detection process, have been employed to enhance detection accuracy by first proposing regions of interest and then classifying them. CenterNet, which operates on a keypoint-based approach, and RetinaNet, which uses focal loss to address class imbalance issues, have also been benchmarked on HRPlanesV2 which is also the aircraft dataset like GDIT. Additionally, DETR (DEtection TRansformers), with its transformer-based architecture, represents a modern approach to object detection, offering robustness against the variability seen in satellite imagery. These models, through extensive testing on HRPlanesV2, provide a comprehensive view of the current capabilities and limitations in the field of aircraft detection from satellite imagery .

The focus of this study is to evaluate the performance of YOLO models from versions 1 to 8 on the GDIT dataset, a collection of aircraft images taken from satellites. This project provides a comprehensive review of the YOLO series' evolution and their respective architectures, followed by an analysis of their performance on this specific dataset. By doing so, the study aims to identify the strengths and limitations of each model in the context of satellite-based aircraft detection, offering insights for future advancements in the field.

The remainder of the paper is organized as follows: Section II provides a comprehensive overview of the YOLO series architecture. Section III demonstrates the experimental setup and discusses the results obtained from training and testing the models. Finally, Section IV concludes with a summary and evaluation of the experiments.

2. Methodology

2.1 Some Basic Concepts

2.1.1 Intersection over Union (IoU)

Intersection over Union (IoU) is a standard metric used to quantify the overlap between a predicted bounding box and the ground truth bounding box. It is computed as the ratio of the area of intersection to the area of union between the two boxes. This metric is fundamental in object detection, as it provides a consistent measure for evaluating localization accuracy. A higher IoU indicates a better alignment between the predicted and ground truth boxes.

In practice, IoU is used in evaluation protocols such as AP50, where a prediction is considered correct if the IoU exceeds 0.5. It also forms the basis of advanced loss functions such as Generalized IoU (GIoU) and Complete IoU (CIoU), which incorporate factors like box size and aspect ratio to refine predictions further. For example, GIoU extends IoU by penalizing non-overlapping regions, while CIoU adds a penalty for aspect ratio inconsistencies.

2.1.2 Non-Maximum Suppression (NMS)

Non-Maximum Suppression (NMS) is a critical post-processing step used to eliminate redundant bounding box predictions. During detection, models often produce multiple overlapping boxes for the same object, each with a confidence score. NMS ensures that only the box with the highest confidence is retained, while others with high overlap (measured by IoU) are suppressed.

The NMS process involves sorting all predicted boxes by confidence scores, selecting the box with the highest score, and discarding any remaining boxes with an IoU greater than a predefined threshold (e.g., 0.5). This procedure is repeated until all boxes are processed. By reducing redundancy, NMS streamlines the detection results and enhances their interpretability, ensuring that each object is represented by a single bounding box.

2.1.3 Bag of Freebies (BoF)

Bag of Freebies (BoF) refers to training techniques that improve the model's accuracy without increasing inference cost. These techniques are typically applied during training and enhance the generalization capability of the model.

For the backbone network, BoF methods include data augmentation strategies like Mosaic and CutMix, which increase the diversity of the training data. Label smoothing, another BoF technique, mitigates overconfidence in predictions by softening class probabilities during training. For the detector, techniques like grid sensitivity elimination and CIoU loss improve localization accuracy by addressing common issues in bounding box prediction. These methods enhance the detector's performance during training but do not add computational overhead during inference.

2.1.4 Bag of Specials (BoS)

Bag of Specials (BoS) includes architectural enhancements and post-processing techniques that slightly increase inference cost but significantly boost detection performance. These components are carefully designed to optimize specific aspects of the detection pipeline.

In the backbone, Cross-Stage Partial Networks (CSP) improve feature reuse and reduce computational redundancy, while activation functions like Mish enhance non-linearity. For the detector, Spatial Pyramid Pooling (SPP) captures multi-scale features by pooling at different kernel sizes, improving object detection across various scales. Path Aggregation Networks (PAN) refine feature propagation and fusion, particularly for small object detection. These architectural improvements enable the model to balance computational efficiency with enhanced accuracy.

2.2 Series of YOLO versions

2.2.1 YOLOv1

Summary:

YOLO V1, the first in the 'You Only Look Once' series, was developed by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi and officially released on June 8, 2015. It marked a groundbreaking shift in object detection by unifying the detection process into a single-stage model. Prior to YOLO, most object detection frameworks relied on two-stage models, such as R-CNN, which separated object classification and localization into distinct steps. YOLO V1 challenged this paradigm by performing classification and localization simultaneously in a single neural network pipeline, making it extremely efficient. The model demonstrated the potential of real-time object detection, achieving 45 FPS on a single GPU, a feat previously unseen.

Architecture:

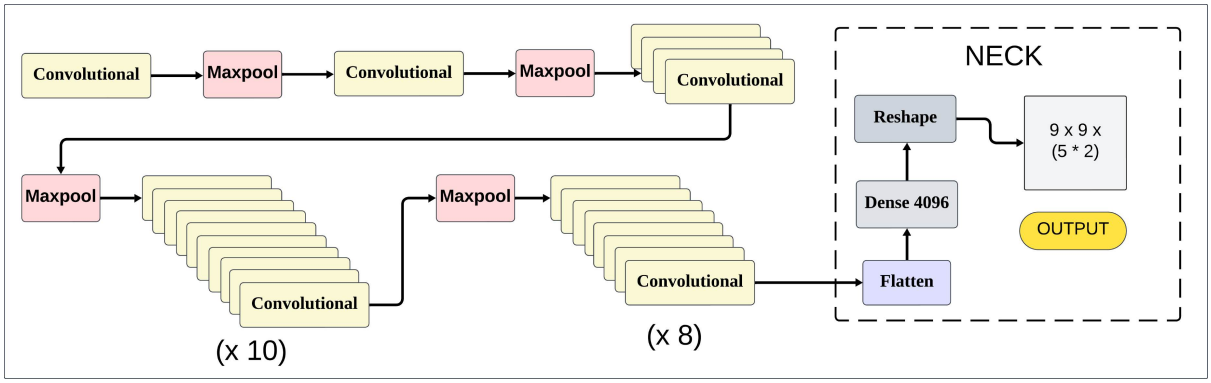


Figure 1: Yolov1's architecture

- Backbone: The architecture of YOLO V1 consisted of a 24-layer convolutional network inspired by GoogLeNet. This network served as the backbone for feature extraction, capturing spatial and contextual information from the entire input image. Despite its simplicity, the backbone provided a balance between computational efficiency and feature extraction capabilities.

- Head: The detection head comprised two fully connected layers. These layers were responsible for bounding box regression and class probability predictions. The final output of the model had dimensions S × S × (B . 5 + C), where S = 7 (grid size), B = 2 (bounding boxes per grid), and C = 20 (number of classes). This configuration allowed the model to predict two bounding boxes per grid cell, each associated with a confidence score and class probabilities.

- Grid Prediction: YOLO V1 divided the input image into a fixed grid of S × S (7 × 7) cells. Each cell was responsible for predicting: Bounding box center coordinates (x, y), width (w), and height (h). A confidence score (C) representing the likelihood of an object

being present in the bounding box and the accuracy of the prediction. Class probabilities (Pc) for each of the 20 classes in the PASCAL VOC dataset. Each grid cell predicted class probabilities independently, which contributed to the overall classification and localization task.

Bag of Freebies (BoF)

YOLO V1 did not employ a pretrained backbone or advanced regularization techniques commonly used today. However, it introduced several innovative practices to improve training and generalization:

- Data Augmentation: Techniques such as random cropping, scaling, and color jittering were used to artificially expand the training data, reducing the risk of overfitting.
- Loss Function: The loss function integrated three components: Regression Loss: Penalized deviations in predicted bounding box dimensions (w, h) and coordinates (x, y). Confidence Loss: Encouraged high confidence scores for grid cells containing objects and suppressed scores for cells without objects. Classification Loss: Penalized incorrect class predictions for grid cells containing objects.

Bag of Specials (BoS)

YOLO V1 utilized ReLU activation functions, enabling it to capture complex patterns effectively.

The entire image was processed in one forward pass, enabling the model to capture global contextual features and reduce false positives caused by incomplete object views.

Key Innovations:

- YOLO V1 pioneered the use of a single neural network for real-time object detection, combining localization and classification in one streamlined process.
- Achieved 45 FPS on a single GPU, setting a benchmark for real-time performance.
- Simplified the training pipeline, requiring only a single neural network instead of separate modules for classification and localization.

Limitations:

- YOLO V1's grid-based design limited each cell to predicting one object, leading to challenges in detecting overlapping objects.
- Detection of small objects was hindered by the coarse grid resolution (7×7).
- The requirement for fixed input dimensions (448×448) reduced flexibility and adaptability to images with varying resolutions.
- Struggled with objects having unusual aspect ratios or configurations unseen during training.

2.2.2 YOLOv2

Summary:

YOLO V2, also known as YOLO9000, was introduced 14 months after YOLO V1 by Joseph Redmon and Ali Farhadi. It aimed to address the shortcomings of its predecessor while maintaining its real-time performance. YOLO V2 achieved significant improvements in accuracy, class detection, and the ability to handle varying object sizes, making it a major milestone in the YOLO series. This model also expanded its scope to detect 9000 classes by leveraging hierarchical classification and multi-task learning.

Architecture:

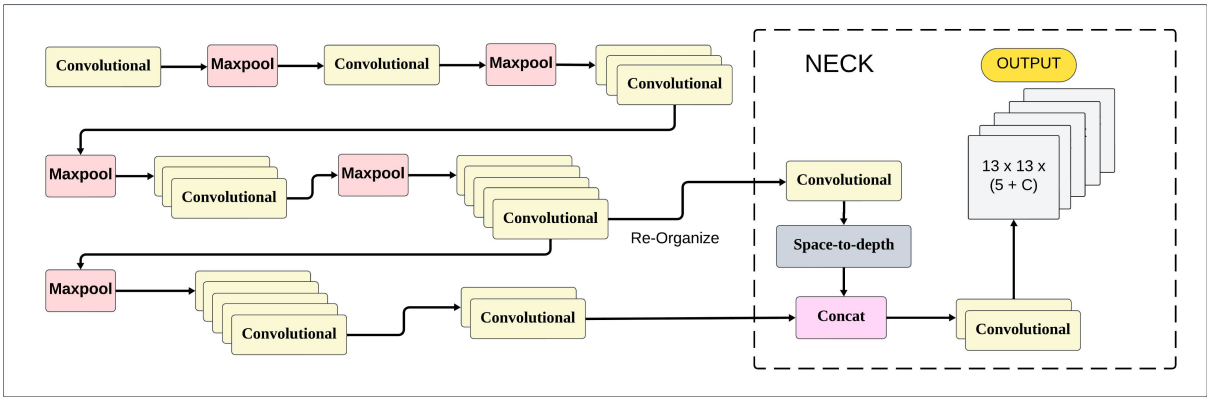


Figure 2: Yolov2's architecture

- Backbone: YOLO V2 introduced Darknet-19, a lightweight and efficient network with 19 convolutional layers and 5 max-pooling layers. This architecture was designed to balance computational efficiency with robust feature extraction.
- Head: YOLO V2 increased the grid size to 13 × 13 using skip connections. This enabled the model to predict objects with greater precision. Anchor boxes were introduced, which were predefined bounding box dimensions based on the training dataset. These allowed the model to predict objects of varying scales and aspect ratios more effectively. Bag of Freebies (BoF). Batch normalization was added to improve training stability and reduce overfitting. Pretraining on the ImageNet dataset provided a strong initialization, significantly enhancing performance on object detection tasks. Random resizing of input images during training improved the model's robustness to scale variations.

Bag of Specials (BoS)

- YOLO V2 introduced IoU-aware confidence scoring, which refined the objectness predictions and improved overall detection accuracy.
- Multi-scale predictions enabled the model to detect small and large objects more effectively by integrating fine-grained features.

Key Innovations:

- Anchor boxes addressed the limitations of YOLO V1 in handling objects of varying sizes and aspect ratios.
- Multi-scale training enhanced the model's adaptability to diverse input resolutions.
- The hierarchical classification system enabled the detection of up to 9000 classes, leveraging relationships between classes.

Limitations:

- The model remained reliant on carefully chosen anchor box dimensions for optimal performance.
- Localization errors from YOLO V1 were not entirely resolved, although significantly reduced.
- The hierarchical classification approach sometimes led to trade-offs between accuracy and speed.

2.2.3 YOLOv3

Summary:

YOLO V3, released on April 8, 2018, by Joseph Redmon and Ali Farhadi, was described by its creators as having minor design changes compared to YOLO V2. Despite these modest adjustments, YOLO V3 introduced significant enhancements in multi-scale detection and classification, making it one of the most robust models in the YOLO family.

Architecture:

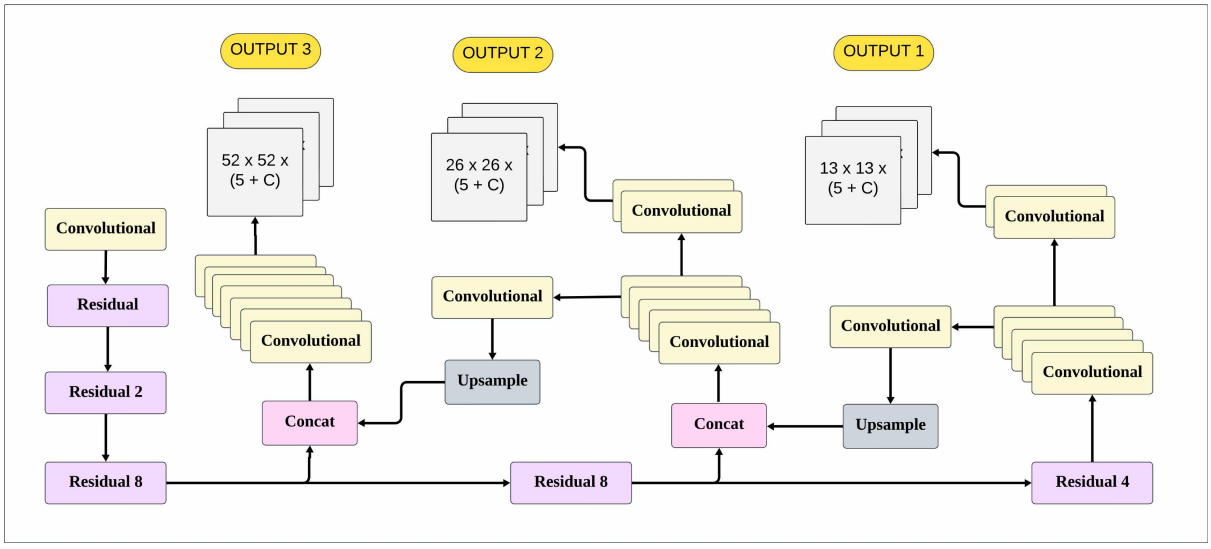


Figure 3: Yolov3's architecture

- Backbone: YOLO V3 replaced Darknet-19 with Darknet-53, which introduced residual connections and strided convolutions. This architecture provided deeper and more efficient feature extraction, significantly improving accuracy while maintaining speed.
- Neck: YOLO V3 incorporated a Feature Pyramid Network (FPN)-style neck, aggregating features from multiple scales using upsampling and concatenation layers. This design enabled better detection of objects at varying scales.
- Head: The detection head generated predictions at three scales, corresponding to large, medium, and small objects. Each scale was optimized to handle objects of specific sizes, refining bounding box and class predictions.

Bag of Freebies (BoF)

- Similar to YOLO V2, YOLO V3 utilized multi-scale training with three different image sizes to improve robustness.
- A new loss function was introduced, replacing softmax loss with binary cross-entropy for multi-label classification, which better supported complex datasets.

Bag of Specials (BoS)

- The addition of residual blocks in Darknet-53 enhanced feature extraction by improving gradient flow and reducing vanishing gradient issues.
- Spatial Pyramid Pooling (SPP) and Feature Pyramid Network (FPN) modules allowed YOLO V3 to excel at detecting small objects by integrating multi-scale features.

Key Innovations:

- Multi-scale predictions at three levels improved performance across diverse object sizes.
- The upgraded backbone, Darknet-53, combined with enhanced feature extraction, significantly increased accuracy.

Limitations:

- YOLO V3's reliance on anchor boxes persisted, requiring careful tuning to achieve optimal results.
- While accuracy improved, the model's computational requirements increased compared to YOLO V2.

2.2.4 YOLOv4

Summary:

YOLOv4, introduced in 2020 by Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao, represents a significant departure from its predecessors, focusing on striking a balance between speed and accuracy in real-time object detection. As the first YOLO model not authored by Joseph Redmon, YOLOv4 incorporates advanced architectural techniques and introduces the concepts of Bag of Freebies (BoF) and Bag of Specials (BoS) to optimize training and inference processes.

Architecture:

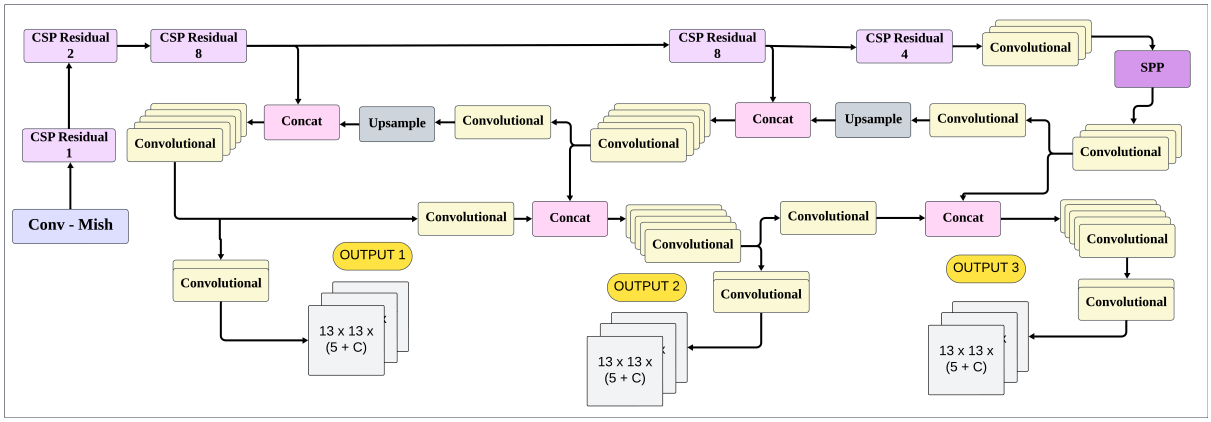


Figure 4: Yolov4's architecture

- Backbone: The backbone integrates Cross-Stage Partial Networks (CSPNet) into the conventional residual block structure, minimizing memory traffic while retaining accuracy. YOLOv4 also employs Mish activation, which outperforms Leaky ReLU by improving gradient flow and convergence.
- Neck: The neck combines Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PANet) for effective multi-scale feature aggregation
- Head: The detection head remains similar to YOLOv3, using anchor-based predictions to detect bounding boxes at three scales: 13×13, 26×26, and 52×52. Each prediction includes bounding box regression, object confidence, and class probabilities.

Bag of Freebies (BoF)

- Mosaic Augmentation: combines four images into one during training, exposing the model to diverse contexts and reducing sensitivity to occlusion.
- Label Smoothing: prevents overconfidence in predictions, reducing overfitting.
- DropBlock Regularization: forces the model to generalize better by randomly dropping regions in feature maps during training.

- Cross mini-Batch Normalization (CmBN): normalizes across multiple mini-batches, allowing training on smaller batches.

Bag of Specials (BoS)

- SPP Block: Expands the receptive field for improved detection of varying object scales.
- PANet: Strengthens feature aggregation and aligns multi-scale outputs for better prediction accuracy.
- CSP Blocks: Reduce computation by efficiently splitting and merging feature maps.
- Mish Activation: Smooths gradients and accelerates training convergence.

Key Innovations: YOLOv4 introduced several new significant advancements, which include a strong backbone and the SPP/PANet structure that are efficient at feature extraction. A new loss function (CIoU) and data augmentation (Mosaic) was also deployed to minimize errors

Limitations: Yolo V4 did not manage to get over the dependence on anchor boxes presented in Yolo V3, requiring careful design for optimal performance. More over, while deploying many optimization steps, it still require high memory requirements when using techniques like CmBN. It also introduced a problem regarding the sensitivity to dataset diversity, making generalization across domains challenging.

2.2.5 YOLOv5

Summary:

YOLO V5, developed by Ultralytics in 2020, marked a significant departure from traditional academic model releases. Unlike its predecessors, YOLO V5 prioritized practicality with an open-source implementation in PyTorch. This approach emphasized ease of use, modularity, and community accessibility. YOLO V5 became widely adopted for its balance of simplicity, performance, and active maintenance.

Architecture:

- Backbone: YOLO V5 introduced CSPDarknet53 as its backbone, which built upon YOLO V4's CSPResidualBlock. The architecture included C3 modules to improve gradient flow and reduce computational overhead. The SiLU (Swish) activation function replaced the Mish activation function, further optimizing computational efficiency.
- Neck: YOLO V5 incorporated the SPPF (Spatial Pyramid Pooling - Fast) module, an optimized version of the SPP block used in YOLO V4. This reduced computational overhead while maintaining robust feature aggregation. YOLO V5 incorporated the SPPF (Spatial Pyramid Pooling - Fast) module, an optimized version of the SPP block used in YOLO V4. This reduced computational overhead while maintaining robust feature aggregation.

- Head: YOLO V5's detection head was derived from YOLO V3 and YOLO V4. It relied on anchor boxes for bounding box predictions across three different scales.

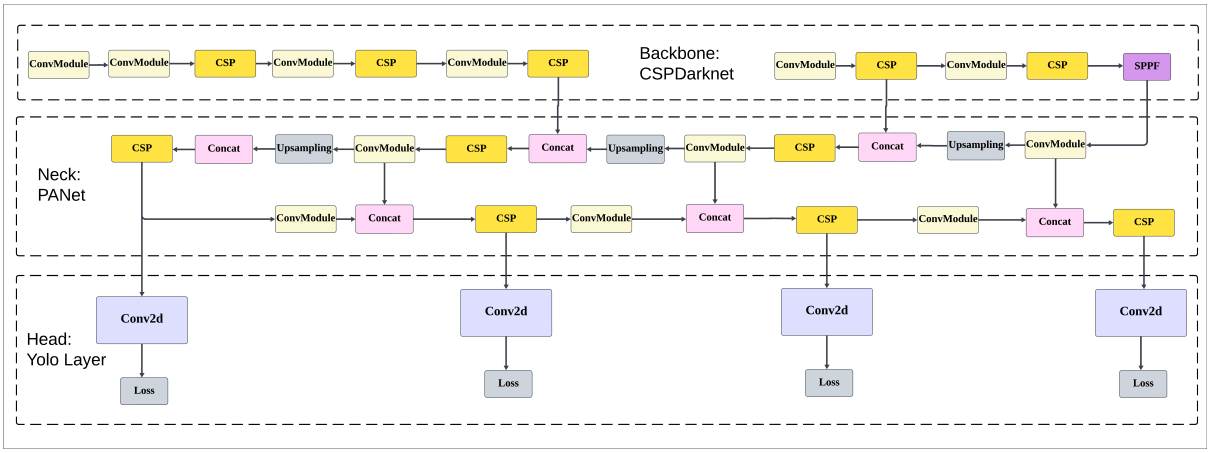


Figure 5: Yolov5's architecture

Bag of Freebies (BoF)

- Data augmentation was significantly enhanced with the introduction of Mosaic, MixUp, and Copy-Paste techniques. These improvements enriched training data diversity, boosting generalization.
- The use of Exponential Moving Average (EMA) ensured more stable training by maintaining a secondary model with weights updated via EMA.
- A genetic algorithm optimized anchor boxes for custom datasets, ensuring better alignment with object distributions.

Bag of Specials (BoS)

- The SPPF block optimized feature aggregation speed compared to the standard SPP block.
- CSP-PANet improved multi-scale feature integration, enhancing the model's ability to detect objects at varying scales.

Key Innovations:

- YOLO V5's lightweight and modular design, featuring C3 modules and the SPPF block, ensured high performance while maintaining user accessibility.
- Enhanced data augmentation techniques and anchor optimization improved training outcomes across diverse datasets.
- Loss scaling for objectness, combined with EMA weighting, further stabilized the training process.

Limitations:

- The absence of an academic paper limited peer-reviewed validation, leading to community debates about its benchmarks relative to YOLO V4.
- YOLO V5 retained dependence on anchor boxes, which required careful tuning for optimal performance.
- Despite its advantages, YOLO V5 faced criticism for lacking novel architectural contributions compared to prior YOLO versions.

2.2.6 YOLOv6

Summary:

YOLO V6, introduced by the Meituan Visual Intelligence Department in 2022, represents a major leap forward in the YOLO family. Specifically optimized for industrial applications, YOLO V6 integrates efficient hardware-aware design principles, advanced quantization techniques, and novel label assignment strategies. These innovations enable it to achieve state-of-the-art performance on benchmarks like the COCO dataset while maintaining scalability across different platforms.

Architecture:

- Backbone: YOLO V6 employs the EfficientRep Backbone, derived from the RepVGG architecture. It features Reparameterized Blocks and CSPStackRep Modules, which optimize feature extraction and scalability across various devices.
- Neck: The Rep-PAN Neck, built on the PAN architecture, is designed for efficient feature aggregation and reparameterization. This enhancement improves multi-scale detection and computational efficiency.
- Head: The Decoupled Head architecture separates classification and regression tasks, resulting in more precise bounding box predictions and improved efficiency compared to the coupled design used in YOLO V5.

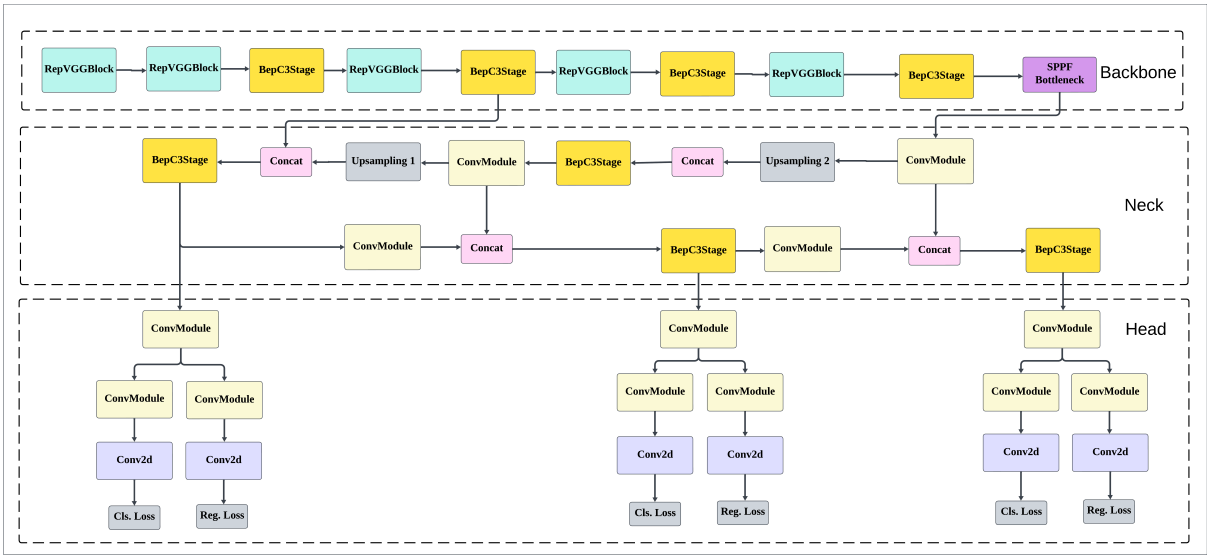


Figure 6: Yolov6's architecture

Bag of Freebies (BoF)

- YOLO V6 uses the model itself as a teacher during training, enabling self-refinement and improved predictive accuracy.
- Task-Aligned Label Assignment, inspired by TOOD, dynamically selects positive samples during training. This alignment mitigates instabilities and enhances overall performance.
- Training epochs were extended from 300 to 400 to allow the model to learn finer details.

Bag of Specials (BoS)

- EfficientRep and CSPStackRep optimize the backbone, reducing complexity while retaining high accuracy and scalability.
- The SIoU Loss function replaced CIoU, further improving spatial alignment and reducing bounding box regression errors.
- YOLO V6 eliminates anchor boxes, reducing randomness in predictions and simplifying the post-processing pipeline.

Key Innovations:

- Advanced Label Assignment dynamically aligns classification and regression tasks, improving detection consistency.
- The anchorless design eliminates reliance on predefined dimensions, enhancing flexibility and inference speed.
- Optimizations in the pipeline's complexity allow YOLO V6 to perform well on low-power systems while delivering high accuracy.

Limitations:

- YOLO V6's focus on industrial optimization may restrict its adaptability for exploratory research.
- Dependence on task-specific hardware design could limit its general-purpose applicability.
- The lack of a broader pretraining framework reduces its ability to adapt to novel tasks beyond its industrial scope.

2.2.7 YOLOv7

Summary:

YOLO V7, released in July 2022 by Chien-Yao Wang and Hong-Yuan Mark Liao, represents the pinnacle of efficiency and accuracy in the YOLO family. It surpasses all prior real-time object detectors on benchmarks like the MS COCO dataset, achieving 56.8% AP while maintaining speeds between 5 to 160 FPS on V100 GPUs. This model is widely regarded as a breakthrough in balancing performance and computational efficiency.

Architecture:

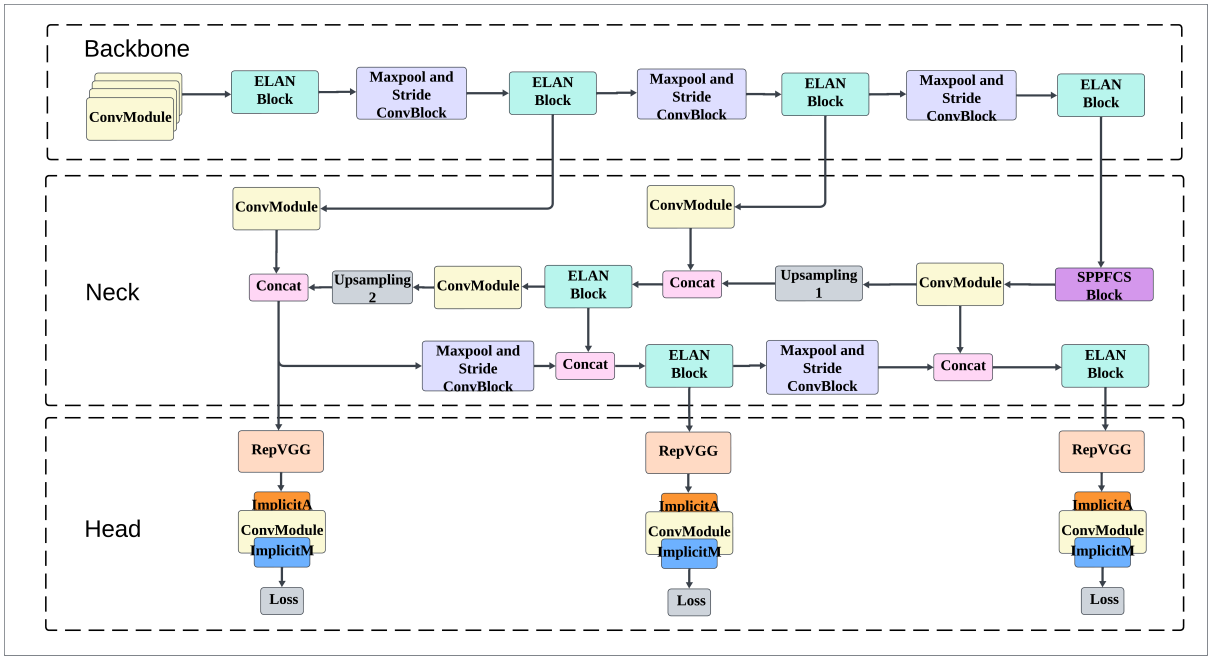


Figure 7: Yolov7's architecture

- Backbone: YOLO V7 employs the Extended Efficient Layer Aggregation Network (E-ELAN) as its backbone. E-ELAN introduces modular feature learning, dynamic scaling, and transition blocks, ensuring stability in deep architectures while enabling efficient feature aggregation.
- Neck: SPPCSPC: A CSP-enhanced Spatial Pyramid Pooling module that captures spatial context by pooling features from varying receptive fields. CSP-PAN: An improved version

of the Path Aggregation Network that incorporates CSP-OSA blocks for intermediate feature processing.

- Head: YOLO V7's detection head combines the strengths of YOLOR with auxiliary supervision. This approach enhances feature learning from coarse to fine levels by refining earlier predictions using auxiliary heads.

Bag of Freebies (BoF)

- Dynamic label assignment was implemented to reduce hyperparameter dependency and simplify training.
- Re-parameterized convolution (RepConv) layers optimized computational overhead while maintaining high accuracy.
- Enhanced batch normalization techniques ensured stability during training.

Bag of Specials (BoS)

- E-ELAN dramatically increased feature extraction capabilities through implicit knowledge layers and dynamic scaling.
- SPPCSPC and CSP-PAN modules refined multi-scale feature aggregation, enabling better detection of small and large objects alike.

Key Innovations:

- YOLO V7 introduced groundbreaking architectural features, including E-ELAN, auxiliary supervision, SimOTA label assignment, and RepConv blocks.
- It achieved unprecedented efficiency and accuracy in object detection, segmentation, and pose estimation tasks.

Limitations:

- The reliance on anchor-based predictions introduces challenges in scenarios with large-scale variability or dense object distributions.
- Auxiliary heads require high-end GPUs, such as NVIDIA V100, for optimal performance, limiting accessibility for resource-constrained users.

2.3 YOLO v8

2.3.1 Overview

YOLOv8, developed by Ultralytics, represents a significant advancement in object detection, pushing the boundaries of speed, accuracy, and usability. As an evolution of the YOLO (You Only Look Once) architecture, which is widely regarded for its real-time object detection capabilities, YOLOv8 builds on the foundation of YOLOv5 with numerous architectural improvements. Unlike traditional object detection systems that rely on multi-stage processes, YOLOv8 performs end-to-end predictions in a single pass, drastically enhancing efficiency. A standout feature of YOLOv8 is its anchor-free design, which eliminates the reliance on predefined anchor boxes for bounding box predictions. This innovation streamlines the Non-Maximum Suppression (NMS) process, reducing complexity while maintaining high precision. Additionally, YOLOv8 is versatile, extending its capabilities beyond object detection to encompass image classification and instance segmentation, making it a robust tool for a broad spectrum of computer vision tasks.

To meet diverse application requirements, YOLOv8 offers a scalable family of models, each balancing speed, accuracy, and computational demand. The model variations are defined by three parameters: depth multiple, width multiple, and max channels. Depth multiple adjusts the network's layer depth by scaling the number of Bottleneck Blocks in the C2f module. Lower values result in shallower, faster models for limited resources, while higher values increase depth and accuracy at the cost of speed. Width multiple determines the number of channels in the convolutional layers, with smaller values creating lightweight models that sacrifice some accuracy, and larger values yielding more accurate but resource-intensive models. Max channels impose an upper limit on the network's width, controlling size and preventing overfitting.

The YOLOv8 family comprises five variants tailored to specific tasks. The "n" (nano) model is the fastest and smallest, optimized for low-resource scenarios but offers lower accuracy. The "s" (small) model balances speed and accuracy for general use. The "m" (medium) model provides higher accuracy while maintaining moderate speed. The "l" (large) model prioritizes accuracy for precision-critical applications, and the "xl" (extra-large) model delivers the highest accuracy, suitable for resource-intensive tasks requiring maximum precision.

2.3.2 Architecture

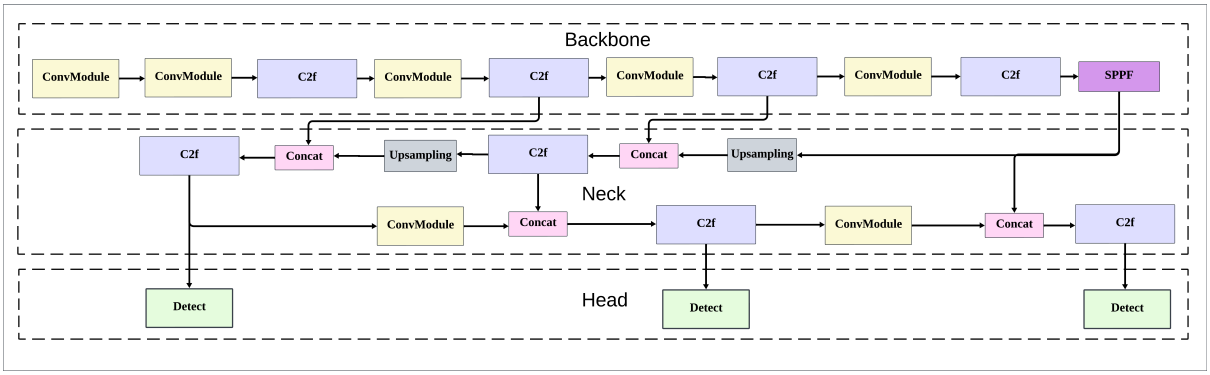


Figure 8: YOLOv8's Architecture

The architecture of YOLOv8 relies on several carefully designed building blocks, each serving a unique purpose in enhancing the model's efficiency, scalability, and performance. These blocks include the Convolutional Block (Conv Block), Bottleneck Block, C2f Block, Spatial Pyramid Pooling Fast (SPPF) Block, and the Detect Block, each contributing to the model's feature extraction and detection capabilities.

Convolutional Block (Conv Block)

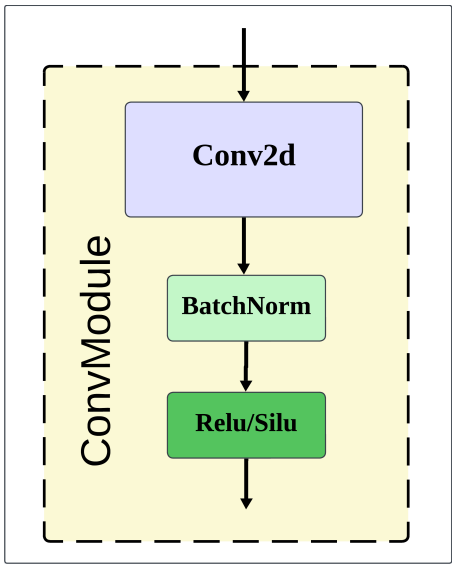


Figure 9: Convolutional Block

The Convolutional Block, commonly referred to as the Conv Block, serves as the foundational unit in the YOLOv8 architecture. This block is instrumental in extracting and processing features from input images, enabling the model to learn intricate patterns essential for object detection tasks. The Conv Block is composed of three primary components: a two-dimensional convolutional layer (Conv2D), a batch normalization layer (BatchNorm2D), and the SiLU activation function.

Conv2d Layer: At the core of the Conv Block is the Conv2D layer, which performs the convolution operation, a fundamental process in convolutional neural networks (CNNs). Convolution involves sliding a small matrix, known as a kernel or filter, across the input data (typically an image) to compute feature maps. This operation is applied in two spatial dimensions, height and width, capturing spatial hierarchies and local patterns within the data. Key parameters of the Conv2D layer include Kernel size (k) that defines the dimensions of the convolutional filter, Stride (s) that specifies the step size with which the kernel moves across the input image, Padding (p) that involves adding a border of zeros around the input image before applying the convolution operation and Input Channels (c) that represents the number of channels in the input data.

Batch Normalization Layer (BatchNorm2D): Following the convolution operation, the output feature maps are passed through a batch normalization layer. Batch normalization is a technique designed to improve the training stability and convergence speed of deep neural networks. The BatchNorm2D layer normalizes the activations across a mini-batch for each feature channel, adjusting the output to have a consistent distribution. The benefits of incorporating batch normalization include stabilizing learning process by normalizing the inputs to each layer, batch normalization mitigates the issue of internal covariate shift, where the distribution of inputs to layers changes during training. This stabilization allows the network to learn more effectively. In addition, normalized inputs enable the use of higher learning rates, which can lead to faster convergence during training. Last but not least batch normalization introduces a slight regularization effect by adding noise through the mini-batch statistics, which can improve the generalization of the model and reduce overfitting.

SiLU Activation Function: The final component of the Conv Block is the SiLU (Sigmoid Linear Unit) activation function, also known as the Swish activation function. The SiLU function is defined mathematically as:

$$\text{SiLU}(x) = x \cdot \sigma(x) \tag{1}$$

where $\sigma(x)$ is the sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

The SiLU activation function combines linear and non-linear characteristics, allowing for smooth and non-monotonic transformations of the input data. Its key properties include:

- **Smooth Gradients:** The function provides continuous and smooth gradients, which can help alleviate issues related to vanishing gradients in deep networks. This smoothness facilitates effective backpropagation of errors during training.

- **Enhanced Performance:** Empirical studies have demonstrated that networks utilizing the SiLU activation function can achieve better performance compared to those using traditional activation functions like ReLU. This improvement is attributed to SiLU's ability to model complex relationships within the data.
- **Non-Monotonic Behavior:** Unlike ReLU, which is strictly monotonic, SiLU's non-monotonic nature allows the network to retain small negative values, providing a richer feature representation.

Bottleneck Block

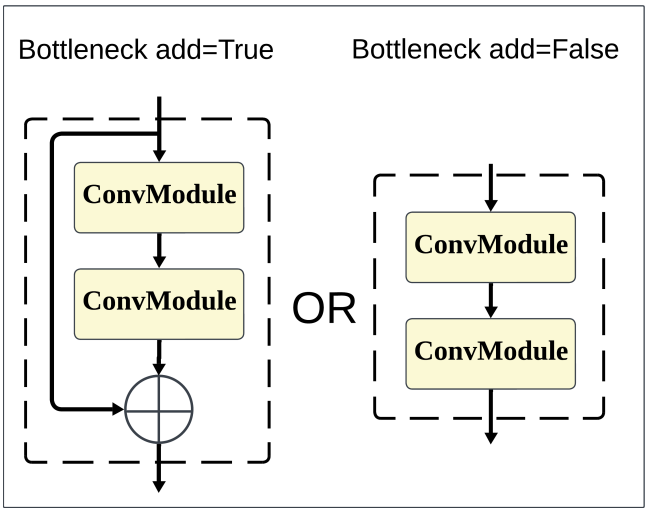


Figure 10: Bottleneck block

The Bottleneck Block is a pivotal component in YOLOv8, designed to enhance feature extraction and gradient flow during training. It builds upon the Conv Block by incorporating a shortcut connection, also known as a residual or skip connection, to address the vanishing gradient problem. Structurally, the Bottleneck Block consists of one or more Conv Blocks with an optional shortcut connection controlled by a boolean parameter. When enabled, the shortcut bypasses the Conv Blocks, allowing the input to be directly added to the output. This residual mapping helps retain input information while applying transformations, enabling the model to learn identity functions efficiently when required. If disabled, the input passes through the Conv Blocks sequentially without bypassing. The shortcut connection facilitates gradient propagation, mitigating the vanishing gradient issue by providing a direct path for gradients during backpropagation. This ensures effective learning in deeper networks and reduces optimization challenges. The vanishing gradient problem, a common issue in deep architectures, occurs when gradients become too small as they propagate, hindering weight updates. By addressing this, the Bottleneck Block allows YOLOv8 to scale to deeper layers, improving its capacity to learn complex representations. This modular design makes the Bottleneck Block essential for the robustness and performance of YOLOv8 in object detection tasks.

C2f Block

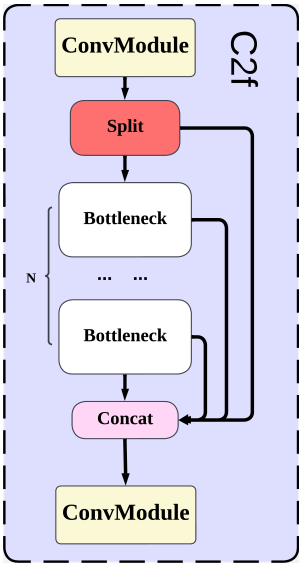


Figure 11: C2f block

The C2f Block (Cross-Stage Partial Bottleneck with Two Convolutions) is a distinctive feature of YOLOv8, designed to improve feature representation and detection accuracy. This block splits the input feature map into two paths: one undergoes a series of Bottleneck Blocks, while the other bypasses them. At the end of the block, the feature maps from both paths are concatenated and processed by a final Conv2D layer. The number of Bottleneck Blocks within the C2f Block is determined by the model's depth multiplier, making it adaptable to varying computational requirements.

Spatial Pyramid Pooling Fast (SPPF) Block

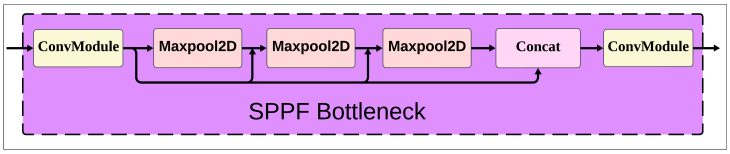


Figure 12: SPPF block

The Spatial Pyramid Pooling Fast (SPPF) Block in YOLOv8 plays a critical role in extracting multi-scale information from input features, enabling efficient and robust object detection. It consists of a convolutional block followed by three sequential MaxPool2d layers. The outputs from these MaxPool2d layers are concatenated and fed into a subsequent convolutional block, creating a composite feature representation. This design allows the network to effectively handle images of varying resolutions, a crucial capability for object recognition tasks where objects may appear at different scales within an image. By pooling features from different grid cells independently, the SPPF block enhances the model's ability to retain spatial context across scales without resizing the input or introducing spatial information loss.

Spatial Pyramid Pooling is inherently computationally intensive due to the need for multiple pooling levels with different kernel sizes. The SPPF Block addresses this limitation by

employing a simplified pooling scheme, using fixed-size kernels across the MaxPool2d layers. This reduces computational complexity while maintaining the effectiveness of multi-scale feature extraction. This trade-off between speed and accuracy makes the SPPF Block an efficient solution for real-time applications, such as autonomous driving and surveillance, where rapid inference is critical.

The MaxPool2d layers in the SPPF Block are pivotal for downsampling the spatial dimensions of feature maps, reducing computational demand while retaining essential information. These layers apply pooling in both the height and width dimensions of the input tensor, selecting the maximum value within each pooling region. This approach not only reduces redundancy in feature maps but also emphasizes dominant features, improving the robustness of the detection model. By integrating the SPPF Block into its architecture, YOLOv8 achieves a balance between computational efficiency and the ability to capture detailed, multi-scale features, which is vital for handling diverse object detection scenarios.

Detect Block

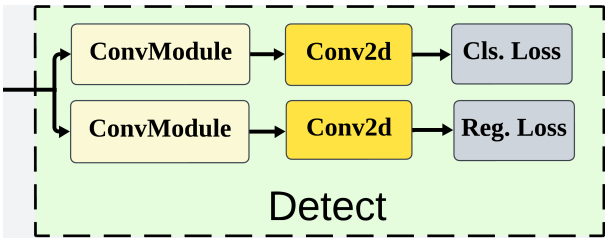


Figure 13: Detect block

The Detect Block is the anchor-free detection component in YOLOv8, responsible for predicting bounding boxes and object classes. Unlike previous YOLO versions, it eliminates the need for anchor boxes by directly predicting object centers. This anchor-free design reduces post-processing complexity, enhancing inference speed. The Detect Block contains two tracks: one for bounding box regression and the other for classification, both utilizing convolutional layers optimized for their respective tasks.

Backbone

The backbone of YOLOv8 forms the feature extraction stage of the architecture, designed to process input images through a series of convolutional and advanced structural blocks. It efficiently reduces spatial dimensions while maintaining critical features, allowing subsequent layers to build on this extracted information.

The backbone starts with Block 0, where the input image of size 640×640×3 (height, width, and channels) is processed through a convolutional block. This block uses a kernel size of 3, a stride of 2, and padding of 1. The stride of 2 reduces the spatial resolution, resulting in a feature map of size 320×320. The output channel of the convolution block is computed using the formula min(64,mc)×w, where mcmc is the maximum channel parameter, and ww is the width multiple. For instance, in the “n” variant of YOLOv8, w=0.25 and mc=1024, yielding an

output channel of 16.

Subsequent processing involves C2f blocks, as seen in Block 2. A C2f block incorporates Bottleneck blocks with optional shortcut connections, determined by the boolean parameter "shortcut." The number of Bottleneck blocks, n, in each C2f block is calculated using the formula n=3×d, where dd is the depth multiple. For example, in the "n" variant, where d=0.33, the number of Bottleneck blocks is approximately 1. The feature map dimensions and output channels remain unchanged within the C2f block.

Finally, Block 9 integrates the Spatial Pyramid Pooling Fast (SPPF) block at the end of the backbone. The SPPF block generates a fixed feature representation by pooling features across multiple scales without resizing the input or losing spatial information. This fixed representation is crucial for enhancing the detection capabilities of YOLOv8 across varying object scales and image resolutions. This structured backbone provides a robust foundation for the Neck and Head components, ensuring efficient feature extraction.

Neck

The Neck section in YOLOv8 acts as a bridge between the Backbone and Head sections, playing a pivotal role in aggregating features from multiple levels of the network and refining them for detection tasks. It achieves this by employing operations such as upsampling and concatenation, which allow it to efficiently combine information from different resolutions.

A key operation in the Neck is the upsample layer, which doubles the spatial dimensions of the feature map without altering the number of output channels. This operation enables the network to recover finer spatial details lost during earlier downsampling stages in the Backbone. By increasing the resolution of feature maps, the upsample layer enhances the network's ability to localize smaller objects effectively.

The Concat block follows the upsample operation, merging feature maps from different stages of the Backbone and the upsampled Neck features. This operation aggregates information from multiple levels of the network hierarchy, preserving both high-level semantic features and low-level spatial details. Importantly, the Concat block ensures that the output channels of the concatenated feature maps are summed, maintaining a consistent resolution and enabling seamless integration into subsequent layers.

The Neck architecture in YOLOv8 is specifically optimized for balancing computational efficiency and detection accuracy. By harmonizing multi-scale features through upsampling and concatenation, the Neck prepares robust feature representations that are well-suited for accurate object detection in the Head section. This carefully designed intermediary stage contributes significantly to YOLOv8's ability to handle diverse object scales and challenging detection scenarios.

Head

The Head section of YOLOv8 is the final stage of the architecture, responsible for producing the model's ultimate predictions: class probabilities and bounding box coordinates for the detected objects. It is specifically designed to optimize the detection performance across a range of object scales, ensuring accurate predictions for both small and large objects within the input image.

The Head comprises three distinct Detect blocks, each tailored for specific object sizes. The first Detect block, connected to the C2f block in Block 15, is dedicated to identifying small objects. By leveraging high-resolution feature maps from earlier layers, it ensures that finer spatial details crucial for detecting smaller objects are retained. The second Detect block, linked to the C2f block in Block 18, focuses on detecting medium-sized objects. This block balances the semantic richness of mid-level features with sufficient spatial resolution, optimizing detection performance for objects that fall between small and large scales. Finally, the third Detect block, fed by the C2f block in Block 21, specializes in detecting large objects, capitalizing on lower-resolution but highly semantic feature maps generated deeper in the network.

Each Detect block processes its input feature map through two primary tasks: classification and regression. Classification predicts the object class probabilities, while regression calculates the bounding box parameters. YOLOv8's anchor-free approach enhances detection efficiency by directly predicting the center points and sizes of bounding boxes without relying on predefined anchor boxes. This not only reduces computational overhead but also simplifies the post-processing step, such as Non-Maximum Suppression (NMS).

The streamlined and specialized design of the Head section, combined with its focus on multi-scale object detection, underpins YOLOv8's robustness in diverse scenarios. By tailoring the detection process to specific object sizes, the Head section ensures that YOLOv8 achieves high precision and recall across a wide range of applications, from small-scale targets in satellite imagery to larger objects in natural scenes.

2.3.3 Major Training Techniques

Data Augmentation

Data augmentation plays a critical role in YOLOv8's performance, particularly during training. One of the key techniques used is Mosaic augmentation, which stitches four images together to create a composite image. This method forces the model to learn from various object positions, occlusions, and backgrounds, enhancing its robustness and generalization capabilities. However, to prevent performance degradation, YOLOv8 disables Mosaic augmentation during the last ten epochs of training, allowing the model to finetune its learning on more conventional images.

Figure 14: Example of Mosaic augmentation

Loss Function

YOLOv8 employs advanced loss functions and innovative assignment techniques to significantly enhance object detection performance. Central to this approach is the Task-Aligned Assigner (TAA), introduced from TOOD, which improves the matching process between predicted bounding boxes and ground truth by simultaneously considering classification and regressionscores. This dual consideration ensures a more balanced and precise assignmentof positive samples, with the assignment score t for a predicted box p and a ground truth box g calculated as:

$$t = s^{\alpha} \times \mathrm{IoU}^{\beta} \tag{3}$$

where s represents the classification score, and α and β are hyperparameters that balance the influence between classification accuracy and localization.

The loss functions within YOLOv8 are bifurcated into two primary branches: classification and regression. The classification branch utilizes Binary Cross-Entropy (BCE) Loss to address discrepancies between the predicted class probabilities and the actual labels. The BCE Loss function is expressed as:

$$\mathrm{Loss}_n = -w\left[y_n \log x_n + (1-y_n)\log(1-x_n)\right] \tag{4}$$

Here, w denotes the weight, y_n the labeled value, and x_n the predicted probability by the model.

For bounding box regression, YOLOv8 integrates Distribution Focal Loss (DFL) and Complete Intersection over Union (CIoU) Loss, both of which contribute to refining bounding box predictions. DFL enhances precision by amplifying the probability distribution surrounding the object's location, computed as:

$$\mathrm{DFL}(S_n, S_{n+1}) = -\left[(y_{n+1}-y)\log S_n + (y-y_n)\log S_{n+1}\right] \tag{5}$$

where S_n is defined by the following relations:

$$S_n = \frac{y_{n+1} - y}{y_{n+1} - y_n}, \quad S_{n+1} = \frac{y - y_n}{y_{n+1} - y_n} \tag{6}$$

In parallel, CIoU Loss extends the traditional IoU metric by incorporating additional factors, such as the distance between the centers of the predicted and ground truth boxes, as well as the consistency of their aspect ratios. The CIoU Loss is mathematically formulated as:

$$\text{CIoU Loss} = 1 - \text{IoU} + \frac{d^2}{c^2} + \alpha \frac{v}{1 - \text{IoU} + v} \tag{7}$$

In this equation, d represents the Euclidean distance between the centers of the predicted and ground truth boxes, c denotes the diagonal length of the smallest enclosing box covering both, and v quantifies the difference in aspect ratios, expressed as:

$$v = \frac{4}{\pi^2} \left(\arctan \frac{w_{gt}}{h_{gt}} - \arctan \frac{w_p}{h_p} \right)^2 \tag{8}$$

These advanced loss functions, combined with the Task-Aligned Assigner, empower YOLOv8 to achieve state-of-the-art results, particularly in scenarios where small object detection and complex environments pose significant challenges. The synergy between the model's architecture and its refined training techniques underscores YOLOv8's capability as a robust and versatile tool for a wide array of computer vision applications.

2.3.4 Other

YOLOv8 distinguishes itself not only through its architectural advancements and training techniques but also through its versatility, ease of use, and integration capabilities. It represents more than just a model update; it serves as a comprehensive platform for modern computer vision applications.

One of YOLOv8's standout features is its streamlined deployment process. The model can be executed seamlessly from the command-line interface (CLI) or installed as a Python package via PIP. This flexibility simplifies integration into various workflows, catering to both novice users and seasoned developers. Moreover, YOLOv8 is equipped with multiple integrations to facilitate the complete lifecycle of computer vision projects, including labeling, training, and deployment, making it highly adaptable to diverse operational needs.

As a refactored version of YOLOv5, YOLOv8 introduces significant improvements to its API and underlying code. It builds upon YOLOv7's ELAN architecture while incorporating additional residual connections, and its decoder is derived from YOLOv6 2.0. These changes reflect its evolution into a technology integration platform, connecting APIs for multiple downstream tasks in a unified framework. The latest iteration, "Glenn," exemplifies this by integrating cutting-edge technologies such as YOLOv9 and YOLO World. While some developers have found the modifications and API usage less intuitive, the platform's optimizations, particularly a 30% boost in training performance, have attracted widespread

adoption among research and development teams.

YOLOv8 also extends its utility by providing a simple API for downstream applications like instance segmentation, pose estimation, multi-object tracking, and even general-purpose object detection tasks. This capability not only enhances its adaptability but also underscores its role as a versatile tool in a wide array of fields, from autonomous systems to medical imaging. These features cement YOLOv8's position as a leading choice for computer vision practitioners, offering unmatched flexibility and performance across applications.

3. Experimental Result and Discussion

3.1 Environment

We worked with YOLO versions 1, 2, 3, 5, 6, 7 and 8, and chose to ignore version 4. This was a deliberate decision as repositories that contain resources for transferring and training YOLOv4 used outdated libraries (eg. Numpy version 1.19.5) or no-longer-supported format (eg. Pretrained weights are saved as .weights files instead of the widely-used .pt files). This created a lot of problems with the training environment and lack of resources, so we have decided to drop version 4.

We started off with building YOLO v1 and v2 models by constructing them from scratch and using pre-trained weights for faster training. We chose to do the training on the Kaggle platform to take advantage of the GPU resources. The unexpected combination of the incapabilities of the early versions of YOLO and a difficult-to-learn dataset had as a final result poor performance. Because the outputs were rather discouraging, we made the decision not to not include YOLO v1 and YOLO v2 in our comparison and discussion since their results will not provide important information for our final analysis.

As we failed with Yolo v1 and v2, we took this as an insight and changed our strategy. For models YOLO v3 and later, we decided to use third-party libraries which are more suitable for training these types of models. To achieve the fairness any academic experiment required, we only use those versions that have optimal parameters: YOLO v3 tiny, YOLO v5 nano, YOLO v6 nano, and YOLOv7-Tiny. This helps combine both the performance evaluation and efficiency evaluation of lightweight models constructed for deployment purposes.

For YOLOv8, we went a step broader and trained 4 variants, namely: n- Nano, s- Small, m- Medium and l- Large. Our goal was to show that YOLO8 could provide varying granularity of performance across the scales of models. Its architecture bested the other models in both efficiency and performance while maintaining a reasonably low parameters count. This was achieved by fine tuning of the model using the Ultralytics library in the case of YOLOv3, YOLOv5 or YOLOv8. For YOLOv6 and YOLOv7, we used the repositories of Meituan, WongKinYiu respectively.

3.2 Setup hyperparameters

To ensure a fair comparison across all models, we trained them using the same hyperparameters, which were as follows:

- Batch Size: 32
- Number of Epochs: 100
- Number of Workers: 8
- Pretrained Weights: Enabled (True)
- Learning Rate: 0.01 (except for YOLOv3, which used 0.001)
- Momentum: 0.937
- Weight Decay: 0.0005
- Optimizer: SGD with step decay (AdamW and cosine decay were used for YOLOv8)
- Image Size: 640

The whole training was finished on Google Colab, making use of its inbuilt T4 GPU. With the exception of Yolo v1 and v2, as stated above, were trained on Kaggle instead with the same GPU of T4.

During the training of YOLOv1 and YOLOv2, we attempted several learning rates, while the epoch numbers have been increased across tests, with some going as far as 500 epochs. But even with these goals in mind and the fact that they used pretrained weights and similar hyperparameters such as a batch size of 32, the output was still unsatisfactory.

3.3 Dataset

For evaluation YOLO versions, we used GDIT Aerial Airport dataset. This dataset consists of several aerial images of parked airplanes. This dataset only contains a single class for all types of airplanes, which are distributed across 810 images with a dimension of 600x600 pixels. The labels are stored in YOLO txt format. The number of airplanes in each image can range from a few to dozens, with a high variability of size, shape and type. 810 instances are divided into 3 separate folders:

- train: 708 images
- valid: 68 images
- test: 34 images

3.4 Evaluation metrics

With the mentioned datasets, precision (P) and recall (R) were used as the evaluation metrics for aircraft detection. These model evaluation metrics are defined as follows:

$$Precision(P) = \frac{TP}{TP+FP} \tag{9}$$

$$Recall(R) = \frac{TP}{TP+FN} \tag{10}$$

where True Positives (TP) corresponds to when the algorithm correctly detects aircrafts with a bounding box, False Positives (FP) indicates when the algorithm computes a bounding box in a false location; and False Negatives (FN) indicates when the algorithm did not detect an aircraft.

The Intersect over Union (IoU) between the bounding box from detection and the ground truth is calculated. For each image, if the IoU is over a certain predetermined threshold, the trained model provides a TP using bounding boxes coordinates and a confidence score. The area under the precision-recall curve represents the Average Precision (AP). AP is usually a number between 0 and 1 used to summarize the different precision values obtained in the recall function. Furthermore, the mean average precision (mAP) is used to evaluate a model and is obtained by averaging the AP for each class.

$$AveragePrecision(AP) = \int_0^1 P(R)\,dR \tag{11}$$

$$MeanAveragePrecision(mAP) = \frac{1}{n}\sum_{i=1}^{n} AP_i \tag{12}$$

, where n is the number of classes.

Two different thresholds are usually applied to produce two different maps: the mAP@0.5 and mAP@0.5:0.95.

$$mAP@0.5 = \frac{1}{n}\sum_{i=1}^{n} AP_i(0.5) \tag{13}$$

The mAP@0.5 calculates mAP at a single threshold of IoU = 0.5, summing the average precision for each class and dividing by the total number of classes.

$$mAP50:95 = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{10}\sum_{t=0.50}^{0.95} AP_i(t)\right) \tag{14}$$

The mAP@0.5-0.95 calculates the average precision for each class at multiple IoU thresholds, from 0.5 to 0.95 in steps of 0.05, and then averages over all thresholds to get mAP@0.5-0.95.

3.5 Result

Model	Parameter	Precision	Recall	mAP50	mAP50-95
YOLOv3-tiny	8.7M	0.898	0.807	0.877	0.462
YOLOv5-n	2.5M	0.917	0.860	0.922	0.513
YOLOv6-n	4.3M	0.877	0.790	0.854	0.448
YOLOv7-tiny	6M	0.918	0.854	0.893	0.458
YOLOv8-n	3.2M	0.925	0.863	0.932	0.519
YOLOv8-s	11.2M	0.923	0.889	0.934	0.538
YOLOv8-m	25.9M	0.933	0.900	0.945	0.545
YOLOv8-l	43.7M	0.944	0.914	0.950	0.552

Table 1: Performance Comparison of YOLOv3-tiny, YOLOv5-n, YOLOv6-n, YOLOv7-tiny vs all variants of YOLOv8

The experimental results provide a thorough assessment of YOLOv3-tiny, YOLOv5-n, YOLOv6-n, YOLOv7-tiny, and all variants of YOLOv8 on the GDIT dataset. Evaluation metrics used are precision, recall, mAP50, and mAP50-95, which offer a detailed comparison of their performance across diverse scenarios. Precision indicates the model's ability to avoid false positives, while recall measures its capability to detect all relevant objects, minimizing false negatives. The mean Average Precision (mAP), computed at thresholds of 50% and 50-95%, provides an overall measure of detection accuracy across varying IoU thresholds.

YOLOv8 showcases superiority across all metrics compared to previous YOLO versions. Specifically, YOLOv8-l achieves the highest recall (0.914) and mAP50-95 (0.552) among all models, underscoring its ability to accurately detect objects even in complex scenarios such as densely packed or overlapping aircraft. Conversely, YOLOv3-tiny and YOLOv7-tiny achieve lower recall scores of 0.807 and 0.854, respectively, showing their limitations in handling such challenging datasets. The recall improvements of YOLOv8 can be attributed to its advanced decoupled head architecture, which independently optimizes objectness, classification, and regression tasks, ensuring specialized processing for each component.

Parameter efficiency is another area where YOLOv8 surpasses its predecessors. For instance, YOLOv8-n achieves an mAP50 of 0.932 with only 3.2M parameters, outperforming YOLOv3-tiny, which requires 8.7M parameters just to achieve a lower mAP50 of 0.877. Similarly, YOLOv8-s, with 11.2M parameters, achieves an mAP50-95 of 0.538, surpassing YOLOv7-tiny's 0.458, despite YOLOv7-tiny having fewer parameters (6M). This effectiveness is primarily due to YOLOv8's anchor-free detection mechanism, which predicts object centers directly, eliminating the need for anchor boxes. This design not only reduces the number of candidate boxes but also accelerates post-processing, enabling faster inference.

The distribution of precision and recall across YOLOv8 variants further demonstrates its adaptability. YOLOv8-m achieves a recall of 0.9, balancing high accuracy and computational demand, while YOLOv8-l achieves a precision of 0.944, the highest among all models. These results emphasize YOLOv8's ability to adapt its performance to varying computational and accuracy requirements. By comparison, YOLOv6-n achieves a lower precision of 0.877 and recall of 0.790, further illustrating the limitations of earlier YOLO versions in addressing

complex object detection tasks.

YOLOv8's advanced loss functions also play an important role in its superior performance. The integration of Distribution Focal Loss (DFL) and Complete Intersection over Union (CIoU) loss refines bounding box predictions, ensuring greater alignment between predicted and ground truth boxes. This improvement is particularly beneficial in scenarios involving small or overlapping objects, common in the GDIT dataset. For example, YOLOv8-s, with its advanced loss optimization, achieves a recall of 0.889, outperforming YOLOv7-tiny's 0.854. These refinements enable YOLOv8 to effectively handle dense clusters of aircraft, a hallmark of the GDIT dataset.

The architectural enhancements of YOLOv8 further contribute to its state-of-the-art performance. The CSPDarknet53 backbone enables efficient multi-scale feature extraction, crucial for maintaining accuracy in high-resolution imagery. The inclusion of the C2f block, which combines high-level features and contextual information, ensures robust detection capabilities while maintaining computational efficiency. Moreover, the Spatial Pyramid Pooling Fast (SPPF) block minimizes spatial information loss during feature aggregation, enhancing YOLOv8's ability to detect objects of varying scales and resolutions. These architectural improvements directly contribute to YOLOv8-l achieving an mAP50-95 of 0.552, far surpassing YOLOv7-tiny's 0.458 and YOLOv3-tiny's 0.462.

In conclusion, YOLOv8 is the most advanced and efficient model when excelling in all metrics. Its architectural innovations, including the decoupled head, anchor-free design, and optimized loss functions, enable it to consistently outperform earlier YOLO versions, particularly in challenging scenarios. With its scalability across variants and parameter efficiency, YOLOv8 is a robust solution for detecting aircraft in high-resolution satellite imagery, setting a new benchmark in real-time object detection.

3.6 Example

The section evaluates the detection performance of different YOLO versions, focusing on the same scenario presented in the dataset, a high-resolution satellite image of an airport terminal with multiple aircraft and complex environmental challenges. This image showcases conditions such as small object sizes, overlapping objects, and cluttered backgrounds, providing a comprehensive test of each model's detection capabilities.

Upon examining the results, YOLOv3-tiny showed poor performance, with sparse detections and a high false-negative rate. Specifically, many aircraft were missed, especially those partially obscured or located in the background. The confidence scores of the detected aircraft were relatively low (e.g., 0.37, 0.53), indicating the model's limitations in recognizing the objects with certainty. YOLOv5-n and YOLOv6-n, on the other hand, improved slightly, detecting more objects but still producing inconsistent confidence scores. In YOLOv6-n, overlapping aircraft or objects near the periphery were either not detected or assigned very low confidence (e.g., 0.39, 0.54), highlighting the model's hindrances in complex scenarios. YOLOv7-tiny marked a significant leap, achieving higher detection accuracy compared to its predecessors.

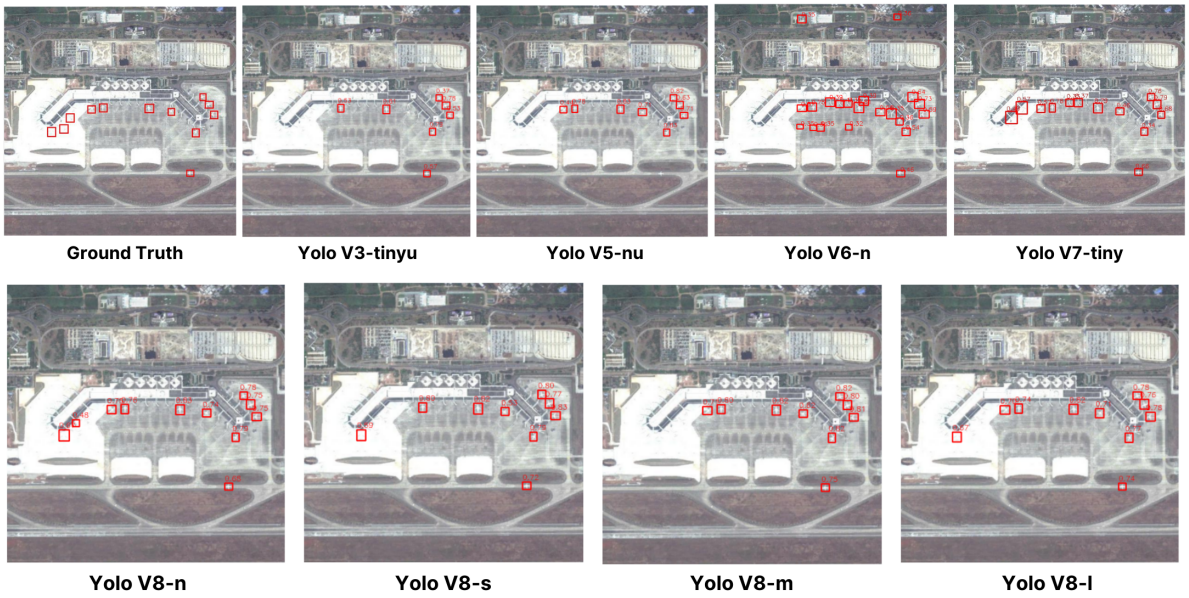


Figure 15: Example of aircraft detection using GDIT dataset

The model correctly identified more aircraft, and its confidence scores were notably improved, with many detections exceeding 0.7. However, challenges persisted in detecting overlapping objects and maintaining uniform confidence across all detections.

The YOLOv8 series demonstrated remarkable advancements over earlier versions, attributable to its architectural innovations. YOLOv8-n, despite having a smaller parameter count (3.2M), outperformed YOLOv7-tiny in both detection completeness and confidence consistency. Aircraft in densely populated areas and at the edges of the image were identified with higher reliability (e.g., confidence scores exceeding 0.74). YOLOv8-s and YOLOv8-m further refined the performance, with YOLOv8-m consistently achieving confidence scores above 0.80 for nearly all detections. This reflects the model's robustness, facilitated by its enhanced C2f block and loss function optimizations like CIoU and DFL, which excel in refining bounding box predictions and managing overlapping objects.

YOLOv8-l showcased the most robust performance, achieving the highest confidence scores (e.g., 0.82, 0.85) across all detected aircraft. Its ability to handle small and overlapping objects, even in cluttered regions, underscores the effectiveness of its advanced Spatial Pyramid Pooling Fast (SPPF) block and the decoupled head design. These innovations enable precise multi-scale feature extraction and reduce interference between classification and regression tasks, resulting in fewer false positives and negatives.

4. Conclusion

This our project provided an in-depth evaluation of YOLO models, specifically focusing on their performance in aircraft detection from satellite imagery using the GDIT dataset. From the early iterations like YOLOv1 and YOLOv2, which struggled with feature learning due to grid limitations and overfitting, to the more recent YOLOv8 model, which consistently outperformed its predecessors across key metrics including precision, recall, mAP50, and mAP50-95.

YOLOv8's significant improvements stem from its architectural advancements, including its decoupled head and anchor-free design, which resulted in more efficient and accurate detection. The use of advanced loss functions such as Distribution Focal Loss and CIoU Loss contributed to better localization accuracy, especially in challenging scenarios involving small or densely packed aircraft. These innovations resulted in superior performance, as demonstrated by YOLOv8's ability to achieve high precision and recall, even with fewer parameters compared to earlier models like YOLOv3 and YOLOv5.

The results indicate that YOLOv8 is not only more computationally efficient but also delivers better detection capabilities, making it a robust choice for real-time aircraft detection in satellite imagery. Future research could explore further optimization of YOLOv8 for specific satellite-based applications or examine the integration of newer machine learning techniques to push the boundaries of object detection in high-resolution imagery. Overall, the YOLO series, and particularly YOLOv8, demonstrates a promising path forward for automated, scalable solutions in civil and military applications where accurate aircraft detection is critical.

References

[1] Luo, R., Xing, J., Chen, L., Pan, Z., Cai, X., Li, Z., ... & Ford, A. (2021). Glassboxing deep learning to enhance aircraft detection from SAR imagery. Remote Sensing, 13(18), 3650.

[2] Lin, S., Chen, T., Huang, X., & Chen, S. (2023). Synthetic aperture radar image aircraft detection based on target spatial imaging characteristics. Journal of Electronic Imaging, 32(2), 021608-021608.

[3] Guo, Q., Wang, H., & Xu, F. (2020). Scattering enhanced attention pyramid network for aircraft detection in SAR images. IEEE Transactions on Geoscience and Remote Sensing, 59(9), 7570-7587.

[4] Zhao, Y., Zhao, L., Li, C., & Kuang, G. (2020). Pyramid attention dilated network for aircraft detection in SAR images. IEEE Geoscience and Remote Sensing Letters, 18(4), 662-666.

[5] Liu, D., & Gao, S. (2022, April). Aircraft Detection in Remote Sensing Imagery Based on Improved YOLOv4. In Journal of Physics: Conference Series (Vol. 2260, No. 1, p. 012063). IOP Publishing.

[6] Liu, Z., Gao, Y., Du, Q., Chen, M., & Lv, W. (2023). YOLO-extract: Improved YOLOv5 for aircraft object detection in remote sensing images. IEEE Access, 11, 1742-1751.

[7] Wei, H., Zhang, Y., Wang, B., Yang, Y., Li, H., & Wang, H. (2020). X-LineNet: Detecting aircraft in remote sensing images by a pair of intersecting line segments. IEEE Transactions on Geoscience and Remote Sensing, 59(2), 1645-1659.

[8] Liu, Q., Xiang, X., Wang, Y., Luo, Z., & Fang, F. (2020). Aircraft detection in remote sensing image based on corner clustering and deep learning. Engineering Applications of Artificial Intelligence, 87, 103333.

[9] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. Pattern recognition, 77, 354-377.

[10] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.

[11] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection.

[12] Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger.

[13] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.

[14] Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934.

[15] Glenn Jocher. (2020). YOLOv5 by Ultralytics.

[16] Li, C., Wang, X., et al. (2022). YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications. arXiv preprint arXiv:2209.02976.

[17] 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. arXiv preprint arXiv:2207.02696.

[18] Glenn Jocher, Ayush Chaurasia, et al. (2023). YOLOv8: Next-Generation Object Detection.