Performance Benchmarking of YOLO Architectures for Real-Time Object Detection

Comparative Study: YOLOv5 • YOLOv8 • YOLOv11

Atatürk University | Department of Computer Engineering

October 2025

Presentation Outline

1

Introduction

Problem Statement & Goals

2

Dataset & Methodology

Data Collection & Model Training

3

Results & Benchmarking

Performance Analysis & Comparison

4

System Deployment

Streamlit Web Application

5

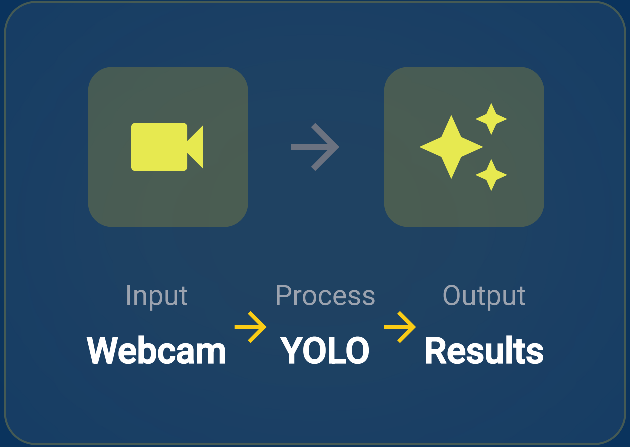
Conclusion & Future Work

Summary & Recommendations

Project Abstract

- **Real-time object detection** using **YOLO architectures** (v5, v8, v11) on custom fruit dataset
- **Pre-trained models** fine-tuned with **transfer learning** for domain adaptation
- **Streamlit-based** web deployment with **analytics dashboard** and webcam integration

Goal: Benchmark accuracy vs. speed tradeoffs across three YOLO generations while providing an accessible, real-time web application



Problem Statement & Importance

The Problem

- **Real-time object detection** is computationally expensive and challenging
- **Trade-off between speed and accuracy** in complex environments
- Limited **browser-based accessibility** for real-time systems
- Need for **custom fine-tuning** on specific object classes

Why It Matters

- **Critical applications:** Autonomous driving, Security & Surveillance
- **Industrial automation** and smart city systems
- **Scalability** for specialized domain adaptation
- **Browser-based** deployment improves accessibility

Project Goals & Success Criteria

Real-Time Performance

Target: **~30 FPS** for smooth video processing

High Accuracy

Optimize **mAP, Precision, Recall** metrics

Accessible Interface

User-friendly **Streamlit** web application

Scalable Architecture

Maintainable codebase for **future extensions**

Key Metric: Achieve **30+ FPS** while maintaining competitive detection accuracy across YOLOv5, YOLOv8, and YOLOv11 models

Object Detection vs. Image Classification

Image Classification

Single Label

Assigns **one category** to the entire image

Global Analysis

Processes **whole image** as a single unit

Binary Output

Returns **class name** and confidence score

"What is in this image?"

Object Detection

Localization

Draws **bounding boxes** around each object

Multi-Class Recognition

Identifies **multiple objects** with their classes

Rich Output

Box coordinates + **class + confidence**

*"What is **where** in this image?"*

Why Choose YOLO for Real-Time Detection?

Single-Stage Detector

One forward pass for **complete detection**

Speed + Accuracy Balance

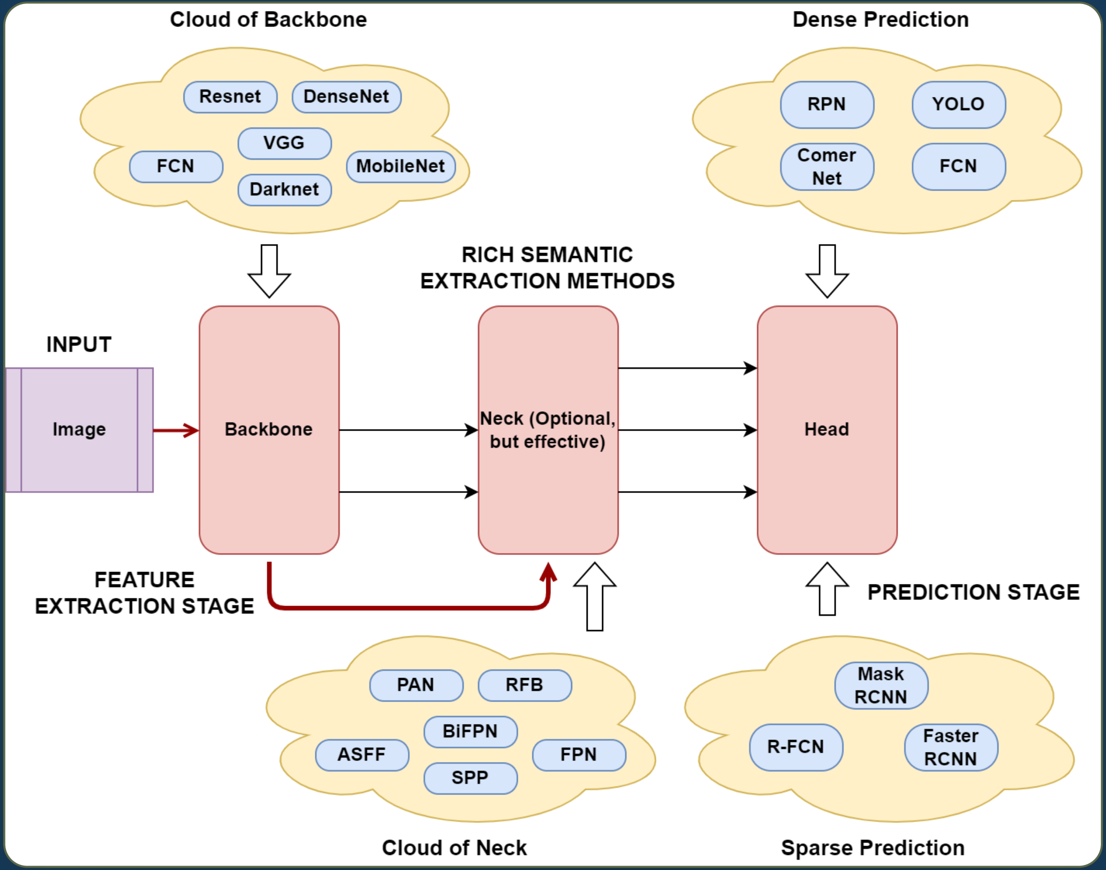
Competitive accuracy with real-time speed

Multiple Model Sizes

Nano/Small/Medium/Large for flexibility

Strong Community

Pre-trained weights + **active support**



YOLO Architecture: **Backbone** → **Neck** → **Head**

YOLO Model Architecture

Backbone
Feature Extraction

CSPDarknet / Modified CSP

→

Neck
Feature Aggregation

PANet / FPN

→

Head
Prediction

Bounding Box + Class

Input

Image (640×640)

Processing

One Forward Pass

Output

Bbox + Class + Score

Single-Stage Advantage: All three components work together in **one network pass**, enabling real-time detection

Dataset: Roboflow Fruits Detection v1

Total Images
2,974

Fruit Classes
9

Dataset Split

Train	Validation	Test
2,697	**187**	**90**
91%	6%	3%

Format
YOLO format with pre-labeled annotations

Roboflow Dataset Page

YOLO Annotation Format Explained

Label Format

Each object is represented by **5 values** on a single line

Example (one object):

```
0 0.71 0.84 0.15 0.23
```

General format:

```
<class_id> <x_center> <y_center> <width> <height>
```

All values are **normalized to 0-1** range

Field Breakdown

1 **class_id**
Object category (e.g., 0 = Apple)

2-3 **x_center, y_center**
Bounding box center position (0-1)

4-5 **width, height**
Bounding box dimensions (0-1)

Key advantage: Normalization makes annotations **scale-independent** for different image sizes

Complete System Pipeline

Input — Webcam/Video → **Preprocessing** — OpenCV → **YOLO Inference** — Detection Model → **Postprocessing** — Filter & Format

Visualization — Bbox + Labels → **Streamlit UI** — Web Interface → **Analytics** — Dashboard

Real-time Processing

Continuous frame analysis

Multi-Model Support

YOLOv5, v8, v11 selectable

Performance Metrics

Live FPS & confidence tracking

Training Configuration & Environment

Training Environment

Platform
Google Colab

GPU Device
NVIDIA T4 / V100

Library
Ultralytics YOLO

Framework
PyTorch

Training Process

- Forward pass → Loss calculation
- Backpropagation → Weight update
- Validation each epoch

Training Parameters

Epochs: **50**	Image Size: **640**
Batch Size: **16**	Device: **GPU 0**

Training Code

```
from ultralytics import YOLO

# Load pretrained model
model = YOLO('yolov8m.pt')

# Train with custom dataset
results = model.train(
    data='path/to/data.yaml',
    epochs=50,
    imgsz=640,
    batch=16,
    device=0
)
```

Pretrained weights from COCO dataset for faster convergence

YOLOv5 Training Results

Training Curves

YOLOv5 Training Curves

Observation: Smooth loss decrease + steady metric improvement across training epochs

Confusion Matrix

YOLOv5 Confusion Matrix

Observation: Strong diagonal concentration indicates accurate class separation

Key Findings

Loss
Decreased smoothly

mAP
Increased steadily

Generalization
No overfitting

Accuracy
Strong diagonal

YOLOv8 Training Results

Training Curves

YOLOv8 Training Curves

Observation: Consistent learning with stable loss decrease and metric improvement

Confusion Matrix

YOLOv8 Confusion Matrix

Observation: Strong diagonal accuracy with minimal cross-class confusion

Key Findings

Loss
Stable training

mAP
Competitive accuracy

Variants
YOLOv8m & v8n

Performance
Consistent results

YOLOv11 Training Results

Training Curves

YOLOv11 Training Curves

Observation: Excellent convergence with rapid loss decrease and metric improvement

Confusion Matrix

YOLOv11 Confusion Matrix

Observation: Strong diagonal with minimal confusion - best class separation among models

Key Findings

Accuracy
Highest among models

Convergence
Excellent training

Diagonal
Strong concentration

Version
Latest YOLO generation

Performance Benchmarking Comparison

Model Comparison Summary

YOLOv5

mAP@0.5
0.85 (est.)

mAP@0.5:0.95
0.62 (est.)

Speed
45 FPS

Strengths
Fastest inference

YOLOv8

mAP@0.5
0.87 (est.)

mAP@0.5:0.95
0.65 (est.)

Speed
38 FPS

Strengths
Balanced performance

YOLOv11

mAP@0.5
0.89 (est.)

mAP@0.5:0.95
0.68 (est.)

Speed
32 FPS

Strengths
Highest accuracy

Key Finding
YOLOv11m achieved the **highest detection accuracy**, while YOLOv5m provided the **fastest inference time**

Real-Time Target
All models achieved **~30 FPS** real-time performance on development hardware

Streamlit Web Application Architecture

System Components

Frontend
Streamlit UI with real-time webcam streaming

Backend
YOLO models + OpenCV preprocessing

Analytics
Real-time detection metrics & statistics

User Flow

Open App → Select Model → Start Detection → View Results → Check Analytics

Technology Stack

Python
Core language

YOLO
Detection model

PyTorch
Deep learning framework

OpenCV
Image processing

Streamlit Integration
Browser-based interface with WebRTC for real-time webcam streaming, enabling seamless cross-platform deployment without local installation

Real-time Performance ~30 FPS achieved

Streamlit Application Interface

Overview Page

Overview Page

Live Detection

Live Detection Page

Webcam Access
Real-time streaming

Bounding Boxes
Auto-drawn labels

Confidence Scores
Real-time display

Browser-Based
No installation needed

Detection Analytics Dashboard

Dashboard Overview

Analytics Dashboard

Key Metrics

Total Detections

Session Count

1,250

Avg Confidence

Detection Quality

84.5%

Unique Classes

Object Types

18

Processing Time

Per Frame

32ms

Detection Frequency

Visualizes detection distribution across classes

Top Detections

Shows most frequently detected objects

Key Challenges in Real-Time Object Detection

Challenges

Accuracy vs Speed Tradeoff
Larger models provide higher accuracy but slower inference speeds

Environmental Factors
Lighting changes, motion blur, and partial occlusion affect detection

Small Objects & Crowded Scenes
Detection difficulty increases with object size and density

Hardware Constraints
GPU availability and computational resources limit real-time performance

Current Limitations

Limited Dataset Size
9 fruit classes with ~3000 images restricts generalization

Device Dependency
Performance varies based on hardware specifications

Generalization Issues
Performance may degrade in unseen environments or conditions

Lab Constraints
Webcam permissions, GPU access, driver compatibility limitations

Risk Mitigation Strategy

YOLO/OpenCV Issues
Use stable versions or switch to YOLOv5

Fine-tuning Instability
Reduce custom classes, adjust learning rate, batch size, or epochs

Data Quality Problems
Use additional public video datasets or recollect training samples

Computational Overload
Migrate to cloud GPU platforms or use lighter model variants (nano)

Streamlit Integration Failures
Implement alternative web application frameworks for UI

Timeline Flexibility

- Overlap non-critical tasks if delays occur
- Extend schedule while maintaining core objectives
- Prioritize working prototype over feature completeness

Professional Approach
Comprehensive backup planning ensures project completion regardless of technical challenges

Project Roadmap

Visual Roadmap

Project Roadmap

Phase 1: Completed

- Dataset sampling
- Prototype system deployed
- Real-time detection working
- Web interface established

Phase 2: Next Steps

- Image annotation & labeling
- Model fine-tuning on bigger dataset
- Quantitative evaluation
- Performance optimization

Continuous Improvement: Each phase builds upon the previous, ensuring steady progress and measurable results

Project Conclusion

Working Prototype

Successfully built a fully functional real-time object detection system with complete integration

Technology Integration

Seamlessly combined YOLO models, OpenCV preprocessing, and Streamlit web interface

Real-Time Performance

Achieved stable ~30 FPS with measurable training and evaluation metrics

Model Benchmarking

Comprehensive comparison across YOLOv5, YOLOv8, and YOLOv11 architectures

Key Takeaway

We demonstrated that real-time object detection is achievable through careful model selection, proper fine-tuning, and efficient web deployment—providing a solid foundation for production-ready applications

Recommendations & Future Directions

Expand Dataset

Train with larger custom dataset: more classes, varied lighting conditions, and diverse environments

Comprehensive Evaluation

Per-class AP analysis, latency profiling, model quantization for edge deployment testing

3D Localization

Implement depth estimation and 3D positioning for enhanced spatial understanding

Mobile Deployment

Optimize for on-device inference on Android/iOS platforms for portable real-time detection

Enhanced User Interface

Add model switcher, confidence threshold controls, result saving, and session logging

Research Impact

- Advance real-time object detection research
- Enable production-ready applications
- Bridge research-to-implementation gap

Next Steps

These recommendations will transform our prototype into a production-grade system suitable for real-world deployment across various applications

Thank You for Your Attention

Questions & Discussion

Performance Benchmarking of YOLO Architectures

for Real-Time Object Detection

Team Members

Beyza GÜLER
YOLOv11 Specialist

Ramazan YILDIZ
AI Project Lead

Abdelrahman MOHAMED
YOLOv5 Specialist