Object-Detection-with-YOLOv3 - Project Documentation

# 1. Project Overview

This project implements a comprehensive microservice-based object detection system utilising the YOLOv3-Tiny model from Ultralytics. The system is designed with a modular architecture comprising two main components that work together to provide real-time object detection capabilities through a user-friendly web interface.  
  
The solution addresses the need for lightweight, CPU-efficient object detection that can be easily deployed and scaled using containerization technologies.

## 1.1 Project Objectives

* Implement a lightweight object detection system using YOLOv3-Tiny model
* Create a modular microservice architecture for scalability
* Provide a user-friendly web interface for image upload and detection visualisation
* Ensure easy deployment through containerization
* Generate structured outputs (JSON detections and annotated images)

## 1.2 Key Features

* CPU-optimized YOLOv3-Tiny model for fast inference
* Flask API returning JSON detections and Base64-encoded annotated images
* Interactive Streamlit web interface for easy testing
* Docker containerization for consistent deployment
* Comprehensive output generation for analysis and documentation

# 2. System Architecture

The system follows a microservice architecture pattern with clear separation of concerns between the AI processing backend and the user interface frontend. This design promotes scalability, maintainability, and allows for independent deployment and scaling of components.

## 2.1 System Components

### AI Service (Backend)

• Framework: Flask web framework  
• Model: Ultralytics YOLOv3-Tiny PyTorch model (.pt format)  
• Functionality: Processes uploaded images and returns detection results  
• Output Format: JSON with detections and Base64-encoded annotated images  
• Endpoint: POST /detect accepting multipart/form-data

### UI Service (Frontend)

• Framework: Streamlit for rapid web application development  
• Functionality: File upload interface and results visualization  
• Integration: Communicates with AI service via HTTP requests  
• Features: Image upload, detection triggering, and results display

# 3. Technical Implementation

## 3.1 Technology Stack

* Python 3.11+ - Core programming language
* Ultralytics - YOLOv3-Tiny model implementation
* Flask - API framework
* Streamlit - Frontend web application framework
* OpenCV - Computer vision operations
* NumPy - Numerical computations
* Pillow - Image processing
* Docker & Docker Compose - Containerization and orchestration

## 3.2 Model Implementation Details

The YOLOv3-Tiny model was chosen for its optimal balance between accuracy and computational efficiency. The implementation leverages the Ultralytics library which provides a simplified Python API for loading and running inference with PyTorch models.  
  
The model processes input images and returns:  
• Object class predictions  
• Confidence scores for each detection  
• Bounding box coordinates (x1, y1, x2, y2 format)  
  
The system automatically handles model loading, preprocessing, inference, and post-processing to generate both structured data and visual outputs.

# 4. Solution Development Process

The development process followed a systematic approach, starting with research and requirements analysis, followed by iterative implementation and testing. Each decision was made considering factors such as performance, scalability, and ease of deployment.

## 4.1 Key Design Decisions

### Model Selection Rationale

YOLOv3-Tiny was selected over other object detection models for several key reasons:  
  
1. Performance Efficiency: Optimised for CPU inference with acceptable accuracy trade-offs  
2. Library Support: Excellent integration with Ultralytics framework  
3. Model Size: Compact model suitable for containerized deployment  
4. Community Support: Well-documented with extensive community resources

### Microservice Architecture Decision

The microservice approach was chosen to:  
  
1. Enable independent scaling of AI processing and UI components  
2. Facilitate easier maintenance and updates  
3. Allow for technology diversity (Flask for API, Streamlit for UI)  
4. Support container-based deployment strategies

# 7. References and Acknowledgments

This project was developed using various open-source technologies and resources. The following references were instrumental in the implementation:

* Ultralytics YOLOv3 Implementation: https://github.com/ultralytics/yolov3
* Flask Web Framework Documentation: https://flask.palletsprojects.com/
* Streamlit Documentation: https://docs.streamlit.io/
* YOLOv3-Tiny Original Paper and Darknet Implementation: https://pjreddie.com/darknet/yolo/
* Docker Documentation: https://docs.docker.com/
* OpenCV Python Documentation: https://docs.opencv.org/4.x/d6/d00/tutorial\_py\_root.html

## 7.1 Acknowledgments

Special thanks to the open-source community for providing robust tools and frameworks that made this implementation possible. The Ultralytics team deserves particular recognition for their excellent YOLOv3 implementation and documentation.