**July 11, 2024**

**OBJECT RECOGNITION**

[Overview 1](#_Toc5878)

[Technical Stack 1](#_Toc5879)

[Implementation Plan: 2](#_Toc5880)

[Team Composition 3](#_Toc5881)

[Our Understanding 3](#_Toc5882)

# Overview

# Object recognition is a fundamental task in computer vision that involves identifying and classifying objects within an image. This process is essential for applications such as image search, autonomous driving, and surveillance. By detecting and labeling objects, systems can interpret visual data and perform tasks that require an understanding of the environment.

# Technical Stack

**Python (3.12.4):** We will use Python as the backend programming language, with FastAPI for serving RESTful APIs to the frontend. FastAPI is chosen for its high performance and ease of use, especially when building APIs that require asynchronous capabilities.

**React Js (18.3.1):** We will use React for the frontend, along with popular libraries like Material-UI or Bootstrap for themes and styling. Axios will handle API calls due to its simplicity and flexibility. React's component-based architecture allows for building a dynamic and responsive user interface.

**Database (NoSQL -** **MongoDB):** We will use MongoDB for the application because of its flexibility and scalability. MongoDB is suitable for applications that require a flexible schema and scalability, especially with unstructured or semi-structured data. Its document-oriented storage is particularly advantageous for handling complex data structures and large volumes of data.

**Logging:** We will use the standard logging library as a robust option. It provides a flexible framework for logging messages from the application, allowing us to record errors, warnings, and other information for debugging and analysis. Its integration with various handlers and formatters makes it versatile for different logging requirements, ensuring that logs can be easily managed, formatted, and directed to various outputs like files, consoles, or external logging services.

# Model Comparison:

|  |  |  |  |
| --- | --- | --- | --- |
| Model Advantages | | Disadvantages | Property |
| YOLO | - Very fast processing speed, able to process 45 frames per second or more, making it suitable for real-time applications | - Lower accuracy compared to other models, especially for small objects and when there is significant overlap between objects | - Number of bounding boxes, confidence threshold  - Small context window due to single pass per image |
| Faster RCNN | - Achieves high accuracy in object detection tasks | - Computationally expensive and slow compared to other models like YOLO | - Anchor boxes, proposal regions, non-max suppression  - Larger context window due to region proposal network |
| SSD | - Balances speed and accuracy well, can process 22-46 frames per second while maintaining good performance | - Less accurate overall compared to Faster RCNN, and may struggle more with tiny objects | - Number of default boxes, feature map sizes  - Moderate context window due to multiple feature maps |
| RetinaNet | - Achieves state-of-the-art object detection accuracy | - Has a slower processing speed and more complex architecture compared to other models | - Focal loss parameters, anchor scales  - Large context window due to FPN (Feature Pyramid Network) |
| EfficientDet | - Efficient in terms of both speed and accuracy | - Less proven than other models and complex to implement | - Compound scaling factors, BiFPN layers  - Variable context window due to compound scaling |
| CenterNet | - Accurate and efficient for real-time object detection | - A newer model, so its performance may depend more on the specific data used | -Key point estimation parameters  - Moderate context window due to heatmap and offset estimation |
| R-FCN | - Faster and more accurate than Faster R-  CNN | - Less flexible and still computationally demanding | - Position-sensitive score maps  - Moderate context window due to position-sensitive RoI pooling |
| Cascade RCNN | - Achieves very high accuracy | - Expensive and requires longer training time compared to other models | - Multiple detection stages, IoU thresholds  - Large context window due to cascading stages |

# 

# Implementation Plan:

We propose approximately 1.5 months for this project, with the following timeline breakdown:

1. **Requirements Gathering**: Conduct thorough discussions with stakeholders to understand their needs and preferences. This phase includes defining the project scope, identifying key features, and documenting technical and non-technical requirements. (4 days)
2. **System Design**: Plan 1 week for developing detailed system architecture and user interface design. This phase includes creating prototypes to validate concepts and gather feedback iteratively. It involves creating wireframes, data flow diagrams, and selecting technology stacks and design patterns. (1 week)
3. **Front-end Development**: Allocate 1 week for developing UI/UX features using ReactJS and selected libraries (e.g., Material-UI, Bootstrap). Focus on responsive design and usability testing. This phase includes implementing user authentication, navigation, and interactive elements while ensuring cross-browser compatibility and performance optimization. (1 week)
4. **Back-end Development**: Allocate 2 weeks for implementing core features using Python with FastAPI for backend APIs and MongoDB for database integration. Include robust error handling, data validation, and security measures. This phase covers setting up the server environment, API endpoints, database schemas, and implementing business logic and third-party integrations. (2 weeks)
5. **Testing**: Dedicate 1 week for rigorous testing, including unit testing, integration testing, and user acceptance testing (UAT). Ensure comprehensive test coverage to identify and resolve issues early. This phase also includes performance testing, security testing, and preparing test documentation to ensure the application meets all quality standards. (1 week)
6. **Deployment**: Allocate 0.5 weeks for deploying the application to a secure hosting environment and configuring it for standalone PC usage. Include the setup of monitoring and logging for ongoing maintenance. This phase involves setting up the production environment, performing final sanity checks, and configuring automated deployment pipelines for continuous integration and continuous deployment . (0.5 weeks)

# Team Composition

Based on the proposed timeline and scope of work, we will handle the development, but we need assistance from a team lead for guidance and support during development and testing. The team lead will help ensure best practices and oversee the testing process.

|  |  |
| --- | --- |
| **Resource Role** | **Resource Name** |
| Designing | Sania Awais, Aleeza Rubab |
| Development | Sania Awais, Aleeza Rubab |
| Testing | Sania Awais, Aleeza Rubab |
| Deployment | Sania Awais, Aleeza Rubab |

# 

# Our Understanding

An object recognition project involves gathering annotated data, selecting a suitable pretrained model (e.g., YOLO, SSD), evaluating its performance, deploying it for real-time processing, and iteratively refining the model to enhance accuracy and adaptability.