Comprehensive List of Final Project Ideas in Object Detection

Project 1: Real-Time Coral Bleaching Detection with YOLOv9 & XAI	
Objective	To build a real-time system that detects and quantifies coral bleaching in underwater images, and use Explainable AI (XAI) to interpret the model's predictions.
Core Novelty	• **State-of-the-Art Model:** Application of a very recent model (YOLOv9) to a pressing environmental issue. • **Explainability:** Integration of XAI adds a layer of trust and scientific value, moving beyond just detection to understanding *why* a decision is made.
Methodology, Tools, & Architecture	• **Tools:** Use **PyTorch** for model training. For annotation, use tools like **CVAT** or **LabelImg**. XAI libraries like **Captum** for PyTorch can be used, or a direct implementation of **Grad-CAM++**. • **Architecture:** A two-class object detector (e.g., 'healthy', 'bleached'). The core comparison will be between a **YOLOv8** baseline and the newer **YOLOv9** architecture, focusing on its specific improvements like Programmable Gradient Information (PGI). • **Workflow:** 1. Collect and annotate a dataset of coral images. 2. Train the YOLOv8 baseline and evaluate performance (mAP, F1-Score, inference time). 3. Train YOLOv9 on the same data and conduct a thorough comparative analysis. 4. Implement Grad-CAM++ to generate heatmaps on correctly detected bleached/healthy corals to visualize the model's focus areas.
Potential Challenges	• **Dataset Availability:** Finding or creating a well-annotated dataset for coral bleaching may be the most time-consuming part. • **Subtle Features:** Differentiating between healthy, stressed, and fully bleached coral can be challenging due to subtle visual cues. • **XAI Interpretation:** The results from XAI methods require careful and critical analysis to derive meaningful conclusions.

Project 2: Few-Shot Learning for Rare Fish Species Identification

Objective	To develop an object detection model that can accurately identify rare fish species using only a very small number of training examples ("few-shot" learning).
Core Novelty	• **Addresses Data Scarcity:** Tackles the common and difficult real-world problem of having insufficient data for rare classes. • **Advanced Technique:** Explores cutting-edge few-shot learning methodologies, which are highly relevant in current deep learning research.
Methodology, Tools, & Architecture	• **Tools: PyTorch** is highly recommended due to its flexibility for research. Libraries like **PyTorch-meta** could provide useful building blocks. Data management will rely on **Pandas** and **NumPy**. • **Architecture:** A common approach is a **two-phase detector**. Use a YOLOv8 backbone pre-trained on COCO or a large set of "base" fish classes. Then, freeze the backbone and re-train or fine-tune only the detection head/classifier on a balanced set of "base" and "novel" (rare) classes. Alternatively, explore methods based on **Prototypical Networks** that operate on the feature vectors extracted by the YOLO backbone. • **Workflow:** 1. Use the **WildFish dataset** and designate species with many images as 'base' classes and those with few as 'novel' classes. 2. Implement the baseline: fine-tune a standard YOLOv8 on all classes and evaluate its poor performance on the rare ones. 3. Implement the two-phase fine-tuning approach and compare its performance on the novel classes against the baseline.
Potential Challenges	• **Algorithm Complexity:** Few-shot learning concepts and implementations can be more complex than standard supervised training. • **Intra-class Variance:** Rare species might still have high visual variance (e.g., different lighting, poses), making it hard to learn from few samples. • **Performance Ceiling:** Achieving very high accuracy is inherently difficult and a primary research challenge in this sub-field.

Project 3: High-Fidelity Fish Tracking & Population Density Estimation

Objective

To create a comprehensive pipeline that detects, classifies, and tracks individual fish in videos to provide an accurate estimation of population density.

Core Novelty

- **Holistic System:** Moves beyond simple object detection to a more complex and applicable task of population analysis, creating a full end-to-end application.
- **Data-Driven Ecology:** The output is not just bounding boxes but a quantitative metric (density/count) that has direct value for ecological monitoring.

Methodology, Tools, & Architecture

- **Tools:** Essential tools include **PyTorch** for detection, **OpenCV** for all video I/O and processing, and **NumPy/Pandas** for analyzing tracking data.
- **Architecture:** This is a **pipeline architecture**. The first stage is a highly efficient object detector, such as **YOLOv8** or **YOLOv9**. The output of the detector (bounding boxes, class scores) is fed frame-by-frame into a tracking algorithm. **ByteTrack** is a strong choice as it is recent, fast, and does not require a separate re-identification model.
- **Workflow:**
  1. Train/fine-tune a YOLO model on the WildFish dataset for robust fish detection.
  2. Integrate the trained model with an open-source ByteTrack implementation.
  3. Develop a Python script to process a video: for each frame, detect objects, pass detections to the tracker, and get back tracked objects with unique IDs.
  4. Design and implement a counting logic (e.g., a virtual line crossing or counting unique IDs within a region of interest over a time window) to estimate density.

Potential Challenges

- **Tracking Errors:** Occlusions (fish swimming behind rocks or each other) and fast motion can cause the tracker to lose or switch object IDs, impacting counts.
- **Real-Time Processing:** Integrating detection and tracking can be computationally expensive; optimizing the pipeline for real-time performance is a challenge.
- **Counting Logic:** Designing a robust counting algorithm that avoids double-counting or missing fish requires careful design and parameter tuning.

Project 4: Real-Time Dynamic Gesture Recognition with Temporal Fusion

Objective	To develop a high-accuracy system for recognizing dynamic, multi-frame hand gestures from video streams, with a focus on optimizing the final model for real-time deployment on an edge device.
Core Novelty	• **Temporal Modeling:** Moves beyond static gesture detection by explicitly modeling the time dimension. This is achieved by fusing features across frames using a recurrent layer. • **Edge Optimization:** Focuses on a practical, challenging aspect of deep learning: model quantization and optimization to achieve real-time inference on low-power devices.

Methodology, Tools, & Architecture	• **Tools: PyTorch/TensorFlow**, **OpenCV** for video handling. For deployment, use **NVIDIA TensorRT** (for Jetson devices) or **TensorFlow Lite** (for general edge/mobile).
	• **Architecture:** A **two-stage hybrid architecture**.
	  – **Stage 1 (Spatial Feature Extraction):** Use a pre-trained, lightweight pose estimator like **YOLOv8-Pose** to extract the '(x, y, confidence)' coordinates of hand keypoints from each frame.
	  – **Stage 2 (Temporal Classification):** Feed the sequence of these keypoint vectors into a lightweight recurrent neural network, such as a **GRU** (Gated Recurrent Unit), followed by a final linear layer and Softmax for classification.
	• **Workflow:**
	  1. Pre-process a dynamic gesture dataset (e.g., **Jester Dataset**) into sequences of frames.
	  2. Run the pose estimator on all frames to generate sequences of keypoint data.
	  3. Train the GRU/LSTM classifier on these prepared sequences.
	  4. (Novelty Focus) Convert and quantize the trained models to a format like ONNX or TFLite and benchmark the performance vs. latency trade-off.

Potential Challenges	• **Real-Time Latency:** The combined execution time of the pose estimator and the recurrent network must be very low, which is a significant engineering challenge. • **Data Pipeline Complexity:** Handling video data, extracting features, and creating sequences for a recurrent model is more complex. • **Synchronization:** Ensuring the smooth flow of data between the two stages of the model in a real-time application requires careful implementation.

Project 5: Fine-Grained Vehicle Recognition in Parking Lots

Objective

To design and implement a system that first detects all vehicles in a dense parking lot image and then performs fine-grained classification on each vehicle to identify its specific make and model (e.g., 'Ford Fiesta', 'BMW 3 Series').

Core Novelty

- **Fine-Grained Classification at Scale:** Moves beyond simple 'car' detection to tackle the much harder problem of classification among hundreds of visually similar classes in a challenging, real-world setting.
- **Robustness to Occlusion:** The core challenge is making the fine-grained classification work on partially visible cars, a common scenario in parking lots but absent from most classification datasets.
- **Hybrid Transformer Architecture:** Proposes a modern pipeline combining the strengths of a convolutional or transformer-based detector with a powerful Vision Transformer (ViT) classifier for the fine-grained task.

Methodology, Tools, & Architecture

- **Tools: PyTorch** and the **Hugging Face Transformers** library for ViT models. **OpenCV** for image processing. **Scikit-learn** for metrics.
- **Datasets:** A two-dataset approach is necessary. Use a detection dataset like **PKLot** or **COCO** to train the detector. For the classifier, use a dedicated fine-grained dataset like the **Stanford Cars Dataset** (196 classes).
- **Architecture:** A **two-stage pipeline**:
  - **Stage 1 (Detector):** A high-performance detector like **YOLOv8** or **RT-DETR** is trained to find bounding boxes for all 'car' objects.
  - **Stage 2 (Classifier):** Each detected bounding box is cropped from the main image and passed to a fine-tuned **Vision Transformer (ViT)** or **Swin Transformer** model that classifies it into one of the 196 makes/models.
- **Workflow:**
  1. Train the YOLO/RT-DETR model on a detection dataset.
  2. Fine-tune the ViT model on the Stanford Cars dataset.
  3. Build a script that connects them: take an image, run detector, loop through detections, crop each car, run classifier on the crop, and display final labeled image.
  4. Evaluate the pipeline on a test set of parking lot images with manual annotations.

Potential Challenges

- **Domain Shift:** The classifier trains on clean, centered images (Stanford Cars) but must perform inference on cluttered, occluded, and variably-angled crops from the detector. This is a major challenge.
- **Extreme Similarity:** Many car models are visually identical from certain angles or across model years, making classification ambiguous even for humans.
- **Computational Cost:** Training and running inference with a Vision Transformer is computationally intensive. The two-stage pipeline will be slower than a single detector.