# **Title:** Real-Time Dual-Head YOLOv8 Architecture for Drone-Based Object and Disaster Detection

---

### 1. Introduction

This project aims to enable a drone to scan and identify objects in real-time, encompassing both everyday objects and disaster-related situations (e.g., floods, earthquakes,etc). Given the need for high-speed, accurate detection in a resource-constrained environment, YOLOv8 (You Only Look Once, version 8) is chosen due to its balance of speed and accuracy. However, since YOLOv8 is pre-trained on the COCO dataset (which covers only everyday objects), a dual-headed architecture is proposed.

---

### 2. Motivation for Dual-Head Architecture

- **Real-Time Constraints:** Real-time processing requires minimal latency. Running two separate models sequentially or in parallel would increase inference time significantly.
- **Modularity:** A dual-head model maintains shared backbone computation, reducing redundant processing and resource load.
- **Task Specialization:** One head can specialize in general object detection (COCO) while the other focuses on disaster detection, leading to better performance in each domain.
- **Unified Deployment:** Simplifies deployment on edge devices like drones compared to handling two separate models.

---

### 3. YOLOv8 Overview

YOLOv8 is an advanced object detection architecture known for its real-time performance. It consists of:

- **Backbone:** For feature extraction.
- **Neck:** For feature fusion.
- **Head:** For object detection and classification.

---

**4. Proposed Architecture**

The modified YOLOv8 model consists of:

- **Shared Backbone & Neck:** Extract and combine features from input images.
- **Head A (General Object Detection):** Uses pre-trained COCO weights.
- **Head B (Disaster Detection):** Fine-tuned on a custom disaster dataset.

**Flowchart:**

```
        +----------------------+
        |   Input Image       |
        +----------+------------+
                   |
        +---------v---------+
        | Shared YOLOv8     |
        | Backbone & Neck   |
        +---------+---------+
                   |
     +---------------+---------------+
     |                               |
+----------v---------+         +----------v----------+
| Head A (COCO)      |         | Head B (Disaster)   |
| Pre-trained        |         | Fine-tuned on       |
| on everyday objects|         | custom dataset      |
+-------------------+         +--------------------+
```

---

**5. Dataset Preparation**

- **COCO Dataset:** Already labeled and structured, used as-is for Head A.
- **Disaster Dataset:**
    - Collected from open-source repositories, satellite images, drone footage.
    - Categories: Flooded areas, collapsed buildings,  etc.

    - Annotations: The images will be annotated manually.

---

### 6. Training Strategy

- **Step 1:** Load pre-trained YOLOv8 with COCO weights.
- **Step 2:** Freeze Backbone and Head A layers.
- **Step 3:** Add Head B and fine-tune only Head B on the disaster dataset.
- **Step 4:** Perform end-to-end evaluation to ensure no conflict in detections.

---

### 7. Inference Strategy

At inference time:

- Image passes through shared layers.
- Both heads produce bounding boxes and class scores.
- Post-processing merges detections with label source information.

---

### 8. Advantages of Dual-Head over Dual-Model

| Feature | Dual-Head Model | Dual Model Setup |
|---|---|---|
| Inference Speed | Faster (shared backbone) | Slower (redundant passes) |
| Memory Footprint | Lower | Higher |
| Deployment Simplicity | Easier | Complex (sync required) |
| Feature Sharing | Yes | No |
| Training Flexibility | High | Moderate |

---

## 11. Conclusion

A dual-head YOLOv8 architecture enables efficient, real-time detection of both everyday and disaster-related objects in drone-based applications. This structure offers a balanced trade-off between specialization and performance while maintaining real-time constraints crucial for autonomous aerial systems.

---

**Visual Example:**
(Insert drone image showing bounding boxes labeled by both heads with color codes: e.g., Blue for COCO objects, Red for Disaster scenarios.)