Implementación Avanzada de Visión por Computadora para el Monitoreo de Salmones: De la Detección a la Estimación de Pose y Segmentación de Instancia con YOLOv8 y CVAT

Parte I: Fundamentos Arquitectónicos: De la Detección al Análisis Granular

La transición de sistemas de detección de objetos básicos a metodologías de análisis más sofisticadas como la segmentación de instancia y la estimación de pose representa un salto cualitativo en la capacidad de monitoreo para la acuicultura. Este informe detalla la implementación de estas técnicas avanzadas utilizando el framework YOLOv8, con un enfoque en la preparación de datos provenientes de la plataforma de anotación CVAT y la aplicación directa al estudio de salmones.

1.1. Las Limitaciones de las Cajas Delimitadoras en la Acuicultura

Los sistemas de detección de objetos basados en cajas delimitadoras (bounding boxes), como el que se encuentra actualmente en uso, son efectivos para tareas fundamentales como el conteo de individuos en un recinto. Proporcionan una localización aproximada y una clasificación de cada pez, lo cual constituye una base valiosa. Sin embargo, esta representación es inherentemente limitada para análisis más profundos.
Una caja delimitadora es una aproximación geométrica burda que no captura la morfología real del pez. Un salmón puede curvar su cuerpo, lo que altera significativamente el área de la caja sin un cambio en su masa real. Por lo tanto, el uso del área de la caja como un proxy para la biomasa es propenso a errores significativos. De manera similar, una caja no puede describir el estado de salud, el comportamiento natatorio, la frecuencia respiratoria o los niveles de estrés de un individuo. Para extraer esta información de alto valor, es imperativo adoptar técnicas que capturen la forma precisa y la postura del animal.

1.2. Introducción a YOLOv8: Un Framework Unificado

YOLOv8 (You Only Look Once, versión 8) es un detector de una sola etapa de última generación, reconocido por su excepcional equilibrio entre velocidad y precisión, lo que lo hace ideal para aplicaciones en tiempo real. Su arquitectura se puede descomponer conceptualmente en tres componentes principales:

1. **Backbone (Columna Vertebral):** Una red neuronal convolucional profunda (por ejemplo, CSPDarknet) que extrae características jerárquicas de la imagen de entrada a diferentes escalas. Las capas iniciales capturan características de bajo nivel como bordes y texturas, mientras que las capas más profundas capturan patrones más complejos.
2. **Neck (Cuello):** Componentes como el Path Aggregation Network (PANet) que fusionan y refinan las características extraídas por el backbone en diferentes niveles. Esto permite que el modelo combine información semántica de alto nivel con detalles espaciales de bajo nivel, mejorando la detección de objetos de diversos tamaños.
3. **Head (Cabeza):** La sección final de la red que realiza las predicciones. Es aquí donde reside la flexibilidad de YOLOv8.

La innovación clave del framework YOLOv8 no es ser un único modelo monolítico, sino una plataforma adaptable. Al modificar o añadir "cabezas" de predicción, el mismo backbone y cuello, potentes y eficientes, pueden ser reutilizados para tareas distintas a la detección de objetos. Esta eficiencia arquitectónica es fundamental, ya que permite entrenar modelos especializados para segmentación y estimación de pose sin necesidad de rediseñar toda la arquitectura desde cero.

1.3. Tarea 1: Segmentación de Instancia con YOLOv8-Seg

La segmentación de instancia es una tarea que va un paso más allá de la detección. Su objetivo es identificar cada objeto en una imagen (instancia) y, simultáneamente, generar una máscara de píxeles que delinee su contorno exacto.

Arquitectónicamente, el modelo YOLOv8-Seg logra esto añadiendo una segunda cabeza de predicción, la "cabeza de máscara", que opera en paralelo a la cabeza de detección de cajas. En lugar de predecir directamente la máscara de píxeles para cada instancia, lo cual sería computacionalmente costoso, YOLOv8-Seg adopta un enfoque más eficiente. La cabeza de máscara predice un pequeño conjunto de máscaras prototipo para toda la imagen y, para cada instancia detectada, predice un conjunto de coeficientes. La máscara final para una instancia se genera mediante una combinación lineal de las máscaras prototipo ponderada por los coeficientes predichos.

Para el monitoreo de salmones, la aplicación más directa y valiosa de la segmentación de instancia es la estimación precisa de la biomasa. Al contar el número exacto de píxeles dentro de la máscara de un salmón, se obtiene su área visible. Esta medida, a diferencia del área de la caja delimitadora, está fuertemente correlacionada con la masa real del pez y es menos

susceptible a las variaciones de pose.

1.4. Tarea 2: Estimación de Pose con YOLOv8-Pose

La estimación de pose, también conocida como detección de puntos clave (keypoints), se enfoca en localizar las coordenadas de puntos de referencia anatómicos específicos en un objeto. Para un salmón, estos puntos podrían ser la punta del hocico, el ojo, la aleta dorsal y la cola.
El modelo YOLOv8-Pose adapta la cabeza de detección para esta tarea. Además de predecir la clase y la caja delimitadora para cada salmón, la cabeza predice un vector que contiene las coordenadas y la confianza para cada punto clave predefinido en un "esqueleto" anatómico. La salida para cada punto clave es típicamente un triplete (x, y, c), donde (x, y) son las coordenadas del punto y c es la confianza de la predicción.
Esta técnica es la puerta de entrada al análisis cuantitativo del comportamiento. La posición de los puntos clave a lo largo del cuerpo permite calcular la curvatura de la columna, un indicador de la eficiencia natatoria y posible estrés. El seguimiento del punto clave en la cola a través de fotogramas de video permite medir la frecuencia y amplitud del batido de cola, un proxy directo del gasto energético. Posturas anómalas o falta de movimiento, cuantificadas a través de los puntos clave, pueden ser indicadores tempranos de enfermedad o malestar.
La elección entre segmentación y estimación de pose no es mutuamente excluyente. Más bien, representan herramientas complementarias para construir un perfil completo del estado de los salmones. La segmentación responde a la pregunta "¿Cuál es la forma y el tamaño del pez?", proporcionando datos morfológicos. La estimación de pose responde a "¿Qué está haciendo el pez y cómo se está moviendo?", proporcionando datos cinemáticos. Un sistema de monitoreo verdaderamente avanzado, como se discutirá en la Parte IV, integra ambas modalidades para obtener una comprensión holística y sin precedentes del bienestar animal.

Parte II: Implementando la Segmentación de Instancia para una Morfología Precisa del Salmón

Esta sección proporciona una guía práctica y detallada para construir la solución de segmentación de instancia, abarcando desde la preparación de datos anotados en CVAT hasta el código de inferencia y la utilización de los resultados.

2.1. Justificación de la Selección del Modelo: YOLOv8-Seg vs. Alternativas

La elección de YOLOv8-Seg para esta tarea se fundamenta en sus ventajas arquitectónicas

para aplicaciones que requieren un procesamiento rápido, como el análisis de video en tiempo real. Su diseño de una sola etapa le confiere una velocidad de inferencia significativamente mayor en comparación con los modelos de dos etapas como Mask R-CNN. Mientras que Mask R-CNN primero propone regiones de interés y luego realiza la clasificación y la segmentación en una segunda etapa, YOLOv8 realiza todas estas tareas en un único paso.

Aunque Mask R-CNN puede, en ciertos benchmarks académicos, ofrecer una calidad de máscara marginalmente superior, el compromiso entre velocidad y precisión se inclina decididamente a favor de YOLOv8-Seg para el monitoreo en acuicultura. La capacidad de procesar un alto número de fotogramas por segundo (FPS) es crítica para no perder eventos de comportamiento y para mantener la viabilidad económica del sistema en hardware comercial.

Tabla 1: Análisis Comparativo de Modelos de Segmentación de Instancia para Acuicultura

Característica	**YOLOv8-Seg**	**Mask R-CNN**	**Implicación para el Monitoreo de Salmones**
Arquitectura	Una Etapa	Dos Etapas	La arquitectura de una etapa permite una inferencia mucho más rápida, crucial para el análisis de video en tiempo real.
Velocidad de Inferencia (FPS)	Alta (e.g., 50-150 FPS en GPU)	Moderada (e.g., 5-20 FPS en GPU)	La alta velocidad de YOLOv8-Seg permite el procesamiento en vivo de múltiples flujos de video con hardware estándar.
Complejidad de Entrenamiento	Baja	Moderada	El proceso de entrenamiento de YOLOv8 es más directo y requiere menos configuración, acelerando el ciclo de desarrollo.
Precisión de Máscara (mAP)	Alta	Muy Alta	Mask R-CNN puede tener una ligera ventaja en la precisión de los bordes, pero la calidad de YOLOv8-Seg es

			excelente para aplicaciones prácticas como la estimación de biomasa.
Adecuación para Despliegue en Borde (Edge)	Excelente	Pobre	Los modelos YOLOv8-Seg son más ligeros y rápidos, lo que los hace adecuados para su despliegue en dispositivos de borde con recursos computacionales limitados.

2.2. Preparación de Datos: De Polígonos de CVAT a Conjuntos de Entrenamiento de YOLOv8

Este es el paso más crítico del proceso. La calidad del modelo final depende directamente de la calidad y el formato correcto de los datos de entrenamiento.
Exportación desde CVAT: El primer paso es exportar las anotaciones desde CVAT. Es fundamental entender que el formato de exportación por defecto "YOLO" solo guarda las cajas delimitadoras. Para la segmentación, se deben preservar los polígonos que definen el contorno de cada salmón. Los formatos de exportación adecuados son **COCO 1.0 (JSON)** o **CVAT for images 1.1 (XML)**, ya que ambos almacenan las coordenadas de los vértices de los polígonos.
Conversión al Formato YOLOv8-Seg: Una vez exportado el archivo de anotaciones (e.g., annotations.json), se debe procesar para generar el formato requerido por YOLOv8. Este formato consiste en un archivo de texto (.txt) por cada imagen, con el mismo nombre. Cada línea dentro de este archivo de texto representa una instancia de un salmón y sigue la estructura: <class-index> <x1> <y1> <x2> <y2>... <xn> <yn>. Todas las coordenadas (x, y) de los vértices del polígono deben estar normalizadas, es decir, divididas por el ancho y el alto de la imagen, respectivamente, para que sus valores estén en el rango $$.
El siguiente script en Python, prepare_segmentation_data.py, realiza esta conversión a partir de una exportación en formato COCO JSON. Este script no solo convierte los datos, sino que también incorpora validaciones para asegurar la calidad de las anotaciones, un paso crucial para evitar errores durante el entrenamiento.

Python

```
# prepare_segmentation_data.py
import json
import os
from pathlib import Path
from tqdm import tqdm
import cv2

def convert_coco_to_yolo_seg(coco_json_path, save_dir):
    """
    Convierte anotaciones de segmentación en formato COCO JSON al formato YOLOv8-Seg.

    Este script asume que las imágenes ya están en una carpeta 'images' y creará
    una carpeta 'labels' paralela.
    """
    save_path = Path(save_dir)
    labels_path = save_path / 'labels'
    images_path = save_path / 'images'
    labels_path.mkdir(parents=True, exist_ok=True)

    with open(coco_json_path, 'r') as f:
        data = json.load(data)

    # Crear un mapeo de image_id a nombre de archivo y dimensiones
    images_info = {img['id']: {'file_name': img['file_name'], 'height': img['height'], 'width':
img['width']} for img in data['images']}

    # Crear un mapeo de category_id a class_index (asumiendo que los IDs de COCO no son
necesariamente 0-indexados)
    categories = {cat['id']: i for i, cat in enumerate(data['categories'])}

    # Agrupar anotaciones por image_id
    annotations_by_image = {}
    for ann in data['annotations']:
        img_id = ann['image_id']
        if img_id not in annotations_by_image:
            annotations_by_image[img_id] =
        annotations_by_image[img_id].append(ann)

    print("Iniciando conversión de anotaciones...")
    for img_id, anns in tqdm(annotations_by_image.items()):
        img_info = images_info[img_id]
        img_h, img_w = img_info['height'], img_info['width']
```

```
        # Validar que las dimensiones de la imagen son válidas
        if img_h == 0 or img_w == 0:
            print(f"Advertencia: La imagen {img_info['file_name']} tiene dimensiones cero. Se omitirá.")
            continue

        label_name = Path(img_info['file_name']).stem + '.txt'
        label_path = labels_path / label_name

        with open(label_path, 'w') as f:
            for ann in anns:
                # Validar que es una anotación de segmentación
                if 'segmentation' not in ann or not ann['segmentation']:
                    continue

                class_index = categories[ann['category_id']]

                # El formato de segmentación de COCO puede tener múltiples polígonos para una instancia
                for seg in ann['segmentation']:
                    # Validar que el polígono tiene al menos 3 vértices
                    if len(seg) < 6:
                        print(f"Advertencia: Polígono inválido (menos de 3 vértices) en {img_info['file_name']}. Se omitirá.")
                        continue

                    # Normalizar coordenadas
                    normalized_coords =
                    for i in range(0, len(seg), 2):
                        x = seg[i] / img_w
                        y = seg[i+1] / img_h
                        normalized_coords.extend([x, y])

                    # Escribir en el archivo de etiqueta
                    line = f"{class_index} " + " ".join(map(str, normalized_coords))
                    f.write(line + '\n')

    print(f"Conversión completada. Las etiquetas se guardaron en: {labels_path}")

# Ejemplo de uso:
# Asegúrese de que su estructura de carpetas sea:
# /path/to/dataset/
```

```
#   - images/
#     - train/
#       - img1.jpg
#       -...
#     - val/
#       - imgN.jpg
#       -...
#   - annotations/
#     - instances_train.json
#     - instances_val.json

# convert_coco_to_yolo_seg('/path/to/dataset/annotations/instances_train.json',
'/path/to/yolo_dataset/train/')
# convert_coco_to_yolo_seg('/path/to/dataset/annotations/instances_val.json',
'/path/to/yolo_dataset/val/')
```

Estructura del Conjunto de Datos y data.yaml:
El entrenamiento con YOLOv8 requiere una estructura de directorios específica y un archivo de configuración data.yaml.
Estructura de directorios:

```
/path/to/salmon_segmentation_dataset/
├── images/
│   ├── train/
│   │   ├── salmon_001.jpg
│   │   └── ...
│   └── val/
│       ├── salmon_101.jpg
│       └── ...
├── labels/
│   ├── train/
│   │   ├── salmon_001.txt
│   │   └── ...
│   └── val/
│       ├── salmon_101.txt
│       └── ...
└── data.yaml
```

Contenido del archivo data.yaml:

YAML

```
# data.yaml
train: /path/to/salmon_segmentation_dataset/images/train
val: /path/to/salmon_segmentation_dataset/images/val

# Número de clases
nc: 1

# Nombres de las clases
names: ['salmon']
```

2.3. El Código Base de Segmentación: Entrenamiento e Inferencia

A continuación se presentan los scripts completos para entrenar el modelo de segmentación y para realizar inferencias con el modelo ya entrenado.
Script de Entrenamiento (train_segmentation.py):
Este script utiliza el aprendizaje por transferencia, comenzando con los pesos de un modelo YOLOv8 pre-entrenado en el conjunto de datos COCO, lo que acelera la convergencia y mejora el rendimiento, especialmente con conjuntos de datos personalizados de tamaño moderado.

Python

```
# train_segmentation.py
from ultralytics import YOLO

def main():
    # Cargar un modelo de segmentación pre-entrenado YOLOv8n-seg.pt
    # 'n' se refiere al modelo nano, el más pequeño y rápido.
    # Se pueden usar otros tamaños como 's', 'm', 'l', 'x' para mayor precisión a costa de velocidad.
    model = YOLO('yolov8n-seg.pt')  # Carga pesos pre-entrenados

    # Entrenar el modelo con el conjunto de datos de salmones
    print("Iniciando el entrenamiento del modelo de segmentación...")
    results = model.train(
        data='path/to/salmon_segmentation_dataset/data.yaml',
        epochs=100,         # Número de épocas de entrenamiento
        imgsz=640,          # Tamaño de imagen de entrada
```

```
        batch=16,           # Tamaño del lote
        patience=20,        # Épocas a esperar sin mejora antes de detener el entrenamiento
        name='yolov8n_salmon_seg', # Nombre del experimento
        # Parámetros de aumento de datos (data augmentation)
        # Cruciales para la robustez en entornos subacuáticos
        hsv_h=0.015,  # Aumento de matiz
        hsv_s=0.7,    # Aumento de saturación
        hsv_v=0.4,    # Aumento de valor (brillo)
        degrees=0.0,  # Rotación
        translate=0.1,# Traslación
        scale=0.5,    # Escala (zoom)
        flipud=0.5,   # Flip vertical
        mosaic=1.0,   # Aumento de mosaico
        mixup=0.1     # Aumento de mixup
    )
    print("Entrenamiento completado.")
    print(f"El mejor modelo se guardó en: {results.save_dir}")

if __name__ == '__main__':
    main()
```

Script de Inferencia (predict_segmentation.py):
Este script carga el modelo personalizado que se acaba de entrenar y lo utiliza para predecir máscaras de segmentación en nuevas imágenes o videos.

Python

```
# predict_segmentation.py
import cv2
import numpy as np
from ultralytics import YOLO

def process_and_visualize(image_path, model):
    """Realiza la inferencia y visualiza los resultados de segmentación."""
    # Realizar la predicción
    results = model.predict(source=image_path)

    # Cargar la imagen original para dibujar sobre ella
    img = cv2.imread(image_path)

    # Verificar si se detectó algo
    if results.masks is None:
```

```
        print("No se detectaron salmones en la imagen.")
        return img

    # Iterar sobre cada instancia detectada
    for i, mask_data in enumerate(results.masks.data):
        # La máscara es un tensor, convertir a array de numpy
        mask = mask_data.cpu().numpy().astype(np.uint8)

        # Redimensionar la máscara al tamaño de la imagen original
        mask = cv2.resize(mask, (img.shape[1], img.shape), interpolation=cv2.INTER_NEAREST)

        # Crear una imagen de color para la máscara
        color = np.random.randint(0, 255, size=3, dtype=np.uint8)
        colored_mask = np.zeros_like(img, dtype=np.uint8)
        colored_mask[mask == 1] = color

        # Superponer la máscara en la imagen original con transparencia
        img = cv2.addWeighted(img, 1, colored_mask, 0.5, 0)

        # Dibujar el contorno de la máscara
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cv2.drawContours(img, contours, -1, (int(color), int(color[1]), int(color[2])), 2)

        # Obtener la caja delimitadora para poner una etiqueta
        box = results.boxes.xyxy[i].cpu().numpy().astype(int)
        conf = results.boxes.conf[i].cpu().numpy()
        label = f"Salmon: {conf:.2f}"
        cv2.rectangle(img, (box, box[1]), (box[2], box[3]), (int(color), int(color[1]), int(color[2])), 2)
        cv2.putText(img, label, (box, box[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255,
255), 2)

    return img

def main():
    # Cargar el modelo de segmentación personalizado entrenado
    # La ruta debe apuntar al archivo 'best.pt' generado durante el entrenamiento
    model_path = 'runs/segment/yolov8n_salmon_seg/weights/best.pt'
    try:
        model = YOLO(model_path)
    except Exception as e:
        print(f"Error al cargar el modelo: {e}")
        print("Asegúrese de que la ruta al archivo 'best.pt' es correcta.")
        return
```

```
    # Ruta a la imagen o video de entrada
    input_source = 'path/to/new_salmon_image.jpg' # o 'path/to/video.mp4'

    # Procesar y visualizar
    result_image = process_and_visualize(input_source, model)

    # Guardar o mostrar la imagen resultante
    output_path = 'segmentation_result.jpg'
    cv2.imwrite(output_path, result_image)
    print(f"Resultado guardado en {output_path}")

    # Para mostrar en una ventana (opcional)
    # cv2.imshow('Segmentation Result', result_image)
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```

2.4. Interpretando y Utilizando las Máscaras de Segmentación

El resultado de la segmentación no es solo una visualización, sino una fuente rica de datos cuantitativos.
Estimación de Biomasa: La aplicación más directa es el cálculo del área. La máscara predicha es una matriz binaria donde los píxeles con valor 1 pertenecen al salmón. El número de estos píxeles es directamente proporcional al área visible.

Python

```
def calculate_area_from_mask(mask_tensor):
    """Calcula el área en píxeles de una máscara de segmentación."""
    # La máscara es un tensor de PyTorch, se convierte a numpy y se suma
    mask_np = mask_tensor.cpu().numpy()
    area_pixels = np.sum(mask_np)
    return area_pixels

# Dentro del bucle de inferencia en predict_segmentation.py:
# for i, mask_data in enumerate(results.masks.data):
#     area = calculate_area_from_mask(mask_data)
```

print(f"Salmón {i+1}: Área visible = {area} píxeles")

Para convertir esta área en píxeles a una estimación de masa (en gramos o kilogramos), se requiere un paso de calibración. Esto típicamente implica:

1. Capturar imágenes de un conjunto de salmones de los cuales se conoce el peso real.
2. Calcular el área en píxeles para cada uno usando el modelo de segmentación.
3. Entrenar un modelo de regresión simple (e.g., regresión lineal o polinómica) que mapee el área en píxeles al peso real. Este modelo puede luego ser usado para estimar el peso de nuevos peces.

Filtrado y Análisis: Las máscaras también permiten filtrar detecciones espurias. Detecciones con un área de máscara muy pequeña pueden ser descartadas como ruido. Además, el análisis de la forma de la máscara (e.g., calculando su excentricidad o momentos de forma) podría revelar información adicional sobre la condición del pez.

Parte III: Desbloqueando Análisis de Comportamiento con la Estimación de Pose del Salmón

Esta sección aborda la tarea más compleja de la estimación de pose, desde la definición conceptual de un esqueleto anatómico para salmones hasta la implementación del código y la extracción de métricas cinemáticas.

3.1. Definiendo un Esqueleto Robusto para el Salmón: Un Enfoque Biomecánico

A diferencia de la estimación de pose humana, para la cual existen esqueletos estándar (e.g., COCO, MPII), no hay una definición preexistente para los salmones. La creación de un esqueleto útil es el primer y más crucial paso. Un esqueleto efectivo no debe basarse en puntos visualmente convenientes, sino en hitos anatómicos que sean **biomecánicamente significativos** y **consistentemente identificables** a través de diferentes individuos, poses y condiciones de iluminación.

La selección de puntos clave debe estar guiada por las preguntas analíticas que se desean responder. Por ejemplo, para analizar la propulsión, son necesarios puntos a lo largo del eje del cuerpo y en la cola. Para monitorear la respiración, un punto en el opérculo es esencial.

Tabla 2: Definición Propuesta del Esqueleto de Puntos Clave para Salmones

ID del Punto Clave	Hito Anatómico	Justificación para su Inclusión	Guía de Anotación
0	**Punta del Hocico**	Punto de inicio para mediciones de longitud total. Indica la	Anotar el punto más anterior de la mandíbula superior.

		dirección de la cabeza y la orientación general.	
1	**Ojo**	Característica clave para determinar la orientación de la cabeza. Su claridad puede ser un indicador de salud.	Anotar el centro de la pupila visible.
2	**Opérculo (Cubierta Branquial)**	El movimiento de esta estructura está directamente relacionado con la respiración. Su seguimiento puede estimar la frecuencia respiratoria.	Anotar el borde posterior y central de la placa opercular.
3	**Origen Aleta Dorsal**	Marca el inicio de la sección media del cuerpo. Un punto clave en la "columna vertebral" flexible del pez.	Anotar el punto donde el borde anterior de la aleta dorsal se une al cuerpo.
4	**Pedúnculo Caudal**	La parte más estrecha del cuerpo antes de la cola. Punto de pivote crucial para el batido de la cola y la propulsión.	Anotar el punto más estrecho del cuerpo justo antes de que comience la aleta caudal.
5	**Horquilla Aleta Caudal**	Punto final para mediciones de longitud. El principal elemento propulsor.	Anotar la muesca central en el borde posterior de la aleta caudal.

Este esqueleto de 6 puntos proporciona un equilibrio entre la simplicidad de la anotación y la riqueza de la información cinemática que se puede extraer. Formalizar esta definición en una tabla asegura la consistencia en la anotación, especialmente si varias personas contribuyen al conjunto de datos.

3.2. Manejo de Datos de Puntos Clave de CVAT

Configuración en CVAT: Para anotar los datos de pose, se debe configurar una nueva etiqueta en CVAT. En la configuración de la etiqueta, se deben definir los puntos clave de la Tabla 2, especificando su nombre y las conexiones entre ellos (e.g., una "extremidad" que conecta el "Hocico" con el "Ojo", y otra que conecta el "Origen Aleta Dorsal" con el "Pedúnculo Caudal") para ayudar a la visualización durante la anotación.
Formato de Exportación: El formato de exportación ideal para datos de pose es **COCO Keypoints (JSON)**. Este es un estándar de la industria que soporta nativamente la información de puntos clave, incluyendo su visibilidad.
Conversión al Formato YOLOv8-Pose: El formato de salida de COCO JSON debe ser convertido al formato de texto requerido por YOLOv8-Pose. Cada línea en el archivo .txt de una imagen debe seguir la estructura: <class-index> <box_x_center> <box_y_center> <box_width> <box_height> <kpt1_x> <kpt1_y> <kpt2_x> <kpt2_y>... <kptn_x> <kptn_y>. Todas las coordenadas (caja y puntos clave) deben estar normalizadas.
Un aspecto fundamental del formato COCO es la bandera de visibilidad para cada punto clave: v=0 (no etiquetado), v=1 (etiquetado pero ocluido), v=2 (etiquetado y visible). Esta bandera es una señal de entrenamiento crítica. Al anotar un punto clave que está temporalmente oculto por otro pez o por el propio cuerpo del animal y marcarlo como v=1, se enseña al modelo a inferir la posición probable de las partes del cuerpo ocultas. Esta práctica hace que el modelo final sea drásticamente más robusto frente a las oclusiones, un problema omnipresente en entornos de alta densidad de peces. El siguiente script de conversión maneja esta información, aunque YOLOv8-Pose actualmente utiliza implícitamente la presencia o ausencia de coordenadas (coordenadas 0,0 si no están presentes) en lugar de una bandera de visibilidad explícita en el archivo de texto final.

Python

```
# prepare_pose_data.py
import json
from pathlib import Path
from tqdm import tqdm

def convert_coco_kpts_to_yolo_pose(coco_json_path, save_dir,
use_segmentation_for_bbox=False):
    """
    Convierte anotaciones de pose en formato COCO JSON al formato YOLOv8-Pose.
    """
    save_path = Path(save_dir)
    labels_path = save_path / 'labels'
    labels_path.mkdir(parents=True, exist_ok=True)

    with open(coco_json_path, 'r') as f:
        data = json.load(data)
```

```
    images_info = {img['id']: {'file_name': img['file_name'], 'height': img['height'], 'width':
img['width']} for img in data['images']}
    categories = {cat['id']: i for i, cat in enumerate(data['categories'])}
    num_kpts = len(data['categories']['keypoints'])

    annotations_by_image = {}
    for ann in data['annotations']:
        img_id = ann['image_id']
        if img_id not in annotations_by_image:
            annotations_by_image[img_id] =
        annotations_by_image[img_id].append(ann)

    print("Iniciando conversión de anotaciones de pose...")
    for img_id, anns in tqdm(annotations_by_image.items()):
        img_info = images_info[img_id]
        img_h, img_w = img_info['height'], img_info['width']

        if img_h == 0 or img_w == 0:
            continue

        label_name = Path(img_info['file_name']).stem + '.txt'
        label_path = labels_path / label_name

        with open(label_path, 'w') as f:
            for ann in anns:
                if 'keypoints' not in ann or ann['num_keypoints'] == 0:
                    continue

                class_index = categories[ann['category_id']]

                # Bounding box
                x, y, w, h = ann['bbox']
                xc = (x + w / 2) / img_w
                yc = (y + h / 2) / img_h
                wn = w / img_w
                hn = h / img_h

                line_parts = [str(class_index), str(xc), str(yc), str(wn), str(hn)]

                # Keypoints
                kpts = ann['keypoints']
                for i in range(num_kpts):
```

```
            kpt_x = kpts[i * 3]
            kpt_y = kpts[i * 3 + 1]
            vis = kpts[i * 3 + 2]

            # YOLO format no usa la bandera de visibilidad directamente,
            # pero los puntos no visibles (v=0) se ponen en (0,0)
            if vis == 0:
                nx, ny = 0, 0
            else:
                nx = kpt_x / img_w
                ny = kpt_y / img_h

            line_parts.extend([str(nx), str(ny)])

        f.write(" ".join(line_parts) + '\n')

    print(f"Conversión completada. Las etiquetas se guardaron en: {labels_path}")

# Ejemplo de uso:
# La estructura de carpetas y el archivo data.yaml son análogos al caso de segmentación.
# convert_coco_kpts_to_yolo_pose('/path/to/dataset/annotations/person_keypoints_train.json',
'/path/to/yolo_pose_dataset/train/')
```

3.3. El Código Base de Estimación de Pose: Entrenamiento e Inferencia

El proceso de entrenamiento e inferencia para la pose es muy similar al de la segmentación, utilizando las variantes de modelo -pose.
Script de Entrenamiento (train_pose.py):
El entrenamiento de un modelo de pose implica la optimización de una función de pérdida compuesta que incluye la pérdida de la caja delimitadora, la pérdida de clasificación de clase, y una pérdida específica para los puntos clave. Esta última a menudo combina una pérdida de "objetividad" (¿está el punto clave presente?) y una pérdida de localización (¿qué tan cerca está la predicción de la verdad fundamental?).

Python

```
# train_pose.py
from ultralytics import YOLO

def main():
```

```
    # Cargar un modelo de estimación de pose pre-entrenado YOLOv8n-pose.pt
    model = YOLO('yolov8n-pose.pt')

    # Entrenar el modelo
    print("Iniciando el entrenamiento del modelo de estimación de pose...")
    results = model.train(
        data='path/to/salmon_pose_dataset/data.yaml',
        epochs=150,
        imgsz=640,
        batch=16,
        patience=30,
        name='yolov8n_salmon_pose'
    )
    print("Entrenamiento completado.")
    print(f"El mejor modelo se guardó en: {results.save_dir}")

if __name__ == '__main__':
    main()
```

Script de Inferencia (predict_pose.py):
El objeto results de una predicción de pose contiene un atributo keypoints que almacena las coordenadas predichas.

Python

```
# predict_pose.py
import cv2
from ultralytics import YOLO

# Definir el esqueleto para la visualización (conexiones entre puntos clave)
# Basado en la Tabla 2
SKELETON = ,  # Hocico -> Ojo
    [1, 2],  # Ojo -> Opérculo
    [2, 3],  # Opérculo -> Origen Aleta Dorsal
    [3, 4],  # Origen Aleta Dorsal -> Pedúnculo Caudal
    [4, 5]   # Pedúnculo Caudal -> Horquilla Aleta Caudal
]
# Colores para los puntos y las conexiones
KEYPOINT_COLORS = [(0, 0, 255), (0, 128, 255), (0, 255, 255), (0, 255, 0), (255, 128, 0), (255, 0, 0)]
LIMB_COLORS = [(0, 255, 0), (0, 255, 0), (0, 255, 0), (255, 0, 0), (255, 0, 0)]
```

```
def visualize_pose(image_path, model):
    """Realiza la inferencia de pose y visualiza los esqueletos."""
    results = model.predict(source=image_path)
    img = cv2.imread(image_path)

    if results.keypoints is None:
        print("No se detectaron poses en la imagen.")
        return img

    # Obtener los datos de los puntos clave (tensor)
    keypoints_data = results.keypoints.xy.cpu().numpy().astype(int)

    for kpts in keypoints_data:
        # Dibujar los puntos clave
        for i, (x, y) in enumerate(kpts):
            if x > 0 and y > 0: # Solo dibujar si el punto fue detectado
                cv2.circle(img, (x, y), 5, KEYPOINT_COLORS[i], -1)

        # Dibujar las conexiones (esqueleto)
        for i, (p1_idx, p2_idx) in enumerate(SKELETON):
            p1 = kpts[p1_idx]
            p2 = kpts[p2_idx]
            if p1 > 0 and p1[1] > 0 and p2 > 0 and p2[1] > 0:
                cv2.line(img, tuple(p1), tuple(p2), LIMB_COLORS[i], 2)

    return img

def main():
    # Cargar el modelo de pose personalizado
    model_path = 'runs/pose/yolov8n_salmon_pose/weights/best.pt'
    try:
        model = YOLO(model_path)
    except Exception as e:
        print(f"Error al cargar el modelo: {e}")
        return

    input_source = 'path/to/new_salmon_image.jpg'
    result_image = visualize_pose(input_source, model)

    output_path = 'pose_result.jpg'
    cv2.imwrite(output_path, result_image)
    print(f"Resultado guardado en {output_path}")
```

```
if __name__ == '__main__':
    main()
```

3.4. De Puntos Clave a Cinemática: Una Introducción a las Métricas de Comportamiento

La verdadera utilidad de la estimación de pose radica en la conversión de las coordenadas de los puntos clave en métricas de comportamiento significativas.
Curvatura Corporal: La flexión del cuerpo se puede cuantificar calculando el ángulo entre segmentos corporales. Por ejemplo, el ángulo formado por los puntos "Origen Aleta Dorsal", "Pedúnculo Caudal" y "Horquilla Aleta Caudal" indica la flexión de la cola.

Python

```
import numpy as np

def calculate_body_curvature(kpts):
    """Calcula el ángulo en el pedúnculo caudal como una medida de curvatura."""
    p_dorsal = kpts[3]
    p_peduncle = kpts[4]
    p_caudal = kpts[5]

    # Crear vectores
    vec1 = p_dorsal - p_peduncle
    vec2 = p_caudal - p_peduncle

    # Calcular el ángulo usando el producto punto
    cosine_angle = np.dot(vec1, vec2) / (np.linalg.norm(vec1) * np.linalg.norm(vec2))
    angle = np.arccos(cosine_angle)

    return np.degrees(angle)
```

Análisis del Batido de Cola: Al procesar un video, se puede rastrear la posición del punto clave del "Pedúnculo Caudal" (ID 4) a lo largo del tiempo. El análisis de la componente lateral de su desplazamiento (perpendicular al eje principal del pez) permite aplicar una Transformada de Fourier para encontrar la frecuencia dominante, que corresponde a la frecuencia del batido de cola (en Hz).
Métricas de Evaluación: La métrica estándar para evaluar la precisión de la estimación de

pose es la Similitud de Puntos Clave de Objeto (Object Keypoint Similarity, OKS). Se calcula como $OKS = \sum_{i} \frac{exp(-d_i^2 / 2s^2k_i^2)}{\delta(v_i > 0)} / \sum_{i} \delta(v_i > 0)$, donde d_i es la distancia euclidiana entre el punto clave predicho y el real, s es la escala del objeto (área de la caja), y k_i es una constante por punto clave que controla la caída de la penalización. OKS es superior a la simple distancia euclidiana porque normaliza el error por el tamaño del objeto, siendo más justo con objetos pequeños.

Parte IV: Creando una Tubería de Análisis Integrada

La combinación de la detección, el seguimiento, la segmentación y la estimación de pose en una única tubería de análisis transforma una colección de herramientas dispares en un sistema de monitoreo sinérgico y potente.

4.1. El Poder del Seguimiento: De Datos Anónimos a Historias Individuales

El código de seguimiento de objetos existente, que asigna un ID único a cada salmón y lo mantiene a través de los fotogramas, actúa como el "pegamento" que une todo el sistema. Sin seguimiento, cada fotograma es una instantánea anónima. Con el seguimiento, es posible construir un perfil de datos temporal para cada individuo, registrando su morfología y comportamiento a lo largo del tiempo. Esto permite pasar de un análisis de población a un monitoreo individualizado, que es fundamental para la gestión de precisión en la acuicultura.

4.2. Un Flujo de Trabajo Multimodal

Un flujo de trabajo eficiente para integrar estas capacidades se puede estructurar de la siguiente manera, optimizando el rendimiento computacional:

1. **Ingesta de Fotograma:** Leer un nuevo fotograma del flujo de video.
2. **Detección y Seguimiento:** Ejecutar un detector de objetos rápido (como el YOLOv8 base) combinado con un algoritmo de seguimiento (e.g., BoT-SORT, ByteTrack) sobre el fotograma completo. Esto produce una lista de cajas delimitadoras y sus correspondientes IDs de seguimiento (track_id).
3. **Recorte de Instancias:** Para cada pez rastreado, recortar la región de la caja delimitadora del fotograma original. Esto crea una serie de imágenes más pequeñas, cada una conteniendo un solo salmón.
4. **Inferencia Paralela:** Pasar estos recortes (en un lote) a los modelos especializados de segmentación y estimación de pose. Ejecutar estos modelos más pesados en imágenes pequeñas y recortadas es mucho más eficiente que ejecutarlos en el fotograma completo de alta resolución.

5. **Agregación de Datos:** Para cada fotograma y cada track_id, almacenar los resultados en una estructura de datos organizada. Esto podría ser un diccionario, un objeto JSON, o una fila en una base de datos. La estructura para cada instancia podría verse así:
   JSON
   ```
   {
    "frame_number": 12345,
    "timestamp": "2023-10-27T10:30:01.500Z",
    "track_id": 17,
    "bounding_box": [x1, y1, x2, y2],
    "segmentation_mask": { "rle": "..." }, // Máscara codificada
    "pixel_area": 5432,
    "keypoints": {
     "snout": [x, y],
     "eye": [x, y],
     ...
    },
    "body_curvature_deg": 155.7,
    "tail_beat_freq_hz": 2.1
   }
   ```

4.3. Análisis Longitudinal: Desbloqueando Nuevos Conocimientos

Esta estructura de datos agregada y longitudinal permite realizar consultas analíticas complejas que antes eran imposibles:

- **Análisis de Crecimiento:** "Trazar el pixel_area para el track_id #17 a lo largo de la última semana. ¿Ha mostrado un crecimiento constante?"
- **Análisis de Comportamiento y Alimentación:** "Correlacionar la tail_beat_freq_hz promedio de todos los peces con la hora del día. ¿Se observa un pico de actividad durante las horas de alimentación?"
- **Detección de Anomalías de Salud:** "Marcar cualquier pez cuya body_curvature_deg promedio se mantenga por debajo de un umbral (indicando rigidez) durante más de 12 horas."
- **Identificación de Individuos en Riesgo:** "Generar una alerta si un pez muestra simultáneamente una disminución en el pixel_area y una reducción en su actividad motora (baja frecuencia de batido de cola)."

Este enfoque transforma el sistema de una simple herramienta de medición a una plataforma de gestión proactiva. En lugar de reaccionar a problemas visibles, como un pez que ya muestra signos evidentes de enfermedad, el sistema puede identificar desviaciones sutiles de la línea de base de comportamiento y morfología de un individuo. Este es el fundamento de un sistema de alerta temprana y monitoreo de salud predictivo. Al establecer lo que es "normal" para cada pez, el sistema puede señalar automáticamente a los individuos que se

desvían de su propia norma, permitiendo una intervención temprana y dirigida, lo que en última instancia mejora el bienestar animal y el rendimiento económico de la operación.

Parte V: Optimización del Rendimiento y Consideraciones de Despliegue

La transición de un prototipo en un entorno de desarrollo a un sistema robusto desplegado para monitoreo continuo requiere una cuidadosa consideración del rendimiento, la eficiencia y los desafíos del mundo real.

5.1. Optimización del Modelo para Inferencia en Tiempo Real

- **Tamaño del Modelo:** YOLOv8 se ofrece en varias escalas (nano n, pequeño s, mediano m, grande l, extra grande x). Existe un compromiso directo entre el tamaño del modelo, su velocidad y su precisión. Un modelo yolov8n-pose será mucho más rápido que un yolov8l-pose, pero menos preciso. La elección óptima depende del hardware disponible (CPU/GPU) y de los requisitos de latencia de la aplicación. Es aconsejable comenzar con un modelo más pequeño y escalar si la precisión resulta insuficiente.
- **Exportación para Producción:** Los modelos entrenados se guardan en formato PyTorch (.pt). Para un despliegue de producción, es altamente recomendable exportar el modelo a un formato de inferencia optimizado como ONNX (Open Neural Network Exchange) o, para hardware NVIDIA, TensorRT. Estos formatos pueden proporcionar aceleraciones de 2x a 5x al realizar optimizaciones como la fusión de capas y la selección de kernels específicos para el hardware. La propia biblioteca ultralytics proporciona un método model.export(format='onnx') para facilitar este proceso.
- **Cuantización:** Para despliegues en dispositivos de borde con recursos muy limitados, se puede aplicar la cuantización post-entrenamiento. Esta técnica reduce la precisión de los pesos del modelo (e.g., de punto flotante de 32 bits a enteros de 8 bits), lo que disminuye el tamaño del modelo y acelera la inferencia, a menudo con una pérdida mínima de precisión.

5.2. Manejando Desafíos del Mundo Real

La visión por computadora subacuática presenta desafíos únicos que deben ser abordados tanto en la preparación de datos como en el despliegue.

- **Iluminación Variable y Turbidez:** Las condiciones de luz en tanques o jaulas marinas pueden cambiar drásticamente con la hora del día, el clima y la turbidez del agua. Para mitigar esto, se deben emplear estrategias robustas de aumento de datos durante el

entrenamiento, como variaciones aleatorias de brillo, contraste y saturación. En el despliegue, se puede implementar un paso de pre-procesamiento en tiempo real, como la ecualización de histograma adaptativa (CLAHE), para normalizar los fotogramas antes de pasarlos a los modelos.
- **Oclusión:** La alta densidad de población en la acuicultura significa que las oclusiones (peces que se tapan unos a otros) son la norma, no la excepción. Como se discutió anteriormente, la estrategia más efectiva para manejar esto es durante la anotación de datos, marcando consistentemente los puntos clave ocluidos. Además, el uso de aumentos de datos como el mosaico, que combina cuatro imágenes de entrenamiento en una, obliga al modelo a aprender a detectar objetos parcialmente visibles.

5.3. Arquitectura del Sistema para el Despliegue

Una arquitectura de sistema para un monitoreo continuo podría estructurarse de la siguiente manera:
- **Servicio de Ingesta:** Un proceso responsable de capturar los flujos de video de las cámaras IP u otras fuentes. Puede realizar una decodificación de video inicial y colocar los fotogramas en una cola de procesamiento.
- **Servicio de Inferencia:** Uno o más procesos de trabajo (workers), idealmente equipados con GPUs, que consumen fotogramas de la cola. Cada worker ejecuta la tubería de análisis integrada descrita en la Parte IV (seguimiento, recorte, inferencia paralela).
- **Almacén de Datos:** Una base de datos diseñada para manejar los datos generados. Para datos de series temporales como las métricas de comportamiento, una base de datos como InfluxDB es una excelente opción. Para datos más estructurados, una base de datos SQL como PostgreSQL puede ser adecuada.
- **Panel de Control y Alertas (Dashboard):** Una interfaz de usuario para visualizar los datos y las tendencias. Herramientas como Grafana son excelentes para crear paneles de control a partir de datos de series temporales. Este componente también sería responsable de ejecutar consultas periódicas para detectar anomalías y enviar alertas (e.g., por correo electrónico o SMS) al personal de la granja.

Conclusión

Este informe ha proporcionado un camino completo y detallado para evolucionar desde una simple detección de salmones mediante cajas delimitadoras hacia un sistema de análisis multimodal y sofisticado. Se han presentado dos códigos base funcionales, uno para la segmentación de instancia y otro para la estimación de pose, ambos diseñados para consumir datos anotados en la plataforma CVAT.
La segmentación de instancia, al delinear el contorno preciso de cada pez, ofrece una vía

para la estimación robusta de la biomasa, superando las limitaciones inherentes de las cajas delimitadoras. La estimación de pose, mediante la definición de un esqueleto biomecánicamente relevante y la localización de sus puntos clave, desbloquea la capacidad de cuantificar el comportamiento, la cinemática natatoria y los posibles indicadores de estrés o enfermedad.

La verdadera potencia de estas herramientas se materializa cuando se integran en una única tubería de análisis, unificada por un sistema de seguimiento de objetos. Este enfoque permite la creación de perfiles longitudinales para individuos, transformando los datos brutos de video en conocimiento accionable sobre el crecimiento, la salud y el bienestar de cada animal. Al adoptar estas metodologías avanzadas, las operaciones de acuicultura pueden pasar de un paradigma de gestión reactiva a uno proactivo y predictivo, sentando las bases para una acuicultura de precisión más sostenible y eficiente.