**Report:**

**The Selected Model & Justification:**

Choosing the appropriate model architecture for a specific task depends on various factors including the nature of the data, computational resources, and the requirements of the application. Here's a justification for using YOLO version 8 and the Segment Anything model for the task of segmenting images:

**YOLO version 8:**

1. **Real-time Object Detection:** YOLO (You Only Look Once) is known for its real-time object detection capabilities. It's designed to detect objects in images quickly and accurately, making it suitable for applications where real-time processing is crucial.
2. **Single-stage Detection:** YOLO operates in a single stage, directly predicting bounding boxes and class probabilities from the full image in one evaluation. This makes it faster compared to two-stage detectors like Faster R-CNN, which require separate region proposal and classification stages.
3. **Efficiency and Speed:** YOLO achieves a good balance between accuracy and speed. YOLOv8 may introduce improvements in terms of speed and accuracy over previous versions, making it a compelling choice for applications where efficiency is important.
4. **Minimal Post-Processing:** YOLO outputs bounding boxes directly, requiring minimal post-processing compared to other detection methods. This simplicity can be advantageous in certain applications.
5. **Wide Range of Applications:** YOLO has been widely used in various applications such as surveillance, autonomous vehicles, and robotics, where real-time object detection is critical.

**Segment Anything Model:**

1. **Versatility:** The "Segment Anything" model, as the name suggests, is versatile in its ability to segment various types of objects or regions in images. It's not limited to a specific set of classes like some traditional segmentation models.
2. **Semantic Segmentation:** The Segment Anything model likely employs semantic segmentation techniques, which assign semantic labels to each pixel in an image. This can be useful for tasks such as image understanding, scene parsing, and pixel-level classification.
3. **Instance Segmentation:** Depending on the specific implementation, the Segment Anything model may also perform instance segmentation, distinguishing between individual object instances within the same class. This provides richer information compared to semantic segmentation alone.
4. **Fine-grained Localization:** Segment Anything models can provide fine-grained localization of objects or regions within an image, which can be valuable for applications requiring precise spatial information.
5. **Adaptability:** Segment Anything models can be adapted to various domains and applications with minimal modifications. They can be trained on custom datasets to segment specific objects or regions relevant to the task at hand.

**Details on Model Retraining:**

The model was retrained using the dataset provided, following a structured workflow. Initially, the dataset was annotated using the CVAT platform, allowing for precise labeling of objects within the images. Subsequently, a Python script named "masks\_to\_polygons" was utilized to generate polygonal representations from these annotations, facilitating further processing.

Afterwards, the dataset was partitioned into training and testing subsets, adhering to an 80:20 ratio for training and testing respectively. This division ensures that the model can learn from a significant portion of the data while also allowing for robust evaluation on unseen samples.

With the dataset prepared, model training commenced with a specified number of epochs, set to 10 in this instance. During training, the model iteratively learns to recognize patterns and features within the data, refining its parameters to improve performance over successive epochs.

By following this methodology, the model is primed to accurately identify and segment objects within images, laying the groundwork for effective application in various computer vision tasks.

**Precision:**

def calculate\_precision(predicted\_labels, true\_labels):

true\_positives = sum((p == 1) and (t == 1) for p, t in zip(predicted\_labels, true\_labels))

false\_positives = sum((p == 1) and (t == 0) for p, t in zip(predicted\_labels, true\_labels))

if true\_positives + false\_positives == 0:

return 0 # Handle division by zero case

precision = true\_positives / (true\_positives + false\_positives)

return precision

# Example usage

predicted\_labels = [1, 0, 1, 1, 0]

true\_labels = [1, 1, 0, 1, 0]

precision = calculate\_precision(predicted\_labels, true\_labels)

print("Precision:", precision)

**Recall:**

def calculate\_recall(predicted\_labels, true\_labels):

true\_positives = sum((p == 1) and (t == 1) for p, t in zip(predicted\_labels, true\_labels))

false\_negatives = sum((p == 0) and (t == 1) for p, t in zip(predicted\_labels, true\_labels))

if true\_positives + false\_negatives == 0:

return 0 # Handle division by zero case

recall = true\_positives / (true\_positives + false\_negatives)

return recall

# Example usage

predicted\_labels = [1, 0, 1, 1, 0]

true\_labels = [1, 1, 0, 1, 0]

recall = calculate\_recall(predicted\_labels, true\_labels)

print("Recall:", recall)

**F1 Score:**

def calculate\_f1\_score(predicted\_labels, true\_labels):

precision = calculate\_precision(predicted\_labels, true\_labels)

recall = calculate\_recall(predicted\_labels, true\_labels)

if precision + recall == 0:

return 0 # Handle division by zero case

f1\_score = 2 \* (precision \* recall) / (precision + recall)

return f1\_score

# Example usage

predicted\_labels = [1, 0, 1, 1, 0]

true\_labels = [1, 1, 0, 1, 0]

f1\_score = calculate\_f1\_score(predicted\_labels, true\_labels)

print("F1 Score:", f1\_score)

**Tools & Libraries:**

* Ultralytics
* YOLO
* Segment Anything
* Numpy
* Opencv
* Matplotlib