NAME – Mansi Dobriyal
Phone no - 9311453903
Email id – mansidobriyal50@gmail.com

Assignments: Image Segmentation & Maskrcnn Practical

1. What is image segmentation, and why is it important?

Ans – Image segmentation is a computer vision technique that partitions an image into meaningful regions, such as objects or areas of interest, to simplify analysis. It is crucial for tasks like object recognition, medical imaging (e.g., detecting tumors), autonomous driving (e.g., identifying roads and obstacles), and satellite imagery (e.g., analyzing land use). By isolating relevant parts of an image, segmentation improves accuracy and efficiency in various applications.

2. Explain the difference between image classification, object detection, and image segmentation

Ans – Here's the difference between **image classification**, **object detection**, and **image segmentation**:

1. **Image Classification**:
   - **What it Does**: Predicts the class or label of the entire image.
   - **Output**: Single label for the whole image (e.g., "cat" or "dog").
   - **Example**: A picture of a dog is classified as "dog" without identifying its location in the image.
2. **Object Detection**:
   - **What it Does**: Identifies objects in an image and their locations.
   - **Output**: Bounding boxes around objects along with their labels (e.g., "dog" at a specific position).
   - **Example**: Detects both a dog and a ball in an image, drawing rectangles around them.
3. **Image Segmentation**:
   - **What it Does**: Divides the image into pixel-level segments, assigning each pixel a label.
   - **Output**: Precise outlines of objects or regions in the image.
   - **Example**: Each pixel belonging to the dog is labeled as "dog," providing a detailed shape instead of just a bounding box.

Summary Table:

Feature	**Image Classification**	**Object Detection**	**Image Segmentation**
Focus	Entire image	Object labels + location	Pixel-level labels
Output	One label	Bounding boxes + labels	Mask outlining objects

Feature	Image Classification	Object Detection	Image Segmentation
Detail Level	Low	Medium	High

3. What is Mask R-CNN, and how is it different from traditional object detection models

Ans – Mask R-CNN is a deep learning model that combines object detection and instance segmentation. It extends Faster R-CNN by adding a mask prediction branch to generate pixel-level object masks along with bounding boxes and labels. Traditional object detection models like YOLO or Faster R-CNN focus only on detecting and localizing objects via bounding boxes, while Mask R-CNN provides a more detailed output by outlining objects' shapes. This makes it highly suitable for tasks like medical imaging, autonomous driving, and any application requiring precise object contours.

4. What role does the "RoIAlign" layer play in Mask R-CNN

The **RoIAlign (Region of Interest Align)** layer in Mask R-CNN ensures accurate feature alignment when extracting features for each proposed region of interest (RoI).

Role of RoIAlign:

1. **Improved Precision**:
   - Traditional methods like RoIPool quantize RoI coordinates, causing misalignment. RoIAlign eliminates this by avoiding rounding operations.
   - It uses bilinear interpolation to precisely sample feature maps, ensuring finer spatial accuracy.
2. **Key to Instance Segmentation**:
   - Accurate alignment is crucial for pixel-level mask prediction, where even small misalignments can degrade mask quality.

Why it Matters:

RoIAlign is a major improvement over RoIPool, making Mask R-CNN more effective for high-resolution tasks such as instance segmentation and dense object detection.

5. What are semantic, instance, and panoptic segmentation

Ans – **Semantic, Instance, and Panoptic Segmentation**

1. **Semantic Segmentation**:
   - Assigns a class label to every pixel in the image.
   - Focuses on **categories**, not individual objects.
   - Example: All cars in an image are labeled as "car," without distinguishing between different cars.
2. **Instance Segmentation**:
   - Similar to semantic segmentation but differentiates between **individual instances** of objects.
   - Example: Each car in the image gets a unique label, such as "car1," "car2," etc.
3. **Panoptic Segmentation**:

- Combines semantic and instance segmentation.
- Assigns a label to every pixel, distinguishing both **stuff** (background regions like sky, road) and **things** (individual objects like cars, people).
- Example: Sky pixels are labeled "sky," while each car is uniquely identified.

Comparison:

Feature	Semantic Segmentation	Instance Segmentation	Panoptic Segmentation
Focus	Categories	Object Instances	Categories + Instances
Pixel Labeling	Shared across objects	Unique for each object	Combines both approaches
Application	Scene understanding	Object-level analysis	Comprehensive image parsing

6. Describe the role of bounding boxes and masks in image segmentation models

Ans – In image segmentation, **bounding boxes** are rectangular areas that roughly define an object's location in an image, often used in object detection and instance segmentation. **Masks** are binary images where each pixel is labeled, providing precise pixel-level segmentation of objects. Bounding boxes help identify regions of interest, while masks offer detailed, pixel-level object boundaries.

7. What is the purpose of data annotation in image segmentation

Ans – Data annotation in image segmentation involves labeling images to train models. The purpose is to provide ground truth for models, helping them learn to recognize and segment objects accurately. Annotations, such as masks or bounding boxes, allow models to differentiate between various regions or objects in an image, enabling precise segmentation for tasks like object detection, instance segmentation, and semantic segmentation.

8. How does Detectron2 simplify model training for object detection and segmentation tasks

Detectron2 simplifies model training for object detection and segmentation by providing a modular, flexible, and efficient framework. It offers pre-built implementations of state-of-the-art models, like Faster R-CNN and Mask R-CNN, along with tools for easy dataset handling, model configuration, and evaluation. Its design supports rapid prototyping and fine-tuning, reducing the complexity of building object detection and segmentation models from scratch. Detectron2 also supports multi-GPU training, which speeds up the training process.

9. Why is transfer learning valuable in training segmentation models

Transfer learning is valuable in training segmentation models because it allows models to leverage pre-trained weights from a related task, reducing the need for large labeled datasets and long training times. By starting with a model trained on a similar problem, it can quickly adapt to new tasks, improving accuracy, especially when data is scarce. This approach accelerates convergence and enhances performance on segmentation tasks by utilizing features learned from general image recognition.

10. How does Mask R-CNN improve upon the Faster R-CNN model architecture

Mask R-CNN improves upon Faster R-CNN by adding a branch for pixel-level segmentation. While Faster R-CNN focuses on object detection with region proposals and classification, Mask R-CNN extends this by introducing an additional **mask branch** that generates binary masks for each detected object. This allows Mask R-CNN to perform **instance segmentation**, accurately segmenting objects at the pixel level, while Faster R-CNN only detects objects without detailed segmentation. The architecture includes a Region of Interest (RoI) alignment step to improve mask accuracy.

11. What is meant by "from bounding box to polygon masks" in image segmentation

"From bounding box to polygon masks" in image segmentation refers to the transition from using simple rectangular bounding boxes to more precise polygonal masks for object delineation. A bounding box is a rough rectangular area that surrounds an object, while a polygon mask represents the object's shape with multiple vertices, providing a much finer, pixel-level boundary. This allows for more accurate segmentation, especially for objects with complex or irregular shapes.

12. How does data augmentation benefit image segmentation model training

Data augmentation benefits image segmentation by increasing the diversity of the training dataset without the need for additional labeled data. Techniques like rotation, flipping, scaling, and cropping help the model generalize better by exposing it to various variations of the objects, improving its ability to segment under different conditions. This reduces overfitting and enhances model robustness, especially when working with limited data.

13. Describe the architecture of Mask R-CNN, focusing on the backbone, region proposal network (RPN), and segmentation mask head

Mask R-CNN's architecture consists of the following key components:

1. **Backbone**: The backbone is a convolutional neural network (CNN) like ResNet or FPN (Feature Pyramid Network) that extracts feature maps from the input image. It serves as the foundation for both object detection and segmentation tasks.
2. **Region Proposal Network (RPN)**: The RPN generates potential object proposals (regions of interest or RoIs) from the feature maps produced by the backbone. It slides over the feature map, producing objectness scores and bounding box coordinates for each region, which are then used for further processing.
3. **Segmentation Mask Head**: For each RoI from the RPN, the mask head is a small fully convolutional network (FCN) that generates a binary mask for each detected object. This mask is produced at the pixel level, representing the object's exact boundary.

Mask R-CNN combines these components to perform **instance segmentation**, identifying objects and generating precise pixel-wise masks along with bounding boxes.

14. Explain the process of registering a custom dataset in Detectron2 for model training

To register a custom dataset in Detectron2:

1. **Prepare Dataset**: Ensure annotations are in a supported format (e.g., COCO).
2. **Register Dataset**: Use DatasetCatalog.register() to load your dataset, and MetadataCatalog to define metadata (e.g., class names).
3. **Example Code**:

python
Copy code

```
from detectron2.data import DatasetCatalog, MetadataCatalog
DatasetCatalog.register("my_train_dataset", lambda: load_coco_json("train.json",
"images"))
MetadataCatalog.get("my_train_dataset").set(thing_classes=["class1", "class2"])
```

15. What challenges arise in scene understanding for image segmentation, and how can Mask R-CNN address them

Challenges in Scene Understanding for Image Segmentation:

1. **Complex Object Boundaries**: Irregular shapes and overlapping objects.
2. **Occlusion**: Partially visible objects.
3. **Class Ambiguity**: Confusion due to similar textures or appearances.
4. **Fine-Grained Details**: Missing small objects or intricate details.
5. **Scale and Lighting Variability**: Objects at different scales and under varying illumination.
6. **Background Complexity**: Cluttered or dynamic backgrounds.

How Mask R-CNN Addresses These Challenges:

1. **Instance-Level Segmentation**: Predicts masks for each detected object, handling overlaps and fine boundaries.
2. **Multi-Task Learning**: Combines detection, classification, and mask generation.
3. **Feature Pyramid Network (FPN)**: Handles objects at various scales.
4. **RoIAlign**: Ensures precise mask alignment.
5. **High-Resolution Masks**: Captures detailed contours.
6. **Generalization**: Pre-trained models adapt to diverse scenes and lighting.

Mask R-CNN effectively resolves segmentation challenges, enabling accurate and detailed scene understanding.

16. How is the "IoU (Intersection over Union)" metric used in evaluating segmentation models

The **Intersection over Union (IoU)** metric evaluates the accuracy of segmentation models by comparing the predicted segmentation output with the ground truth. It is calculated as:

IoU=Area of OverlapArea of UnionIoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}

Steps to Compute IoU:

1. **Intersection**: The overlapping area between the predicted segmentation mask and the ground truth mask.
2. **Union**: The total area covered by both the predicted and ground truth masks.

Use in Evaluation:

1. **Thresholding**: IoU is often used with a threshold (e.g., 0.5 or 0.75) to decide if a prediction is a correct match (IoU ≥ threshold).
2. **Mean IoU (mIoU)**: Averages the IoU across all classes, providing a single metric for overall model performance.
3. **Pixel-Level Accuracy**: Helps assess how well the model captures object boundaries and overlaps.

Advantages:

- Robust against imbalanced datasets since it directly measures area overlap.
- Focuses on both precision and recall aspects of segmentation.

IoU is a widely adopted metric in segmentation tasks because it provides a clear and interpretable measure of overlap quality.

17. Discuss the use of transfer learning in Mask R-CNN for improving segmentation on custom datasets

Transfer Learning in Mask R-CNN:

Transfer learning uses a pre-trained Mask R-CNN model (e.g., on COCO) and fine-tunes it for a custom dataset.

Benefits:

1. **Faster Training**: Pre-trained weights accelerate convergence.
2. **Less Data Needed**: Reduces dependence on large datasets.
3. **Better Accuracy**: Leverages general features for improved results.

Steps:

1. Load a pre-trained Mask R-CNN.
2. Annotate the custom dataset (e.g., COCO format).
3. Fine-tune higher layers; freeze backbone.
4. Adjust parameters (e.g., classes, anchors).
5. Train with a small learning rate.

Applications:

- Medical imaging, satellite analysis, autonomous vehicles.

Transfer learning enhances segmentation performance with minimal data and effort.

18. What is the purpose of evaluation curves, such as precision-recall curves, in segmentation model assessment

Purpose of Evaluation Curves in Segmentation Model Assessment:

Evaluation curves, like **Precision-Recall (PR) curves**, provide insights into the performance of segmentation models under varying thresholds for classification or mask prediction.

Key Benefits:

1. **Threshold Sensitivity**: Shows how precision and recall change with the decision threshold.
2. **Performance Trade-off**:
    - High precision often reduces recall and vice versa. The PR curve visualizes this balance.
3. **Class Imbalance Handling**:
    - Particularly useful for datasets with imbalanced classes, focusing on positive predictions.
4. **Model Comparison**:
    - Helps compare models; a higher area under the PR curve indicates better performance.

Typical Use in Segmentation:

- **Pixel-level** or **instance-level** evaluations to measure how well the model predicts true masks while avoiding false positives and negatives.

By analyzing PR curves, practitioners can fine-tune segmentation models to optimize performance for specific use cases.

19. How do Mask R-CNN models handle occlusions or overlapping objects in segmentation

Handling Occlusions and Overlapping Objects in Mask R-CNN:

Mask R-CNN effectively addresses occlusions and overlapping objects using the following mechanisms:

1. **Region Proposal Network (RPN)**:
    - Generates multiple object proposals, ensuring occluded or overlapping objects are likely included in different proposals.
2. **Instance Segmentation**:
    - Predicts separate masks for each object, isolating instances even if they overlap.
3. **Bounding Box Regression**:
    - Refines the bounding boxes for precise localization of each object.
4. **Non-Maximum Suppression (NMS)**:
    - Removes duplicate detections by keeping the box with the highest confidence score, reducing overlap issues.
5. **RoIAlign**:
    - Accurately maps features to object proposals, improving boundary segmentation for occluded areas.

Mask R-CNN's instance-level segmentation and feature refinement ensure robust handling of occlusions and overlaps in complex scenes.

20. Explain the impact of batch size and learning rate on Mask R-CNN model training

Impact of Batch Size and Learning Rate on Mask R-CNN Training:

1. **Batch Size**:
    - **Small Batch Size**:
        - Pros: Allows training on limited GPU memory.
        - Cons: Increases training noise and instability, potentially requiring more iterations to converge.
    - **Large Batch Size**:
        - Pros: Stabilizes gradients and speeds up convergence.

        - Cons: Requires more GPU memory and may lead to worse generalization if learning rate is not adjusted.
2. **Learning Rate**:
    - **High Learning Rate**:
        - Pros: Speeds up training.
        - Cons: Risks overshooting minima, leading to unstable training or poor convergence.
    - **Low Learning Rate**:
        - Pros: Provides stable updates and fine-grained weight adjustments.
        - Cons: Slows down training and may get stuck in local minima.

Interaction Between Batch Size and Learning Rate:

- Larger batch sizes typically require proportionally higher learning rates to maintain stable updates (e.g., Linear Scaling Rule).
- Small batch sizes often benefit from lower learning rates to avoid gradient noise.

Best Practices:

- Start with moderate batch size and learning rate.
- Use learning rate schedules (e.g., step decay, warm-up) to improve convergence.
- Experiment to find the optimal balance for the dataset and hardware.

Both batch size and learning rate significantly influence Mask R-CNN's training stability, speed, and final performance.

21. Describe the challenges of training segmentation models on custom datasets, particularly in the context of Detectron2

Challenges of Training Segmentation Models on Custom Datasets with Detectron2:

1. **Data Preparation**:
    - **Challenge**: Formatting custom datasets to Detectron2's required structure (e.g., COCO JSON format).
    - **Solution**: Use annotation tools like LabelImg or CVAT and convert outputs to the appropriate format.
2. **Class Imbalance**:
    - **Challenge**: Some classes may dominate, leading to poor performance on minority classes.
    - **Solution**: Use techniques like weighted loss functions or data augmentation for underrepresented classes.
3. **Limited Data**:
    - **Challenge**: Small datasets can lead to overfitting.
    - **Solution**: Leverage pre-trained weights for transfer learning and apply data augmentation.
4. **Hyperparameter Tuning**:
    - **Challenge**: Finding optimal learning rates, batch sizes, and anchors for specific datasets.
    - **Solution**: Use grid search or automated tools for tuning.
5. **Annotation Errors**:
    - **Challenge**: Inconsistent or incorrect annotations reduce model accuracy.
    - **Solution**: Verify annotations manually or with automated checks.
6. **Hardware Limitations**:
    - **Challenge**: Detectron2 requires significant GPU memory for training.
    - **Solution**: Reduce batch size, use mixed-precision training, or train on subsets of data iteratively.
7. **Evaluation Metrics**:
    - **Challenge**: Assessing performance with meaningful metrics like IoU or mAP for segmentation tasks.
    - **Solution**: Use Detectron2's in-built evaluators and customize as needed for specific use cases.

By addressing these challenges, Detectron2 can be effectively used to train robust segmentation models on custom datasets.

22. How does Mask R-CNN's segmentation head output differ from a traditional object detector's output

Difference Between Mask R-CNN's Segmentation Head and Traditional Object Detector Output:

1. **Mask R-CNN's Segmentation Head**:
   - Outputs **pixel-level masks** for each detected object, providing precise boundaries.
   - Each instance has an associated binary mask for segmentation.
   - It extends object detection by adding a **segmentation branch** to predict masks.
2. **Traditional Object Detector**:
   - Outputs **bounding boxes** around objects and their corresponding class labels.
   - Lacks the granularity to identify object shapes or detailed boundaries.
   - Focuses solely on localization and classification.

Key Difference:

Mask R-CNN enhances traditional object detection by providing instance-level segmentation, making it suitable for tasks requiring fine-grained object delineation.

Practical

1. Perform basic color-based segmentation to separate the blue color in an image

   Color-based segmentation involves isolating a specific color from an image using its pixel values in a color space like RGB or HSV. For separating the blue color, the **HSV (Hue, Saturation, Value)** color space is often preferred because it makes it easier to specify a color range.

   Here's how you can perform basic blue color segmentation:

   Step-by-Step Implementation

   1. Import Required Libraries:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

2. Read the Image:

Load the image in which you want to segment the blue color.

```
image = cv2.imread('D:\mice project\Mouse_Lab-1120x630.jpg')
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

Define Blue Range and Create Mask:

```
lower_blue = np.array([100, 150, 50])
upper_blue = np.array([140, 255, 255])
blue_mask = cv2.inRange(hsv_image, lower_blue, upper_blue)
```

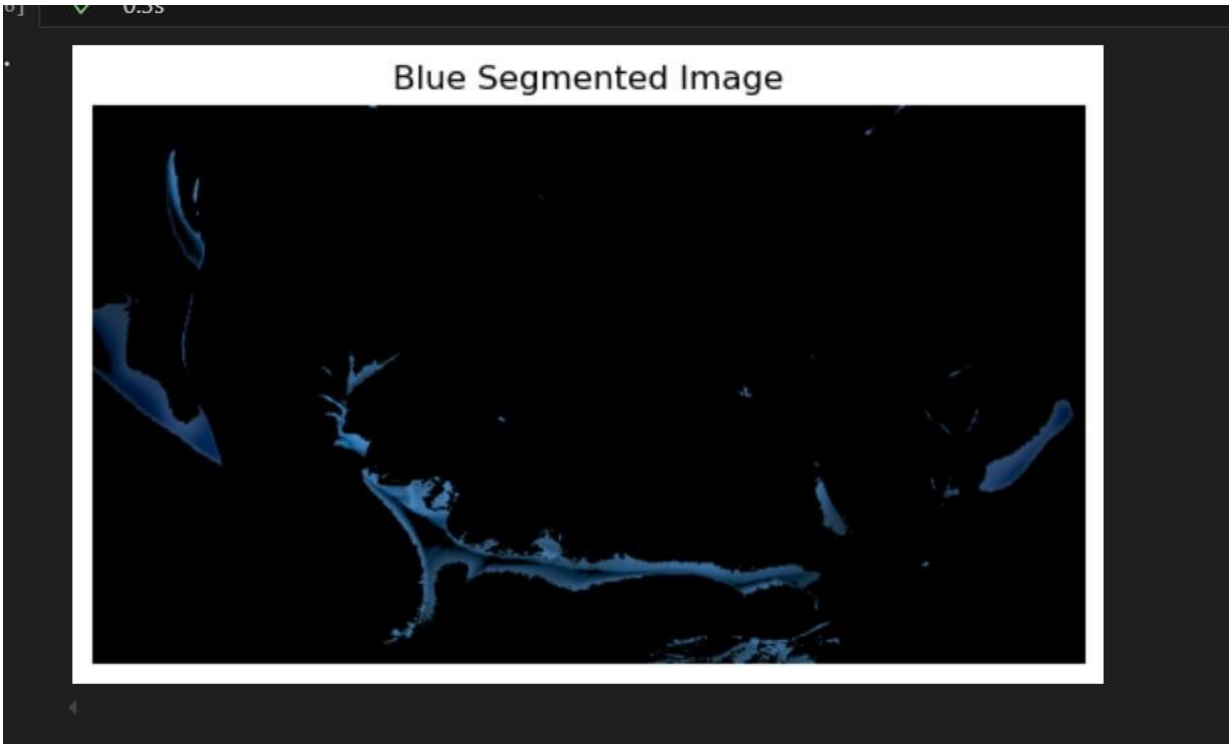
Extract Blue Region:

```
blue_segment = cv2.bitwise_and(image, image, mask=blue_mask)
```

Display Results:

```
plt.imshow(cv2.cvtColor(blue_segment, cv2.COLOR_BGR2RGB))
plt.title('Blue Segmented Image')
```

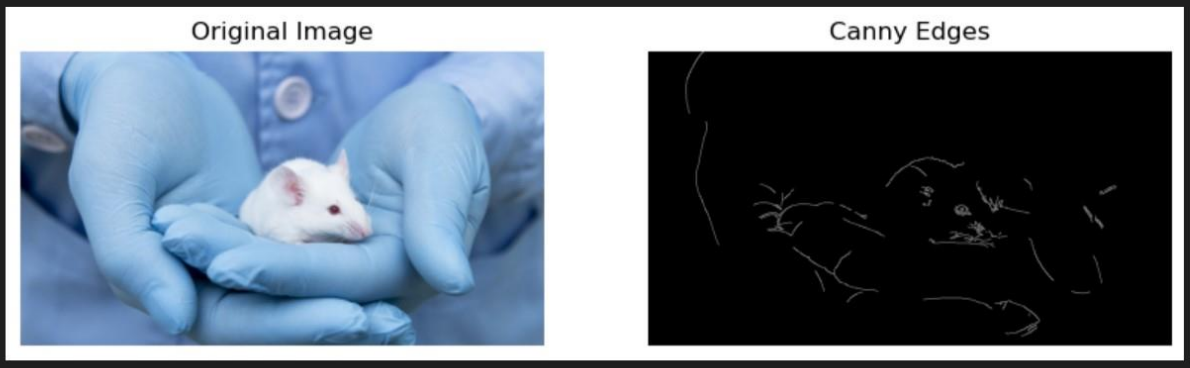
```
plt.axis('off')
plt.show()
```



2. Use edge detection with Canny to highlight object edges in an image loaded
   Code Example

```
import cv2
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('D:\mice project\Mouse_Lab-1120x630.jpg', cv2.IMREAD_COLOR)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

# Apply Canny Edge Detection
edges = cv2.Canny(gray_image, threshold1=100, threshold2=200)

# Display the results
plt.figure(figsize=(10, 5))

# Original Image
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis("off")

# Edge Detection Output
plt.subplot(1, 2, 2)
plt.imshow(edges, cmap='gray')
plt.title("Canny Edges")
plt.axis("off")

plt.show()
```

```
<>:5: SyntaxWarning: invalid escape sequence '\m'
C:\Users\hp\AppData\Local\Temp\ipykernel_8624\4229272512.py:5: SyntaxWarning: invalid escape sequence '\m'
  image = cv2.imread('D:\mice project\Mouse_Lab-1120x630.jpg', cv2.IMREAD_COLOR)
```



3. Load a pretrained Mask R-CNN model from PyTorch and use it for object detection and segmentation on an image

Code Example

```
import torch
from torchvision.models.detection import maskrcnn_resnet50_fpn
from torchvision.transforms import functional as F
from PIL import Image
import matplotlib.pyplot as plt

# Load the pretrained Mask R-CNN model
model = maskrcnn_resnet50_fpn(pretrained=True)
model.eval()  # Set to evaluation mode

# Load and preprocess the image
image_path ="D:\mice project\Mouse_Lab-1120x630.jpg"
image = Image.open(image_path).convert("RGB")
image_tensor = F.to_tensor(image).unsqueeze(0)  # Convert to tensor and add batch
dimension

# Perform object detection and segmentation
with torch.no_grad():
    predictions = model(image_tensor)

# Extract results
boxes = predictions[0]['boxes']
labels = predictions[0]['labels']
scores = predictions[0]['scores']
masks = predictions[0]['masks']

# Filter results by confidence threshold
threshold = 0.8
filtered_indices = [i for i, score in enumerate(scores) if score > threshold]

# Visualize results
plt.figure(figsize=(10, 10))
plt.imshow(image)
ax = plt.gca()
```

```
for i in filtered_indices:
    box = boxes[i].cpu().numpy()
    mask = masks[i, 0].cpu().numpy()
    score = scores[i].item()

    # Draw bounding box
    x1, y1, x2, y2 = box
    rect = plt.Rectangle((x1, y1), x2 - x1, y2 - y1, fill=False, color='red', linewidth=2)
    ax.add_patch(rect)

    # Display mask
    plt.imshow(mask, alpha=0.5, cmap='Reds')

    # Add label and score
    ax.text(x1, y1 - 10, f"{labels[i]}: {score:.2f}", color='white', fontsize=10,
backgroundcolor='red')

plt.axis("off")
plt.show()
```

Explanation

- **Model: maskrcnn_resnet50_fpn is a Mask R-CNN model pretrained on COCO dataset.**
- **Boxes: Bounding boxes of detected objects.**
- **Masks: Segmentation masks for each object.**
- **Labels: Object categories.**

4. Generate bounding boxes for each object detected by Mask R-CNN in an image

To generate bounding boxes for objects detected by a Mask R-CNN model, you can follow these steps in Python. The bounding boxes can be extracted from the Mask R-CNN outputs, typically under the rois key (Region of Interest).

Here's an example code snippet:

Code to Generate Bounding Boxes:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from mrcnn import model as modellib, visualize
from mrcnn.config import Config

# Step 1: Define Configuration
class InferenceConfig(Config):
    NAME = "coco_inference"
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1
    NUM_CLASSES = 1 + 80  # COCO has 80 classes + background

config = InferenceConfig()

# Step 2: Load Mask R-CNN model
model = modellib.MaskRCNN(mode="inference", config=config, model_dir="./")
model.load_weights('mask_rcnn_coco.h5', by_name=True)
```

```
# Step 3: Load an image
image = plt.imread("D:\mice project\Mouse_Lab-1120x630.jpg")
# Step 4: Detect objects
results = model.detect([image], verbose=1)
r = results[0]

# Step 5: Extract and plot bounding boxes
fig, ax = plt.subplots(1, figsize=(12, 12))
ax.imshow(image)

for i, box in enumerate(r['rois']):
    y1, x1, y2, x2 = box
    # Draw bounding box
    rect = Rectangle((x1, y1), x2 - x1, y2 - y1, linewidth=2, edgecolor='red',
facecolor='none')
    ax.add_patch(rect)
    # Annotate with class label
    class_id = r['class_ids'][i]
    score = r['scores'][i]
    label = f"{class_id} ({score:.2f})"
    ax.text(x1, y1 - 5, label, color='red', fontsize=12, backgroundcolor='white')

plt.axis('off')
plt.show()
```

Explanation:

1. **Configuration Setup**:
   - InferenceConfig is used to define the parameters for inference.
   - Set the number of classes to the number used in your dataset (e.g., COCO).
2. **Model Loading**:
   - Load the pre-trained Mask R-CNN weights. Replace 'mask_rcnn_coco.h5' with the correct weights file.
3. **Object Detection**:
   - Use the detect() method to get the detection results. The results dictionary contains rois (bounding boxes), class_ids, and scores.
4. **Plotting**:
   - Extract each bounding box (rois) and draw it using Matplotlib's Rectangle along with annotations for class ID and confidence score.
5. **Output**:
   - The bounding boxes are drawn on the input image with labels and confidence scores.

5. Convert an image to grayscale and apply Otsu's thresholding method for segmentation

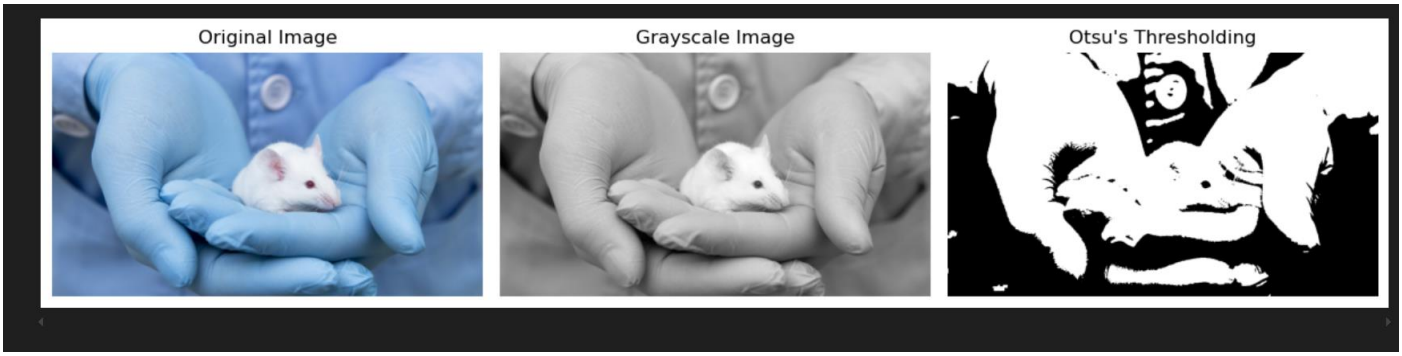
To convert an image to grayscale and apply Otsu's thresholding for segmentation, we can use Python with the **OpenCV** library. Otsu's thresholding is a technique that automatically determines the optimal threshold value for separating foreground and background pixels in an image.

Here's the code:

Code Example

```
import cv2
import matplotlib.pyplot as plt

# Step 1: Read the image
image_path ="D:\mice project\Mouse_Lab-1120x630.jpg"
image = cv2.imread(image_path)

# Step 2: Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 3: Apply Otsu's thresholding
# Otsu's method automatically finds the optimal threshold
_, otsu_thresh = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Step 4: Display results
plt.figure(figsize=(12, 6))

# Original image
plt.subplot(1, 3, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

# Grayscale image
plt.subplot(1, 3, 2)
plt.title("Grayscale Image")
plt.imshow(gray_image, cmap="gray")
plt.axis("off")

# Otsu Thresholding Result
plt.subplot(1, 3, 3)
plt.title("Otsu's Thresholding")
plt.imshow(otsu_thresh, cmap="gray")
plt.axis("off")

plt.tight_layout()
plt.show()
```

output



Explanation

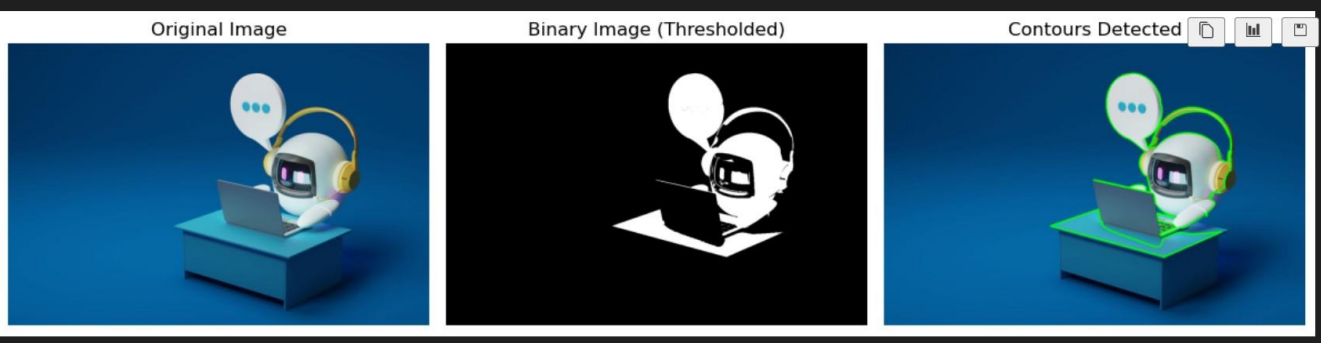
1. **Convert to Grayscale**:
   - cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) converts the input image to a grayscale image.
2. **Apply Otsu's Thresholding**:
   - cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU):
     - The first threshold parameter (0) is ignored because Otsu's method calculates the optimal threshold value.
     - 255 specifies the maximum intensity for binary output.
     - cv2.THRESH_BINARY + cv2.THRESH_OTSU combines binary thresholding with Otsu's optimization.
3. **Display Results**:
   - Using matplotlib, we show the original image, the grayscale version, and the result of Otsu's thresholding.

6. Perform contour detection in an image to detect distinct objects or shapes
   To perform contour detection in an image, we can use OpenCV's cv2.findContours() function. Contour detection identifies boundaries of distinct objects or shapes in a binary image (e.g., after applying thresholding or edge detection)
   Code Example: Contour Detection

```
import cv2
import matplotlib.pyplot as plt

# Step 1: Read the image
image_path = 'D:\mice project\AI-and-customer-experience.jpg'
image = cv2.imread(image_path)

# Step 2: Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 3: Apply thresholding (Otsu's method)
_, binary_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Step 4: Detect contours
# RETR_EXTERNAL: retrieves only the outermost contours
# CHAIN_APPROX_SIMPLE: compresses horizontal, vertical, and diagonal segments
contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Step 5: Draw contours on the original image
# Create a copy of the original image to draw on
contour_image = image.copy()
cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 3)  # Green contours
```

```python
# Step 6: Display results
plt.figure(figsize=(12, 6))

# Original image
plt.subplot(1, 3, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

# Binary image
plt.subplot(1, 3, 2)
plt.title("Binary Image (Thresholded)")
plt.imshow(binary_image, cmap="gray")
plt.axis("off")

# Contour detection result
plt.subplot(1, 3, 3)
plt.title("Contours Detected")
plt.imshow(cv2.cvtColor(contour_image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.tight_layout()
plt.show()

# Optional: Print the number of contours detected
print(f"Number of contours detected: {len(contours)}")
```

```
<>:5: SyntaxWarning: invalid escape sequence '\m'
C:\Users\hp\AppData\Local\Temp\ipykernel_8624\853490903.py:5: SyntaxWarning: invalid escape sequence '\m'
  image_path = 'D:\mice project\AI-and-customer-experience.jpg'
```



7. Apply Mask R-CNN to detect objects and their segmentation masks in a custom image and display them

```python
import torch
from torchvision.models.detection import maskrcnn_resnet50_fpn
from torchvision.transforms import functional as F
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

# Load the pretrained Mask R-CNN model
model = maskrcnn_resnet50_fpn(pretrained=True)
```

```
model.eval()

# Load and preprocess the custom image
image_path = 'D:\mice project\Mouse_Lab-1120x630.jpg'
image = Image.open(image_path).convert("RGB")
image_tensor = F.to_tensor(image).unsqueeze(0)  # Convert image to tensor

# Perform inference
with torch.no_grad():
    predictions = model(image_tensor)

# Extract predictions
boxes = predictions[0]['boxes'].cpu().numpy()
masks = predictions[0]['masks'].cpu().numpy()
scores = predictions[0]['scores'].cpu().numpy()

# Set confidence threshold
threshold = 0.8
filtered_indices = np.where(scores > threshold)[0]

# Visualize results
plt.figure(figsize=(10, 10))
plt.imshow(image)
ax = plt.gca()

for i in filtered_indices:
    # Bounding box
    x1, y1, x2, y2 = boxes[i]
    rect = plt.Rectangle((x1, y1), x2 - x1, y2 - y1, fill=False, color='red', linewidth=2)
    ax.add_patch(rect)

    # Overlay segmentation mask
    mask = masks[i][0] > 0.5  # Convert to binary mask
    colored_mask = np.zeros((*mask.shape, 3), dtype=np.uint8)
    colored_mask[mask] = [255, 0, 0]  # Red mask
    plt.imshow(np.where(mask[..., None], colored_mask, np.array(image)), alpha=0.5)

plt.axis("off")
plt.show()
```

8. Apply k-means clustering for segmenting regions in an image.

```
import cv2
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Step 1: Load the image
image_path = 'path_to_your_image.jpg'  # Replace with your image path
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
# Step 2: Reshape the image into a 2D array of pixels
pixels = image.reshape(-1, 3)  # Each pixel has 3 color channels (R, G, B)

# Step 3: Normalize the pixel values
pixels = pixels / 255.0  # Scale RGB values to [0, 1]

# Step 4: Apply K-Means clustering
num_clusters = 4  # Number of segments
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(pixels)

# Step 5: Replace each pixel with its cluster center
segmented_pixels = kmeans.cluster_centers_[kmeans.labels_]
segmented_image = segmented_pixels.reshape(image.shape)

# Step 6: Visualize the segmented image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Segmented Image')
plt.imshow(segmented_image)
plt.axis('off')

plt.show()
```

THE END……