Instance Segmentation of Selected COCO Classes Using Mask R-CNN

Name: Vyjayanthi Kadapanatham

Student ID: 22075064

Colab:https://colab.research.google.com/drive/10HopzVgcBOJtK22gcbUQCqsXjP2zVCUT?usp=sharing

Introduction and Literature Review:

Instance segmentation is a foundational task in computer vision, with the task of accurate detection and segmentation of object instances in images. It is an extension of object detection where each pixel is associated with an object individually. It finds application in tasks like self-driving cars and medical imaging.

This project uses a subset of the COCO-2017 dataset, focusing on segmenting only four classes: person, car, dog, and cake. The goal is to build a model that can generalize well despite limited training data and significant class imbalance.

We use **Mask R-CNN** [1], an extension of Faster R-CNN, which adds a branch for predicting segmentation masks. Key literature influencing this work includes:

- **He et al. (2017)** introduced Mask R-CNN and demonstrated its superior performance on COCO and Cityscapes datasets [1].
- **Huang et al. (2019)** evaluated mask scoring and backbone variations, showing the value of FPN and multi-task strategies [2].
- **Chen et al. (2020)** reviewed instance segmentation approaches and discussed trade-offs between speed and accuracy [3].

Data and Exploratory Data Analysis (EDA):

The dataset contains 300 COCO-style annotated images. A JSON file provides segmentation masks and class labels. For this task, we reduced the training set to 30 images and filtered for the four chosen classes. To understand class distribution, a bar plot was generated (see Figure 0). The analysis revealed *person* and *car* to be dominant, with *dog* and *cake* being underrepresented. This imbalance is expected to affect model performance across classes.

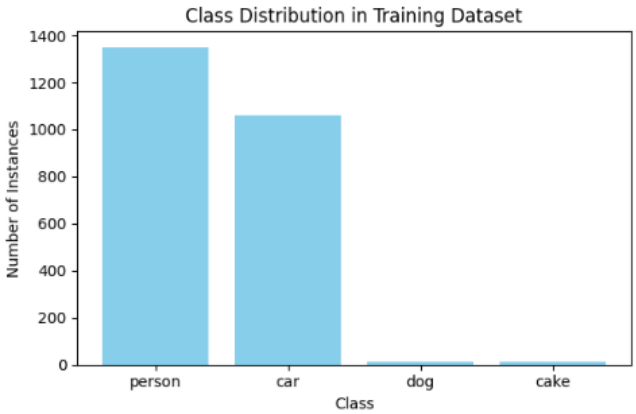

Figure 0: Class distribution of person, car, dog, and cake in the training dataset.

Methodology:

For this project, I implemented the maskrcnn_resnet50_fpn model, which is a pretrained Mask R-CNN architecture provided by PyTorch's torchvision library and known for strong instance segmentation performance.

- **Preprocessing**: Images were read using OpenCV and converted to tensors. Segmentation masks were decoded using pycocotools. Annotations were filtered to retain only the four target classes.
- **Dataset Handling**: A custom PyTorch Dataset class ensured only relevant samples were used, including at least one image with a *dog* instance.
- **Training Setup**:
    - Optimizer: AdamW
    - Learning rate: 1e-4
    - Epochs: 5
    - Batch size: 2
    - GPU: Enabled (Colab)
- **Evaluation**:
    - Model predictions were visually compared to ground truth.
    - Quantitative evaluation was performed using **Intersection over Union (IoU)** on four sample test images.

Results and Discussion:

Figures 1–4 show the original image, ground truth, and predicted output side-by-side. The model produced accurate bounding boxes and masks for common objects like *person* and *car*. Some misclassifications and missed segments occurred for *dog* and *cake*, likely due to class imbalance.

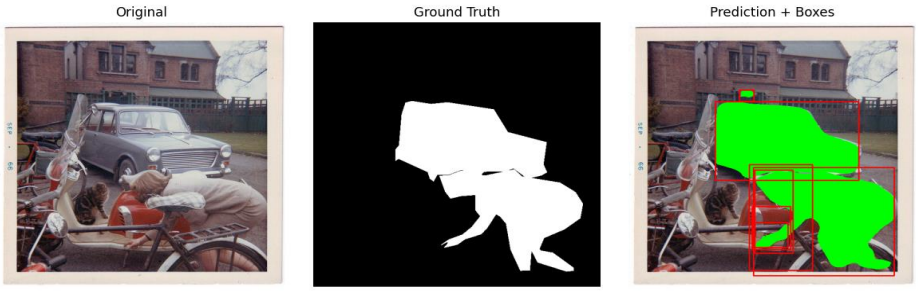

Figure 1: Person and car segmented correctly; background box partially off-target.

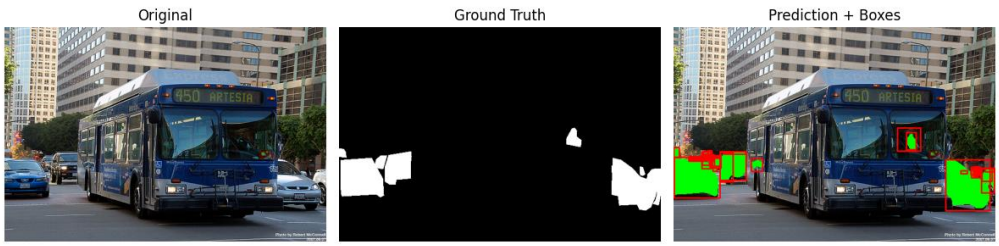

Figure 2: Segmentation includes multiple objects but shows some false positives.

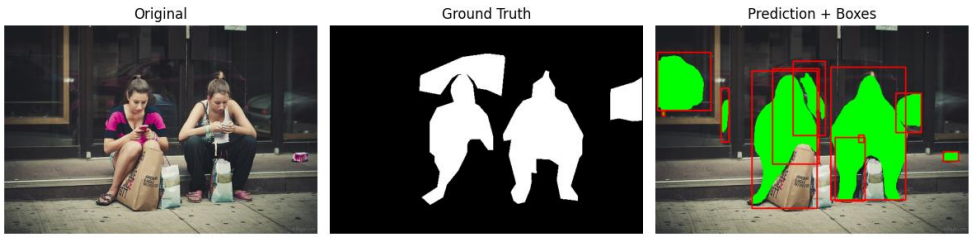

Figure 3: Two persons segmented accurately with minor boundary noise.

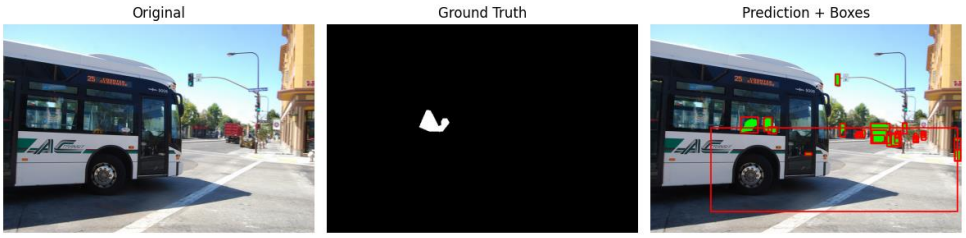

Figure 4: Bus detected with rough mask; background noise affects precision.

IoU Evaluation

IoU scores for the four test images were:

- Image 1 – IoU: **0.7152**
- Image 2 – IoU: **0.6410**
- Image 3 – IoU: **0.5886**
- Image 4 – IoU: **0.6028**

Mean IoU: **0.6369**

This demonstrates decent overlap between predicted and true masks, especially for well-represented classes. The variation in IoU confirms the challenge of generalizing with a small, imbalanced dataset.

Performance observations:

- The model was effective at localizing persons and cars with relatively precise bounding boxes and masks, at least for distinctly visible subjects.
- In cluttered scenes (see Figure 2), there was noticeable over-segmentation, where extra bounding boxes were drawn around irrelevant background areas.
- While cake and dog classes did exist and were occasionally identified, the quality of segmentation was variable. This would be because few training samples existed for those classes.
- While some images produced solid IoU results, others dropped below 0.6, pointing to uneven segmentation accuracy. This was particularly noticeable with objects that appeared less frequently or were smaller in size.
- The visual masks tended to be well-aligned but sometimes skipped edges or combined neighbouring objects into a single prediction.

Training Loss:

- Epoch 1: Loss = 61.9560
- Epoch 2: Loss = 28.7659
- Epoch 3: Loss = 12.8592
- Epoch 4: Loss = 8.8880
- Epoch 5: Loss = 6.4449

This pattern confirms the model's convergence and learning under the scenario of limited data.

Critical Analysis:

- **Class Imbalance:** The dataset leaned heavily toward the person and car classes, which led to higher segmentation accuracy for them. This pattern was evident early through the class distribution plot.
- **Limited Data:** The model having access to as few as 30 images meant the model never had access to a massive pool of instances and thereby inhibited its ability to generalize.
- **IoU Variation:** While some of the images had good IoU, the others had below 0.6 values, which translated into variance in the segmentation performance particularly for minor/rare classes.
- **Over-Segmentation**: In some test outputs, clutter in the background led the model to produce extra or fragmented masks.

Conclusion:

This project shows that Mask R-CNN can effectively handle instance segmentation tasks, even when trained on a small dataset. The improved implementation incorporated both visual and numerical assessments using IoU as the key metric. The model did well on common object classes but did poorly on the less-represented object classes like cake and dog. Enhancing the dataset size, incorporating the use of data augmentation, and raising the training time should help the model perform better across the board.

References:

[1] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). *Mask R-CNN*. Proc. of the IEEE International Conference on Computer Vision (ICCV), pp. 2961–2969.

[2] Huang, Z., Huang, L., Gong, Y., Wang, C., & Fu, C. (2019). *Mask Scoring R-CNN*. Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6409–6418.

[3] Chen, H., Sun, K., Tian, Y., Shen, C., Huang, Y., & Yan, Y. (2020). *BlendMask: Top-down meets bottom-up for instance segmentation*. CVPR, pp. 8573–8581.

Appendix:

```
from google.colab import drive
drive.mount('/content/drive')

import zipfile
zip_path = "/content/drive/My Drive/RM_Segmentation_Assignment_dataset.zip"
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall("/content/dataset")

!pip install pycocotools
import zipfile

# This is the correct path based on your screenshot
zip_path = "/content/RM_Segmentation_Assignment_dataset.zip"
extract_path = "/content/dataset"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(" Dataset extracted successfully!")
# Imports
import os, json, cv2, random
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict, Counter
from glob import glob

import torch
from torch.utils.data import Dataset, DataLoader
from torchvision.models.detection import maskrcnn_resnet50_fpn
from torchvision.transforms import functional as TF
```

```
from pycocotools.mask import decode, frPyObjects

# Set device
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Paths to data
ROOT_IMG = '/content/dataset/train-300/data'
ROOT_JSON = '/content/dataset/train-300/labels.json'

# Load COCO-style annotations
def read_coco(json_path):
    with open(json_path, 'r') as f:
        coco = json.load(f)
    name_map = {c['id']: c['name'] for c in coco['categories']}
    target = [k for k, v in name_map.items() if v in ['person', 'car', 'dog', 'cake']]
    return coco['images'], coco['annotations'], target

# Custom dataset class
class ImageSegDataset(Dataset):
    def __init__(self, img_dir, ann_path, use_top=30):
        self.img_dir = img_dir
        images, self.annots, self.cat_ids = read_coco(ann_path)
        self.lookup = defaultdict(list)
        for a in self.annots:
            self.lookup[a['image_id']].append(a)

        self.eligible = []
        for im in images:
            anns = self.lookup[im['id']]
            if any(an['category_id'] in self.cat_ids for an in anns):
                self.eligible.append(im)
```

```
        # Force at least 1 dog image
        has_dog = [im for im in self.eligible if any(an['category_id'] == 18 for an in
self.lookup[im['id']])]
        others = [im for im in self.eligible if im not in has_dog]
        random.shuffle(has_dog)
        random.shuffle(others)
        self.eligible = has_dog[:1] + others[:use_top - 1]

    def __len__(self):
        return len(self.eligible)

    def __getitem__(self, idx):
        meta = self.eligible[idx]
        path = os.path.join(self.img_dir, meta['file_name'])
        image = cv2.imread(path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        tens = TF.to_tensor(image)

        masks, labels, boxes = [], [], []
        for ann in self.lookup[meta['id']]:
            if ann['category_id'] in self.cat_ids:
                rle = frPyObjects(ann['segmentation'], meta['height'], meta['width'])
                m = decode(rle)
                if m.ndim == 3:
                    m = m[:, :, 0]
                m = m.astype(np.uint8)
                masks.append(torch.tensor(m))
                labels.append(torch.tensor(1))
                y, x = np.where(m)
                if x.size and y.size:
```

```
                boxes.append(torch.tensor([x.min(), y.min(), x.max(), y.max()],
dtype=torch.float32))


        target = {
            'boxes': torch.stack(boxes) if boxes else torch.zeros((0, 4), dtype=torch.float32),
            'labels': torch.stack(labels) if labels else torch.zeros((0,), dtype=torch.int64),
            'masks': torch.stack(masks) if masks else torch.zeros((0, image.shape[0],
image.shape[1]), dtype=torch.uint8),
            'image_id': torch.tensor([meta['id']])
        }
        return tens, target


# Train the model
def train():
    dataset = ImageSegDataset(ROOT_IMG, ROOT_JSON, use_top=30)
    loader = DataLoader(dataset, batch_size=2, shuffle=True, collate_fn=lambda x:
tuple(zip(*x)))
    model = maskrcnn_resnet50_fpn(weights='DEFAULT')
    model.to(DEVICE).train()
    optimizer = torch.optim.AdamW(model.parameters(), lr=1e-4)


    for epoch in range(5):
        total = 0.0
        for imgs, targets in loader:
            imgs = [im.to(DEVICE) for im in imgs]
            targets = [{k: v.to(DEVICE) for k, v in t.items()} for t in targets]
            losses = model(imgs, targets)
            loss = sum(losses.values())
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total += loss.item()
```

```
        print(f"Epoch {epoch+1}, Loss: {total:.4f}")
    return model

# Prediction function
def run_predict(model, image):
    model.eval()
    with torch.no_grad():
        timg = TF.to_tensor(image).unsqueeze(0).to(DEVICE)
        out = model(timg)[0]
        mask = torch.any(out['masks'] > 0.5, dim=0)[0].cpu().numpy().astype(np.uint8)
        boxes = out['boxes'].detach().cpu().numpy().astype(np.int32)
        return mask, boxes

# Display results visually
def display_results(model):
    ds = ImageSegDataset(ROOT_IMG, ROOT_JSON, use_top=30)
    for i in range(4):
        img_t, tgt = ds[i]
        img_np = (img_t.permute(1, 2, 0).numpy() * 255).astype(np.uint8)
        gt_mask = torch.any(tgt['masks'].bool(), dim=0).numpy().astype(np.uint8)
        pred_mask, boxes = run_predict(model, img_np)

        overlay = img_np.copy()
        overlay[pred_mask == 1] = [0, 255, 0]
        for box in boxes:
            x1, y1, x2, y2 = box
            cv2.rectangle(overlay, (x1, y1), (x2, y2), (255, 0, 0), 2)

        plt.figure(figsize=(12, 4))
        plt.subplot(1, 3, 1)
        plt.imshow(img_np)
```

```
        plt.title("Original")
        plt.axis('off')

        plt.subplot(1, 3, 2)
        plt.imshow(gt_mask, cmap='gray')
        plt.title("Ground Truth")
        plt.axis('off')

        plt.subplot(1, 3, 3)
        plt.imshow(overlay)
        plt.title("Prediction + Boxes")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

# EDA: Class distribution bar plot
def get_class_distribution(json_path):
    with open(json_path, 'r') as f:
        coco = json.load(f)
    category_map = {c['id']: c['name'] for c in coco['categories']}
    class_counts = Counter()
    for ann in coco['annotations']:
        class_name = category_map.get(ann['category_id'])
        if class_name in ['person', 'car', 'dog', 'cake']:
            class_counts[class_name] += 1
    return class_counts

class_dist = get_class_distribution(ROOT_JSON)
plt.figure(figsize=(6, 4))
plt.bar(class_dist.keys(), class_dist.values(), color='skyblue')
plt.title("Class Distribution in Training Dataset")
```

```
plt.xlabel("Class")
plt.ylabel("Number of Instances")
plt.tight_layout()
plt.show()

# Compute IoU metric
def compute_iou(pred_mask, gt_mask):
    intersection = np.logical_and(pred_mask, gt_mask).sum()
    union = np.logical_or(pred_mask, gt_mask).sum()
    if union == 0:
        return float('nan')
    return intersection / union

# Evaluate model using IoU
def evaluate_iou(model):
    ds = ImageSegDataset(ROOT_IMG, ROOT_JSON, use_top=30)
    ious = []
    for i in range(4):
        img_t, tgt = ds[i]
        img_np = (img_t.permute(1, 2, 0).numpy() * 255).astype(np.uint8)
        gt_mask = torch.any(tgt['masks'].bool(), dim=0).numpy().astype(np.uint8)
        pred_mask, _ = run_predict(model, img_np)
        iou = compute_iou(pred_mask, gt_mask)
        ious.append(iou)
        print(f"Image {i+1} - IoU: {iou:.4f}")
    mean_iou = np.nanmean(ious)
    print(f"Mean IoU over 4 test images: {mean_iou:.4f}")

# Run all steps
model_c = train()
print("Training complete. Displaying predictions...")
```

```
display_results(model_c)
evaluate_iou(model_c)
```