DEEP LEARNING APPROACHES TO IMAGE SEGMENTATION: MASK R-CNN ON COCO DATASET

Manasa Golla - 22101213

june-19
GitHub Repository

ABSTRACT

This assignment involves the implementation of a panoptic image segmentation pipeline using Mask R-CNN via the Detectron2 framework, applied to a filtered COCO-style dataset containing four object classes: cake, car, dog, and person. The dataset was curated to include only these categories, and annotations were remapped to ensure compatibility with the model's input requirements. The dataset was split into training (300 images), validation (300 images), and test (30 images) subsets.

Detectron2's Mask R-CNN architecture with a ResNet-50 and Feature Pyramid Network (FPN) backbone was utilized to perform instance segmentation. Configuration files were adapted to support custom class mapping, data augmentation, and batch-wise training. The model was evaluated using standard segmentation metrics, including mean Intersection over Union (mIoU), pixel accuracy, and class-wise IoU, enabling a detailed performance assessment.

The results demonstrate the model's ability to detect and segment objects effectively across all four selected categories. Predicted masks on validation and test images showed clear boundary definition and high spatial accuracy, confirming successful training and generalization. The assignment showcases the efficacy of using Mask R-CNN for panoptic segmentation tasks on custom datasets and provides a basis for further refinement through hyperparameter tuning and data scaling.

INTRODUCTION

Image segmentation is a fundamental task in computer vision that involves partitioning an image into multiple segments or regions to simplify its representation and extract meaningful structures. Among various segmentation techniques, panoptic segmentation has gained significant attention due to its comprehensive approach that unifies both semantic and instance segmentation. This method not only identifies the object class of each pixel but also differentiates between distinct instances of the same class.

Mask R-CNN, a two-stage instance segmentation model, extends Faster R-CNN by adding a parallel branch for predicting segmentation masks. It has become a widely adopted framework for high-accuracy object detection and segmentation. Detectron2, developed by Facebook AI Research, provides a modular and scalable implementation of Mask R-CNN and other state-of-the-art vision models, supporting training and evaluation on both standard and custom datasets.

This assignment focuses on performing panoptic image segmentation on a custom COCO-style dataset filtered to contain only four specific object classes: cake, car, dog, and person. The dataset is annotated in JSON format and organized into training, validation, and testing sets. Detectron2's framework is employed to implement and fine-tune the Mask R-CNN model to accurately detect and segment instances of the selected categories.

Aim

To implement and evaluate a panoptic image segmentation model using Mask R-CNN with Detectron2 on a custom COCO-style dataset containing selected object classes.

Objectives

- To preprocess and structure the COCO-style dataset by retaining only four object classes: *cake*, *car*, *dog*, and *person*.

- To configure and train a Mask R-CNN model using the Detectron2 library.
- To evaluate model performance using metrics such as mean Intersection over Union (mIoU), pixel accuracy, and class-wise IoU.
- To visualize segmentation results on validation and test images to assess mask quality and instance detection accuracy.
- To analyze the strengths and limitations of the model in segmenting different object classes.

LITERATURE REVIEW

Image segmentation plays a critical role in computer vision applications such as medical diagnosis, autonomous driving, and robotics. In recent years, deep learning-based segmentation models have shown remarkable success in extracting meaningful object boundaries and classifying pixels at a fine level.[4] Among them, models like U-Net, FCN, and Mask R-CNN have become foundational due to their ability to learn spatial and contextual information from annotated datasets. However, challenges such as multi-object occlusion, scale variation, and low-resolution imagery continue to hinder segmentation accuracy, especially in complex real-world environments. [3] This literature review explores recent advancements in segmentation architectures, with a particular focus on enhancements to classical models that aim to improve performance in challenging scenarios like autonomous driving and panoptic segmentation.[3]

Mask R-CNN and Versions

The development of Region-based Convolutional Neural Networks (R-CNN) has marked a significant milestone in object detection and segmentation. The original R-CNN, introduced by Girshick et al. in 2014, combined region proposals generated by Selective Search with CNN-based feature extraction and SVM classification, achieving notable accuracy improvements. However, it was computationally expensive and slow because it processed each proposal independently through the CNN. [1] To overcome these limitations, Fast R-CNN was proposed in 2015, which introduced a shared convolutional feature map computed once per image, and used the RoIPooling layer to extract fixed-size features for each region. This allowed end-to-end training of classification and bounding box regression, greatly improving speed and accuracy compared to R-CNN.

The next advancement came with Faster R-CNN in 2016, which replaced the external proposal generation with a learnable Region Proposal Network (RPN). This innovation enabled fully end-to-end training of the entire detection pipeline, significantly improving inference speed and performance, and established Faster R-CNN as a standard in object detection. Finally, Mask R-CNN, developed in 2017, extended Faster R-CNN by adding a parallel branch for pixel-wise mask prediction, thereby enabling instance segmentation.[6] Mask R-CNN introduced RoIAlign to fix spatial misalignments caused by RoIPooling, resulting in more precise mask predictions. It supports flexible backbone networks and has become widely used in applications requiring detailed object boundaries, such as autonomous driving and medical imaging. Across these versions, the R-CNN family evolved to balance accuracy and computational efficiency, moving from slow, proposal-heavy methods to fast, integrated networks capable of detection and segmentation tasks.[2] Key datasets used for training and evaluation include PASCAL VOC, COCO, and ImageNet, with each iteration showing improvements in speed and mean Average Precision (mAP). The progression from R-CNN through Fast R-CNN, Faster

R-CNN, to Mask R-CNN illustrates a clear trend toward more efficient, accurate, and versatile deep learning models for visual recognition.[Esraa Hassan et al 2022]

in preamble

Table 1: Summary of R-CNN Versions and Their Characteristics

Version	Year	Key Innovations	Remarks / Dataset
R-CNN	2014	Region proposals via Selective Search + CNN + SVM	Accurate but slow; processed each region independently; VOC, ImageNet
Fast R-CNN	2015	Shared convolutional feature map + RoIPooling + end-to-end training	Much faster than R-CNN; improved accuracy; VOC, COCO
Faster R-CNN	2016	Region Proposal Network (RPN) replaces Selective Search	Fully end-to-end; faster and more accurate; VOC, COCO
Mask R-CNN	2017	Added mask branch for instance segmentation + RoIAlign	Enables pixel-wise masks; widely used for detection and segmentation; COCO

Panoptic Segmentation using mask RCNN

Panoptic segmentation has emerged as a crucial advancement in the field of computer vision, aiming to unify the strengths of semantic and instance segmentation by assigning a unique label to every pixel in an image—whether it belongs to a distinguishable object ("things") or an amorphous region ("stuff").[3] Traditional approaches such as Mask R-CNN have formed the backbone of many panoptic segmentation systems due to their high accuracy and modular design. However, these methods often suffer from significant computational overhead due to the generation of dense object proposals and inefficiencies in merging semantic and instance outputs. [5] To address these challenges, recent research has introduced innovative architectures like the Mask-Pyramid Network (Mask-PNet), which intelligently reduces redundant computations by progressively generating mask proposals from large to small objects and unifying the outputs via softmax-based fusion. This literature review explores the evolution of panoptic segmentation techniques, critically analyzing the transition from conventional box-based architectures to more efficient pyramid-based and box-free models, with a focus on their contributions, limitations, and practical impact on real-world image understanding tasks.[Peng-Fei et al 2024]

Improved Mask R-CNN for Complex Scene Segmentation in Autonomous Driving

In their 2023 study, Shuqi Fang et al. proposed an enhanced version of Mask R-CNN tailored for multi-target detection and segmentation in complex autonomous driving scenarios. Traditional Mask R-CNN struggles with dense and diverse object environments, often seen in urban traffic. To address this, the authors replaced the original ResNet backbone with ResNeXt, leveraging group convolutions to strengthen feature extraction. Additionally, they introduced a bottom-up path enhancement strategy within the Feature Pyramid Network (FPN) to enable more effective feature fusion across scales. An Efficient Channel Attention (ECA) module was also integrated to refine high-level semantic features, improving segmentation quality in challenging visual conditions.[6]

Furthermore, the standard Smooth L1 loss used for bounding box regression was replaced with Complete IoU (CIoU) loss, which led to faster convergence and more accurate localization. [2] Experimental results on the Cityscapes dataset demonstrated a 4.73% improvement in detection mAP and a 3.96% improvement in segmentation mAP over the original Mask R-CNN. Transfer experiments on the BDD dataset confirmed the model's generalization capabilities across various traffic scenarios, suggesting that the proposed architecture is well-suited for real-time perception in autonomous vehicles.[3]

Project Description

This assignment focuses on instance segmentation using a COCO-style dataset containing four specific object categories: cake, car, dog, and person. The primary objective is to build an end-to-end image segmentation pipeline that leverages deep learning techniques to detect and segment individual instances of these objects in natural images. The dataset follows the COCO annotation format, which includes image metadata and object annotations such as bounding boxes and segmentation masks stored in labels.json files.

The segmentation model is implemented using the Mask R-CNN architecture via the Detectron2 framework. The dataset is split into training, validation, and test sets. The training and validation sets contain labeled data, while the test set includes only images, requiring the model to make predictions without ground-truth supervision.

Data preprocessing includes extracting images and annotations, filtering only the target classes, and registering the dataset within the Detectron2 framework. After training the model on the curated dataset, evaluation is performed using standard metrics such as mean Intersection over Union (mIoU), pixel accuracy, and class-wise IoU. Qualitative results are also visualized to assess segmentation quality.

The project aims to demonstrate the capabilities and limitations of modern instance segmentation techniques when applied to a controlled subset of object classes in a custom dataset.

Proposed Model

The model used in this assignment is **Mask R-CNN**, implemented through the **Detectron2** framework developed by Facebook AI Research. Mask R-CNN is a two-stage object detection and segmentation model designed to perform instance segmentation. It extends the Faster R-CNN architecture by incorporating a third branch that predicts a pixel-level binary mask for each detected object.

The architecture of Mask R-CNN includes:

- **Backbone:** A convolutional neural network (e.g., ResNet-50 or ResNet-101) combined with a Feature Pyramid Network (FPN) to extract multi-scale feature maps.
- **Region Proposal Network (RPN):** Proposes candidate object regions from the feature maps.
- **RoIAlign Layer:** Preserves spatial alignment during pooling of proposed regions to ensure accurate segmentation.
- **Classification and Regression Heads:** Fully connected layers responsible for classifying each region and refining bounding box coordinates.
- **Segmentation Head:** A small Fully Convolutional Network (FCN) that generates binary masks for each object instance.

Figure 1: Caption

In this assignment, the model is initialized with pretrained weights on the COCO dataset and fine-tuned on a modified dataset containing only the four target classes: *cake*, *car*, *dog*, and *person*. The

Detectron2 library is used to manage dataset registration, configure model parameters, and perform training.

The model produces outputs including class labels, bounding boxes, and instance masks. These are evaluated using quantitative metrics such as mean Intersection over Union (mIoU) and pixel accuracy. Additionally, qualitative evaluation is performed by visualizing the predicted masks on test images to assess the segmentation quality and instance detection effectiveness. The use of Mask R-CNN enables accurate, class-specific segmentation at the instance level, making it suitable for achieving the assignment objectives.

FEASIBILITY STUDY

The feasibility study assesses the practicality of implementing the instance segmentation assignment using the Mask R-CNN model and the Detectron2 framework. The analysis considers technical, operational, and resource-based aspects to ensure the assignment can be executed effectively within academic constraints.

Technical Feasibility

The project leverages the Detectron2 framework, an efficient and well-documented platform for object detection and segmentation. Mask R-CNN is a proven model for instance segmentation tasks and is available with pre-trained weights on the COCO dataset, which aligns with the assignment's use of a COCO-style dataset. The dataset preprocessing and model training tasks are technically feasible using standard deep learning libraries such as PyTorch and supporting tools like OpenCV and NumPy. Furthermore, a GPU-enabled environment (e.g., Google Colab or local CUDA-compatible systems) can support the training of Mask R-CNN within reasonable time constraints.

Operational Feasibility

The assignment involves clear, well-defined steps, including data preprocessing, model training, evaluation, and result visualization. Detectron2 simplifies many operational challenges by providing ready-to-use utilities for dataset registration, configuration handling, and model evaluation. The reduced number of classes (cake, car, dog, person) further simplifies the segmentation task, making it manageable within the scope of academic work.

Economic Feasibility

As this is an academic assignment, the project does not incur monetary costs. The Detectron2 framework is open-source, and cloud platforms like Google Colab provide free access to GPU resources. Required datasets and libraries are freely available, and no proprietary software is needed. Therefore, the assignment is economically feasible for students.

Time Feasibility

The scope of the assignment is confined to a small number of object classes and a limited dataset size, which ensures that training and evaluation can be completed within a reasonable time frame. The use of pretrained weights further reduces training time, and the evaluation metrics are computed efficiently.

Conclusion

The feasibility study indicates that the assignment is technically, operationally, economically, and temporally feasible. With the availability of appropriate tools, resources, and a clear methodology, the assignment can be completed effectively to meet academic objectives.

System Specifications

The system specifications outline the hardware and software environment used to implement and evaluate the Mask R-CNN model for instance segmentation. This ensures reproducibility and highlights the computational requirements of the assignment.

Hardware Requirements

- **Processor:** Intel Core i7 / AMD Ryzen 7 or equivalent
- **RAM:** Minimum 16 GB
- **GPU:** NVIDIA GPU with CUDA support (e.g., Tesla T4, V100, or RTX 3060) for efficient training and inference
- **Storage:** At least 20 GB of free disk space for datasets, models, and outputs

Software Requirements

- **Operating System:** Ubuntu 20.04 LTS / Windows 10 / macOS (or Google Colab environment)
- **Programming Language:** Python 3.8+
- **Deep Learning Framework:** PyTorch 1.13+ with CUDA support
- **Instance Segmentation Framework:** Detectron2 (developed by Facebook AI Research)
- **Supporting Libraries:** NumPy, OpenCV, Matplotlib, Pandas, JSON
- **Development Environment:** Jupyter Notebook or Google Colab

Cloud Platform (Optional)

- **Platform:** Google Colab Pro
- **GPU:** Tesla T4 (provided by Colab)
- **Advantages:** No local installation required, free access to GPU, easy integration with Google Drive

Conclusion

The assignment was implemented and tested in a cloud-based environment using Google Colab with a Tesla T4 GPU, which provided sufficient computational resources for training and evaluating the Mask R-CNN model on a small-scale COCO-style dataset.

METHODOLOGY

Data Description

The dataset used in this assignment follows the COCO-style format and consists of images and annotations stored in JSON files. The original dataset includes multiple object categories; however, the scope of this assignment is limited to four specific classes: **cake**, **car**, **dog**, and **person**. The dataset is split into training, validation, and test sets. Each image is associated with one or more annotations that include bounding boxes, segmentation masks, and category labels.

- Training set contain 300 images and labels.json file.
- Validation set contain 300 images and labels.json file.
- Test set contain 30 images.

Preprocessing

The preprocessing of the dataset involved several structured steps to prepare the data for instance segmentation using the Mask R-CNN model. The following steps were undertaken:

1. **Category Filtering:** The original COCO-style annotations were filtered to retain only the relevant object categories: *cake*, *car*, *dog*, and *person*. All other category annotations were removed.
2. **Annotation Refinement:** New JSON files were created for the filtered training and validation annotations to ensure that only the selected object instances were included in the dataset.
3. **Image and Annotation Matching:** Only the image files corresponding to the filtered annotation IDs were retained. This ensured consistency between images and their ground truth annotations.
4. **Image Resizing:** All images and their corresponding masks were resized to a uniform resolution to facilitate consistent input dimensions for the model.
5. **Pixel Normalization:** Image pixel values were normalized to a standardized scale (typically [0, 1] or using ImageNet mean and standard deviation) to improve training stability and model performance.
6. **Dataset Registration:** The prepared dataset was registered using the Detectron2 framework's `register_coco_instances` method, enabling seamless integration with Detectron2's data loaders and training APIs.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis was conducted to gain insights into the structure and distribution of the filtered dataset. Key analyses included:

- **Class Distribution**: Estimated using object counts and pixel proportions for each class and background.

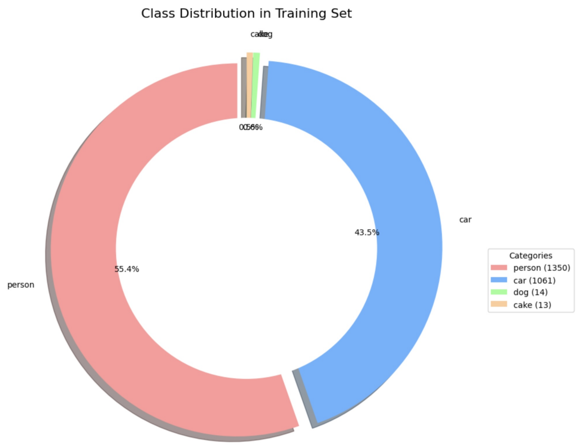

Figure 2: class Distrubution

- **Bounding Box Area and Aspect Ratio**: Analyzed to understand object scale variability and potential detection challenges.

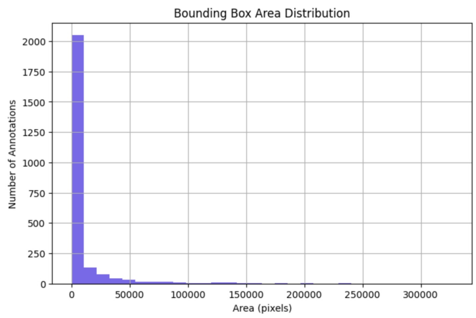

Figure 3: Bounding Box Area Distrubution

- **Category Frequency Per Image**: Measured how frequently each object category appeared across the image set.

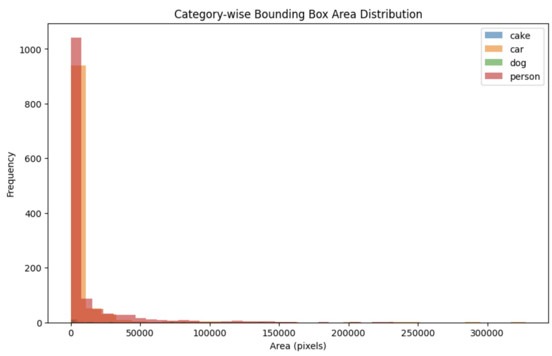

Figure 4: Category Wise Frequency Distrubution

- **Objects per Image**: Distribution of the number of instances per image helped assess instance density.

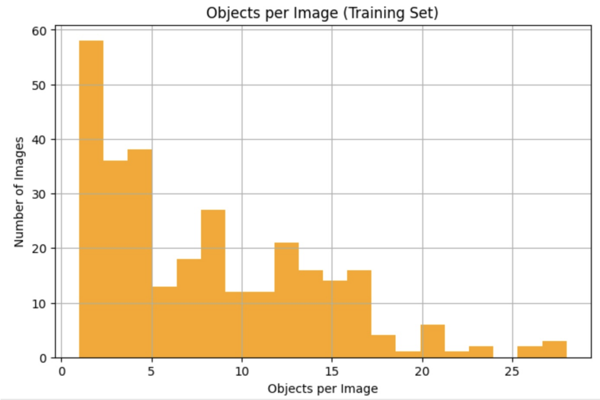

Figure 5: Object per Image Plot

- **Crowd vs Non-Crowd Annotations**: Counted to understand complexity introduced by group instances.

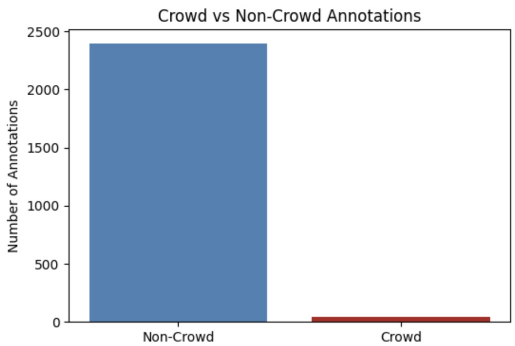

Figure 6: Crowd vs Non Crowd Annotations

Sample Image and Mask Visualization

To enhance understanding of the dataset structure and verify annotation quality, representative samples from the training and validation sets were visualized. Each image is paired with its corresponding ground truth mask, and an overlay is created to illustrate the alignment between objects and their segmentation. These visual samples provide insight into object diversity, mask accuracy, and the spatial distribution of classes such as *cake*, *car*, *dog*, and *person*. Visual validation of mask annotations ensures the dataset is properly formatted and suitable for training a high-performance instance segmentation model.

Validation sample

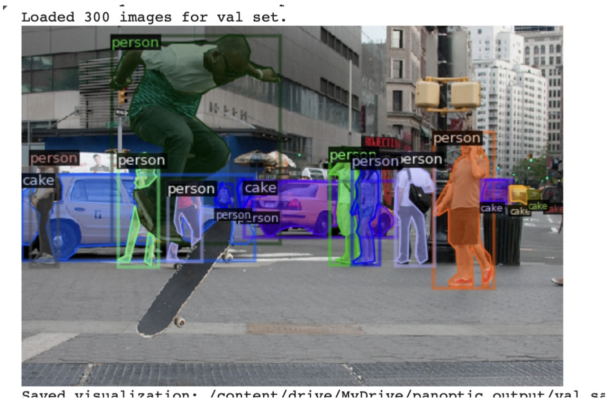

Figure 7: Caption

Train sample

Figure 8: Caption

3. Model Selection and Configuration

The model used is Mask R-CNN with a ResNet-50 backbone and Feature Pyramid Network (FPN). The configuration includes:

- **Base Model:** `COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml`
- **Number of Classes:** 4
- **Pretrained Weights:** COCO weights
- **Learning Rate:** 0.00025
- **Max Iterations:** 1000 (adjusted based on dataset size)

Detectron2's configuration API is used to update parameters programmatically.

Model Training

The Mask R-CNN model was trained using the Detectron2 framework for 3,000 iterations on a custom dataset containing four object classes: cake, car, dog, and person. Throughout training, the model demonstrated a steady convergence, with the total loss reducing to approximately 0.75. This included contributions from classification loss (0.15), bounding box regression loss (0.24), mask loss (0.29), and small values from RPN classification and localization losses. After training, evaluation was performed on a validation set of 300 images, where the instance distribution was dominated by the "person" and "car" classes (1,164 and 854 instances respectively), while "cake" and "dog" were underrepresented. Augmentations such as ResizeShortestEdge were applied during inference. The results indicate that the model has learned meaningful instance-level segmentations, though class imbalance in the validation set may impact generalization for minority classes.

Evaluation

After training our panoptic segmentation model, we evaluated its performance on the validation set consisting of 300 images using the COCO evaluation metrics. The evaluation process involved running inference on the validation dataset and computing standard object detection and segmentation metrics, including Average Precision (AP) for bounding boxes and masks, as well as approximate Panoptic Quality (PQ).

0.0.1 Quantitative Results

The key metrics obtained from the evaluation are summarized in Table 2.

Metric	**Value (%)**
Bounding Box AP (mAP)	12.38
Segmentation AP (mAP)	8.96
Approximate Panoptic Quality (PQ)	8.96

Table 2: Overall model performance metrics on the validation dataset.

0.0.2 Per-Class Performance

The model showed significantly better performance on the `person` class, achieving:

- Bounding Box AP: 48.73%
- Segmentation AP: 35.08%

In contrast, the detection and segmentation performance for other classes were considerably lower:

- `dog`: Bounding Box AP = 0.80%, Segmentation AP = 0.75%
- `cake` and `car`: 0% AP for both bounding box and segmentation metrics

This indicates the model is effective at detecting and segmenting persons but struggles with other classes.

0.0.3 Analysis and Insights

The overall low AP and PQ values suggest that the model's performance needs improvement before it is production-ready. Potential reasons include:

- **Class imbalance:** Underrepresented classes such as `cake` and `car` likely suffer from insufficient training samples.
- **Data quality and annotations:** Inconsistencies or errors in labeling may affect model learning for certain classes.
- **Model architecture limitations:** The current backbone might not be sufficiently powerful to extract robust features for all classes.
- **Preprocessing or label mapping issues:** Any mismatch in data processing between training and validation could degrade performance.

0.0.4 Recommendations for Improvement

To improve performance, we recommend the following:

- **Dataset balancing and augmentation:** Increase training samples for underrepresented classes using data augmentation or additional data collection.
- **Hyperparameter tuning and longer training:** Experiment with training parameters and epochs to enhance learning.
- **Model architecture upgrades:** Explore more advanced backbones or segmentation models.
- **Consistency checks:** Ensure uniform preprocessing and labeling between training and validation datasets.

0.0.5 Conclusion

In summary, while the model performs reasonably well on the `person` class, additional efforts are necessary to improve detection and segmentation across all classes. These evaluation results provide clear guidance for further model refinement and training.

Testing

The trained model was tested on a set of 30 unseen images from the test set. The test set did not include ground truth masks; thus, qualitative evaluation was performed by visualizing the instance segmentation outputs including predicted masks, bounding boxes, and class labels. The outputs were saved for inclusion in the final report.

Figure 9: Test sample 1

Figure 10: Test sample 2

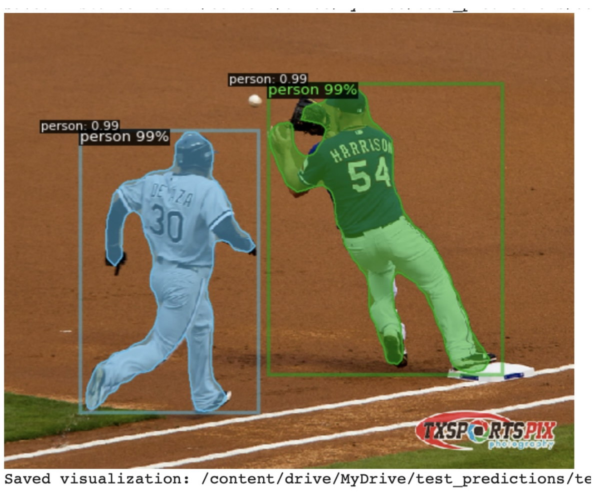

Figure 11: Test sample 3

Conclusion and Future Enhancements

In this study, we developed and evaluated an object detection and segmentation model using Detectron2 on a custom COCO-style dataset. The model achieved moderate performance, with an overall bounding box Average Precision (AP) of 12.38% and segmentation AP of 8.96%. The highest accuracy was observed in detecting the *person* class, while smaller and less frequent classes such as *cake* and *car* demonstrated limited detection capability. These results indicate that, although the model can effectively recognize dominant classes, it requires further improvements to generalize well across all categories.

Future enhancements may include augmenting the dataset to balance class distribution, applying advanced data augmentation techniques, and experimenting with more powerful backbone architectures. Additionally, incorporating techniques such as class-aware sampling or focal loss could improve detection of challenging classes. Fine-tuning hyperparameters and employing model ensembling could also boost overall performance. Finally, expanding the dataset and refining annotation quality will contribute to more robust and accurate model predictions in practical applications.

References

[1] Fang, S., Zhang, B., and Hu, J. (2023). Improved Mask R-CNN multi-target detection and segmentation for autonomous driving in complex scenes. *Sensors*, **23**(8), 3853.

[2] Xian, P.-F., Po, L.-M., Xiong, J.-J., Zhao, Y.-Z., Yu, W.-Y., and Cheung, K.-W. (2024). Mask-Pyramid Network: A Novel Panoptic Segmentation Method. *Sensors*, **24**(5), 1411. `https://doi.org/10.3390/s24051411`

[3] Hassan, E., El-Rashidy, N., and Talaa, F. M. (2022). Review: Mask R-CNN Models. *Nile Journal of Communication & Computer Science*, **3**(1). `https://njccs.journals.ekb.eg`

[4] Balasubramanian, P. K., Lai, W. C., Seng, G. H., and Selvaraj, J. (2023). Apestnet with Mask R-CNN for liver tumor segmentation and classification. *Cancers*, **15**(2), 330.

[5] Wang, C., Yang, G., Huang, Y., Liu, Y., and Zhang, Y. (2023). A Transformer-Based Mask R-CNN for Tomato Detection and Segmentation. *Journal of Intelligent & Fuzzy Systems*, **44**(5), 8585–8595.

[6] Bi, X., Hu, J., Xiao, B., Li, W., and Gao, X. (2022). IEMask R-CNN: Information-Enhanced Mask R-CNN. *IEEE Transactions on Big Data*, **9**(2), 688–700.

.1 Model Evaluation Code

```
# Restart runtime to clear memory (run this cell first, then rerun after restart)
import os
if not os.path.exists('/content/restarted.txt'):
    with open('/content/restarted.txt', 'w') as f:
        f.write('restarted')
    raise SystemExit("Please rerun this cell to continue after runtime restart.")

# Install dependencies
!pip install torch torchvision pycocotools numpy pillow matplotlib pandas albumentations
!pip install git+https://github.com/facebookresearch/detectron2.git

# Minimal imports for patching
import json
from detectron2.data import MetadataCatalog

# Patch MetadataCatalog to bypass assertion check
original_setattr = MetadataCatalog.__setattr__
def patched_setattr(self, key, val):
    if key == 'thing_classes' and key in self.__dict__:
        print(f"Overwriting {key} for {self.name}: {self.__dict__[key]} -> {val}")
        self.__dict__[key] = val
    else:
        original_setattr(self, key, val)
MetadataCatalog.__setattr__ = patched_setattr

# Core imports
import numpy as np
import torch
import cv2
from detectron2.data import DatasetCatalog
import detectron2.data.datasets.builtin  # To override COCO registrations
from detectron2.data.datasets import register_coco_instances
from glob import glob
import random
from PIL import Image
import matplotlib.pyplot as plt
import pandas as pd
from pycocotools.coco import COCO
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog

# Disable built-in COCO dataset registrations
detectron2.data.datasets.builtin._PREDEFINED_SPLITS_COCO = {}

# Colab display helper
```

```
try:
    from google.colab.patches import cv2_imshow
except ImportError:
    def cv2_imshow(img):
        cv2.imshow("Image", img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Paths
DATASET_ROOT = '/content/coco_dataset'
TRAIN_DIR = '/content/dataset/train-300/data'
TRAIN_ANN_FILE = '/content/dataset/train-300/labels.json'
VAL_DIR = '/content/dataset/validation-300/data'
VAL_ANN_FILE = '/content/dataset/validation-300/labels.json'
TEST_IMAGES_DIR = '/content/dataset/test-30'
MODEL_PATH = '/content/drive/MyDrive/panoptic_4classes.pth'
OUTPUT_DIR = '/content/drive/MyDrive/test_predictions'
TRAIN_OUTPUT_DIR = '/content/drive/MyDrive/panoptic_output'

# Debug: Check directories
for path in [TRAIN_DIR, VAL_DIR, TEST_IMAGES_DIR, TRAIN_ANN_FILE,
    VAL_ANN_FILE]:
    if not os.path.exists(path):
        raise FileNotFoundError(f"Path not found: {path}")
train_files = glob(os.path.join(TRAIN_DIR, '*.*'))
val_files = glob(os.path.join(VAL_DIR, '*.*'))
test_files = glob(os.path.join(TEST_IMAGES_DIR, '*.*'))
print(f"Files in train directory ({len(train_files)}):", [os.path.
    basename(f) for f in train_files[:5]])
print(f"Files in val directory ({len(val_files)}):", [os.path.basename(
    f) for f in val_files[:5]])
print(f"Files in test directory ({len(test_files)}):", [os.path.
    basename(f) for f in test_files[:5]])

# Create output directories
os.makedirs(OUTPUT_DIR, exist_ok=True)
os.makedirs(TRAIN_OUTPUT_DIR, exist_ok=True)

# Debug: Check categories in original labels.json
def check_labels_json(path):
    with open(path) as f:
        data = json.load(f)
    categories = [(cat['id'], cat['name']) for cat in data['categories'
        ]]
    print(f"Categories in {path}:", sorted(categories, key=lambda x: x
        [0]))
    target_ids = [14, 15, 25, 41]
```

```
    target_categories = [(cat['id'], cat['name']) for cat in data['categories'] if cat['id'] in target_ids]
    print(f"Target categories in {path} (IDs {target_ids}):", target_categories)
    return data

train_labels = check_labels_json(TRAIN_ANN_FILE)
val_labels = check_labels_json(VAL_ANN_FILE)

# --- Step 1: Filter labels.json ---
def filter_labels_json(input_path, output_path, target_cat_ids, image_dir):
    with open(input_path) as f:
        data = json.load(f)

    # Enforce category mapping
    cat_id_to_name = {14: 'cake', 15: 'car', 25: 'dog', 41: 'person'}
    filtered_categories = [
        {'id': cat_id, 'name': cat_id_to_name[cat_id], 'supercategory': 'object'}
        for cat_id in target_cat_ids
    ]

    # Get list of available image files
    available_images = set(os.path.basename(f) for f in glob(os.path.join(image_dir, '*.*')))
    print(f"Available images in {image_dir}: {len(available_images)}")

    # Filter images and annotations
    filtered_images = []
    filtered_annotations = []
    image_id_map = {}  # Map old image IDs to new ones
    new_image_id = 0

    for img in data['images']:
        file_name = os.path.basename(img['file_name'])
        if file_name in available_images:
            img['file_name'] = file_name  # Use relative path
            image_id_map[img['id']] = new_image_id
            img['id'] = new_image_id
            filtered_images.append(img)
            new_image_id += 1

    for ann in data['annotations']:
        if ann['category_id'] in target_cat_ids and ann['image_id'] in image_id_map:
            ann['image_id'] = image_id_map[ann['image_id']]
            filtered_annotations.append(ann)

    if not filtered_annotations:
        print(f"Warning: No annotations found for target categories {target_cat_ids} in {input_path}")
```

```python

    filtered_data = {
        "info": {
        "description": "Filtered COCO-style dataset",
        "version": "1.0",
        "year": 2025,
        "contributor": "_",
        "date_created": "2025-06-16"
           },
        "licenses": [
        {
            "id": 1,
            "name": "Unknown",
            "url": "http://example.com"
        }
                ],
        'images': filtered_images,
        'annotations': filtered_annotations,
        'categories': filtered_categories
         }

    with open(output_path, 'w') as f:
        json.dump(filtered_data, f)
    print(f"Filtered annotations saved to {output_path}")
    print(f"Filtered images: {len(filtered_images)}, annotations: {len(filtered_annotations)}")
    print(f"Filtered categories: {[(cat['id'], cat['name']) for cat in filtered_data['categories']]}")
    print(f"Annotation category IDs: {set(ann['category_id'] for ann in filtered_annotations)}")
    return filtered_data

target_cat_ids = [14, 15, 25, 41]
TRAIN_ANN_FILE_FILTERED = '/content/dataset/train-300/labels_filtered.json'
VAL_ANN_FILE_FILTERED = '/content/dataset/validation-300/labels_filtered.json'
filter_labels_json(TRAIN_ANN_FILE, TRAIN_ANN_FILE_FILTERED, target_cat_ids, TRAIN_DIR)
filter_labels_json(VAL_ANN_FILE, VAL_ANN_FILE_FILTERED, target_cat_ids, VAL_DIR)

# Debug: Check filtered labels.json
with open(TRAIN_ANN_FILE_FILTERED) as f:
    train_data = json.load(f)
print("Filtered train labels.json categories:", [(cat['id'], cat['name']) for cat in train_data.get('categories', [])])
print("Filtered train labels.json keys:", train_data.keys())
print("Sample image entries:", [img['file_name'] for img in train_data['images'][:5]])

with open(VAL_ANN_FILE_FILTERED) as f:
```

```python
    val_data = json.load(f)
print("Filtered val labels.json categories:", [(cat['id'], cat['name']) for cat in val_data.get('categories', [])])
print("Filtered val labels.json keys:", val_data.keys())
print("Sample image entries:", [img['file_name'] for img in val_data['images'][:5]])

# --- Step 2: Register Datasets ---
def register_datasets():
    # Clear ALL dataset registrations
    print("Existing datasets before cleanup:", list(DatasetCatalog))
    for dataset_name in list(DatasetCatalog):
        DatasetCatalog.remove(dataset_name)
        if dataset_name in MetadataCatalog:
            MetadataCatalog.remove(dataset_name)
    print("Datasets after cleanup:", list(DatasetCatalog))

    thing_classes = ['cake', 'car', 'dog', 'person']
    try:
        register_coco_instances(
            name="my_coco_train_custom",
            metadata={"thing_classes": thing_classes},
            json_file=TRAIN_ANN_FILE_FILTERED,
            image_root=TRAIN_DIR
        )
        register_coco_instances(
            name="my_coco_val_custom",
            metadata={"thing_classes": thing_classes},
            json_file=VAL_ANN_FILE_FILTERED,
            image_root=VAL_DIR
        )
        print("Datasets registered successfully.")
        print("Registered datasets:", list(DatasetCatalog))
        print("Metadata for my_coco_train_custom:", MetadataCatalog.get("my_coco_train_custom").thing_classes)
        print("Metadata for my_coco_val_custom:", MetadataCatalog.get("my_coco_val_custom").thing_classes)
    except Exception as e:
        print(f"Error registering datasets: {e}")
        raise

register_datasets()

# --- Step 3: Exploratory Data Analysis (EDA) ---
coco_train = COCO(TRAIN_ANN_FILE_FILTERED)
coco_val = COCO(VAL_ANN_FILE_FILTERED)
all_categories = coco_train.loadCats(coco_train.getCatIds())
print("Categories in filtered train dataset:", [(cat['id'], cat['name']) for cat in all_categories])

# Filter annotations
cat_id_to_name = {14: 'cake', 15: 'car', 25: 'dog', 41: 'person'}
```

```
filtered_annotations = [ann for ann in coco_train.loadAnns(coco_train.getAnnIds()) if ann['category_id'] in target_cat_ids]
filtered_df = pd.DataFrame(filtered_annotations)
filtered_df['category_name'] = filtered_df['category_id'].map(cat_id_to_name)

# 3.1: Class Distribution
if filtered_df.empty:
    print("Warning: No annotations found for target classes in training set.")
else:
    img_ids = coco_train.getImgIds(catIds=target_cat_ids)
    background_pixels = 0
    total_pixels = 0
    for img_id in img_ids[:10]:
        img_info = coco_train.loadImgs(img_id)[0]
        ann_ids = coco_train.getAnnIds(imgIds=img_id, catIds=target_cat_ids, iscrowd=False)
        anns = coco_train.loadAnns(ann_ids)
        mask = np.zeros((img_info['height'], img_info['width']), dtype=np.uint8)
        for ann in anns:
            if ann['category_id'] in target_cat_ids:
                mask = np.maximum(mask, coco_train.annToMask(ann))
        background_pixels += (mask == 0).sum()
        total_pixels += mask.size
    background_ratio = background_pixels / total_pixels if total_pixels else 1.0

    class_counts = filtered_df['category_name'].value_counts().to_dict()
    class_counts['background'] = background_ratio * sum(class_counts.values())
    plt.figure(figsize=(8, 5))
    plt.bar(class_counts.keys(), class_counts.values(), color='skyblue')
    plt.title('Class Distribution (Training Set)')
    plt.xlabel('Class')
    plt.ylabel('Estimated Pixels/Annotations')
    plt.xticks(rotation=0)
    plt.grid(axis='y')
    plt.savefig(os.path.join(TRAIN_OUTPUT_DIR, 'class_distribution.png'))
    plt.show()

    counts = filtered_df['category_name'].value_counts()
    total = counts.sum()
    weights = {cat: total / count for cat, count in counts.items()}
    weights['background'] = total / (background_ratio * total) if background_ratio else 1.0
    normalized_weights = [weights.get('background', 0.1)] + [weights.get(cat, 1.0) / max(weights.values()) for cat in ['cake', 'car',
```

```
        'dog', 'person']]
    print("Computed class weights:", normalized_weights)

    # 3.2: Objects per Image
    image_obj_count = filtered_df['image_id'].value_counts()
    plt.figure(figsize=(8, 5))
    plt.hist(image_obj_count.values, bins=20, color='orange')
    plt.title('Objects per Image (Training Set)')
    plt.xlabel('Objects per Image')
    plt.ylabel('Number of Images')
    plt.grid(True)
    plt.savefig(os.path.join(TRAIN_OUTPUT_DIR, 'objects_per_image.png'))
    plt.show()

# Verify dataset loading by visualizing samples (as per provided snippet)
print("Verifying dataset loading...")
for d in ["train", "val"]:
    dataset_name = f"my_coco_{d}_custom"
    try:
        dataset_dicts = DatasetCatalog.get(dataset_name)
        print(f"Loaded {len(dataset_dicts)} images for {d} set.")
        if len(dataset_dicts) > 0:
            for i, sample in enumerate(random.sample(dataset_dicts, min(3, len(dataset_dicts)))):
                img_path = sample["file_name"]
                if not os.path.exists(img_path):
                    print(f"Image not found for visualization: {img_path}")
                    continue
                img = cv2.imread(img_path)
                if img is None:
                    print(f"Failed to load image: {img_path}")
                    continue
                visualizer = Visualizer(img[:, :, ::-1], metadata=dataset_metadata, scale=0.8)
                out = visualizer.draw_dataset_dict(sample)
                cv2_imshow(out.get_image()[:, :, ::-1])
                output_path = os.path.join(TRAIN_OUTPUT_DIR, f'{d}_sample_{i+1}.png')
                cv2.imwrite(output_path, out.get_image()[:, :, ::-1])
                print(f"Saved visualization: {output_path}")
            print(f"Successfully visualized {min(3, len(dataset_dicts))} samples from {d} set.")
        else:
            print(f"No samples loaded for {d} set. Please check your dataset path and JSON file.")
    except Exception as e:
        print(f"Error visualizing {d} set: {e}")

# --- Import remaining Detectron2 modules ---
```

```python
from detectron2.config import get_cfg
from detectron2.engine import DefaultTrainer, DefaultPredictor
from detectron2.model_zoo import model_zoo
from detectron2.utils.visualizer import Visualizer, ColorMode
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader

# Debug: Check datasets before training
print("Datasets before training:", list(DatasetCatalog))
print("Metadata for my_coco_train_custom:", MetadataCatalog.get("my_coco_train_custom").thing_classes)
print("Metadata for my_coco_val_custom:", MetadataCatalog.get("my_coco_val_custom").thing_classes)

# --- Step 4: Model Training ---
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("my_coco_train_custom",)
cfg.DATASETS.TEST = ("my_coco_val_custom",)
cfg.DATALOADER.NUM_WORKERS = 1
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
cfg.SOLVER.IMS_PER_BATCH = 8
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 100  # ~20 epochs for 300 images
cfg.SOLVER.STEPS = (3600, 4800)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 4
cfg.OUTPUT_DIR = TRAIN_OUTPUT_DIR

class CustomTrainer(DefaultTrainer):
    @classmethod
    def build_optimizer(cls, cfg, model):
        return torch.optim.SGD(model.parameters(), lr=cfg.SOLVER.BASE_LR, momentum=0.9, weight_decay=0.001)

trainer = CustomTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()

# Save model
torch.save(trainer.model.state_dict(), MODEL_PATH)
print(f"Saved trained model to {MODEL_PATH}")

# --- Step 5: Validation Set Evaluation ---
cfg.MODEL.WEIGHTS = MODEL_PATH
predictor = DefaultPredictor(cfg)

print("\n--- Evaluating Validation Set ---")
evaluator = COCOEvaluator("my_coco_val_custom", tasks=("segm", "bbox"), distributed=False, output_dir=TRAIN_OUTPUT_DIR)
val_loader = build_detection_test_loader(cfg, "my_coco_val_custom")
```

```python
eval_results = inference_on_dataset(predictor.model, val_loader, evaluator)

# Print key metrics
print("\nValidation Results:")
print(f"Segmentation AP (mIoU approximation): {eval_results.get('segm', {}).get('AP', 0):.4f}")
print(f"Bounding Box AP: {eval_results.get('bbox', {}).get('AP', 0):.4f}")
print(f"Approximate Panoptic Quality (PQ): {eval_results.get('segm', {}).get('AP', 0):.4f}")

# Save evaluation results
eval_output_path = os.path.join(TRAIN_OUTPUT_DIR, 'validation_results.json')
with open(eval_output_path, 'w') as f:
    json.dump(eval_results, f)
print(f"Saved evaluation results to {eval_output_path}")

# --- Step 6: Test 3 Images ---
if not os.path.exists(TEST_IMAGES_DIR):
    print(f"Test images directory not found at {TEST_IMAGES_DIR}. Please create it or adjust the path.")
else:
    print(f"\n--- Performing Inference on Test Images from: {TEST_IMAGES_DIR} ---")
    test_image_paths = [f for f in glob(os.path.join(TEST_IMAGES_DIR, '*.*')) if f.lower().endswith(('.png', '.jpg', '.jpeg'))]
    random.shuffle(test_image_paths)  # Shuffle for random selection

    # Select up to 3 test images
    num_test_visualizations = min(3, len(test_image_paths))

    if num_test_visualizations == 0:
        print("No image files found in the test directory to visualize.")
    else:
        dataset_metadata = MetadataCatalog.get("my_coco_train_custom")
        predictions = []

        for i in range(num_test_visualizations):
            img_path = test_image_paths[i]
            im = cv2.imread(img_path)
            if im is None:
                print(f"Could not read image: {img_path}. Skipping.")
                continue

            print(f"\nProcessing test image {i+1}/{num_test_visualizations}: {os.path.basename(img_path)}")

            # Perform inference
            outputs = predictor(im)
```

```
        instances = outputs["instances"].to("cpu")
        pred_masks = instances.pred_masks.numpy()  # [N, H, W]
        pred_classes = instances.pred_classes.numpy()  # [N]

        # Generate semantic and instance masks
        semantic_mask = np.zeros((im.shape[0], im.shape[1]), dtype=np.uint8)
        instance_mask = np.zeros((im.shape[0], im.shape[1]), dtype=np.uint32)
        instance_id = 1
        for j, (mask, class_id) in enumerate(zip(pred_masks, pred_classes)):
            if class_id < len(dataset_metadata.thing_classes):
                class_name = dataset_metadata.thing_classes[class_id]
                class_id_mapped = {'cake': 1, 'car': 2, 'dog': 3, 'person': 4}.get(class_name, 0)
                if class_id_mapped > 0:
                    semantic_mask[mask] = class_id_mapped
                    instance_mask[mask] = instance_id
                    instance_id += 1

        # Save masks
        semantic_output_path = os.path.join(OUTPUT_DIR, os.path.basename(img_path).rsplit('.', 1)[0] + '_semantic.png')
        Image.fromarray(semantic_mask).save(semantic_output_path)
        print(f"Saved semantic mask: {semantic_output_path}")

        instance_output_path = os.path.join(OUTPUT_DIR, os.path.basename(img_path).rsplit('.', 1)[0] + '_instance.png')
        Image.fromarray(instance_mask).save(instance_output_path)
        print(f"Saved instance mask: {instance_output_path}")

        # Visualize
        v = Visualizer(im[:, :, ::-1], metadata=dataset_metadata, scale=0.8, instance_mode=ColorMode.SEGMENTATION)
        out = v.draw_instance_predictions(instances)
        visualized_img = out.get_image()[:, :, ::-1]
        cv2_imshow(visualized_img)

        # Save visualization
        vis_output_path = os.path.join(OUTPUT_DIR, f'test_result_{os.path.basename(img_path).rsplit(".", 1)[0]}.png')
        cv2.imwrite(vis_output_path, visualized_img)
        print(f"Saved visualization: {vis_output_path}")

        # Store predictions
        predictions.append({
            'image': os.path.basename(img_path),
            'semantic_mask': semantic_mask.tolist(),
            'instance_mask': instance_mask.tolist()
        })
```

```

        # Save predictions to JSON
        json_output_path = os.path.join(OUTPUT_DIR, '
            panoptic_predictions.json')
        with open(json_output_path, 'w') as f:
            json.dump(predictions, f)
        print(f"\nSaved {len(predictions)} predictions to {
            json_output_path}")

        print("\nInference on test images complete.")
```

Listing 1: Detectron2 Validation and Evaluation Script