University of Hertfordshire UH

School of Physics, Engineering and Computer Science

Research Methods in Data Science

7PAM2015-0509-2024

Department of Physics, Astronomy and Mathematics

Assignment Title:

Image Segmentation

Student Name and SRN:

Hari Bahadur Gharti Magar (22075765)

GitHub address: https://github.com/harimagar/Research-Methods-in-Data-Science-Image-segmentation

Kaggle Colab Link: https://www.kaggle.com/code/harighartimagar/image-segmentation

Google Colab Link: https://colab.research.google.com/drive/1q-MItcd4lrOIHDsJ_c8xggaKSpmiqwN4?usp=sharing

Introduction

Image segmentation is the field of computer vision that breaks images into several segments to minimize the complexity of the image and allow further processing (Yu et al., 2023). It uses a pixel-by-pixel approach that seeks segmentation masks, making it easy to analyze the images. Instance segmentation, which is a type of image segmentation, is used in this assignment. Instance segmentation sorts each pixel of an image into a category and also differentiates between individual objects of the same class. For image segmentation, a subset of the COCO-2017 dataset is used, which is a popular dataset for instance segmentation due to its diverse content, large-scale annotations and complexity (Pan et al., 2024). It is applicable and best suited for many modern technologies and innovations, including autonomous driving, smart surveillance, augmented reality, robotics and medical imaging.

The method that is being used in this assignment is Mask R-CNN. Mask R-CNN is built by extending Faster R-CNN, which also predicts object masks in parallel (He et al., 2018). Its advantage is that it is easier to train and implement, and does not add much computational overhead. Therefore, it achieves state-of-the-art results on the COCO dataset and has many adaptations in the research field and industry (Wu et al. 2020).

The data collection and processing is a tedious task. Creating a high-quality dataset requires higher skill and human effort. Further, inconsistent object scales, noisy labels and other factors are important to consider while processing the data before performing the analysis.

Thus, the assignment focuses on applying instance segmentation techniques using the Mask R-CNN model to a subset of the COCO-2017 dataset. The EDA (Exploratory Data Analysis) is performed to analyze the dataset and identify data quality issues. Only four classes, cake, car, dog and person are used to model training, validation and testing purposes.

Data and EDA (Exploratory Data Analysis)

As mentioned earlier, a subset of the COCO-2017 dataset is used in this assignment. The folder structure of the dataset with three different folders is shown below:

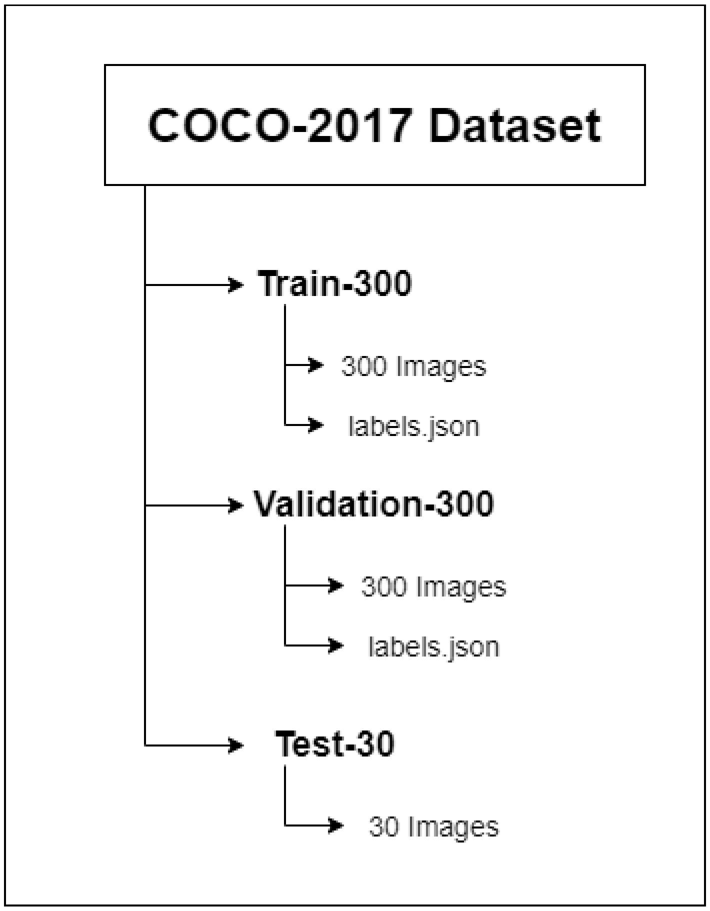


Figure 1 Folder Structure

The JSON file contains image information along with segmentation masks and object bounding boxes for both the train and validation datasets. The dataset is focused on only four target classes: cake, car, dog and person. The instances of these target classes in train dataset is shown by the image below:

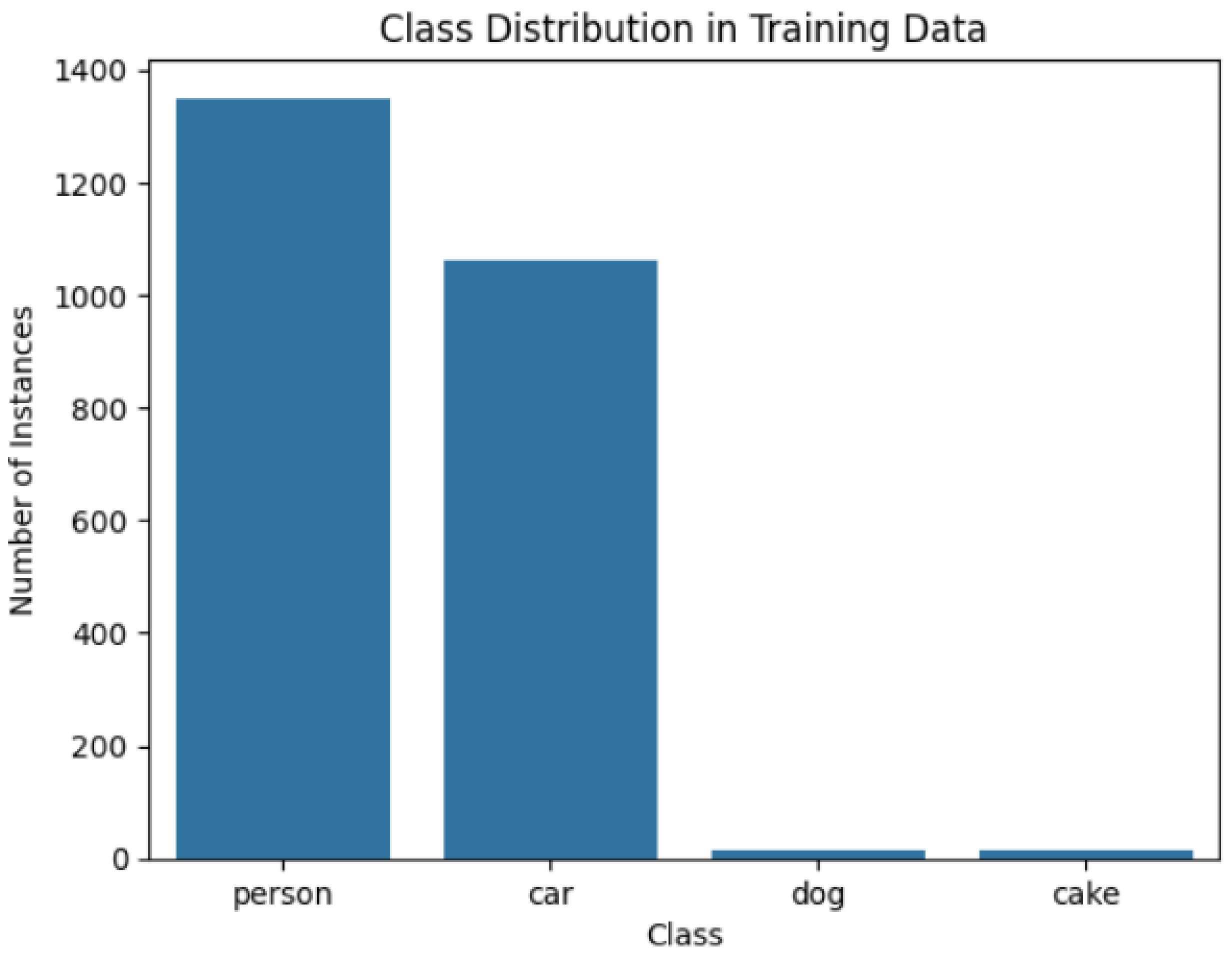


Figure 2 Instances of Target Classes

The diagram shows the class imbalances where instances of person and car are much higher than those of dog and cake. This might have effects on the model training and evaluation.

The variation in image dimensions is shown by the diagrams below:

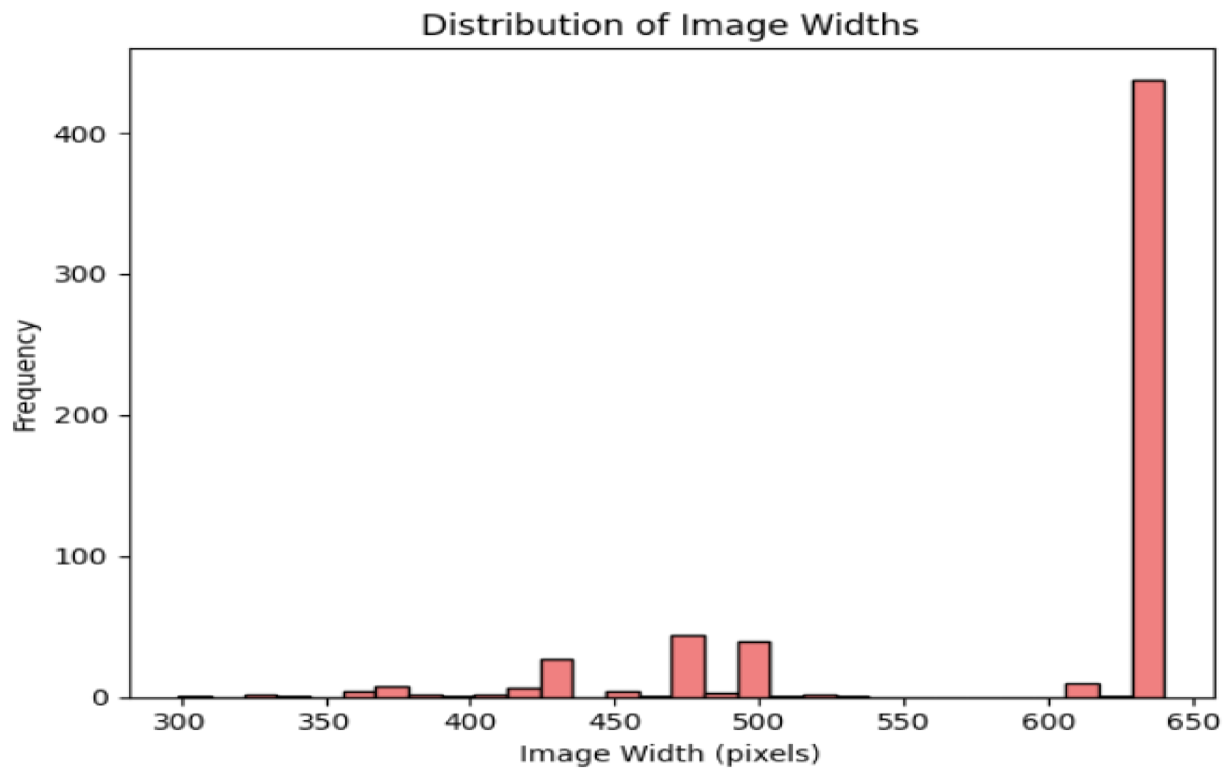


Figure 3 Image Widths Distribution

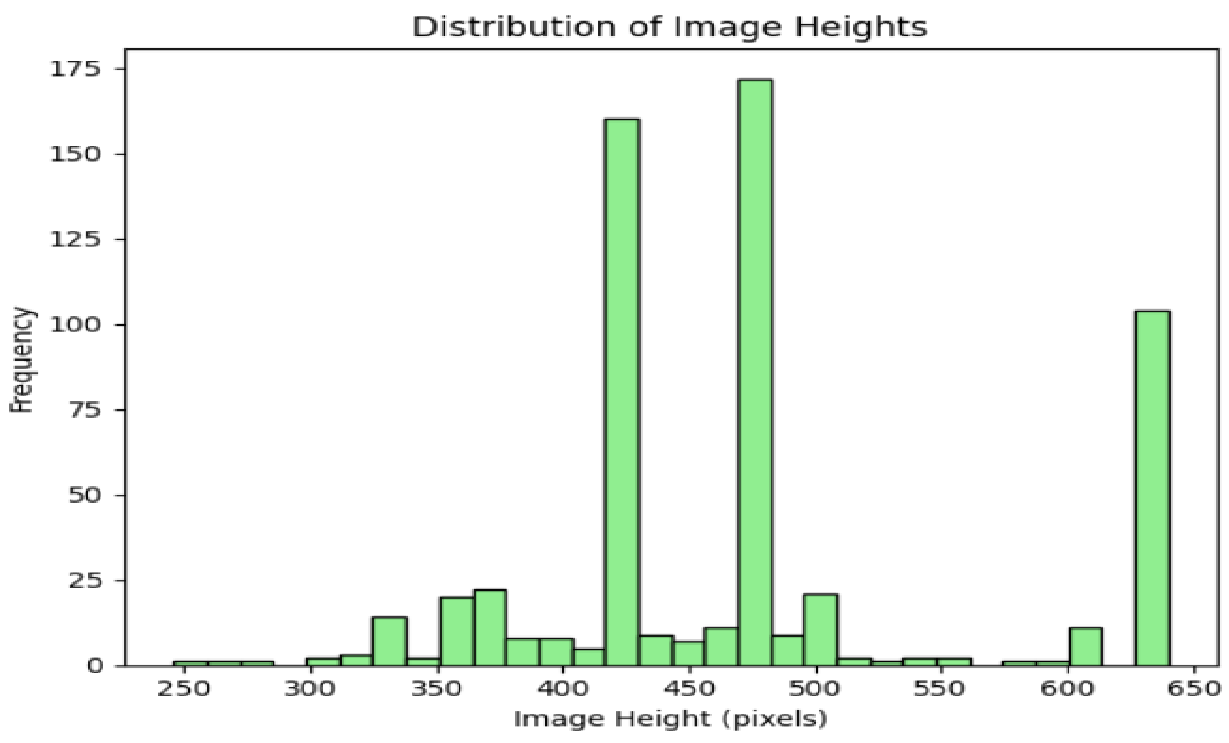


Figure 4 Image Heights Distribution

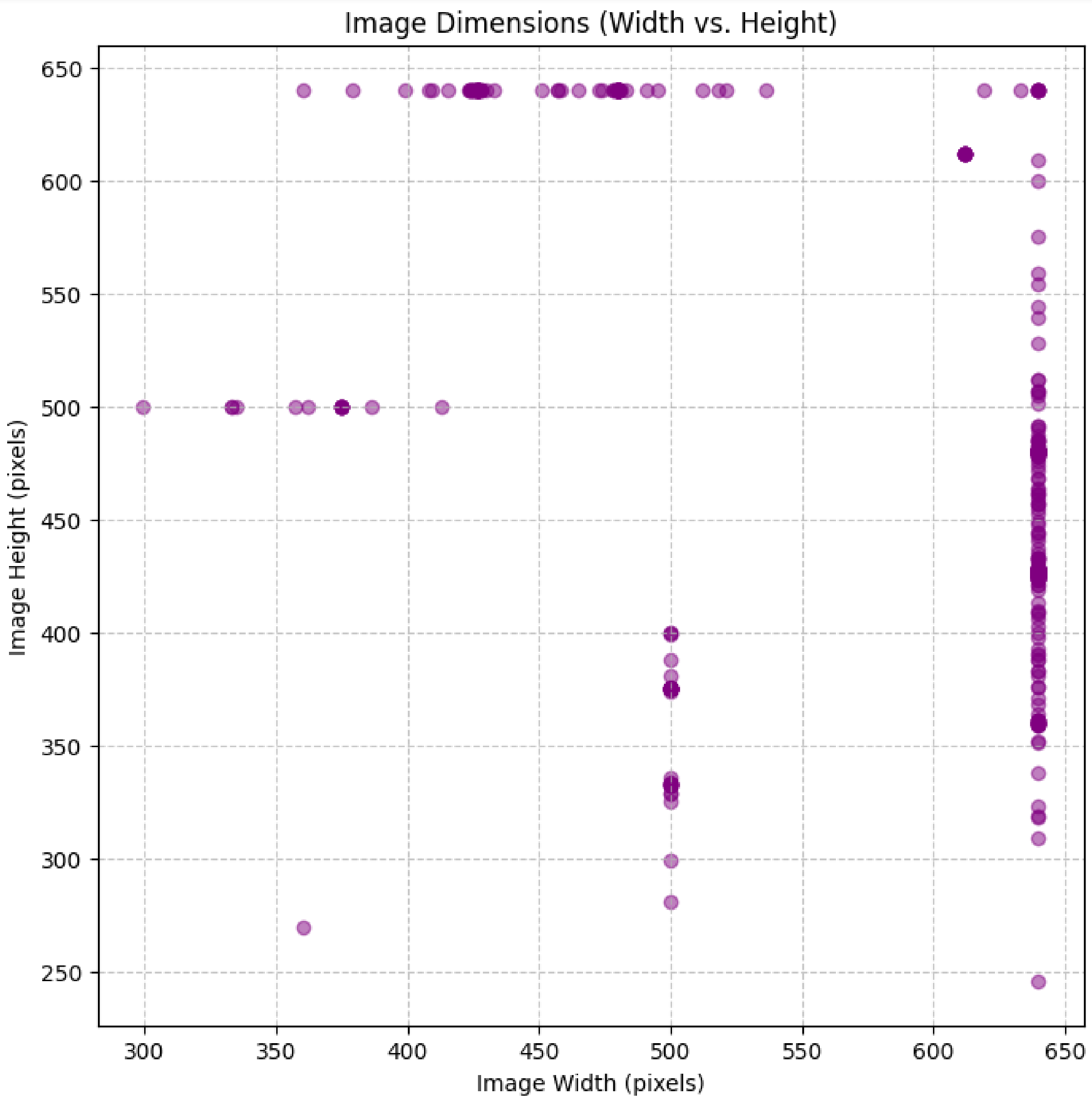
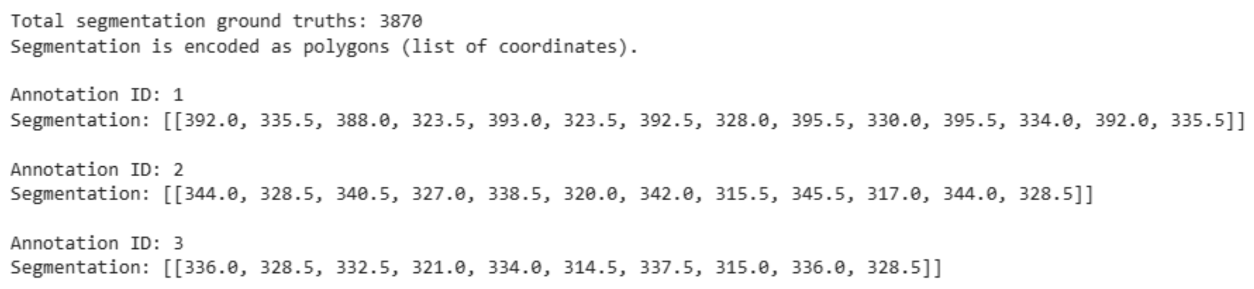


Figure 5 Image Dimensions

The above images show that there is a diverse range of aspect ratios of images with a wide distribution in width and height. These dimensions are analyzed to perform an image resizing technique for better model training and evaluation.

The polygons created for segmentation ground truth encoding are shown below:



```
Total segmentation ground truths: 3870
Segmentation is encoded as polygons (list of coordinates).

Annotation ID: 1
Segmentation: [[392.0, 335.5, 388.0, 323.5, 393.0, 323.5, 392.5, 328.0, 395.5, 330.0, 395.5, 334.0, 392.0, 335.5]]

Annotation ID: 2
Segmentation: [[344.0, 328.5, 340.5, 327.0, 338.5, 320.0, 342.0, 315.5, 345.5, 317.0, 344.0, 328.5]]

Annotation ID: 3
Segmentation: [[336.0, 328.5, 332.5, 321.0, 334.0, 314.5, 337.5, 315.0, 336.0, 328.5]]
```

Figure 6 Segmentation Encoding

Further, data preprocessing is done by removing duplicate images, labelling categories, registering the dataset with Detectron2, and more.

Methodology

The methodology employed in the analysis is Mask R-CNN, a deep learning model. It is a supervised learning algorithm that requires labelled training data provided via the labels.json file. It is implemented using Detectron2, which offers a flexible approach to training segmentation models. The ResNet-FPN serves as the backbone network for detecting objects of varying sizes. Mask R-CNN is implemented in this assignment because it provides state-of-the-art performance, which provides high accuracy on instance segmentation. Detectron2 is implemented as it offers a robust implementation of the Mask R-CNN model and is flexible (C and P, 2023).

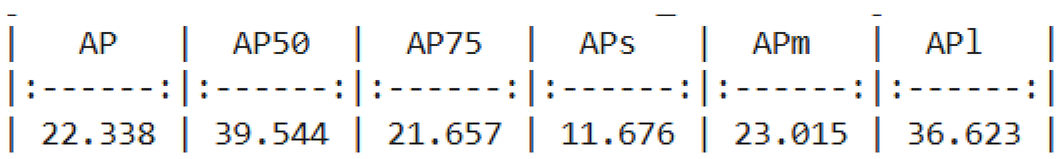
The model is implemented by training the Mask R-CNN using the COCO-2017 dataset that utilizes transfer learning. The Detectron2 model is applied, and the dataset is registered using the register_coco_instances() method. A customized training configuration is provided trainer.train() is used for the fine-tuning process. The CUDA or GPU is used for faster training of the model, and when it is not available CPU is used. The COCOEvaluator is implemented for the evaluation of the validation dataset. Finally, using three images from the test dataset, the model results are tested.

Various challenges are faced during method implementation, including the need for a GPU as training on CPU was slow, the need for further computational resources, limited training images resulting in overfitting and a need for data augmentation and class imbalance.

The metric used to assess the model is Average Precision (AP). It is the average of the maximum precision at different recall levels across multiple IoU (Intersection over Union) thresholds (Bharati and Pramanik, 2019). Further, Ap50, Ap75, APs, APm and APl are also used.

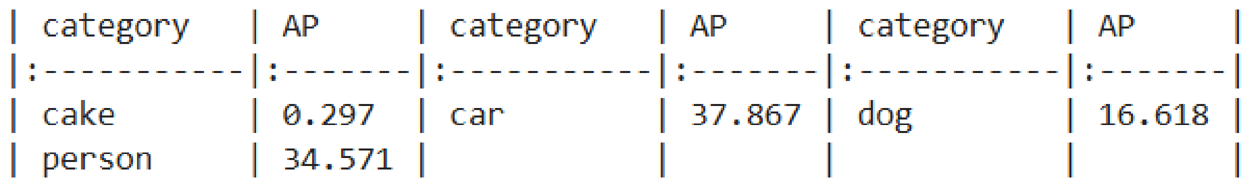
Results

Despite having limited training data, the Mask R-CNN model showed competitive performance. The results of the evaluation is shown below:



```
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 22.338 | 39.544 | 21.657 | 11.676 | 23.015 | 36.623 |
```

Figure 7 Evaluation Results



```
| category   | AP     | category   | AP     | category   | AP     |
|:-----------|:-------|:-----------|:-------|:-----------|:-------|
| cake       | 0.297  | car        | 37.867 | dog        | 16.618 |
| person     | 34.571 |            |        |            |        |
```

Figure 8 AP of the 4 Categories

The evaluation results show that the model performed well, as it has 22.338 AP and 39.544 AP50 and can accurately locate and segment objects. However, lower AP75 suggests the pixel-level mask precision needs improvement. The model can accurately segment a cat, a dog and a person. However, the AP of cake is just 0.297, which suggests the model struggles with cake segmentation, and there is a need for more labelled training data for cake.

The result of three test images are:

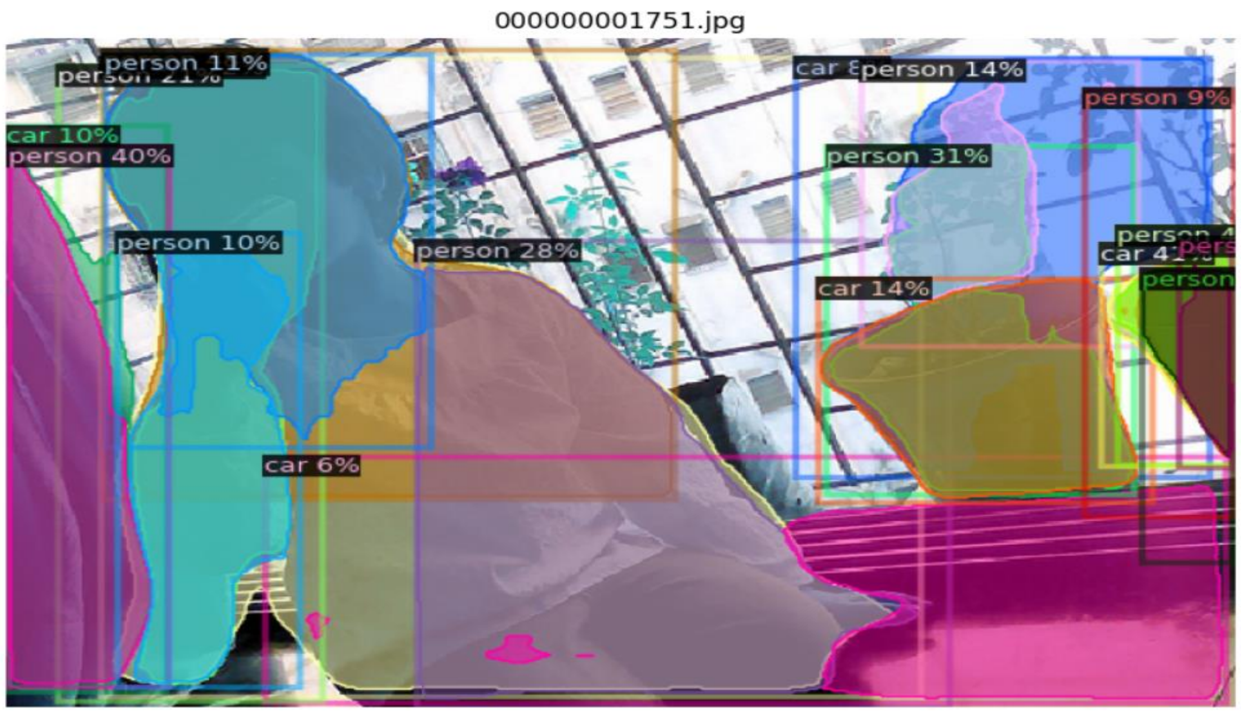


Figure 9 Test-Image-1

000000001650.jpg

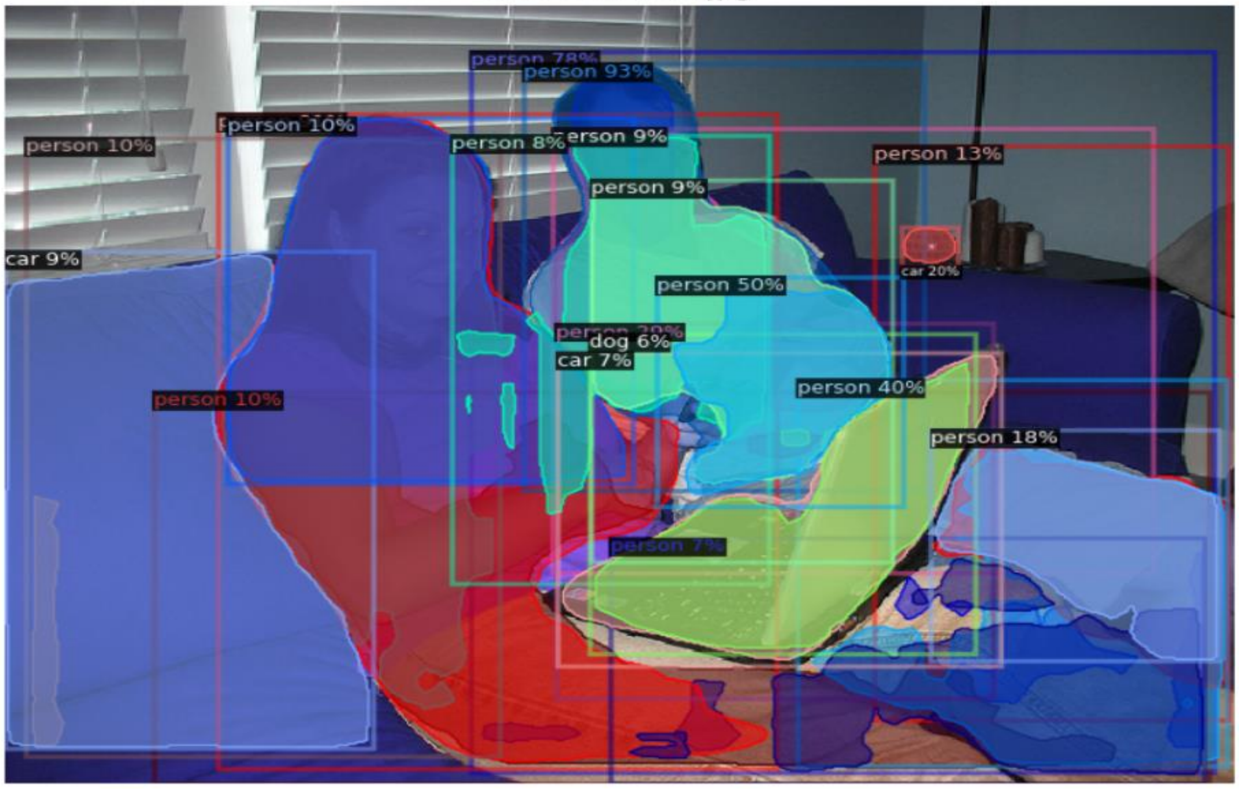


Figure 10 Test-Image-2

000000001439.jpg

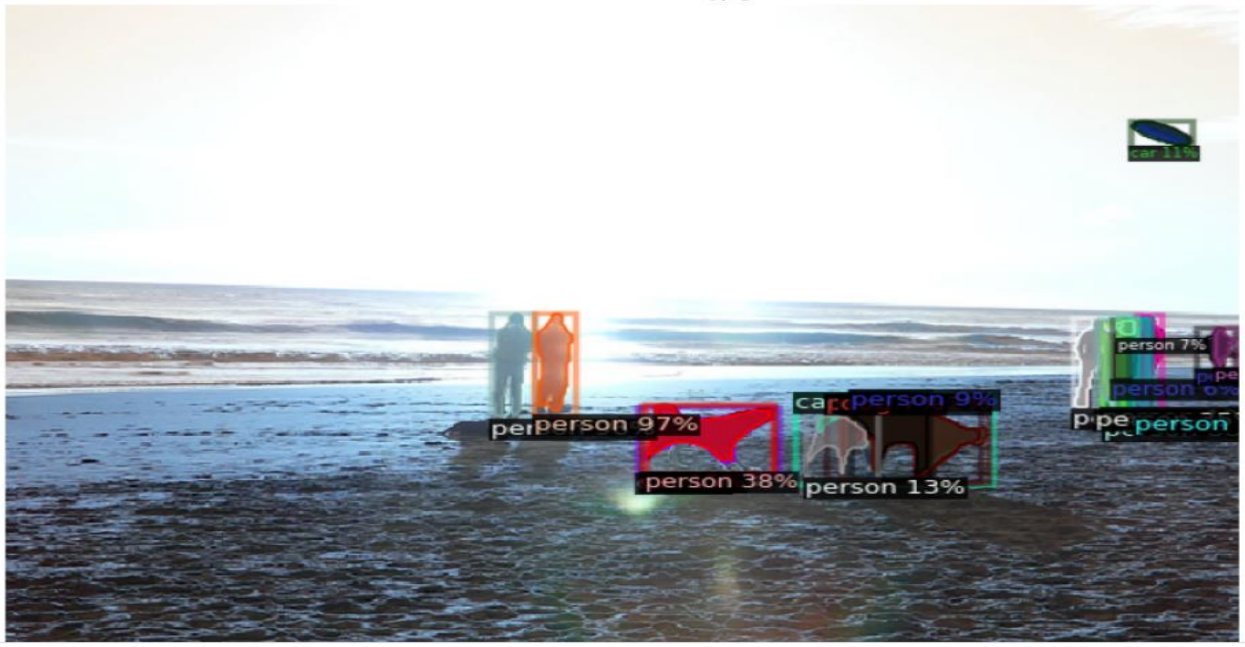


Figure 11 Test-Image-3

The above results show that it can detect objects in the images. The person, dog and car are detected well, but not the cake.

Compared to the literature review, the Mask R-CNN model detected segmentation of all four target classes well, and the AP, which is 30s and higher, are similar to that of the literature mentioned. However, for better results, a bigger dataset with correctly labelled data is required (Tong and Wu, 2023).

References

Bharati, P., & Pramanik, A. (2019). Deep Learning Techniques—R-CNN to Mask R-CNN: A survey. In Advances in intelligent systems and computing (pp. 657–668). https://doi.org/10.1007/978-981-13-9042-5_56

C, A., V., & P, K. (2023). Deep Learning-Based Instance Segmentation of Aircraft in Aerial Images using Detectron2. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.4485468

He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2018). Mask R-CNN. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(2), 386–397. https://doi.org/10.1109/tpami.2018.2844175

Pan, H., Guan, S., & Zhao, X. (2024). LVD-YOLO: An efficient lightweight vehicle detection model for intelligent transportation systems. Image and Vision Computing, 151, 105276. https://doi.org/10.1016/j.imavis.2024.105276

Tong, K., & Wu, Y. (2023). Rethinking PASCAL-VOC and MS-COCO dataset for small object detection. Journal of Visual Communication and Image Representation, 93, 103830. https://doi.org/10.1016/j.jvcir.2023.103830

Wu, M., Yue, H., Wang, J., Huang, Y., Liu, M., Jiang, Y., Ke, C., & Zeng, C. (2020). Object detection based on RGC mask R-CNN. IET Image Processing, 14(8), 1502–1508. https://doi.org/10.1049/iet-ipr.2019.0057

Yu, Y., Wang, C., Fu, Q., Kou, R., Huang, F., Yang, B., Yang, T., & Gao, M. (2023). Techniques and Challenges of Image Segmentation: A review. Electronics, 12(5), 1199. https://doi.org/10.3390/electronics12051199

Appendix

Environment Setup and Data Loading and Exploration

```
# Importing relevant modules.
import random
import numpy as np
import matplotlib.pyplot as plt
# official COCO dataset Python API
from pycocotools.coco import COCO
# for reading and writing images
import skimage.io as io

%matplotlib inline
```

```
# # Import and mount Google Drive.
from google.colab import drive
drive.mount("/content/drive")
# # Unzip the dataset file and store it in a folder called images.
# !unzip "/content/drive/MyDrive/Colab
Notebooks/coco2017/RM_Segmentation_Assignment_dataset.zip" -d
"/content/drive/MyDrive/Colab Notebooks/coco2017/"
```

```
# Definining location of training data.
train_data_path = "/content/drive/MyDrive/Colab Notebooks/coco2017/train-
300"
train_annotation_file = f"{train_data_path}/labels.json"
```

```
# Definining location of validation data.
validation_data_path = "/content/drive/MyDrive/Colab
Notebooks/coco2017/validation-300"
validation_annotation_file = f"{validation_data_path}/labels.json"
coco_validation = COCO(validation_annotation_file)
```

```
# Initialising COCO API for instance annotations.
coco = COCO(train_annotation_file)
```

Dataset Analysis

```
# Extract and display COCO categories and supercategories for the training
dataset.
category_IDs = coco.getCatIds()
categories = coco.loadCats(category_IDs)
```

```
print(categories)

names_cats = [cats["name"] for cats in categories]
print(len(names_cats), "COCO categories:", " ".join(names_cats))

names_scats = set([cats["supercategory"] for cats in categories])
print(len(names_scats), "COCO supercategories:", " ".join(names_scats))
```

```
# Define function to extract category name from ID.
def get_category_name(class_ID, categories):
    for i in range(len(categories)):
        if categories[i]["id"] == class_ID:
            return categories[i]["name"]
    return "None"
```

```
# examples of categories
category_name_41 = get_category_name(41, categories)
print(f"The category name is {category_name_41}.")
category_name_15 = get_category_name(15, categories)
print(f"The category name is {category_name_15}.")
```

Filtering and Displaying Example Images

```
# Get all training images containing a given object category or
categories.
filter_class = ["cat"]
category_IDs = coco.getCatIds(catNms=filter_class)
image_IDs = coco.getImgIds(catIds=category_IDs)

print(f"Number of images containing specified category(ies):
{len(image_IDs)}.")
print(f"IDs of images containing specified category(ies): {image_IDs}.")
```

```
# Load and display one of the example images.
example_image = coco.loadImgs(image_IDs[0])[0]
print(example_image)

image = io.imread(f'{train_data_path}/data/{example_image["file_name"]}')
plt.axis("off")
plt.imshow(image)

plt.show()
```

```python
# Get COCO annotation IDs and content of annotations, i.e., bounding boxes
and segmentation masks.
test_image_annotations_ID = coco.getAnnIds(
    imgIds=example_image["id"], catIds=category_IDs, iscrowd=None
)
print(test_image_annotations_ID)

test_image_annotations = coco.loadAnns(test_image_annotations_ID)
print(test_image_annotations)
```

```python
# Load and display test image with instance annotations.
plt.imshow(image)
plt.axis("off")

coco.showAnns(test_image_annotations, draw_bbox=True)
```

```python
# Extract the training images that contain any combination of the four
target classes.
target_classes = ["cake", "car", "dog", "person"]
target_classes_IDs = coco.getCatIds(catNms=target_classes)
print("Target class IDs:", target_classes_IDs)
training_images = []

# Iterate over each individual class in the list.
for class_name in target_classes:
    # Get all images containing target class.
    print(class_name)
    training_images_categories = coco.getCatIds(catNms=class_name)
    training_images_IDs =
coco.getImgIds(catIds=training_images_categories)
    training_images += coco.loadImgs(training_images_IDs)

print(
    f"Number of images with target classes including repetitions:
{len(training_images)}."
)
```

```python
# Filter out repeated images.
unique_training_images = []

for i in range(len(training_images)):
    if training_images[i] not in unique_training_images:
        unique_training_images.append(training_images[i])

# Shuffle the training data.
random.seed(0)
random.shuffle(unique_training_images)

print(
    f"Number of unique images in training data containing the target
classes: {len(unique_training_images)}"
)
```

```
# Load and display example training image with segmentation masks.
training_image = unique_training_images[10]
print(training_image)

image = io.imread(f'{train_data_path}/data/{training_image["file_name"]}')
plt.axis("off")
plt.imshow(image)

training_image_annotations_ID = coco.getAnnIds(
    imgIds=training_image["id"], catIds=target_classes_IDs, iscrowd=None
)
training_image_annotations = coco.loadAnns(training_image_annotations_ID)
coco.showAnns(training_image_annotations, draw_bbox=False)

plt.show()
```

```
# Generating and Plotting Segmentation Masks
mask_example = coco.annToMask(training_image_annotations[0])

print(type(mask_example))
print(mask_example)
print(mask_example.shape)
print(np.max(mask_example))
print(np.min(mask_example))
```

```
# Plotting the segmentation masks with different colours.
mask = np.zeros((training_image["height"], training_image["width"]))

for i in range(len(training_image_annotations)):
    # Get object category name.
    object_category = get_category_name(
        training_image_annotations[i]["category_id"], categories
    )
    # Assign pixel value based on location in target_classes list.
    pixel_value = target_classes.index(object_category) + 1
    # Assign pixel value to mask based on annToMask output.
    mask = np.maximum(coco.annToMask(training_image_annotations[i]) * 3,
mask)

print(f"Unique pixel values in the mask: {np.unique(mask)}")

plt.imshow(mask)
plt.show()
```

```
# How many images are in our training/validation/test datasets?
# How many images are in the downloaded data folders?
import os

def count_images_in_folder(folder_path):
    return len([f for f in os.listdir(folder_path) if
f.lower().endswith((".jpg", ".jpeg", ".png"))])

train_image_count = count_images_in_folder("/content/drive/MyDrive/Colab
Notebooks/coco2017/train-300/data")
val_image_count = count_images_in_folder("/content/drive/MyDrive/Colab
Notebooks/coco2017/validation-300/data")
test_image_count = count_images_in_folder("/content/drive/MyDrive/Colab
Notebooks/coco2017/test-30")

print(f"Train Images: {train_image_count}")
print(f"Validation Images: {val_image_count}")
print(f"Test Images: {test_image_count}")
print(f"Total Images in downloaded data folders: {train_image_count +
val_image_count + test_image_count}")
```

Data Statistics and Visualization

```
# Load annotations
import json
with open("/content/drive/MyDrive/Colab Notebooks/coco2017/train-
300/labels.json") as f:
    coco_json = json.load(f)

# Count category occurrences
from collections import Counter
target_ids = [15, 16, 25, 41]
```

```python
counter = Counter()

for ann in coco_json["annotations"]:
    if ann["category_id"] in target_ids:
        counter[ann["category_id"]] += 1

print("Instances of each target class in train-300:")
for cid in target_ids:
    print(f"Class ID {cid}: {counter[cid]} instances")

anns = coco.loadAnns(coco.getAnnIds(catIds=target_classes_IDs))

from collections import Counter
label_names = [coco.loadCats(ann["category_id"])[0]["name"] for ann in
anns]
label_counts = Counter(label_names)

import seaborn as sns
sns.barplot(x=list(label_counts.keys()), y=list(label_counts.values()))
plt.title("Class Distribution in Training Data")
plt.ylabel("Number of Instances")
plt.xlabel("Class")
plt.show()
```

```python
# How often does each COCO/target class appear?
from collections import defaultdict

# Define your target classes
target_classes = ["cake", "car", "dog", "person"]

# Get COCO category IDs for those target classes
target_category_ids = coco.getCatIds(catNms=target_classes)

# Create a mapping from category ID to name
category_id_to_name = {cat["id"]: cat["name"] for cat in
coco.loadCats(target_category_ids)}

# Count instances
target_category_counts = defaultdict(int)

for ann in coco.dataset["annotations"]:
    cat_id = ann["category_id"]
    if cat_id in target_category_ids:
        target_category_counts[category_id_to_name[cat_id]] += 1
```

```
# Print result
print("Target class frequencies in train-300:")
for category in target_classes:
    print(f"{category}: {target_category_counts[category]}")
```

```
# How large are the images? What is their distribution?
# Function to collect image sizes
def collect_image_sizes(coco_obj):
    img_ids = coco_obj.getImgIds()
    images_info = coco_obj.loadImgs(img_ids)

    widths = []
    heights = []
    for img_info in images_info:
        widths.append(img_info['width'])
        heights.append(img_info['height'])
    return widths, heights

# Collect sizes for training and validation
train_widths, train_heights = collect_image_sizes(coco)
val_widths, val_heights = collect_image_sizes(coco_validation)

all_widths = train_widths + val_widths
all_heights = train_heights + val_heights

print(f"\n--- Image Size Distribution (Train & Val) ---")
print(f"Total images analyzed for size: {len(all_widths)}")
print(f"Min Width: {min(all_widths)}, Max Width: {max(all_widths)}, Avg
Width: {np.mean(all_widths):.2f}")
print(f"Min Height: {min(all_heights)}, Max Height: {max(all_heights)},
Avg Height: {np.mean(all_heights):.2f}")

# Plotting histograms for width and height
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.hist(all_widths, bins=30, color='lightcoral', edgecolor='black')
plt.xlabel("Image Width (pixels)")
plt.ylabel("Frequency")
plt.title("Distribution of Image Widths")

plt.subplot(1, 2, 2)
plt.hist(all_heights, bins=30, color='lightgreen', edgecolor='black')
plt.xlabel("Image Height (pixels)")
plt.ylabel("Frequency")
plt.title("Distribution of Image Heights")

plt.tight_layout()
plt.show()

# Plotting scatter plot for width vs height
plt.figure(figsize=(8, 8))
plt.scatter(all_widths, all_heights, alpha=0.5, color='purple')
plt.xlabel("Image Width (pixels)")
plt.ylabel("Image Height (pixels)")
plt.title("Image Dimensions (Width vs. Height)")
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

```
# How many segmentation ground truths are given? How are these encoded?
annotations = coco.dataset["annotations"]
total_segmentations = len(annotations)
print(f"Total segmentation ground truths: {total_segmentations}")

# Check format of segmentation
example_seg = annotations[0]["segmentation"]

if isinstance(example_seg, list):
    print("Segmentation is encoded as polygons (list of coordinates).")
elif isinstance(example_seg, dict) and "counts" in example_seg:
    print("Segmentation is RLE encoded (Run Length Encoding).")
else:
    print("Unknown segmentation format.")

# Show some example segmentations
for i in range(3):
    print(f"\nAnnotation ID: {annotations[i]['id']}")
    print(f"Segmentation: {annotations[i]['segmentation']}")
```

```
import torch
import torchvision
import torchaudio

print("Torch version:", torch.__version__)
print("Torchvision version:", torchvision.__version__)
print("Torchaudio version:", torchaudio.__version__)
```

```
# Checking CUDA version compatibility
print("CUDA available:", torch.cuda.is_available())
print("Torch CUDA version:", torch.version.cuda)
```

Data Preprocessing

```
# filter dataset (only 4 classes)
def filter_coco_by_classes(json_file, classes_to_keep):
    with open(json_file) as f:
        data = json.load(f)

    class_name_to_id = {cat["name"]: cat["id"] for cat in
data["categories"]}
    keep_ids = set(class_name_to_id[name] for name in classes_to_keep)

    filtered_anns = [a for a in data["annotations"] if a["category_id"] in
keep_ids]
    image_ids = set([a["image_id"] for a in filtered_anns])
    filtered_imgs = [i for i in data["images"] if i["id"] in image_ids]
    filtered_cats = [c for c in data["categories"] if c["id"] in keep_ids]

    filtered = {
        "images": filtered_imgs,
        "annotations": filtered_anns,
        "categories": filtered_cats
    }
    return filtered

input_path = f"/content/drive/MyDrive/Colab Notebooks/coco2017/train-
300/labels.json"
output_path = f"/content/drive/MyDrive/Colab
Notebooks/coco2017/filtered_labels_train.json"

filtered = filter_coco_by_classes(input_path, target_classes)
with open(output_path, "w") as f:
    json.dump(filtered, f)

input_path2 = f"/content/drive/MyDrive/Colab
Notebooks/coco2017/validation-300/labels.json"
output_path2 = f"/content/drive/MyDrive/Colab
Notebooks/coco2017/filtered_labels_validation.json"

filtered2 = filter_coco_by_classes(input_path2, target_classes)
with open(output_path2, "w") as f:
    json.dump(filtered2, f)
```

```
!pip install 'git+https://github.com/facebookresearch/detectron2.git'
!pip install pycocotools opencv-python matplotlib seaborn albumentations

import torch, torchvision
import numpy as np
import os, json, random
import cv2
import matplotlib.pyplot as plt
from pycocotools.coco import COCO
```

Model Training

```
# model training using Detectron2
from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg
from detectron2.data.datasets import register_coco_instances
from detectron2.utils.logger import setup_logger
setup_logger()

# Register dataset
register_coco_instances("coco_train2", {}, "/content/drive/MyDrive/Colab
Notebooks/coco2017/filtered_labels_train.json",
"/content/drive/MyDrive/Colab Notebooks/coco2017/train-300/data")
register_coco_instances("coco_val2", {}, "/content/drive/MyDrive/Colab
Notebooks/coco2017/filtered_labels_validation.json",
"/content/drive/MyDrive/Colab Notebooks/coco2017/validation-300/data")
```

```
from detectron2.config import get_cfg
from detectron2 import model_zoo
import torch

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))

cfg.DATASETS.TRAIN = ("coco_train2",)
cfg.DATASETS.TEST = ("coco_val2",)
cfg.DATALOADER.NUM_WORKERS = 2
cfg.OUTPUT_DIR = "/content/output"
cfg.MODEL.WEIGHTS = "detectron2://COCO-
InstanceSegmentation/mask_rcnn_R_50_FPN_3x/137849600/model_final_f10217.pk
l"
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 1000
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 4

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)

if torch.cuda.is_available():
    print("CUDA available.")
    cfg.MODEL.DEVICE = "cuda"
else:
    print("CUDA not available. Training will run on CPU.")
    cfg.MODEL.DEVICE = "cpu"

from detectron2.engine import DefaultTrainer

trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

Evaluation and Testing

```
# Evaluation on Validation set
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader

evaluator = COCOEvaluator("coco_val2", output_dir="./output")
val_loader = build_detection_test_loader(cfg, "coco_val2")
print(inference_on_dataset(trainer.model, val_loader, evaluator))
```

```
from detectron2.engine import DefaultPredictor
from detectron2.utils.visualizer import Visualizer, ColorMode
from detectron2.data import MetadataCatalog
import random
import os
import cv2
import matplotlib.pyplot as plt

# Set weights and create predictor
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
predictor = DefaultPredictor(cfg)

test_images_path = "/content/drive/MyDrive/Colab Notebooks/coco2017/test-
30"
```

```
test_images = [f for f in os.listdir(test_images_path) if
f.endswith(".jpg") or f.endswith(".png")]

# Get metadata for dataset (this contains category names)
metadata = MetadataCatalog.get(cfg.DATASETS.TEST[0])

for img_name in random.sample(test_images, 3):
    path = os.path.join(test_images_path, img_name)
    img = cv2.imread(path)
    outputs = predictor(img)

    v = Visualizer(img[:, :, ::-1], metadata=metadata, scale=1.2,
instance_mode=ColorMode.SEGMENTATION)
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))

    plt.figure(figsize=(10, 7))
    plt.imshow(out.get_image()[:, :, ::-1])
    plt.title(img_name)
    plt.axis("off")
    plt.show()
```