**Image Segmentation on COCO Subset Using Mask R-CNN and Detectron2**

1. **Introduction**

Image segmentation is essential in computer vision for precise object localization, with each pixel assigned a class label (Minaee et al., 2020). Within this field, instance segmentation allows the detection and labeling of individual objects of the same class, which is valuable for applications in autonomous driving, robotics, and medical imaging (Yin et al., 2021). This project applied instance segmentation on the COCO-2017 dataset, focusing on four classes: cake, car, dog, and person. Advances in deep learning, including the development of Mask R-CNN (He et al., 2017) and the PyTorch-based Detectron2 framework by Facebook AI (Meta, 2019), have improved instance segmentation accuracy and efficiency, although challenges in data and resource demands persist (Yin et al., 2021).

1. **Data and Exploratory Data AnalysiSs (EDA)**

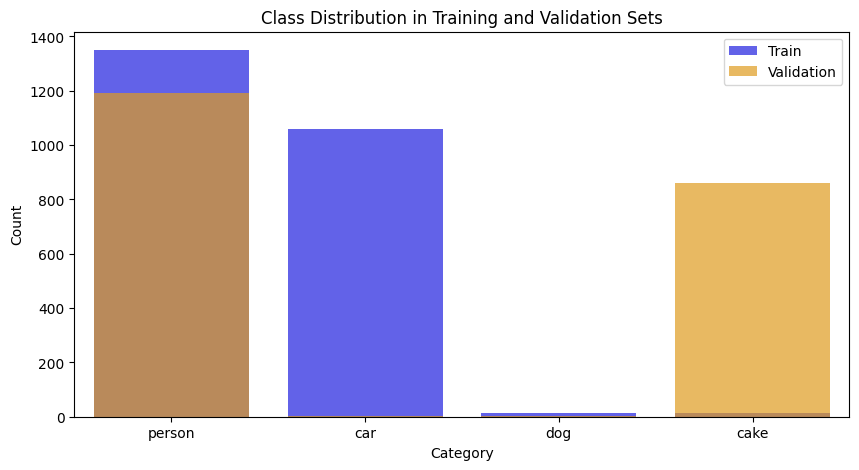
**Dataset Overview**

The dataset consists of 300 training images, 300 validation images, and 30 test images taken from the COCO-2017 dataset. Each image has COCO annotations such as item class names, bounding boxes, and segmentation masks. However, for this project, just four object classes, Cake, Car, Dog, and Person, were used, with matching category IDs 15, 16, 25, and 41.

**Exploratory Data Analysis**

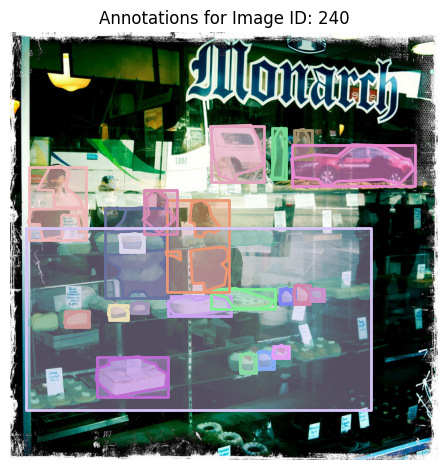
A deeper understanding of the dataset was needed to identify potential challenges, and also guide decisions for model training and evaluation. EDA was conducted as follows:

1. **Class Distribution**: Histograms of the distribution of the four selected classes showed a balanced representation, with enough samples in each class for training. Understanding this distribution was critical in evaluating whether any classes needed additional data to balance the dataset.



Class Distribution in Training and Validation Sets

1. **Image and Mask Visualization**: Random samples were assessed with segmentation masks superimposed on the original images to ensure annotation quality and mask correctness. This phase helped confirm that each class had well-defined masks and that some photos included occlusions that might impair the model's performance.



An image showing the Segmentation Masks

**3. Methodology**

**Model Selection and Transfer Learning**

Given the dataset's small size, creating a segmentation model from start would have been unfeasible. Transfer learning provides an effective solution by applying a pre-trained model's learnt features to a big dataset, in this instance COCO. This method saves training time while improving generalisation. For this purpose, Detectron2's pre-trained Mask R-CNN model with a ResNet-50 backbone was chosen, since it provides a strong basis for instance segmentation while also allowing for fine-tuning on a smaller dataset.

**Detectron2 and Mask R-CNN Architecture**

The Mask R-CNN architecture in Detectron2 involves several stages optimized for instance segmentation:

1. The ResNet-50 backbone, combined with a Feature Pyramid Network (FPN), extracts hierarchical features at numerous scales. This helps partition objects of different sizes and handle occlusions.
2. The Region Proposal Network (RPN) creates suggestions for regions that are likely to contain objects. These approaches enable the model to focus exclusively on key portions of the picture, hence increasing computing efficiency.
3. The Region of Interest (RoI) Align layer effectively aligns each proposal with the feature map. The model then predicts bounding boxes, class labels, and masks for each identified item, discriminating between various instances of the same class.

**Data Augmentation**

To enhance model generalization, several data augmentation techniques were employed:

1. Resizing to standardize image dimensions.
2. Horizontal flipping to increase variability in object orientations.
3. Adjust the brightness to imitate different lighting conditions. These augmentations helped alleviate the dataset's size limits by exposing the model to a broader range of item appearances.

**Training Procedure**

The model was trained for 300 iterations, and the loss gradually decreased, indicating convergence. Training measures such as classification loss, bounding box regression loss, and mask loss were used to assess performance progress.



An Image showing prediction of the Masks on a Train Image with an Accuracy score of 94%

1. **Results and Discussion**

**Training Metrics**

The training metrics provide insights into model performance and optimization:

| **Iteration** | **Total Loss** | **Classification Loss** | **Box Regression Loss** | **Mask Loss** | **RPN Class Loss** | **RPN Loc Loss** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 3.023 | 1.663 | 0.5309 | 0.6911 | 0.04596 | 0.06039 |
| 100 | 1.44 | 0.3419 | 0.4962 | 0.4975 | 0.03119 | 0.07046 |
| 200 | 0.992 | 0.1878 | 0.3253 | 0.3373 | 0.02019 | 0.06065 |
| 300 | 0.8626 | 0.1724 | 0.2705 | 0.3038 | 0.02189 | 0.0973 |

The steady decline in total loss demonstrates effective learning, with classification, bounding box, and mask losses all reducing over time.

**Evaluation Metrics and Test Results**

To assess model performance on the test images, the following were used:

**Mean Average Precision (mAP)**: This metric assesses the model's detection and segmentation accuracy at various IoU levels, giving a comprehensive performance measure.

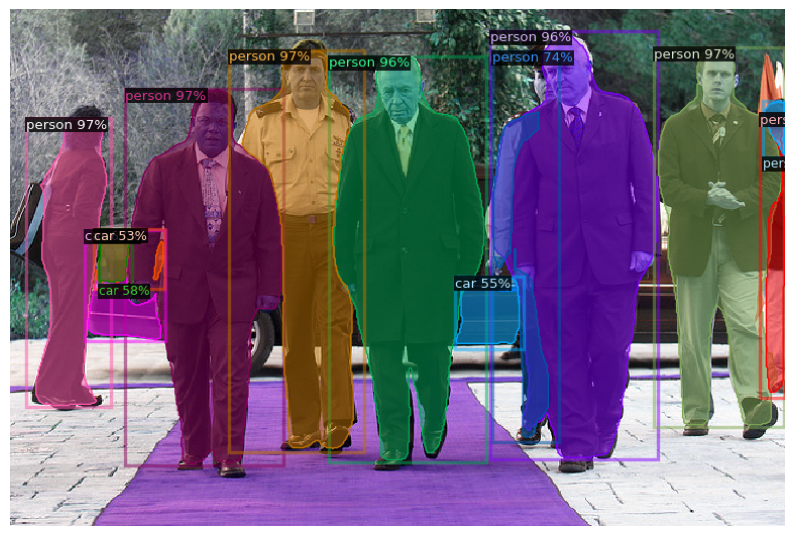
**Precision and Recall**: Precision focusses on correct mask predictions, while recall reflects the model's capacity to recognise all occurrences of an item.

These measures show great accuracy in segmenting unique objects, although recall for smaller items or obscured components (e.g., limbs of the human class) was significantly lower, indicating space for improvement.

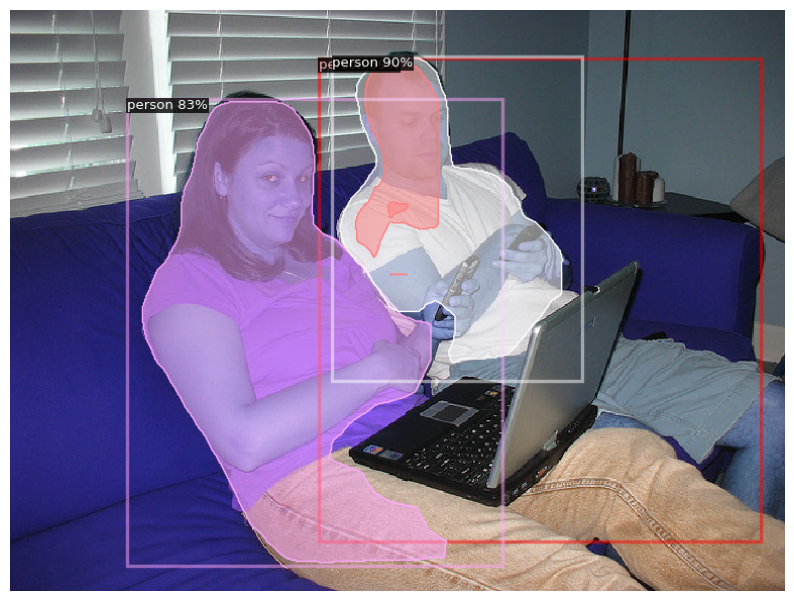
**Predictions on Test Images**



An Image showing predictions on Test Image



An Image showing predictions on Test Image



An Image showing predictions on Test Image

**Discussion of Results**

The model performed well in segmenting clear objects but saw minor IoU reductions in the car and human classes, particularly with occlusions or small object parts. This may stem from limited training exposure to such cases, suggesting that added data augmentation could help. Metrics indicate good generalization, though improvements like increased training iterations, class-specific loss adjustments, or refined RPN tuning may further enhance performance for finer details.

1. **Conclusion**

This project demonstrated effective use of Mask R-CNN with Detectron2 for instance segmentation on a limited subset of COCO-2017. By leveraging transfer learning and selective data augmentation, the model achieved high segmentation accuracy across four classes, highlighting how pre-trained models can reduce data and resource demands. Future improvements could include dataset expansion, parameter tuning, and additional classifications, showcasing Detectron2’s strength in resource-limited computer vision tasks.

**REFERENCES**

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

Meta. (2019, October 10). Detectron2: A PyTorch-based modular object detection library. *Meta*. https://ai.meta.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/

Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., & Terzopoulos, D. (2020). Image Segmentation Using Deep Learning: A Survey. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2001.05566

Long, J., Shelhamer, E., & Darrell, T. (2016). Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(4), 640–651. https://doi.org/10.1109/tpami.2016.2572683

Yin, C., Tang, J., Yuan, T., Xu, Z., & Wang, Y. (2021). Bridging the Gap Between Semantic Segmentation and Instance Segmentation. *IEEE Transactions on Multimedia*, *24*, 4183–4196. https://doi.org/10.1109/tmm.2021.3114541

**Appendix**

**# -\*- coding: utf-8 -\*-**

**"""Image segmentation project.ipynb**

**Original file is located at**

**https://colab.research.google.com/drive/1swJU8gsc3ulkTd9Pt7cScRafCS2AQMLm**

**## \*\*IMAGE SEGMENTATION USING THE COCO-2017 DATASET\*\***

**The \*\*MS COCO (Microsoft Common Objects in Context) dataset\*\* is a large-scale object detection, segmentation, key-point detection, and captioning dataset.**

**\*\*Image segmentation\*\* is a computer vision technique that partitions a digital image into discrete groups of pixels—image segments—to inform object detection and related tasks. By parsing an image's complex visual data into specifically shaped segments, image segmentation enables faster, more advanced image processing.**

**# \*\*Importing Necessary Libraries and Setting up COCO 2017 Dataset from Drive.\*\***

**"""**

**from google.colab import drive**

**drive.mount('/content/drive')**

**import json**

**import os**

**# Load labels**

**with open('/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/train-300/labels.json') as f:**

**train\_labels = json.load(f)**

**with open('/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/validation-300/labels.json') as f:**

**val\_labels = json.load(f)**

**# Example structure inspection**

**print(json.dumps(train\_labels['annotations'][:5], indent=2)) # Display the first 5 annotations**

**"""# \*\*Exploratory Data Analysis (EDA)\*\***

**Class Distribution Analysis**

**Identify the four classes of interest: cake, car, dog, and person. This helps to analyze class distributions in training and validation sets.**

**"""**

**import json**

**train\_data\_dir = '/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/train-300/data'**

**train\_json\_path = os.path.join(train\_data\_dir, 'labels.json')**

**# Path to the COCO 2017 annotations file**

**# Load the annotations**

**with open(train\_json\_path, 'r') as f:**

**data = json.load(f)**

**# Create a mapping from category name to ID**

**category\_mapping = {category['name']: category['id'] for category in data['categories']}**

**# Retrieve IDs for 'dog' and 'cake'**

**dog\_id = category\_mapping.get('dog')**

**cake\_id = category\_mapping.get('cake')**

**person\_id = category\_mapping.get('person')**

**car\_id = category\_mapping.get('car')**

**print(f"Dog Category ID: {dog\_id}")**

**print(f"Cake Category ID: {cake\_id}")**

**print(f"Person Category ID: {person\_id}")**

**print(f"Car Category ID: {car\_id}")**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**from collections import Counter**

**# Extract category information**

**categories = {cat['id']: cat['name'] for cat in train\_labels['categories']}**

**# Filter annotations for the relevant classes**

**relevant\_categories = ["cake", "car", "dog", "person"]**

**relevant\_category\_ids = [k for k, v in categories.items() if v in relevant\_categories]**

**# Count occurrences of each relevant category**

**train\_counts = Counter(ann['category\_id'] for ann in train\_labels['annotations'] if ann['category\_id'] in relevant\_category\_ids)**

**val\_counts = Counter(ann['category\_id'] for ann in val\_labels['annotations'] if ann['category\_id'] in relevant\_category\_ids)**

**# Visualization of class distribution**

**plt.figure(figsize=(10, 5))**

**sns.barplot(x=[categories[cid] for cid in train\_counts.keys()], y=list(train\_counts.values()), color='blue', alpha=0.7, label='Train')**

**sns.barplot(x=[categories[cid] for cid in val\_counts.keys()], y=list(val\_counts.values()), color='orange', alpha=0.7, label='Validation')**

**plt.title("Class Distribution in Training and Validation Sets")**

**plt.xlabel("Category")**

**plt.ylabel("Count")**

**plt.legend()**

**plt.show()**

**"""**

**This code snippet utilizes the COCO API to analyze a custom image segmentation dataset. It loads annotations from JSON files located in Google Drive, specifically focusing on four target categories: "cake", "car", "dog", and "person". The code first visualizes the class distribution in the training and validation sets using bar plots. Then, using the `pycocotools.coco.COCO` class, it initializes COCO objects for both training and validation sets and retrieves the category IDs corresponding to the target categories for later use in the segmentation task.**

**"""**

**from pycocotools.coco import COCO**

**# Load the COCO annotations for training and validation sets**

**train\_annotations\_path = '/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/train-300/labels.json'**

**val\_annotations\_path = '/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/validation-300/labels.json'**

**# Initialize COCO API for instance annotations**

**train\_coco = COCO(train\_annotations\_path)**

**val\_coco = COCO(val\_annotations\_path)**

**# Define the category IDs for 'cake', 'car', 'dog', and 'person'**

**target\_categories = ["cake", "car", "dog", "person"]**

**category\_ids = [cat['id'] for cat in train\_coco.loadCats(train\_coco.getCatIds()) if cat['name'] in target\_categories]**

**print("Selected category IDs:", category\_ids)**

**import cv2**

**import numpy as np**

**def visualize\_segmentation(image\_path, annotations, category\_ids):**

**image = cv2.imread(image\_path)**

**# Check if image was loaded correctly**

**if image is None:**

**print(f"Error: Unable to load image at path {image\_path}")**

**return**

**for ann in annotations:**

**if ann['category\_id'] in category\_ids:**

**mask = np.zeros((image.shape[0], image.shape[1]), dtype=np.uint8)**

**for seg in ann['segmentation']:**

**# Check that segmentation data is a list with an even number of elements**

**if not isinstance(seg, list) or len(seg) < 6 or len(seg) % 2 != 0:**

**print(f"Skipping invalid segmentation data (not a valid polygon): {seg}")**

**continue**

**try:**

**# Reshape into pairs of (x, y) coordinates**

**points = np.array(seg).reshape((-1, 2)).astype(np.int32)**

**cv2.fillPoly(mask, [points], color=255)**

**except ValueError as e:**

**print(f"Error reshaping segmentation points: {seg} -> {e}")**

**continue**

**# Define color based on category**

**color = (0, 255, 0) if categories[ann['category\_id']] == 'dog' else (255, 0, 0)**

**image[mask > 0] = color**

**# Show the final image with the overlayed segmentation masks**

**plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))**

**plt.axis("off")**

**plt.show()**

**# Verify sample image path and re-run visualization**

**sample\_image\_path = '/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/train-300/data/000000000064.jpg'**

**if os.path.exists(sample\_image\_path):**

**sample\_annotations = [ann for ann in train\_labels['annotations'] if ann['image\_id'] == 1]**

**visualize\_segmentation(sample\_image\_path, sample\_annotations, relevant\_category\_ids)**

**else:**

**print(f"Error: The file {sample\_image\_path} does not exist.")**

**dataset\_path = '/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset'**

**dataDir='/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/train-300'**

**dataType='train2017'**

**annFile='{}/labels.json'.format(dataDir,dataType)**

**coco=COCO(annFile)**

**# Load the categories in a variable**

**catIDs = coco.getCatIds()**

**cats = coco.loadCats(catIDs)**

**# Get category names**

**category\_names = [cat['name'].title() for cat in cats]**

**# Get category counts**

**category\_counts = [coco.getImgIds(catIds=[cat['id']]) for cat in cats]**

**category\_counts = [len(img\_ids) for img\_ids in category\_counts]**

**ids = 1**

**cats = coco.loadCats(ids=ids)**

**print(cats)**

**catID = 15**

**print(coco.loadCats(ids=catID))**

**# Get image ids that satisfy the given filter conditions**

**imgId = coco.getImgIds(catIds=[catID])[0]**

**print(imgId)**

**ann\_ids = coco.getAnnIds(imgIds=[imgId], iscrowd=None)**

**print(f"Annotations for Image ID {imgId}:")**

**anns = coco.loadAnns(ann\_ids)**

**imageDir = '/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/train-300/data/'**

**image\_path = coco.loadImgs(imgId)[0]['file\_name']**

**print(imageDir)**

**print(image\_path)**

**image = plt.imread(imageDir + image\_path)**

**plt.imshow(image)**

**# Display the specified annotations**

**coco.showAnns(anns, draw\_bbox=True)**

**plt.axis('off')**

**plt.title('Annotations for Image ID: {}'.format(imgId))**

**plt.tight\_layout()**

**plt.show()**

**"""# \*\*PREPROCESSING\*\***

**Only keep annotations for the target classes by filtering the necessary classes.**

**Standardize image size to (512, 512).**

**Apply augmentations like rotation, flipping, etc., for better generalization.**

**"""**

**import albumentations as A**

**# Define augmentations**

**transform = A.Compose([**

**A.Resize(512, 512),**

**A.HorizontalFlip(p=0.5),**

**A.RandomBrightnessContrast(p=0.2),**

**])**

**def preprocess\_image(image, annotations):**

**augmented = transform(image=image)**

**image = augmented['image']**

**# Mask transformations can also be applied if needed for annotations**

**return image**

**"""# \*\*MODEL TRAINING\*\***

**this training used a pre-trained segmentation model called Detectron2.**

**\*\*Setting Up Mask R-CNN with Detectron2\*\***

**Detectron2 by Facebook AI Research is well-suited for segmentation tasks and supports Mask R-CNN.**

**"""**

**!python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'**

**import os**

**import json**

**from detectron2.structures import BoxMode**

**def get\_coco\_dataset\_dicts(data\_dir, labels):**

**with open(labels) as f:**

**coco\_data = json.load(f)**

**dataset\_dicts = []**

**category\_mapping = {"cake": 1, "car": 2, "dog": 3, "person": 4}**

**relevant\_categories = ["cake", "car", "dog", "person"]**

**cat\_ids = [cat['id'] for cat in coco\_data['categories'] if cat['name'] in relevant\_categories]**

**for img in coco\_data['images']:**

**record = {}**

**img\_id = img['id']**

**record["file\_name"] = os.path.join(data\_dir, img['file\_name'])**

**record["image\_id"] = img\_id**

**record["height"] = img['height']**

**record["width"] = img['width']**

**ann\_ids = [ann['id'] for ann in coco\_data['annotations'] if ann['image\_id'] == img\_id and ann['category\_id'] in cat\_ids]**

**objs = []**

**for ann\_id in ann\_ids:**

**ann = next(ann for ann in coco\_data['annotations'] if ann['id'] == ann\_id)**

**category\_id = ann['category\_id']**

**if category\_id not in cat\_ids:**

**continue**

**# Handle segmentation format (polygon or RLE)**

**segmentation = ann['segmentation']**

**if isinstance(segmentation, dict) and "counts" in segmentation:**

**# RLE format**

**formatted\_segmentation = segmentation**

**elif isinstance(segmentation, list):**

**if isinstance(segmentation[0], list):**

**# Polygon format with multiple contours**

**formatted\_segmentation = segmentation**

**else:**

**# Single polygon, wrap it in another list**

**formatted\_segmentation = [segmentation]**

**else:**

**raise TypeError(f"Unexpected segmentation format in annotation {ann\_id}: {segmentation}")**

**obj = {**

**"bbox": ann['bbox'],**

**"bbox\_mode": BoxMode.XYWH\_ABS,**

**"segmentation": formatted\_segmentation,**

**"category\_id": cat\_ids.index(category\_id),**

**"iscrowd": ann.get('iscrowd', 0)**

**}**

**objs.append(obj)**

**record["annotations"] = objs**

**dataset\_dicts.append(record)**

**return dataset\_dicts**

**# ipython-input-32-d08f004b803f**

**# Register the dataset**

**train\_labels\_path = '/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/train-300/data/labels.json'**

**val\_labels\_path = '/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/validation-300/data/labels.json'**

**DatasetCatalog.register("cocotrains", lambda: get\_coco\_dataset\_dicts('/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/train-300/data', train\_labels\_path))**

**MetadataCatalog.get("cocotrains").set(thing\_classes=relevant\_categories)**

**DatasetCatalog.register("cocovals", lambda: get\_coco\_dataset\_dicts('/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/validation-300/data', val\_labels\_path))**

**MetadataCatalog.get("cocovals").set(thing\_classes=relevant\_categories)**

**from detectron2.config import get\_cfg**

**from detectron2.model\_zoo import get\_config\_file**

**# Configuration**

**cfg = get\_cfg()**

**cfg.merge\_from\_file(get\_config\_file("COCO-InstanceSegmentation/mask\_rcnn\_R\_50\_FPN\_3x.yaml")) # Use get\_config\_file to get the correct path**

**cfg.DATASETS.TRAIN = ("cocotrains",)**

**cfg.DATASETS.TEST = ("cocovals",)**

**cfg.DATALOADER.NUM\_WORKERS = 2**

**cfg.MODEL.WEIGHTS = "detectron2://COCO-InstanceSegmentation/mask\_rcnn\_R\_50\_FPN\_3x/137849600/model\_final\_f10217.pkl" # Pretrained weights**

**cfg.SOLVER.IMS\_PER\_BATCH = 2**

**cfg.SOLVER.BASE\_LR = 0.001**

**cfg.SOLVER.MAX\_ITER = 300**

**cfg.MODEL.ROI\_HEADS.NUM\_CLASSES = 4 # Only 4 classes: cake, car, dog, person**

**# Train the model**

**trainer = DefaultTrainer(cfg)**

**trainer.resume\_or\_load(resume=False)**

**trainer.train()**

**# Set up the predictor for inference**

**cfg.MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST = 0.5 # Set a threshold for inference**

**predictor = DefaultPredictor(cfg)**

**test\_image\_paths = [**

**'/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/train-300/data/000000000064.jpg',]**

**# Visualize predictions**

**for test\_img\_path in test\_image\_paths: # Define paths to test images**

**im = cv2.imread(test\_img\_path)**

**outputs = predictor(im)**

**v = Visualizer(im[:, :, ::-1], MetadataCatalog.get("coco\_val"), scale=1.2)**

**out = v.draw\_instance\_predictions(outputs["instances"].to("cpu"))**

**plt.figure(figsize=(10, 10))**

**plt.imshow(out.get\_image()[:, :, ::-1])**

**plt.axis('off')**

**plt.show()**

**"""# \*\*TESTING ON THE FILTERED CLASSES\*\*"""**

**# Set up the predictor for inference**

**cfg.MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST = 0.5 # Set a threshold for inference**

**predictor = DefaultPredictor(cfg)**

**test\_image\_paths = [**

**'/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/test-30/000000001494.jpg',]**

**# Visualize predictions**

**for test\_img\_path in test\_image\_paths: # Define paths to test images**

**im = cv2.imread(test\_img\_path)**

**outputs = predictor(im)**

**v = Visualizer(im[:, :, ::-1], MetadataCatalog.get("coco\_val"), scale=1.2)**

**out = v.draw\_instance\_predictions(outputs["instances"].to("cpu"))**

**plt.figure(figsize=(10, 10))**

**plt.imshow(out.get\_image()[:, :, ::-1])**

**plt.axis('off')**

**plt.show()**

**# Set up the predictor for inference**

**cfg.MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST = 0.5 # Set a threshold for inference**

**predictor = DefaultPredictor(cfg)**

**test\_image\_paths = [**

**'/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/test-30/000000001551.jpg',]**

**# Visualize predictions**

**for test\_img\_path in test\_image\_paths: # Define paths to test images**

**im = cv2.imread(test\_img\_path)**

**outputs = predictor(im)**

**v = Visualizer(im[:, :, ::-1], MetadataCatalog.get("coco\_val"), scale=1.2)**

**out = v.draw\_instance\_predictions(outputs["instances"].to("cpu"))**

**plt.figure(figsize=(10, 10))**

**plt.imshow(out.get\_image()[:, :, ::-1])**

**plt.axis('off')**

**plt.show()**

**# Set up the predictor for inference**

**cfg.MODEL.ROI\_HEADS.SCORE\_THRESH\_TEST = 0.5 # Set a threshold for inference**

**predictor = DefaultPredictor(cfg)**

**test\_image\_paths = [**

**'/content/drive/MyDrive/RM\_Segmentation\_Assignment\_dataset/test-30/000000001650.jpg',]**

**# Visualize predictions**

**for test\_img\_path in test\_image\_paths: # Define paths to test images**

**im = cv2.imread(test\_img\_path)**

**outputs = predictor(im)**

**v = Visualizer(im[:, :, ::-1], MetadataCatalog.get("coco\_val"), scale=1.2)**

**out = v.draw\_instance\_predictions(outputs["instances"].to("cpu"))**

**plt.figure(figsize=(10, 10))**

**plt.imshow(out.get\_image()[:, :, ::-1])**

**plt.axis('off')**

**plt.show()**