National College of Ireland

Configuration Manual

MSc Research Project
Artificial Intelligence

Ayodeji Michael Adedeji
Student ID: x23170476

School of Computing
National College of Ireland

Supervisor: Anh Duong Trinh

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ayodeji Michael Adedeji
Student ID:	x23170476
Programme:	Artificial Intelligence
Year:	2024
Module:	MSc Research Project
Supervisor:	Anh Duong Trinh
Submission Due Date:	12/12/2024
Project Title:	Configuration Manual
Word Count:	655
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ayodeji Michael Adedeji
x23170476

1 Introduction

This manual provides information regarding the environment where the project was developed. It also provides the outline of how to replicate the project "Evaluation of the Detectron2 framework for Instance Segmentation of Multi-Component Meal Images".

2 System Configuration

The implementation was done in the Google Colab Pro environment. It is a web-based interactive Jupyter notebook. It provides access to virtual resources like GPU, CPU and Storage devices. Figure 1 shows some important system information of the Google Colab Pro environment.

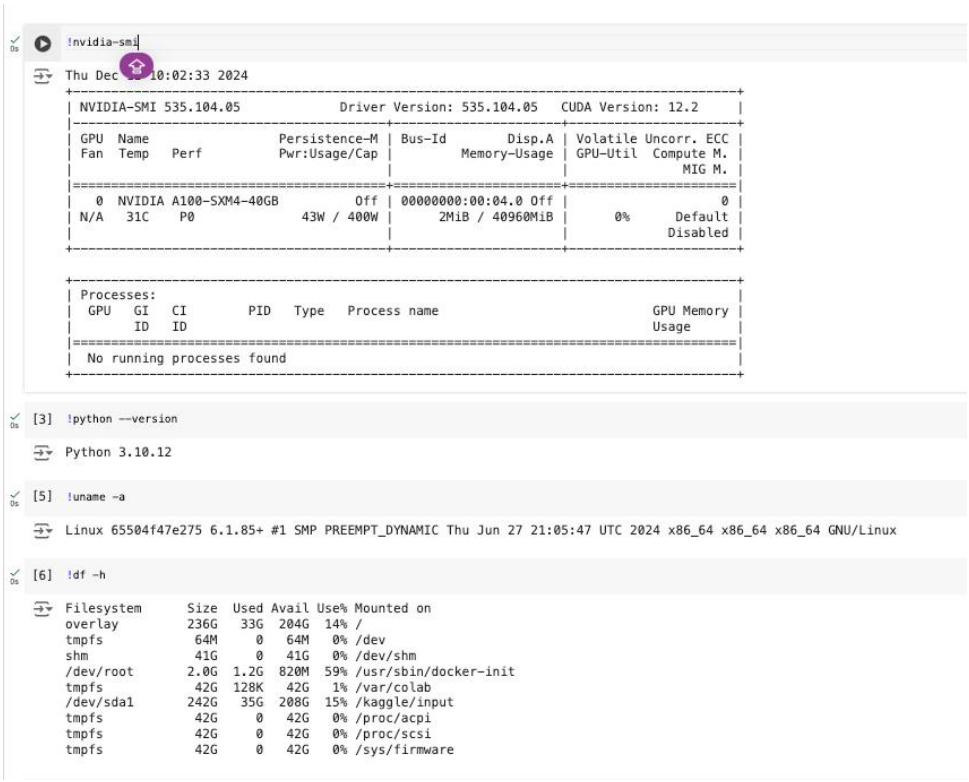


Figure 1: Google Colab Pro System Information

3 Libraries

The following libraries were used for the development of this project.

- Os
- PIL
- CV2
- TQDM
- Json
- Torch
- Numpy
- Pandas
- Google
- Random
- Detectron2
- Concurrent
- Matplotlib

While most of the libraries came pre-installed on the Google Colab Pro environment, libraries such as Detectron2 and OpenCV had be to installed, the pre-installed torch version was also downgraded to solve the compatibility issues that arose during the development. Figure 2 shows the code snippets for the downgrading of the torch library and installation of the Detectron2 and OpenCV libraries, while Figure 3 contains the code snippet for the import of all required libraries.

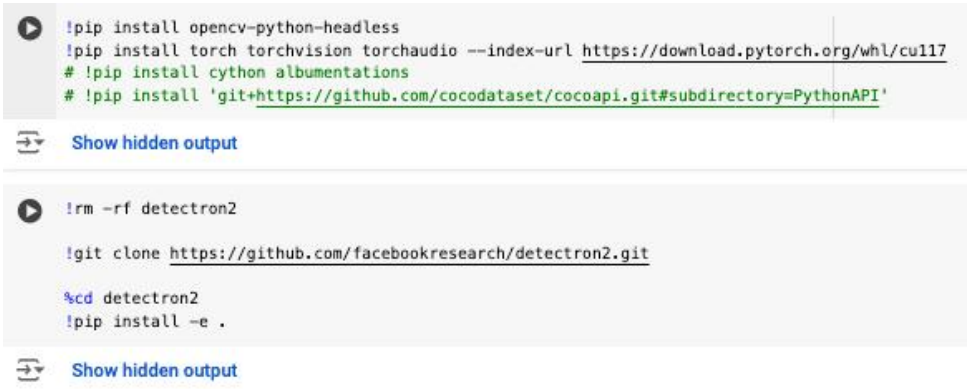


Figure 2: Downgrading Torch and Installation of Detectron2 and OpenCV

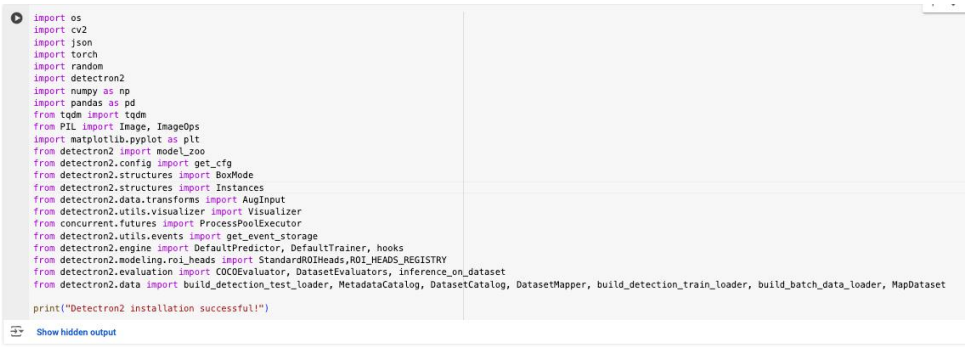


Figure 3: Importing Required Libraries

4 Dataset

This section contains the links to the datasets used. The FoodSeg103 was saved to a Google Drive folder [1], while UECFOODPIXComplete was saved to a different Google Drive folder [2].

5 Execution

This section details the step by step execution stages from dataset loading to evaluation. The code snippet in Figure 4 covers the following:

- Mounting of Google Drive in the Google Colab Pro environment.
- Extraction of the food labels from a text file into a Python dictionary.
- Creating a chart to display the proportion of test and train datasets.

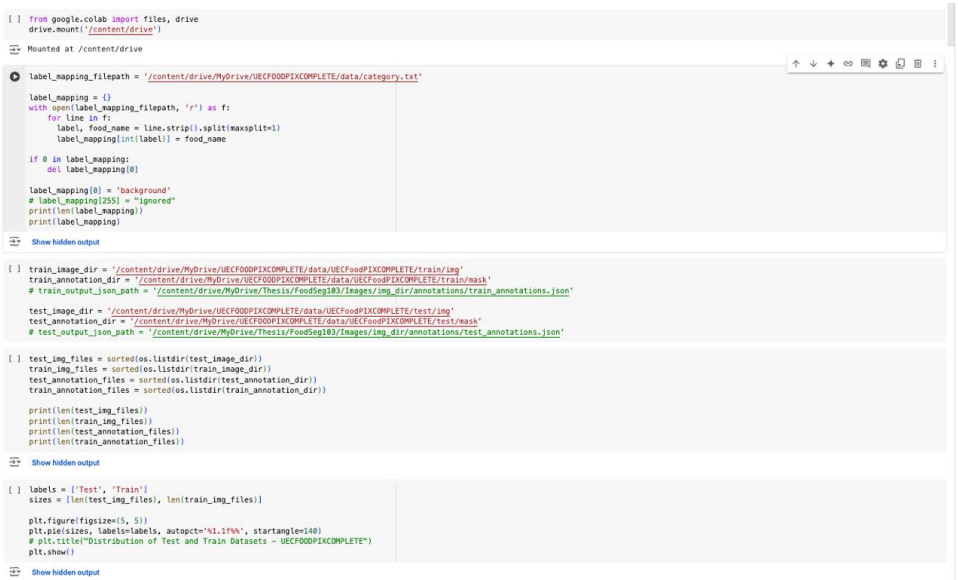


Figure 4: Dataset Exploration

[1] FoodSeg103 dataset link
[2] UECFOODPIXComplete dataset link

The code snippets in Figure 5 and 6 shows the random visualisation of images with a list of the food components present in the images from both datasets.

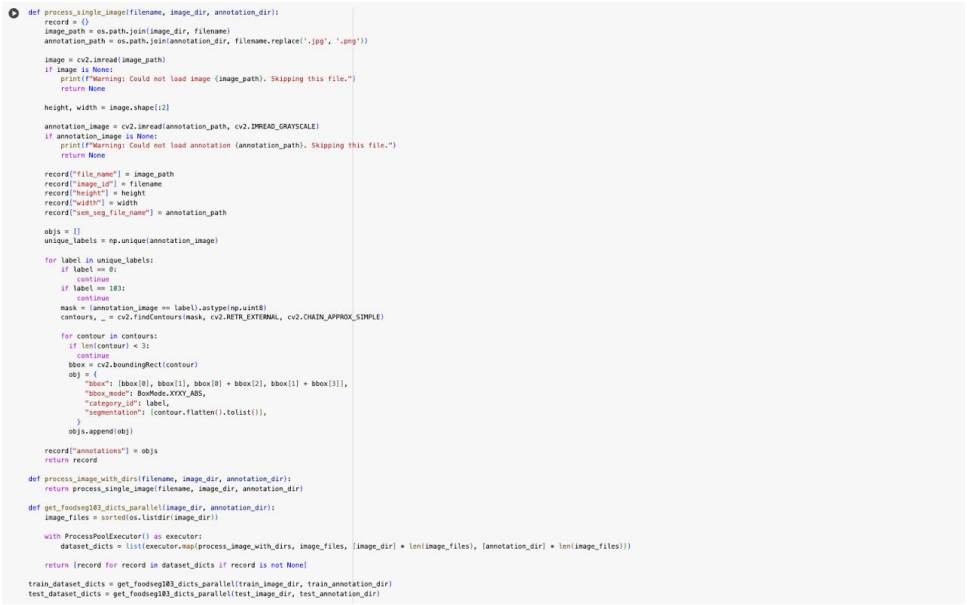
```
color_palette = plt.cm.tab20(np.linspace(0, 1, len(label_mapping)))[:, :3] * 255

def overlay_annotations(image, annotation, label_mapping):
    overlay = np.zeros_like(image, dtype=np.uint8)

    unique_ids = np.unique(annotation)
    for label_id in unique_ids:
        if label_id == 0:
            continue
        mask = annotation == label_id
        color = color_palette[label_id % len(color_palette)].astype(np.uint8)
        overlay[mask] = color

    blended = cv2.addWeighted(image, 0.7, overlay, 0.3, 0)

    plt.figure(figsize=(5, 5))
    plt.imshow(cv2.cvtColor(blended, cv2.COLOR_BGR2RGB))
    # plt.title("Image with Annotations")
    plt.axis("off")

    for label_id in unique_ids:
        if label_id == 0:
            continue
        if label_id == 103:
            continue
        plt.scatter([], [], color=color_palette[label_id % len(color_palette)] / 255, label=label_mapping[label_id])
    plt.legend(loc="upper right", title="Classes", bbox_to_anchor=(1.5, 1))
    plt.show()

for annotation_file in random.sample(train_annotation_files, 10):
    annotation_path = os.path.join(train_annotation_dir, annotation_file)
    image_path = os.path.join(train_image_dir, annotation_file.replace('.png', '.jpg'))

    annotation = cv2.imread(annotation_path, cv2.IMREAD_UNCHANGED)
    image = cv2.imread(image_path)

    if image is None or annotation is None:
        print(f"Skipping {annotation_file}: Could not load corresponding image or annotation.")
        continue

    overlay_annotations(image, annotation, label_mapping)
```

Figure 5: Visualization of images - FoodSeg103

```
[ ] color_palette = plt.cm.tab20(np.linspace(0, 1, len(label_mapping)))[:, :3] * 255

    def visualize_image_with_labels(image_path, mask_path, label_mapping):
        image = np.array(Image.open(image_path))
        mask_image = Image.open(mask_path)
        mask = np.array(mask_image)[:, :, 0]

        plt.figure(figsize=(6, 6))
        plt.imshow(image)
        plt.axis("off")

        unique_labels = np.unique(mask)
        for label_id in unique_labels:
            if label_id == 0:
                continue
            if label_id == 103:
                continue
            plt.scatter([], [], color=color_palette[label_id % len(color_palette)] / 255, label=label_mapping.get(label_id, "Unknown"))

        plt.legend(loc="upper right", title="Classes", bbox_to_anchor=(1.5, 1))
        plt.show()

    sampled_images = random.sample(train_img_files, 10)

    for image_name in sampled_images:
        image_path = os.path.join(train_image_dir, image_name)
        mask_path = os.path.join(train_annotation_dir, os.path.splitext(image_name)[0] + ".png")

        if os.path.exists(mask_path):
            print(f"Processing {image_name}...")

            visualize_image_with_labels(image_path, mask_path, label_mapping)
```

Figure 6: Visualization of images - UECFOODPIXComplete

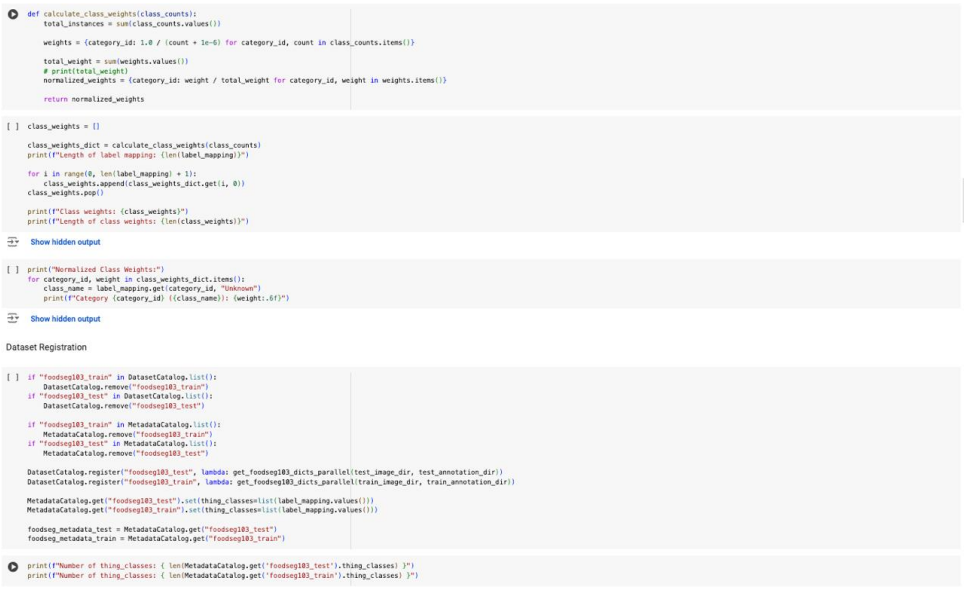
The code snippet in Figure 7 shows the conversion of the images and annotations into the standard COCO format.



```
def process_single_image(filename, image_dir, annotation_dir):
    record = {}
    image_path = os.path.join(image_dir, filename)
    annotation_path = os.path.join(annotation_dir, filename.replace('.jpg', '.png'))

    image = cv2.imread(image_path)
    if image is None:
        print(f"Warning: Could not load image {image_path}. Skipping this file.")
        return None

    height, width = image.shape[:2]

    annotation_image = cv2.imread(annotation_path, cv2.IMREAD_GRAYSCALE)
    if annotation_image is None:
        print(f"Warning: Could not load annotation {annotation_path}. Skipping this file.")
        return None

    record["file_name"] = image_path
    record["image_id"] = filename
    record["height"] = height
    record["width"] = width
    record["sem_seg_file_name"] = annotation_path

    objs = []
    unique_labels = np.unique(annotation_image)

    for label in unique_labels:
        if label == 0:
            continue
        if label == 103:
            continue
        mask = (annotation_image == label).astype(np.uint8)
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        for contour in contours:
          if len(contour) < 3:
            continue
          bbox = cv2.boundingRect(contour)
          obj = {
              "bbox": [bbox[0], bbox[1], bbox[0] + bbox[2], bbox[1] + bbox[3]],
              "bbox_mode": BoxMode.XYXY_ABS,
              "category_id": label,
              "segmentation": [contour.flatten().tolist()],
          }
          objs.append(obj)

    record["annotations"] = objs
    return record

def process_image_with_dirs(filename, image_dir, annotation_dir):
    return process_single_image(filename, image_dir, annotation_dir)

def get_foodseg103_dicts_parallel(image_dir, annotation_dir):
    image_files = sorted(os.listdir(image_dir))

    with ProcessPoolExecutor() as executor:
        dataset_dicts = list(executor.map(process_image_with_dirs, image_files, [image_dir] * len(image_files), [annotation_dir] * len(image_files)))

    return [record for record in dataset_dicts if record is not None]

train_dataset_dicts = get_foodseg103_dicts_parallel(train_image_dir, train_annotation_dir)
test_dataset_dicts = get_foodseg103_dicts_parallel(test_image_dir, test_annotation_dir)
```

Figure 7: Conversion of images and annotations to COCO format

The code snippet in Figure 8 shows the calculation of class weights and the registration of the COCO format dataset and class labels into Detectron2's Dataset and Metadata catalog.



```
def calculate_class_weights(class_counts):
    total_instances = sum(class_counts.values())

    weights = {category_id: 1.0 / (count + 1e-6) for category_id, count in class_counts.items()}

    total_weight = sum(weights.values())
    # print(total_weight)
    normalized_weights = {category_id: weight / total_weight for category_id, weight in weights.items()}

    return normalized_weights
```

```
[ ] class_weights = []

    class_weights_dict = calculate_class_weights(class_counts)
    print(f"Length of label mapping: {len(label_mapping)}")

    for i in range(0, len(label_mapping) + 1):
        class_weights.append(class_weights_dict.get(i, 0))
    class_weights.pop()

    print(f"Class weights: {class_weights}")
    print(f"Length of class weights: {len(class_weights)}")
```

Show hidden output

```
[ ] print("Normalized Class Weights:")
    for category_id, weight in class_weights_dict.items():
        class_name = label_mapping.get(category_id, "Unknown")
        print(f"Category {category_id} ({class_name}): {weight:.6f}")
```

Show hidden output

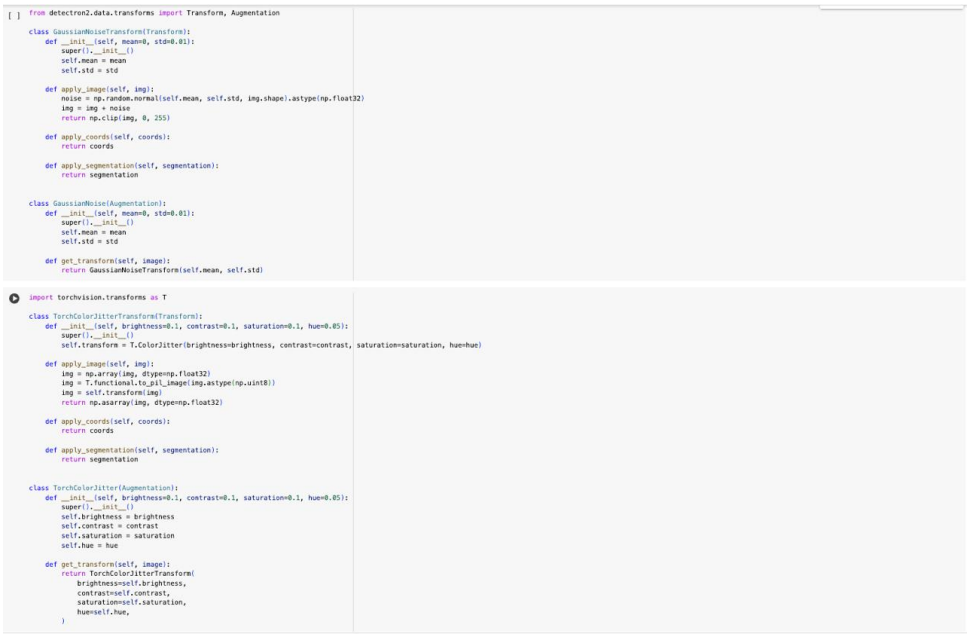
Dataset Registration

```
[ ] if "foodseg103_train" in DatasetCatalog.list():
        DatasetCatalog.remove("foodseg103_train")
    if "foodseg103_test" in DatasetCatalog.list():
        DatasetCatalog.remove("foodseg103_test")

    if "foodseg103_train" in MetadataCatalog.list():
        MetadataCatalog.remove("foodseg103_train")
    if "foodseg103_test" in MetadataCatalog.list():
        MetadataCatalog.remove("foodseg103_test")

    DatasetCatalog.register("foodseg103_test", lambda: get_foodseg103_dicts_parallel(test_image_dir, test_annotation_dir))
    DatasetCatalog.register("foodseg103_train", lambda: get_foodseg103_dicts_parallel(train_image_dir, train_annotation_dir))

    MetadataCatalog.get("foodseg103_test").set(thing_classes=list(label_mapping.values()))
    MetadataCatalog.get("foodseg103_train").set(thing_classes=list(label_mapping.values()))

    foodseg_metadata_test = MetadataCatalog.get("foodseg103_test")
    foodseg_metadata_train = MetadataCatalog.get("foodseg103_train")
```

```
print(f"Number of thing_classes: { len(MetadataCatalog.get('foodseg103_test').thing_classes) }")
print(f"Number of thing_classes: { len(MetadataCatalog.get('foodseg103_train').thing_classes) }")
```

Figure 8: Class weights calculation and Registration of Dataset and Class labels

The code snippet in Figure 9 show the implementation of custom data augmentation techniques such as the addition of Gaussian Noise and colour jittering to the data augmentation pipeline.

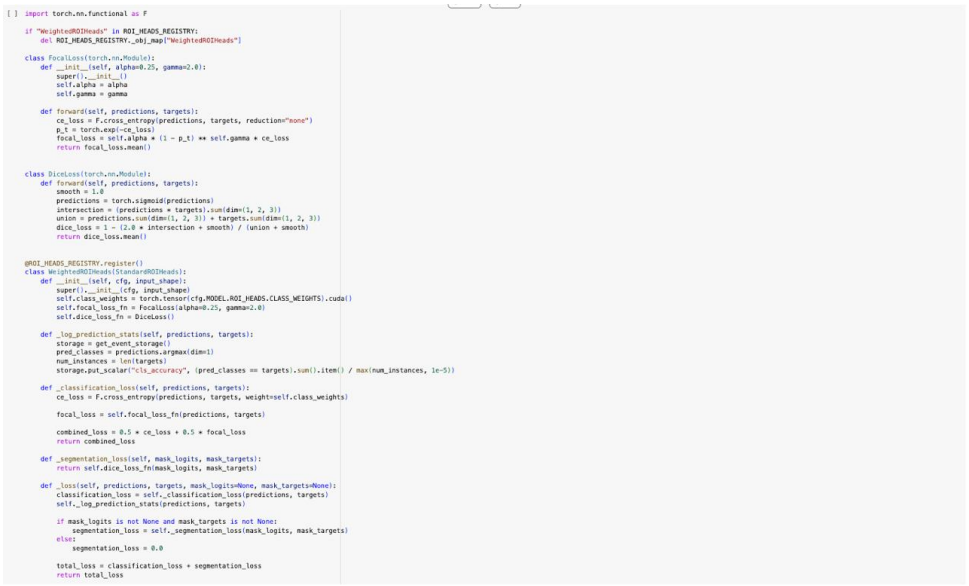


```
[ ] from detectron2.data.transforms import Transform, Augmentation

    class GaussianNoiseTransform(Transform):
        def __init__(self, mean=0, std=0.01):
            super().__init__()
            self.mean = mean
            self.std = std

        def apply_image(self, img):
            noise = np.random.normal(self.mean, self.std, img.shape).astype(np.float32)
            img = img + noise
            return np.clip(img, 0, 255)

        def apply_coords(self, coords):
            return coords

        def apply_segmentation(self, segmentation):
            return segmentation


    class GaussianNoise(Augmentation):
        def __init__(self, mean=0, std=0.01):
            super().__init__()
            self.mean = mean
            self.std = std

        def get_transform(self, image):
            return GaussianNoiseTransform(self.mean, self.std)
```

```
    import torchvision.transforms as T

    class TorchColorJitterTransform(Transform):
        def __init__(self, brightness=0.1, contrast=0.1, saturation=0.1, hue=0.05):
            super().__init__()
            self.transform = T.ColorJitter(brightness=brightness, contrast=contrast, saturation=saturation, hue=hue)

        def apply_image(self, img):
            img = np.array(img, dtype=np.float32)
            img = T.functional.to_pil_image(img.astype(np.uint8))
            img = self.transform(img)
            return np.asarray(img, dtype=np.float32)

        def apply_coords(self, coords):
            return coords

        def apply_segmentation(self, segmentation):
            return segmentation


    class TorchColorJitter(Augmentation):
        def __init__(self, brightness=0.1, contrast=0.1, saturation=0.1, hue=0.05):
            super().__init__()
            self.brightness = brightness
            self.contrast = contrast
            self.saturation = saturation
            self.hue = hue

        def get_transform(self, image):
            return TorchColorJitterTransform(
                brightness=self.brightness,
                contrast=self.contrast,
                saturation=self.saturation,
                hue=self.hue,
            )
```

Figure 9: Data Augmentation - Gaussian Noise and Colour Jittering

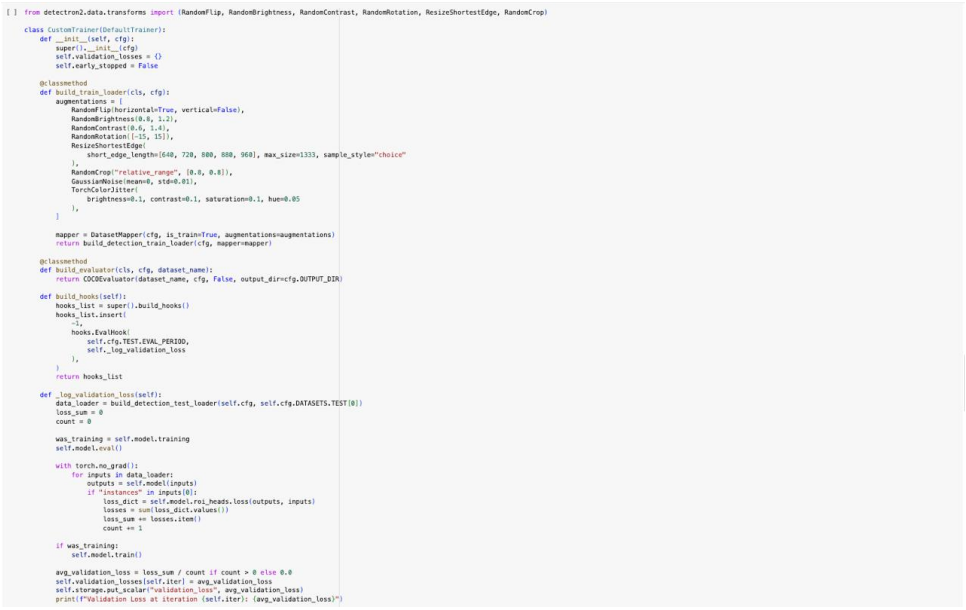
The code snippet in Figure 10 shows the implementation of Focal Loss, Dice Loss, modification of the Cross-Entropy Loss to the Weighted Cross-Entropy Loss and computation of the total loss.



```
[ ] import torch.nn.functional as F

    if "WeightedROIHeads" in ROI_HEADS_REGISTRY:
        del ROI_HEADS_REGISTRY._obj_map["WeightedROIHeads"]

    class FocalLoss(torch.nn.Module):
        def __init__(self, alpha=0.25, gamma=2.0):
            super().__init__()
            self.alpha = alpha
            self.gamma = gamma

        def forward(self, predictions, targets):
            ce_loss = F.cross_entropy(predictions, targets, reduction="none")
            p_t = torch.exp(-ce_loss)
            focal_loss = self.alpha * (1 - p_t) ** self.gamma * ce_loss
            return focal_loss.mean()


    class DiceLoss(torch.nn.Module):
        def forward(self, predictions, targets):
            smooth = 1.0
            predictions = torch.sigmoid(predictions)
            intersection = (predictions * targets).sum(dim=(1, 2, 3))
            union = predictions.sum(dim=(1, 2, 3)) + targets.sum(dim=(1, 2, 3))
            dice_loss = 1 - (2.0 * intersection + smooth) / (union + smooth)
            return dice_loss.mean()


    @ROI_HEADS_REGISTRY.register()
    class WeightedROIHeads(StandardROIHeads):
        def __init__(self, cfg, input_shape):
            super().__init__(cfg, input_shape)
            self.class_weights = torch.tensor(cfg.MODEL.ROI_HEADS.CLASS_WEIGHTS).cuda()
            self.focal_loss_fn = FocalLoss(alpha=0.25, gamma=2.0)
            self.dice_loss_fn = DiceLoss()

        def _log_prediction_stats(self, predictions, targets):
            storage = get_event_storage()
            pred_classes = predictions.argmax(dim=1)
            num_instances = len(targets)
            storage.put_scalar("cls_accuracy", (pred_classes == targets).sum().item() / max(num_instances, 1e-5))

        def _classification_loss(self, predictions, targets):
            ce_loss = F.cross_entropy(predictions, targets, weight=self.class_weights)

            focal_loss = self.focal_loss_fn(predictions, targets)

            combined_loss = 0.5 * ce_loss + 0.5 * focal_loss
            return combined_loss

        def _segmentation_loss(self, mask_logits, mask_targets):
            return self.dice_loss_fn(mask_logits, mask_targets)

        def _loss(self, predictions, targets, mask_logits=None, mask_targets=None):
            classification_loss = self._classification_loss(predictions, targets)
            self._log_prediction_stats(predictions, targets)

            if mask_logits is not None and mask_targets is not None:
                segmentation_loss = self._segmentation_loss(mask_logits, mask_targets)
            else:
                segmentation_loss = 0.0

            total_loss = classification_loss + segmentation_loss
            return total_loss
```

Figure 10: Loss Functions

In Figures 11 and 12, the default trainer is modified to include in-built data augmentation techniques such as random flipping, random rotation etc. The early stopping functionality is also integrated with the custom trainer.



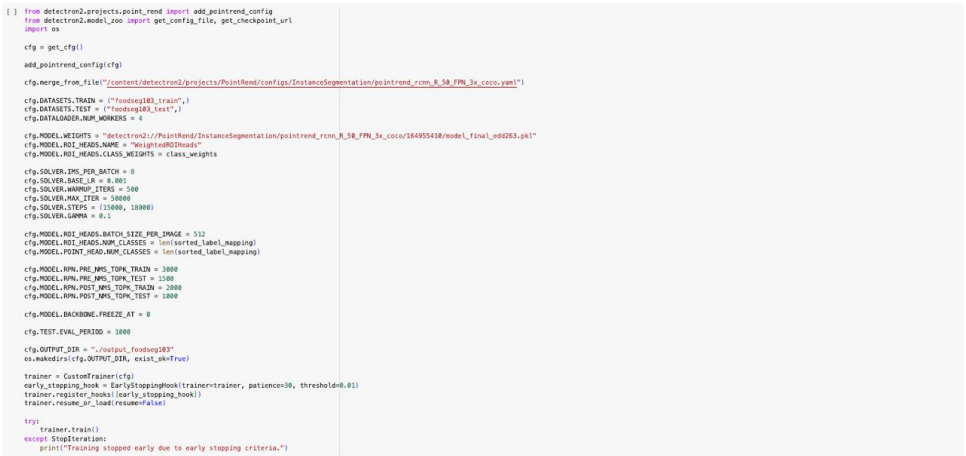
```
[ ] from detectron2.data.transforms import (RandomFlip, RandomBrightness, RandomContrast, RandomRotation, ResizeShortestEdge, RandomCrop)

    class CustomTrainer(DefaultTrainer):
        def __init__(self, cfg):
            super().__init__(cfg)
            self.validation_losses = {}
            self.early_stopped = False

        @classmethod
        def build_train_loader(cls, cfg):
            augmentations = [
                RandomFlip(horizontal=True, vertical=False),
                RandomBrightness(0.8, 1.2),
                RandomContrast(0.6, 1.4),
                RandomRotation([-15, 15]),
                ResizeShortestEdge(
                    short_edge_length=[640, 720, 800, 880, 960], max_size=1333, sample_style="choice"
                ),
                RandomCrop("relative_range", [0.8, 0.8]),
                GaussianNoise(mean=0, std=0.01),
                TorchColorJitter(
                    brightness=0.1, contrast=0.1, saturation=0.1, hue=0.05
                ),
            ]

            mapper = DatasetMapper(cfg, is_train=True, augmentations=augmentations)
            return build_detection_train_loader(cfg, mapper=mapper)

        @classmethod
        def build_evaluator(cls, cfg, dataset_name):
            return COCOEvaluator(dataset_name, cfg, False, output_dir=cfg.OUTPUT_DIR)

        def build_hooks(self):
            hooks_list = super().build_hooks()
            hooks_list.insert(
                -1,
                hooks.EvalHook(
                    self.cfg.TEST.EVAL_PERIOD,
                    self._log_validation_loss
                ),
            )
            return hooks_list

        def _log_validation_loss(self):
            data_loader = build_detection_test_loader(self.cfg, self.cfg.DATASETS.TEST[0])
            loss_sum = 0
            count = 0

            was_training = self.model.training
            self.model.eval()

            with torch.no_grad():
                for inputs in data_loader:
                    outputs = self.model(inputs)
                    if "instances" in inputs[0]:
                        loss_dict = self.model.roi_heads.loss(outputs, inputs)
                        losses = sum(loss_dict.values())
                        loss_sum += losses.item()
                        count += 1

            if was_training:
                self.model.train()

            avg_validation_loss = loss_sum / count if count > 0 else 0.0
            self.validation_losses[self.iter] = avg_validation_loss
            self.storage.put_scalar("validation_loss", avg_validation_loss)
            print(f"Validation Loss at iteration {self.iter}: {avg_validation_loss}")
```

Figure 11: In-built data augmentation

```
            avg_validation_loss = loss_sum / count if count > 0 else 0.0
            self.validation_losses[self.iter] = avg_validation_loss
            self.storage.put_scalar("validation_loss", avg_validation_loss)
            print(f"Validation Loss at iteration {self.iter}: {avg_validation_loss}")

        def save_model(self):
            model_path = os.path.join(self.cfg.OUTPUT_DIR, "model_final.pth")
            torch.save(self.model.state_dict(), model_path)
            print(f"Model saved to {model_path}")


    class EarlyStoppingHook(hooks.HookBase):
        def __init__(self, trainer, patience=5, threshold=0.01):
            self.patience = patience
            self.threshold = threshold
            self.best_loss = float("inf")
            self.counter = 0
            self.trainer = trainer

        def after_step(self):
            if self.trainer.iter in self.trainer.validation_losses:
                current_loss = self.trainer.validation_losses[self.trainer.iter]

                if current_loss < self.best_loss - self.threshold:
                    self.best_loss = current_loss
                    self.counter = 0
                else:
                    self.counter += 1

                if self.counter >= self.patience:
                    print(f"Early stopping triggered at iteration {self.trainer.iter}.")
                    self.trainer.early_stopped = True
                    self.trainer.save_model()
                    raise StopIteration
```

Figure 12: Early stopping

The code snippet in Figure 13 shows the instantiation of our Detectron2 trainer using the PointRend configuration yaml. The training parameters are also set in the code snippet.



```
[ ] from detectron2.projects.point_rend import add_pointrend_config
    from detectron2.model_zoo import get_config_file, get_checkpoint_url
    import os

    cfg = get_cfg()

    add_pointrend_config(cfg)

    cfg.merge_from_file("/content/detectron2/projects/PointRend/configs/InstanceSegmentation/pointrend_rcnn_R_50_FPN_3x_coco.yaml")

    cfg.DATASETS.TRAIN = ("foodseg103_train",)
    cfg.DATASETS.TEST = ("foodseg103_test",)
    cfg.DATALOADER.NUM_WORKERS = 4

    cfg.MODEL.WEIGHTS = "detectron2://PointRend/InstanceSegmentation/pointrend_rcnn_R_50_FPN_3x_coco/164955410/model_final_edd263.pkl"
    cfg.MODEL.ROI_HEADS.NAME = "WeightedROIHeads"
    cfg.MODEL.ROI_HEADS.CLASS_WEIGHTS = class_weights

    cfg.SOLVER.IMS_PER_BATCH = 8
    cfg.SOLVER.BASE_LR = 0.001
    cfg.SOLVER.WARMUP_ITERS = 500
    cfg.SOLVER.MAX_ITER = 50000
    cfg.SOLVER.STEPS = (15000, 18000)
    cfg.SOLVER.GAMMA = 0.1

    cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = len(sorted_label_mapping)
    cfg.MODEL.POINT_HEAD.NUM_CLASSES = len(sorted_label_mapping)

    cfg.MODEL.RPN.PRE_NMS_TOPK_TRAIN = 3000
    cfg.MODEL.RPN.PRE_NMS_TOPK_TEST = 1500
    cfg.MODEL.RPN.POST_NMS_TOPK_TRAIN = 2000
    cfg.MODEL.RPN.POST_NMS_TOPK_TEST = 1000

    cfg.MODEL.BACKBONE.FREEZE_AT = 0

    cfg.TEST.EVAL_PERIOD = 1000

    cfg.OUTPUT_DIR = "./output_foodseg103"
    os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)

    trainer = CustomTrainer(cfg)
    early_stopping_hook = EarlyStoppingHook(trainer=trainer, patience=30, threshold=0.01)
    trainer.register_hooks([early_stopping_hook])
    trainer.resume_or_load(resume=False)

    try:
        trainer.train()
    except StopIteration:
        print("Training stopped early due to early stopping criteria.")
```

Figure 13: Model Training

The code snippet in Figure 14 shows the upload of the final model, the weights and the training outputs to a custom location for reusability.

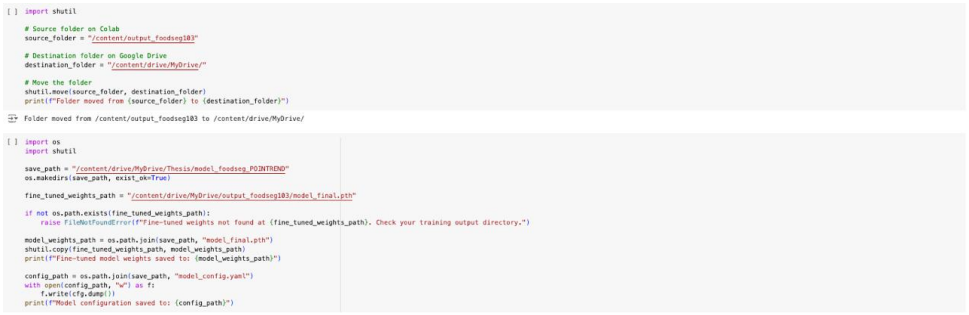


Figure 14: Upload of final model, its weights and the training outputs to a custom location

The code snippet in Figure 15 is used to generate a plot of the training loss and mean average precision across different iterations.

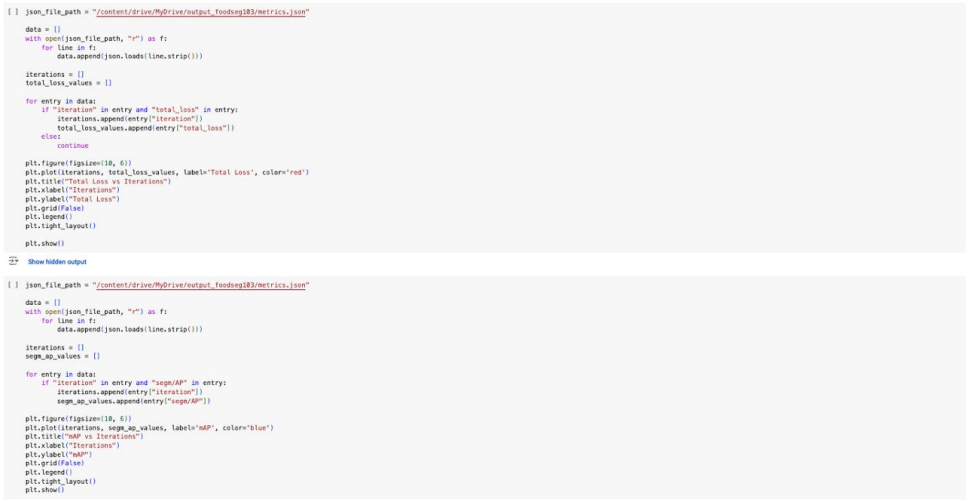
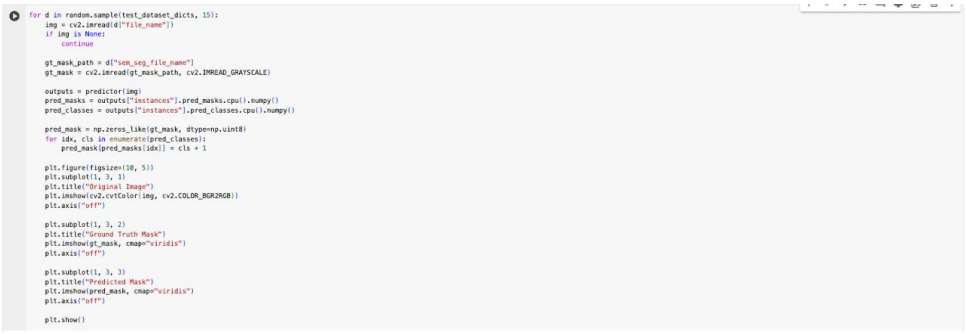


Figure 15: Plot of training loss and mean average precision across different iterations

The code snippet in Figure 16 shows the model evaluation and generation of predictions with the final model.

Figure 16: Evaluation of model and generation of predictions

The code snippet in Figure 17 is used to visualize the image alongsides the ground truth masks and predicted masks.



```
for d in random.sample(test_dataset_dicts, 15):
    img = cv2.imread(d["file_name"])
    if img is None:
        continue

    gt_mask_path = d["sem_seg_file_name"]
    gt_mask = cv2.imread(gt_mask_path, cv2.IMREAD_GRAYSCALE)

    outputs = predictor(img)
    pred_masks = outputs["instances"].pred_masks.cpu().numpy()
    pred_classes = outputs["instances"].pred_classes.cpu().numpy()

    pred_mask = np.zeros_like(gt_mask, dtype=np.uint8)
    for idx, cls in enumerate(pred_classes):
        pred_mask[pred_masks[idx]] = cls + 1

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 3, 1)
    plt.title("Original Image")
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.axis("off")

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

    plt.subplot(1, 3, 3)
    plt.title("Predicted Mask")
    plt.imshow(pred_mask, cmap="viridis")
    plt.axis("off")

    plt.show()
```

Figure 17: Early stopping