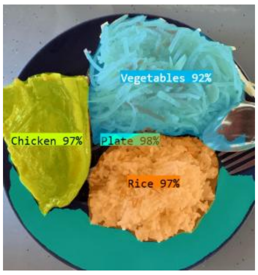


Project

TEB3123: Machine Learning

Mix-Food Rice Price Prediction

Name	**ID**	**Course**
Cheng Pin-Jie	21000548	Computer Science
Wong Xing Hao	21000612	Computer Science
Loo Pei Xin	21000483	Computer Science
Amaro Ong	21001982	Computer Science

AP Ts Dr Said Jadid Abdulkadir

Table of Contents

1.0 Introduction

Universiti Teknologi PETRONAS has more than 6 cafes, including V1 Café Warisan, V2 Gee & S, V3 Island One Café, and more. Most of the food price calculations are relied on human interpretations and human vision. This prone to human errors and unpredictable price variation, causing unsatisfaction in customers and students. Hence, our project objectives are to develop an AI model that can read and interpret a picture of mix food rice, ultimately providing reliable and consistent price predictions. By using **supervised** machine learning method, the project goals stated as below:

i. To identify and detect food on a round plate using segmentation process.
ii. To calculate the percentage of each type of food in the meal (Rice, Chicken, Fish, Vegetables).
iii. To determine and calculate the prices based on food composition using preset correlation between food portion and prices.

2.0 Data Understanding & Preparation

The project begins by acquiring an open-source dataset from internet. The dataset that we used is a combination of different datasets, allowing broader type of food segmentation: https://universe.roboflow.com/fyp-ys280

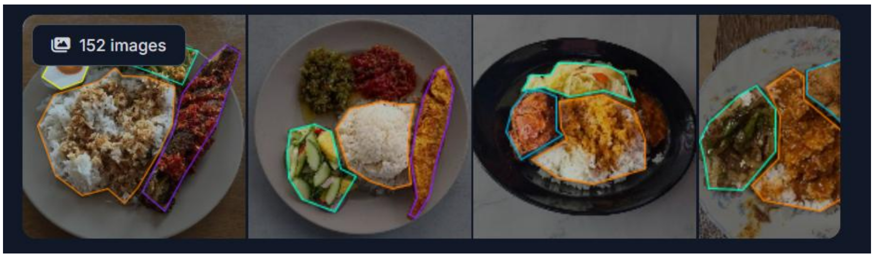


Figure 2a: Mix Food Rice Datasets

2.1 Data Pre-Processing

The datasets need to be clean and pre-processed before we pass them into algorithm for modelling purpose. Hence, by using functionality provided by *Roboflow*, the duplicated images are removed, image orientation are aligned, and figure size are adjusted to 640x640 pixels.

2.2 Data Augmentation

In this stage, we create multiple augmented images from pre-processed datasets by undergoing mosaic augmentation, random affine transformations, HSV adjustments and random flips. Data augmentation allows us to improve model generalization by creating modified versions out of our original datasets. After importing all the necessary Python libraries, we developed a function called *augment_yolov8_dataset* to perform data augmentation on our dataset. This function firstly retrieve all images from the training folder, then each image's polygon annotations (keypoints) are extracted in YOLO format labels. Afterwards, augmentation pipeline is performed multiple times per image while transformed annotations are ensured valid within the range of 0-1 range. Lastly, the augmented images with labels are saved. Due to document length constraints, the code snippet below represents only a portion of the entire data augmentation process:

This is an example snippet of a transformation function (Geometric transformations). The other transformation are colour, blur and weather transformation.

```
transforms = A.Compose([
        # Geometric transformations
        A.OneOf([
            A.HorizontalFlip(p=0.8),
            A.VerticalFlip(p=0.5),
            A.RandomRotate90(p=0.5),
            A.Affine(
                scale=(0.85, 1.15),
                translate_percent={"x": (-0.1, 0.1), "y": (-0.1, 0.1)},
                rotate=(-15, 15),
                shear={"x": (-5, 5), "y": (-5, 5)},
                p=0.5),
        ], p=0.7),

        # Color transformations …

        # Blur and noise …

        # Weather effects …

    ],
keypoint_params=A.KeypointParams(format='xy',label_fields=['class_labels'],
remove_invisible=False))
```

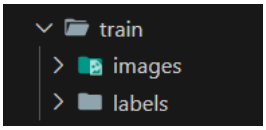
This next snippet provides us to retrieve the image annotations (keypoints):

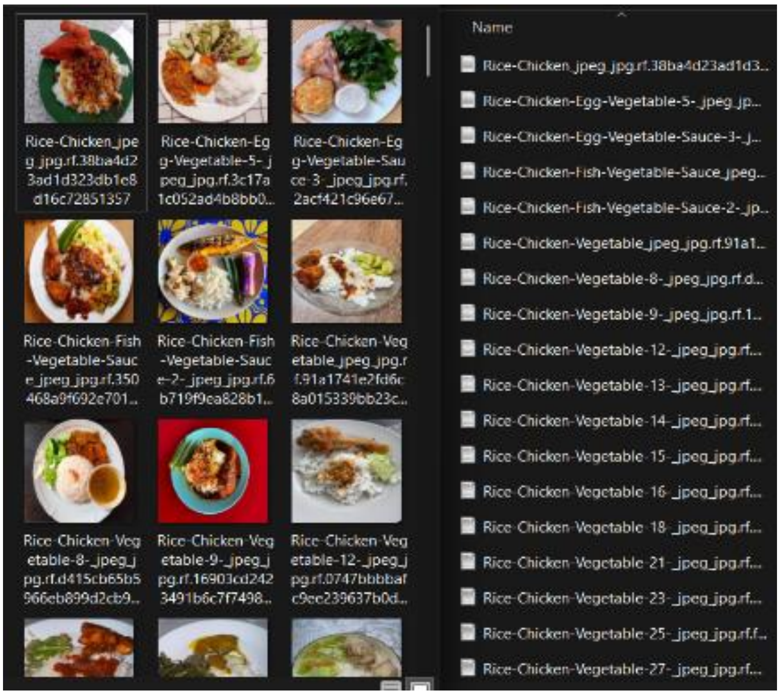
```
# Read polygon labels
            polygons = []
            class_labels = []

            with open(label_file, 'r') as f:
                for line in f.readlines():
                    parts = line.strip().split()
                    if len(parts) >= 3:  # class_id followed by x,y pairs
                        class_id = int(parts[0])
                        # Convert pairs of coordinates into keypoints
                        keypoints = []
                        for i in range(1, len(parts), 2):
                            if i + 1 < len(parts):
                                x = float(parts[i])
                                y = float(parts[i + 1])
                                keypoints.append((x, y))

                        if keypoints:
                            polygons.append(keypoints)
                            class_labels.append(class_id)
```

Augmentation process is carried out per image in a loop as shown below:

```
# Apply augmentation
for aug_idx in range(augmentations_per_image):
    transformed = transforms(
        image=image,
        keypoints=keypoints_flattened,
        class_labels=keypoint_class_labels
    transformed_image = transformed['image']
    transformed_keypoints = transformed['keypoints']
    transformed_class_labels = transformed['class_labels
    # Skip if no keypoints after augmentation
    if len(transformed_keypoints) == 0:
        contin
    # Group keypoints back into polygons
    transformed_polygons = {}
    for kp, class_id, poly_id in zip(transformed_keypoints,
transformed_class_labels, polygon_idx):
        if poly_id not in transformed_polygons:
            transformed_polygons[poly_id] = {
                "points": [], "class_id": class_id}
        transformed_polygons[poly_id]["points"].append(k
    # Save augmented image directly in the same folder as original images
    aug_image_name = f"{image_name}_aug_{aug_idx}{ext}"
    aug_image_path = os.path.join(train_images_path, aug_image_name)
    cv2.imwrite(aug_image_path, cv2.cvtColor(
```

```
        transformed_image, cv2.COLOR_RGB2BGR
    # Save augmented labels directly in the same folder as original labels
    aug_label_path = os.path.join(train_labels_path,
f"{image_name}_aug_{aug_idx}.txt")
    with open(aug_label_path, 'w') as f:
        for poly_id, poly_data in transformed_polygons.items():
            line = str(poly_data["class_id"])
            for x, y in poly_data["points"]:
                # Ensure values are within valid range
                x = max(0.0, min(1.0, x))
                y = max(0.0, min(1.0, y))
                line += f" {x} {y}"
            f.write(line + "\n
```
After processing, images are saved in local path with labels:





3.0 Modelling

The dataset is now ready to be pass into algorithm for modelling. By using *Yolov11* and *ultralytics* framework, we able to train a segmentation model. First, we load a pre-trained YOLOv11 nano segmentation model ('*yolo11n-seg.pt*'):

```
from ultralytics import YOLO
model = YOLO('yolo11n-seg.pt')
```

After that, we start the model training using *train* function with the following parameters:

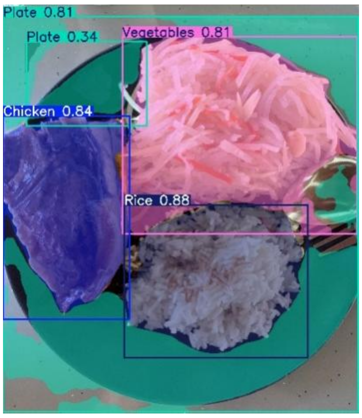
```
results = model.train(
    data=DATASET_CONFIG_PATH, #Path to YAML file
    epochs=300, #number of training cycles
    patience=50, #Early stopping if no improvement after 50 epochs
    imgsz=640, #Image input resolution (640x640 pixels)
    batch=32, #Number of images processed in each training batch
    cache=True, #Cache memory for faster training
    seed=42, #Random seed for reproducibility
    plots=True, #Generates performance visualizations during training
    save=True #Saves model checkpoints and final weights
)
```

4.0 Results & Conclusion

After roughly 120~150 training cycles due to early stopping, we test our model by uploading a picture from UTP cafes.

```
results = model(
    Path('test-image/1.jpg').absolute())
results[0].show()
```

We iterate *results* to extract segmentation masks *result.masks* to identify the calculate their mask area and class IDs.




```
{'total_area': 214240,
 'class_areas': {'Rice': 37126,
  'Chicken': 34591,
  'Plate': 82893,
  'Vegetables': 60058},
 'class_percentages': {'Rice': 17.33,
  'Chicken': 16.15,
  'Plate': 38.69,
  'Vegetables': 28.03}}
```

Furthermore, based on the masks area, we set a pricing rules based on different food type and food portion:

Food Type	Pricing
Rice	RM2 if > 10%, default = RM0
Chicken	RM4 per 20%
Fish	RM6 per 20%
Vegetables	RM2 per 20%

```python
def calculate_price(area_results):
    # Initialize price dictionary
    price_breakdown = {
        "Rice": 0,
        "Chicken": 0,
        "Fish": 0,
        "Vegetables": 0,
        "total": 0
    }

    # Check if plate exists
    if "Plate" not in area_results["class_percentages"]:
        return {"error": "Plate not detected", "total": 0}

    # Calculate price for each food item
    percentages = area_results["class_percentages"]

    # Rice: RM2 if > 10%
    if "Rice" in percentages and percentages["Rice"] > 10:
        price_breakdown["Rice"] = 2

    # Chicken: RM4 per 20%
    if "Chicken" in percentages:
        chicken_price = 4 * (percentages["Chicken"] / 20)
        price_breakdown["Chicken"] = round(chicken_price, 2)

    # Fish: RM6 per 20%
    if "Fish" in percentages:
        fish_price = 6 * (percentages["Fish"] / 20)
        price_breakdown["Fish"] = round(fish_price, 2)

    # Vegetables: RM2 per 20%
    if "Vegetables" in percentages:
        veg_price = 2 * (percentages["Vegetables"] / 20)
        price_breakdown["Vegetables"] = round(veg_price, 2)

    # Calculate total price
    price_breakdown["total"] = round(sum([
        price_breakdown["Rice"],
        price_breakdown["Chicken"],
        price_breakdown["Fish"],
        price_breakdown["Vegetables"]
    ]), 2)

    return price_breakdown
```

Lastly, export the result and price is calculated:

```python
price_result = calculate_price(area_results)

# Print results
print("\nFood composition percentages:")
for class_name, percentage in area_results["class_percentages"].items():
    print(f"{class_name}: {percentage}%")

print("\nPrice breakdown:")
for item, price in price_result.items():
    if item != "total" and item != "error":
        print(f"{item}: RM{price:.2f}")

print(f"\nTotal price: RM{price_result['total']:.2f}")
```

```
Food composition percentages:
Rice: 17.33%
Chicken: 16.15%
Plate: 38.69%
Vegetables: 28.03%

Price breakdown:
Rice: RM2.00
Chicken: RM3.23
Fish: RM0.00
Vegetables: RM2.80

Total price: RM8.03
```