**Project Capstone 5**

**Object Detection for Fresh Produce Quality Control**

**with Interactive Assistant**

**Introduction**

For this **Capstone 5** project, the primary objective is to identify and classify vegetables from images. Additionally, the project includes an intelligent chatbot capable of answering questions about the project, its implementation, and the characteristics of various vegetables. To automate and improve the accuracy of this process, AI-powered object detection has proven to be an effective solution. In this project, we utilize the **YOLO (You Only Look Once)** model for object detection, which is renowned for its speed and efficiency in real-time object identification, making it well-suited for the dynamic task of vegetable classification. To ensure optimal performance, we set the minimum requirements for the model to achieve a **Mean Average Precision (mAP) > 0.75** and an **inference time < 100ms per image** on a CPU.Use an AI-powered object detection

**Data Collection**

In this section, we created a custom dataset consisting of five vegetable classes: **Cucumber, Capsicum, Carrot, Cauliflower,** and **Potato**. To compile this dataset for training our model, we sourced images from multiple platforms. First, we utilized the **Kaggle Vegetable Image Dataset**, which provided a large collection of vegetable images. However, this dataset had some limitations, as many of the images were low-resolution or blurry, which could hinder the model's ability to learn effectively. To address this, we supplemented the dataset with high-resolution **online images** that offered more detailed views of the vegetables. While these images were of better quality, some came with watermarks that could interfere with the model's ability to identify key features. Lastly, we took a more hands-on approach by photographing the vegetables ourselves in the **vegetable aisle** of a local supermarket. This method allowed for greater control over image quality and composition but was more time-consuming and meticulous. Ultimately, this combination of sources provided a diverse and robust dataset for training the model

**Data Preparation**

Once we were satisfied with the data collection, the next step was to process the images into a format that would be more suitable and efficient for the model to learn from. For image detection, we used the **bounding box** method, which serves as a mask to help the model identify and classify objects. To label and annotate the dataset, we utilized **Roboflow**, a tool that allowed us to process the images into a format that could be easily read by our YOLO model. We then split the images across all classes with a distribution of **70% for training**, **15% for validation**, and **15% for testing**.

After labelling the images, we moved on to the next phase: **resizing** and **augmentation**. The resizing process standardizes the image dimensions, either by stretching, compressing, or rescaling the images. We set the target resolution to **640x640 pixels** to maintain consistency across the dataset. For image augmentation, we applied a simple transformation to the training set by rotating images **90° upside down**, which increased the variety of training data and helped the model generalize better. Once the images were labelled, resized, and augmented, we exported the dataset in **YAML format**, ready to be used for training with our YOLO model.

**Model Development**

We used **Jupyter Notebook** as our coding platform and imported the necessary libraries for model development, as outlined below.

# Import Packages

from ultralytics import YOLO

import ultralytics

import numpy as np

import cv2

import matplotlib.pyplot as plt

import os

To simplify dataset access, we employed a relative path method, which avoids the need to specify long file paths.

# Load our custom dataset

dataset = os.path.join(os.getcwd(),'Dataset','Sayuran 2.0.v12i.yolov11','data.yaml')

Next, we imported and loaded our **YOLO model**. For this project, we used **YOLOv11**, the latest model from Ultralytics, selecting the smallest model variant for faster training (Given more powerful hardware, I would have chosen a larger YOLO model for potentially better performance).

# Load the pretrained model

model = YOLO("yolo11n.pt")

Following this, we proceeded with training the model using our custom dataset. The training was conducted with the following settings: **epochs = 50**, **patience = 10**, **imgsz = 640**, and **seed = 42**.

# Training with custom Dataset

result = model.train(data=dataset, epochs=50, imgsz=640, seed=42, patience=10 ) # training the model with custom dataset

**Result**

After waiting for the model to train for nearly 1 hour and 40 minutes, we obtained the performance summary, which shows the best results from 50 epochs of training.

The model performance summary:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Class** | **Images** | **Instances** | **Box(P)** | **Box(R)** | **Box(mAP50)** | **Box(mAP50-95)** |
| All | 68 | 165 | 0.769 | 0.9 | 0.898 | 0.714 |
| Capsicum | 18 | 42 | 0.925 | 0.879 | 0.94 | 0.801 |
| Carrot | 10 | 33 | 0.533 | 0.934 | 0.78 | 0.559 |
| Cauliflower | 14 | 20 | 0.731 | 1 | 0.975 | 0.792 |
| Cucumber | 13 | 31 | 0.716 | 0.893 | 0.86 | 0.696 |
| Potato | 12 | 39 | 0.939 | 0.795 | 0.933 | 0.723 |

Speed: 1.4ms preprocess, 37.3ms inference, 0.0ms loss, 1.4ms postprocess per image

It appears that Cauliflower have highest for its mAP50, followed by Capsicum, Potato, Cucumber and lastly Carrot. mAP50 (mean Average Precision at IoU threshold 0.5) it is a key metric for evaluating object detection models because it balances **accuracy (precision)** and **completeness (recall)** at a specific level of overlap (IoU ≥ 0.5). In simpler terms: **mAP50** tells you how well the model identifies and localizes objects in images, considering that a bounding box must overlap with the ground truth by at least 50% to be counted as correct.

We then used this best-performing model for object detection on vegetable images.

# Deploy custom trained model

model\_trained = YOLO("runs/detect/train42/weights/best.pt") # Best model based on previous training

To make predictions, we set the confidence threshold to 0.7.

# Prediction

result = model\_trained.predict(r"C:\Users\suhaimi\Downloads\test2.png", show=True, conf=0.7)

Then we plot out the result using this code below

# plot out the result

img = result[0].plot() # this convert to numpy image

img = cv2.cvtColor(img, cv2.COLOR\_RGB2BGR)

plt.imshow(img)

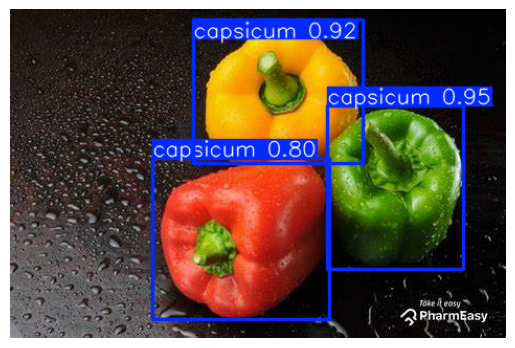
plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.axis('off')

plt.show()



The model successfully identified and labelled the vegetables, particularly recognizing **capsicums** in the images with high confidence. The results were visualized using the code above, clearly showing how the model labelled the capsicums with its predicted confidence scores.

**Conclusion**

In this Capstone 5 project, we successfully developed a vegetable identification and classification system using AI-powered object detection. The project aimed to identify five vegetable classes—Cucumber, Capsicum, Carrot, Cauliflower, and Potato—by training a YOLO model on a custom dataset. The model was designed to detect vegetables in images with high accuracy and real-time performance, meeting the goal of achieving a **Mean Average Precision (mAP) > 0.75** and an inference time of less than **100ms per image** on a CPU.

In summary, this project successfully leveraged the YOLO model for real-time vegetable classification and object detection, offering an efficient solution for identifying vegetables in images. The system's performance, including the mAP and inference time, met the project’s objectives, and the model can be further improved by using larger YOLO variants or additional data sources. This system can serve as a valuable tool for applications like automated inventory management in supermarkets or agricultural tech solutions.

**Project Code**

# Import Packages

from ultralytics import YOLO

import ultralytics

import numpy as np

import cv2

import matplotlib.pyplot as plt

import os

# Load our custom dataset

dataset = os.path.join(os.getcwd(),'Dataset','Sayuran 2.0.v12i.yolov11','data.yaml')

# Load the pretrained model

model = YOLO("yolo11n.pt")

# Training with custom Dataset

result = model.train(data=dataset, epochs=50, imgsz=640, seed=42, patience=10 ) # training the model with custom dataset

# Deploy custom trained model

model\_trained = YOLO("runs/detect/train42/weights/best.pt") # Best model based on previous training

# Deploy custom trained model

model\_trained = YOLO("runs/detect/train42/weights/best.pt") # Best model based on previous training

# Trained model metrics

metrics = model\_trained.val()

# Prediction

result = model\_trained.predict(r"C:\Users\suhaimi\Downloads\test2.png", show=True, conf=0.7)

# plot out the result

img = result[0].plot() # this convert to numpy image

img = cv2.cvtColor(img, cv2.COLOR\_RGB2BGR)

plt.imshow(img)

plt.xticks([])

plt.yticks([])

plt.grid(False)

plt.axis('off')

plt.show()

# Predict from Webcam

result1 = model\_trained.predict(source=0, stream=True, show = True, conf = 0.7) # 0 in this case for ultralytics are represent "Webcam",

print(type(result1))

# Using Webcam as Source for detection object

cam = cv2.VideoCapture(0)

try:

    for i in result1:

        print(i.boxes)

    cv2.destroyAllWindows()

    cam.release()

except Exception as e:

    cv2.destroyAllWindows()

    cam.release()