Capstone 5 – Vegetable Project Documentation

Project Implementation

The vegetable identification project using YOLOv11 aims to detect and classify five specific vegetables: cauliflower, tomato, cucumber, potato, and cabbage. The implementation involves collecting a diverse dataset of images, which are then annotated with bounding boxes using Roboflow. The data is split into training and validation sets and configured through a `data.yaml` file specifying class labels and paths. YOLOv11 is used for training with transfer learning from pretrained weights. Key hyperparameters include 30 epochs, a batch size of 16, and an image size of 240x240 pixels. The model's performance is evaluated using metrics (precision, recall, and mean Average Precision (mAP)). Adjustments like increasing epochs or balancing the dataset with synthetic augmentation are made to fine-tune the model.

For inference, the trained model (`best.pt`) is loaded to detect vegetables in new images, displaying bounding boxes with labels.

YOLO

YOLO (You Only Look Once) is a fast and efficient object detection algorithm that performs both object detection and localization in a single pass through an image, making it ideal for real-time applications like video surveillance, autonomous driving, and robotics. It works by dividing the image into a grid and predicting bounding boxes, class probabilities, and confidence scores for each grid cell simultaneously, enabling quick and accurate detection. YOLO's unified model approach distinguishes it from other methods that require multiple passes over the image, contributing to its speed. Over time, YOLO has evolved through several versions, each improving accuracy and performance, with the latest being YOLOv7, known for state-of-the-art results. Its versatility makes it suitable for various applications, including detecting objects in images, tracking moving objects, and even recognizing anomalies in medical images.

YOLOv11

YOLOv11 is an advanced version of the YOLO (You Only Look Once) object detection model, designed to improve both speed and accuracy across various computer vision tasks like object detection, segmentation, classification, and pose estimation. It introduces several variants (YOLO11n, YOLO11s, YOLO11m, YOLO11l, and YOLO11x), catering to different levels of computational resources, from lightweight models for faster inference to larger models that prioritize accuracy. YOLOv11 is optimized for real-time applications, with enhancements in training and inference speeds, and supports advanced capabilities such as recognizing poses and segmenting objects. It is efficient for deployment on both CPUs and GPUs, making it ideal for diverse applications, from object detection to real-time video analysis. These improvements are backed by rigorous training on large datasets like COCO, contributing to its versatility and high performance

Methodology

1. Data collection

A custom dataset was created by collecting and annotating images of various vegetables. The dataset was collected from https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset.

The dataset contains images of the following classes:

- Cabbage
- Cucumber
- Cauliflower
- Potato
- Tomato

2. Data annotation

Images was collected with different lighting conditions and angles. A total of 267 images with different lighting conditions and angles was annotated using Roboflow. The annotated images were saved in YOLO format (vegetables.v8i.yolov11).

The dataset is split into train, validation and test sets with respective 70/15/15 ratio using Roboflow. A *data.yaml* file was also created in the folder. Below is the content for the *data.yaml* file:

```yaml
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 5
names: ['cabbage', 'cauliflower', 'cucumber', 'potato', 'tomato']

roboflow:
  workspace: vegetables-yzlet
  project: vegetables-4g7xy
  version: 8
  license: CC BY 4.0
  url: https://universe.roboflow.com/vegetables-yzlet/vegetables-4g7xy/dataset/8
```

The downloaded dataset folder from Roboflow also included 2 README.txt files which is for the dataset and Roboflow:

i. README.dataset.txt
   # vegetables > 2024-11-11 11:41am
   https://universe.roboflow.com/vegetables-yzlet/vegetables-4g7xy

   Provided by a Roboflow user
   License: CC BY 4.0

ii. README.roboflow.txt

vegetables - v8 2024-11-11 11:41am
==========

This dataset was exported via roboflow.com on November 11, 2024 at 11:46 AM GMT

Roboflow is an end-to-end computer vision platform that helps you
* collaborate with your team on computer vision projects
* collect & organize images
* understand and search unstructured image data
* annotate, and create datasets
* export, train, and deploy computer vision models
* use active learning to improve your dataset over time

For state of the art Computer Vision training notebooks you can use with this dataset,
visit https://github.com/roboflow/notebooks

To find over 100k other datasets and pre-trained models, visit https://universe.roboflow.com

The dataset includes 641 images.
Vegetables are annotated in YOLOv11 format.

The following pre-processing was applied to each image:
* Auto-orientation of pixel data (with EXIF-orientation stripping)
* Resize to 640x640 (Stretch)

The following augmentation was applied to create 3 versions of each source image:
* 50% probability of horizontal flip
* 50% probability of vertical flip
* Equal probability of one of the following 90-degree rotations: none, clockwise, counter-clockwise, upside-down
* Randomly crop between 0 and 20 percent of the image
* Random rotation of between -15 and +15 degrees
* Random brigthness adjustment of between -15 and +15 percent
* Random exposure adjustment of between -10 and +10 percent

Data Augmentation

Flip: Horizontal, Vertical

90° Rotate: Clockwise, Counter-Clockwise, Upside Down

Crop: 0% Minimum Zoom, 20% Maximum Zoom

Rotation: Between -15° and +15°

Grayscale: Apply to 15% of images

Saturation: Between -25% and +25%

Brightness: Between -15% and +15%

Exposure: Between -10% and +10%

3. Model Development

Model Architecture

Model: YOLOv11x

Training Parameters:

- **Data file**: data.yaml
- **Epochs**: 30
- **Image size**: 240x240 pixels
- **Augmentation**: Enabled with horizontal flips (fliplr=0.5), no shear, and no vertical flip (flipud=0.0).
- **Batch size**: Based on the GPU memory constraints.

Code Explanation for Model Definition

The model is built using a **Sequential** container that contains various layers like convolutions, residuals, and attention mechanisms. These layers work together to extract features, process them, and eventually make predictions. The **Detect** module performs the task of final prediction by classifying the objects detected in the image.

Coding:

a. Import packages

```
# IMPORT PACKAGES
from ultralytics import YOLO
import ultralytics
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

The above code imports the necessary packages for a computer vision project, involving object detection using the YOLO (You Only Look Once) model. This code sets up the environment for an object detection task using a YOLO model from the Ultralytics library. It imports tools for image processing (`cv2`), numerical computations (`numpy`), and visualization (`matplotlib`).

b. Load the pretrained model

```
# LOAD THE PRETRAINED MODEL
model = YOLO("yolo11x.pt")
```

This code is for loading a pre-trained YOLO model using the Ultralytics library. The model uses yolo11x.pt model to to make predictions, detect objects in images, and perform various tasks related to computer vision. This step sets up the object detection model by loading the weights from the model file.

c. Plotting the predictions with the YOLO model

```
# PLOT OUT THE RESULT
result =
model.predict(r'D:\YPAI09\Capstone\Capstone_5_Vegetable_Zaryth_new\Capstone_5_Veg
etable_Zaryth\Capstone_5_Vegetable_Zaryth\test_image_2.jpg')
img = result[0].plot() # gives numpy array
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.xticks([])
plt.yticks([])
plt.grid(False)
plt.axis('off')
plt.show()
```

The coding is performing object detection on an image using the pre-trained YOLO model loaded previously. It then visualizes the detection results using OpenCV and Matplotlib. The code loads an image, runs it through the YOLO model for object detection, and visualizes the results with bounding boxes and labels. Below is the test image using the pretrained YOLO model:

Figure 1 Testing image with YOLO model before train

As you can see, the model does not detect the vegetables because it is not trained with the 5 classes yet.

d. Start train with custom dataset

```
# TRAINING WITH CUSTOM DATASET
results =
model.train(data=r'C:\Users\User\Desktop\zaryth\Capstone_5_Vegetable_Zaryth\Capstone_5_Vegetable_Zaryth\dataset\vegetables.v8i.yolov11\data.yaml', epochs=30, imgsz=240, augment=True, shear=0.0, flipud=0.0, fliplr=0.5)
```

The coding is used to train the YOLO model on a custom dataset using the Ultralytics library's `train` function. This code trains a YOLO model for 30 epochs on a custom vegetable dataset with 240x240 image resolution, using data augmentation (including horizontal flips) to improve detection accuracy based on the specifications in the `data.yaml` file. After completing training, the model should be fine-tuned to detect specific objects (like different vegetables) as defined in the dataset.

Here is the summary of layers in the YOLO based model:

	from	n	params	module	arguments
0	-1	1	2784	ultralytics.nn.modules.conv.Conv	[3, 96, 3, 2]
1	-1	1	166272	ultralytics.nn.modules.conv.Conv	[96, 192, 3, 2]
2	-1	2	389760	ultralytics.nn.modules.block.C3k2	[192, 384, 2, True, 0.25]
3	-1	1	1327872	ultralytics.nn.modules.conv.Conv	[384, 384, 3, 2]
4	-1	2	1553664	ultralytics.nn.modules.block.C3k2	[384, 768, 2, True, 0.25]
5	-1	1	5309952	ultralytics.nn.modules.conv.Conv	[768, 768, 3, 2]
6	-1	2	5022720	ultralytics.nn.modules.block.C3k2	[768, 768, 2, True]
7	-1	1	5309952	ultralytics.nn.modules.conv.Conv	[768, 768, 3, 2]
8	-1	2	5022720	ultralytics.nn.modules.block.C3k2	[768, 768, 2, True]
9	-1	1	1476864	ultralytics.nn.modules.block.SPPF	[768, 768, 5]
10	-1	2	3264768	ultralytics.nn.modules.block.C2PSA	[768, 768, 2]
11	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
12	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
13	-1	2	5612544	ultralytics.nn.modules.block.C3k2	[1536, 768, 2, True]
14	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
15	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
16	-1	2	1700352	ultralytics.nn.modules.block.C3k2	[1536, 384, 2, True]
17	-1	1	1327872	ultralytics.nn.modules.conv.Conv	[384, 384, 3, 2]
18	[-1, 13]	1	0	ultralytics.nn.modules.conv.Concat	[1]
...					

Figure 2 Summary of layers

Model Layer Breakdown

This model consists of various layers that each serve a specific function in detecting objects in images. Here's a quick summary of each type of layer:

1. **Convolutional Layers (Conv)**: These layers extract essential features from the input image, like edges and shapes. The numbers in the "arguments" column specify details like kernel size, number of filters, and stride.

2. **Residual Blocks (C3k2 and C3k)**: These blocks allow the model to learn complex features by stacking layers with skip connections. They help the model capture both detailed and high-level patterns in the image. The "True" argument indicates the use of residual connections, while values like "0.25" represent a fraction used for certain internal calculations.

3. **Spatial Pyramid Pooling (SPPF)**: This module combines features from different parts of the image to make detection more robust. It helps the model focus on important areas regardless of the object's size or location.

4. **Attention Mechanism (C2PSA)**: This layer gives the model the ability to focus on specific areas of the image that are more likely to contain objects. This is particularly helpful for identifying overlapping or similar-looking vegetables.

5. **Upsampling and Concatenation (Upsample and Concat)**: The Upsample layer increases the image resolution, helping the model retain details. The Concat layers combine information from multiple stages in the network, making it easier for the model to detect objects across various scales.

Model's Performance

```
Validating runs\detect\train8\weights\best.pt...
Ultralytics 8.3.24  Python-3.12.7 torch-2.5.1+cu118 CUDA:0 (NVIDIA GeForce RTX
2050, 4096MiB)
YOLO11x summary (fused): 464 layers, 56,832,799 parameters, 0 gradients, 194.4
GFLOPs
```

Class	Images	Instances	P	R	mAP50	mAP-95
all	40	136	0.858	0.808	0.892	0.592
cabbage	7	14	0.922	0.849	0.889	0.493
cauliflower	9	11	0.812	0.909	0.933	0.55
cucumber	9	52	0.803	0.75	0.857	0.652
potato	6	33	0.803	0.788	0.885	0.633
tomato	9	26	0.951	0.746	0.896	0.632

```
Speed: 0.0ms preprocess, 27.1ms inference, 0.0ms loss, 1.5ms postprocess per
image
Results saved to runs\detect\train8
```

1. Overall Accuracy:
   - The model achieves a high **mAP50 of 0.892**, indicating robust detection accuracy. This shows that the model can reliably identify objects with moderate overlap, which is crucial in scenarios where vegetables may be grouped closely together.
   - The **mAP50-95 score of 0.592** reflects the model's accuracy across a range of stricter IoU thresholds, which is a more challenging measure. While lower than mAP50, this result is still reasonable and suggests that the model can handle tighter object boundaries, though there may be room for improvement in precision.
2. Class-wise Performance:
   - The model performs well on most vegetable classes, with particularly strong scores for **cauliflower** (mAP50 of 0.933) and **potato** (mAP50 of 0.885). These high scores suggest the model effectively captures the distinguishing features of these vegetables, possibly due to their unique shapes and textures.
   - **Cucumber** and **tomato** also show good detection rates with mAP50 scores of 0.857 and 0.896, respectively. However, they display slightly lower mAP50-95 values, indicating that the model may have some difficulty distinguishing these vegetables in situations with significant overlap or tighter detection requirements.
   - **Cabbage** has the lowest mAP50-95 score (0.493), suggesting it may be harder for the model to detect accurately in complex or tightly-packed scenarios. This might be due to cabbage's similarity in shape and texture to other vegetables, highlighting a potential area for further model refinement.
3. Resource Efficiency:
   - The model's **inference speed of 27.1 ms per image** shows that it is capable of real-time processing, even on moderately powerful hardware (NVIDIA GeForce RTX 2050, 4GB). This makes the model suitable for real-time applications, such as use in agricultural sorting systems or in-store vegetable recognition tools.

e. Deploying custom trained model for pictures

```
# DEPLOY CUSTOM TRAINED MODEL
# For picture
model_custom =
YOLO(r"D:\YPAI09\Capstone\Capstone_5_Vegetable_Zaryth_new\Capstone_5_Vegetable_Za
ryth\Capstone_5_Vegetable_Zaryth\runs\detect\train8\weights\best.pt")
result =
model_custom(source=r'D:\YPAI09\Capstone\Capstone_5_Vegetable_Zaryth_new\Capstone
_5_Vegetable_Zaryth\Capstone_5_Vegetable_Zaryth\test_image_2.jpg') # picture -->
source="link image"
```

This code demonstrates the process of loading a custom-trained YOLO model, using the best weights obtained during training, to perform object detection on a given image. By specifying the image path, it applies the trained model to detect and classify objects within the image, effectively showcasing the model's ability to identify specific items (such as various vegetables) based on the custom dataset used during training.

f. Plot out the result

```
# PLOT OUT THE RESULT
img = result[0].plot() # gives numpy array
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.xticks([])
plt.yticks([])
plt.grid(False)
plt.axis('off')
plt.show()
```

This code extracts the detection results from the YOLO model and visualizes them by plotting the image with bounding boxes and labels for identified objects. It uses OpenCV to convert the color format for accurate display and Matplotlib to render a clear, clean image without grid lines or axis ticks.

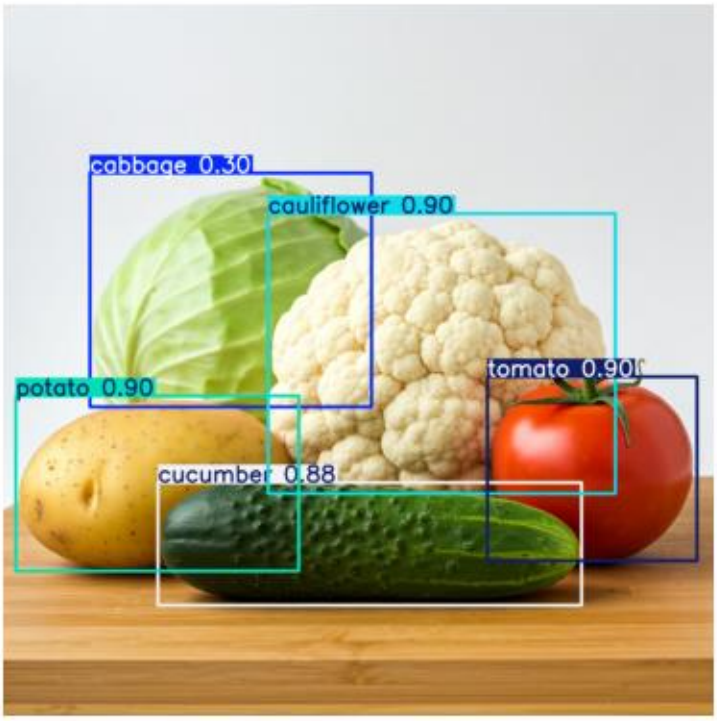

Figure 3 Testing image after train

The object detection has finally improved since we've trained the model. All classes are present in the picture and are classified correctly.

g. Deploying custom trained model using webcam

```
# DEPLOY CUSTOM TRAINED MODEL
# For camera
model_custom =
YOLO(r"C:\Users\User\Desktop\zaryth\Capstone_5_Vegetable_Zaryth\Capstone_5_Vegetable_Zaryth\runs\detect\train8\weights\best.pt")
result = model_custom(source=0,show=True)

# train8 is the best
```

This code demonstrates the deployment of a custom-trained YOLO model for real-time object detection using a webcam. The model is loaded from a specific path, which contains the best-performing weights from the training phase. The `source=0` argument configures the webcam as the input source, while `show=True` ensures the results are displayed in real time.

h. To print the model's architecture

```
# Print model architecture
print(model)
```

Model' architecture can be found in the model_architecture.txt file

Conclusion

The YOLO model demonstrates effective performance for vegetable identification, showing high accuracy and efficient processing speed. Overall, the model is a viable solution for real-time vegetable detection applications, providing a solid foundation that can be further refined for enhanced precision. The model effectively balances speed and accuracy for real-time detection tasks. It accurately detects and classifies multiple vegetables in a single pass, showcasing its efficiency and suitability for real-time applications like surveillance, autonomous driving, or robotics. The model performs with a high **mAP** (mean average precision) of 0.892, demonstrating its robustness and versatility across diverse object categories. It maintains consistent performance in various environments, handling different lighting conditions, object scales, and orientations with resilience.

Vegetables Detection Complete Coding

```
# IMPORT PACKAGES
from ultralytics import YOLO
import ultralytics
import numpy as np
import cv2
import matplotlib.pyplot as plt
# LOAD THE PRETRAINED MODEL
model = YOLO("yolo11x.pt")
# TRAINING WITH CUSTOM DATASET
results =
model.train(data=r'C:\Users\User\Desktop\zaryth\Capstone_5_Vegetable_Zaryth\Capstone_5_Vegetable_Zaryth\dataset\vegetables.v8i.yolov11\data.yaml', epochs=30, imgsz=240, augment=True, shear=0.0, flipud=0.0, fliplr=0.5)
# DEPLOY CUSTOM TRAINED MODEL
# For picture
model_custom =
YOLO(r"D:\YPAI09\Capstone\Capstone_5_Vegetable_Zaryth_new\Capstone_5_Vegetable_Zaryth\Capstone_5_Vegetable_Zaryth\runs\detect\train8\weights\best.pt")
result =
model_custom(source=r'D:\YPAI09\Capstone\Capstone_5_Vegetable_Zaryth_new\Capstone_5_Vegetable_Zaryth\Capstone_5_Vegetable_Zaryth\test_image_2.jpg') # picture --> source="link image"
# PLOT OUT THE RESULT
img = result[0].plot() # gives numpy array
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.xticks([])
plt.yticks([])
plt.grid(False)
plt.axis('off')
plt.show()
# DEPLOY CUSTOM TRAINED MODEL
# For camera
model_custom =
YOLO(r"C:\Users\User\Desktop\zaryth\Capstone_5_Vegetable_Zaryth\Capstone_5_Vegetable_Zaryth\runs\detect\train8\weights\best.pt")
result = model_custom(source=0,show=True)
# train8 is the best
# Print model architecture
print(model)
```