

CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY



SPECIALIZED THESIS
BACHELOR OF ENGINEERING IN
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

VIETNAMESE FOOD DETECTION WITH YOLOv8
(Upgraded version)

Advisor: Dr. Lam Nhut Khang

Student: Tran Thien Phuc
Student ID: B2015003
Class: 2023-2024 (K46)

Can Tho, 10/2023

**CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY**



**GRADUATION THESIS
BACHELOR OF ENGINEERING IN
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)**

**VIETNAMESE FOOD DETECTION WITH YOLOv8
(Upgraded version)**

Advisor: Dr. Lam Nhut Khang

**Student: Tran Thien Phuc
Student ID: B2015003
Class: 2023-2024 (K46)**

Can Tho, 10/2023

THANK YOU !

First of all, I would like to express my sincere thanks to the teacher of the Faculty of Electrical and Electronics Engineering for imparting and providing me with the necessary knowledge to apply in practice as well as in this topic.

And especially Dr. Lam Nhut Khang enthusiastically guided and helped me throughout the process of writing the essay, she enthusiastically guided the stages to do, what to prepare and how to approach the research topic effectively best.

Due to limited knowledge, in the process of writing the essay, i cannot avoid mistakes. I hope that the teacher can give more guidance so that i can learn from experience and complete better in the next time.

I sincerely thank you!

CONTENTS :

CHAPTER 1. INTRODUCTION.....	5
1.1 Introduction.....	5
1.2 The goal of the subject.....	5
1.3 Subject limit.....	5
1.4 Research Methods.....	5
1.5 Theme layout.....	6
CHAPTER 2: THEORETICAL BASIS.....	7
2.1 Network architecture of Yolo.....	7
Figure 1. YOLO network architecture diagram.....	7
Figure 2. Layers in the Darknet-53 network.....	8
2.2 Grid system.....	9
Figure 3. Converting the end Fully Connected layers into convolutional layers.....	9
Figure 4. The position and information of the object is maintained until the end.....	10
2.3 The concept of IoU index and Non-max suppression algorithm.....	10
Figure 5. Description of the IoU index.....	11
2.4 Label the patterns.....	12
Figure 6. The centers of the two objects coincide and lie in the same grid.....	13
2.5 Features of YOLOv8.....	13
Figure 7. Performance of YOLOv8 compared with older versions.....	14
Figure 8. YOLOv8 models for detection.....	15
Figure 9. The architecture of YOLOv8.....	15
CHAPTER 3: SYSTEM DESIGN.....	16
3.1 Steps to be able to train YOLOv8 on Custom Dataset:.....	16
3.2 Prepare the training data folder.....	16
Figure 10. The Dataset folder.....	17
Figure 11. Dataset folder structure.....	17
Figure 12. Labeling shapes.....	18
Figure 13. Downloading .zip package.....	18
3.3 Setting up an environment for training and identifying Vietnamese dishes.....	19
Figure 14. Set up model YOLOv8.....	20
Figure 15. Configuring the .yaml file.....	20
Figure 16. Upload and extract the dataset file.....	20
3.4 Training.....	21
Figure 17. Training on YOLOv8.....	21
CHAPTER 4. RESULTS.....	22
Figure 18. results directory after training.....	22
Figure 19. predicted results.....	23
CHAPTER 5 CONCLUSIONS AND DEVELOPMENT ORIENTATIONS.....	24
5.1 Conclude.....	24
5.2 Advantage.....	24

Figure 20. prediction results after training with dataset.....	24
5.3 Disadvantage.....	24
5.4 Development.....	25
APPENDIX.....	25
REFERENCES.....	26

CHAPTER 1. INTRODUCTION

1.1 Introduction

Deep learning network is the field of study of algorithms, computer programs so that computers can learn and make predictions like humans. It is applied to many different applications such as science, engineering, other fields of life as well as applications of object classification and detection. A typical example is CNN (Convolutional Neural Network) applied to automatic recognition, learn distinguishing patterns from images by successively stacking layers on top of each other, and in many applications, CNN is now considered as powerful image classifier and leverages technologies in the field of computer vision that leverage machine learning. But besides that, to classify an object, CNN technology consumes a lot of resources such as bandwidth, memory and processing capacity of hardware.

In order to reduce these consumable resources, more and more time-based algorithms and models have been developed, including the YOLOv8 model for the object classification problem, specifically applied to the problem of object classification “VIETNAMESE FOOD DETECTION WITH YOLOv8”.

1.2 The goal of the subject

- Learn about Deep Learning and its applications.
- Understand the theoretical and architectural basis of the YOLOv8 model for the object recognition problem.
- Use support libraries, virtual environment to execute the model.

1.3 Subject limit

In this topic, 20 different types of Vietnamese dishes can be identified. The system only stops at research and cannot be applied to the market. Another upgraded part is that model can work with realtime webcam tracking on PyCharm.

1.4 Research Methods

- Collect documents, refer to previous related applications.
- Based on the knowledge learned about how to train a neural network.
- Read more documents and search online.
- Consult and follow the instructor's instructions.

1.5 Theme layout

- **Chapter 1:** Overview. In this chapter, the objectives, limitations, research methods, layout and problem are presented.
- **Chapter 2:** Theoretical basis. In this chapter, the formation and development of Yolo and the algorithm of Yolov8 are presented.
- **Chapter 3:** System design. In this chapter, the operation diagram, software design, and software functions are presented.
- **Chapter 4:** Result. In this chapter, the results obtained after design and construction are presented.
- **Chapter 5:** Conclusion and directions for development. This chapter summarizes and presents the advantages and disadvantages in the research process and the development direction of the topic.
- **Appendix**
- **References**

CHAPTER 2: THEORETICAL BASIS

2.1 Network architecture of Yolo

The YOLO architecture includes: base networks are convolution networks that perform feature extraction. The back part is the Extra Layers applied to detect objects on the feature map of the base network. YOLO's base network uses mainly convolutional layers and fully connected layers as shown in Figure 2-1. YOLO architectures are also quite diverse and can be customized into versions for many different input shapes.

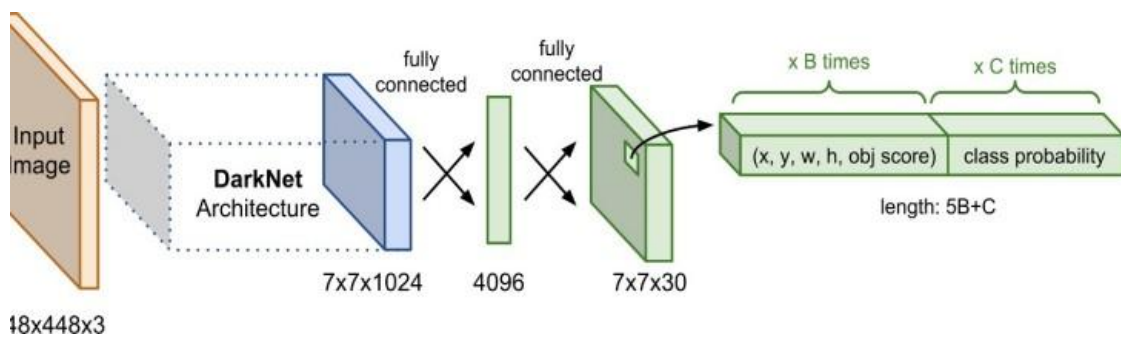


Figure 1. YOLO network architecture diagram.

The Darknet Architecture component called the base network has a feature extraction effect. The output of the base network is a 7x7x1024 feature map that will be used as input for the Extra layers that predict the object's label and bounding box coordinates.

In YOLO version 3, apply a feature extractor network darknet-53 as shown in Figure 2-2. This network consists of 53 consecutively connected convolutional layers, each of which is followed by a batch normalization and a Leaky Relu activation. To reduce the size of the output after each convolution layer, down sample is used with filters of size 2. This trick has the effect of minimizing the number of parameters for the model.

	Type	Filters	Size	Output
	Convolutional	32	3×3	256×256
	Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1	
	Convolutional	64	3×3	
	Residual			128×128
	Convolutional	128	$3 \times 3 / 2$	64×64
2x	Convolutional	64	1×1	
	Convolutional	128	3×3	
	Residual			64×64
	Convolutional	256	$3 \times 3 / 2$	32×32
8x	Convolutional	128	1×1	
	Convolutional	256	3×3	
	Residual			32×32
	Convolutional	512	$3 \times 3 / 2$	16×16
8x	Convolutional	256	1×1	
	Convolutional	512	3×3	
	Residual			16×16
	Convolutional	1024	$3 \times 3 / 2$	8×8
4x	Convolutional	512	1×1	
	Convolutional	1024	3×3	
	Residual			8×8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2. Layers in the Darknet-53 network.

The images, when fed into the model, will be scaled to the same size as the input shape of the model and then aggregated into a batch for training.

Currently YOLO is supporting 2 main inputs, 416x416 and 608x608. Each input will have its own design of layers that match the shape of the input. After going through the convolutional layers, the shape decreases exponentially by 2. Finally, we get a feature map with a relatively small size to predict the feature on each cell of the feature map.

The size of the feature map will depend on the input. For input 416x416, the feature map has dimensions of 13x13, 26x26 and 52x52. And when input is 608x608 will generate feature map 19x19, 38x38, 72x72.

2.2 Grid system

The idea of implementing Grid System comes from the fact that instead of implementing Fully Connected classes at the end of the model, these layers will be converted into convolutional layers like the front layers in the object classification problem. Then in the final layer instead of a fully connected layer of size (number of classes, 1) into a convolution layer of size 3 dimensions (1,1, number of classes).

This transformation does not affect the prediction results, but also shows that we can completely classify objects by convolution operations with the prediction results located in the final and superior convolutional layer. while maintaining the position of the object. The transformation takes place as follows, for example, in Figure 2-3, the image containing the object to be classified is $14 \times 14 \times 3$. After performing the convolution steps, the final convolution layer is obtained. $1 \times 1 \times 4$, the convolution layer itself carries the result of the object classification.

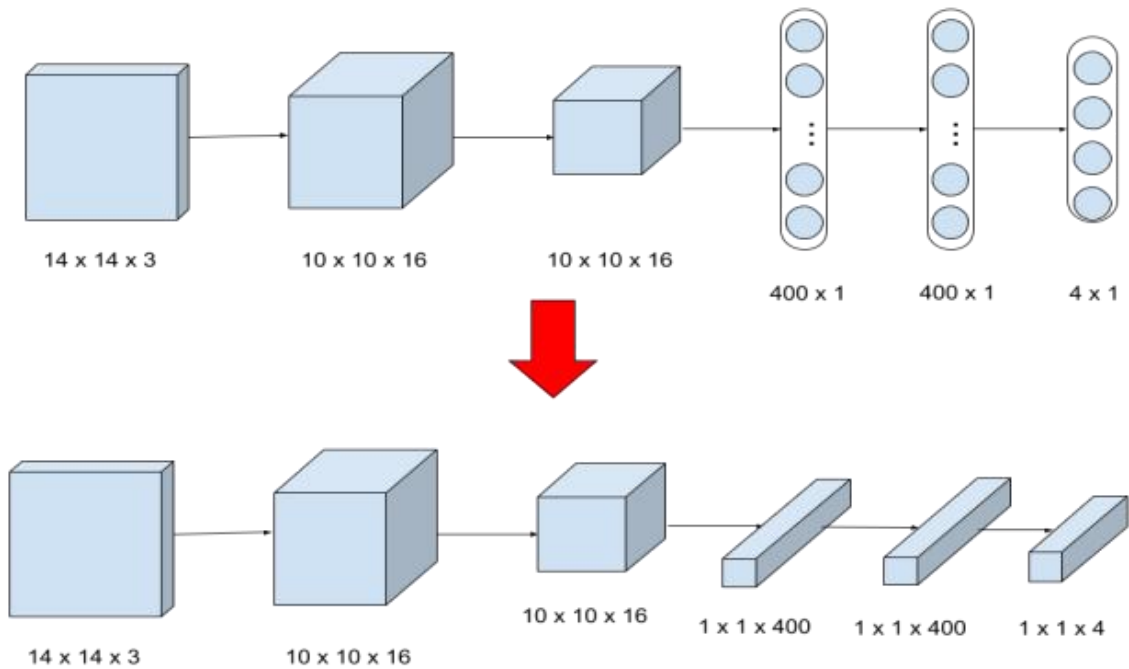


Figure 3. Converting the end Fully Connected layers into convolutional layers.

To prove that the object position does not change and can still be determined through the last layer, we assume that Figure 2-4 has dimensions of $27 \times 27 \times 3$

divided into 3 x 3 grids as shown below, the position of the object can be located in the colored box, after performing the convolution steps, we get the final convolution layer with size 3 x 3 x 4.

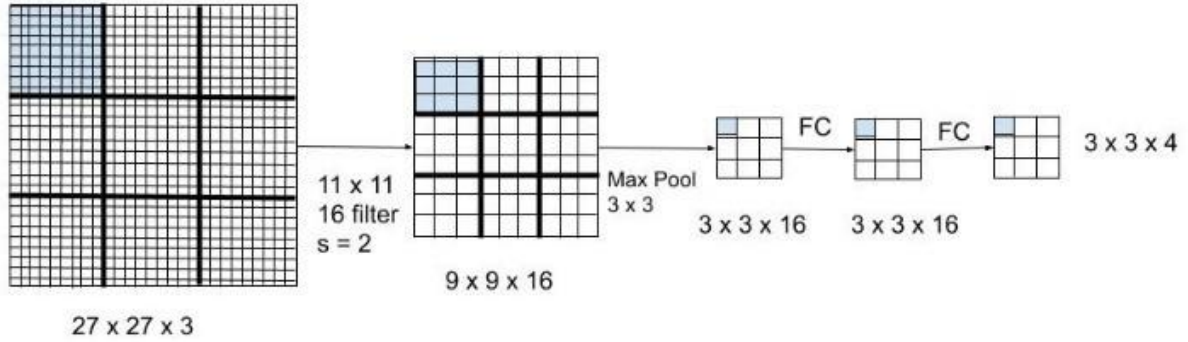


Figure 4. The position and information of the object is maintained until the end.

We can see that after convolution the data and the position of the object are maintained until the last layer, and in the corresponding color box there is the classification result of the object. Thus, we can both classify objects and determine the location of objects.

Thereby, the grid system will divide the original image into a grid number equivalent to the size of the last layer (not to mention the number of dimensions), as the example on the last layer has a size of 3 x 3 so we will divide the original image into 3 x 3 grid (bold line). There each grid will carry 3 main information: whether the grid contains objects or not, the coordinates of the bounding boxes (including the x,y coordinates of the upper left corner and the length and width of the bounding box), determine object classification.

2.3 The concept of IoU index and Non-max suppression algorithm

The IoU (Intersection over Union) index tells us the overlap of two boxes. Where $A \cap B$ is the intersection (Intersection) of box A and box B, $A \cup B$ is the common part of 2 boxes (Union) equal to the sum of the areas of the 2 boxes minus the intersection. Based on Figure 2-6, determining the IoU helps to calculate the ability to accurately detect the object, in which box A is usually the anchor box (or groundtruth bounding box) assigned label in the training phase and box B is the bounding box of the system

specified in the test phase. Calculate IoU to evaluate whether the model has detected the correct object or not.

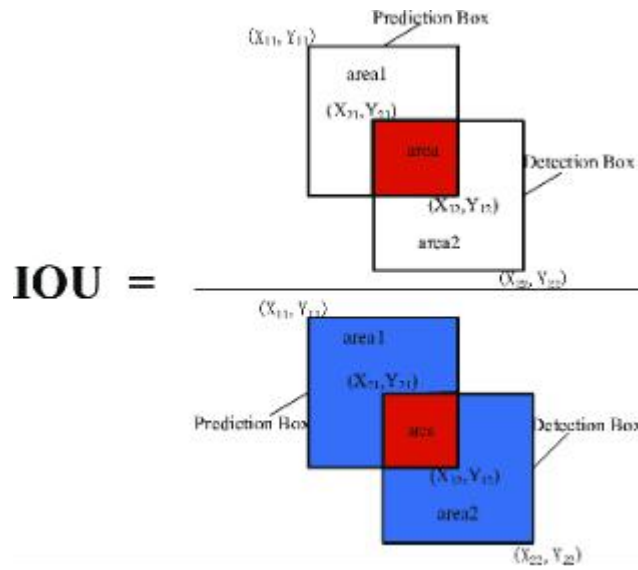


Figure 5. Description of the IoU index.

During the test phase, the system will produce many different bounding boxes with different prediction probabilities and different IoU ratios, so the Non-max suppression algorithm helps to eliminate the bounding boxes with low prediction rates. and keep only the last 1 bounding box with the highest prediction rate. The Non-max suppression algorithm goes like this:

Step 1: Remove all bounding boxes where the probability of occurrence of the pc object is lower than the threshold. Such removal so that grids that do not contain objects have a low probability of object occurrence will not display the bounding box.

Step 2: Select the bounding boxes with the highest probability of object occurrence.

Step 3: If there are multiple bounding boxes with the same highest probability of occurrence of the object, we will remove by IoU, the bounding box with IoU index lower than the threshold will be discarded. At the end of step three, we will get the bounding box with the best object recognition ratio.

2.4 Label the patterns

Similar to "Machine Learning" / "Deep Learning" problems, the object recognition problem also includes 2 phases, a test phase and a training phase. In the learning phase of the YOLO algorithm, we will perform location and classification labels for the object, then y has the following form:

$$y = [pc, bx, by, bh, bw, c1, c2, c3, \dots]$$

For example for Figure 2-5, suppose the problem needs to find the object location and classify with 3 labels, in which the sign of the figure belongs to the first label. Then, in the grid that detects traffic signs, y will be assigned as follows $y = [1, 230, 120, 20, 30, 1, 0, 0]$, ie that grid has objects, coordinates (x, y) in the upper left of the anchor box is $(230, 120)$, the anchor box has a length of 20 and a width of 30, the sign belongs to label 1 and not to the other two labels. If the grid does not contain objects y will be assigned $y = [0, x, x, x, x, x, x, x]$ ie there are no objects in that grid and the remaining values do not matter.

However, if the same 2 objects appear in the same grid, the same assignment cannot be performed. With that problem, we can handle it in 2 ways:

Method 1: Subdivide the grid until 2 objects are in 2 different grids. However, if the grid is divided as small, the learning cannot take place deeply, making it difficult to classify objects accurately because the following layers do not learn high features. And if the centers of the two objects are almost the same, it is also impossible. solved as above. Then we have to do the 2nd way.

Method 2 : Instead of y being assigned to only 1 object, y will be extended to many objects. Consider the example with Figure 2-7 below, the girl and the car are on the same grid. Then y will be assigned as follows $y = [1, 120, 20, 20, 120, 1, 0, 1, 90, 50, 90, 20, 0, 1]$. That is, in this grid, there are cars ($y[0] = 1$), $(120, 20, 20, 120)$ are 4 anchor box parameters of cars, ($y[5]=1, y[6]=0$) to classify this anchor box as a car, not a girl, similarly $y[7]=1$ means this grid also has a girl, the next 4 parameters to determine the anchor box coordinates for her girl and ($y[12]=0, y[13]=1$) to classify this anchor box as a girl, not a car. Thus, with method 2 we will combine the

y of 2 or more objects in the same grid into one, $y[0:6]$ to determine for detecting the vehicle, and $y[7:13]$ to determine intended for the girl. However, with the second way, if we combine as much processing speed as possible, the more operations we perform, the more we should not abuse this way, but we should coordinate it harmoniously with the first method, increasing the number of operations. grid must split.

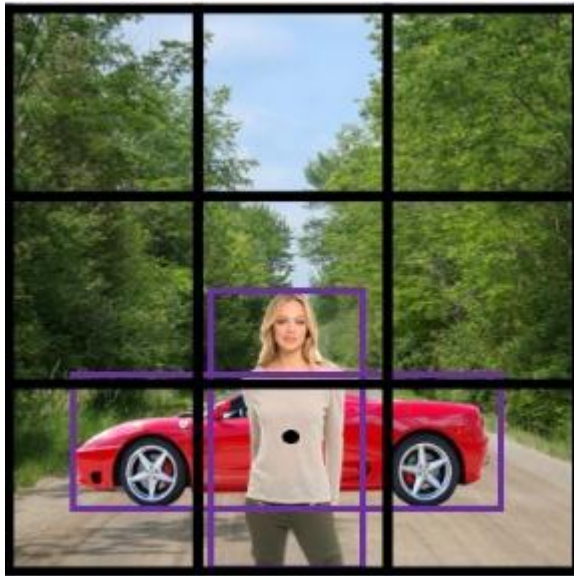


Figure 6. The centers of the two objects coincide and lie in the same grid.

2.5 Features of YOLOv8

[Ultralytics YOLOv8](#) is a cutting-edge, state-of-the-art (SOTA) model that builds upon the success of previous YOLO versions and introduces new features and improvements to further boost performance and flexibility. YOLOv8 is designed to be fast, accurate, and easy to use, making it an excellent choice for a wide range of object detection and tracking, instance segmentation, image classification and pose estimation tasks.

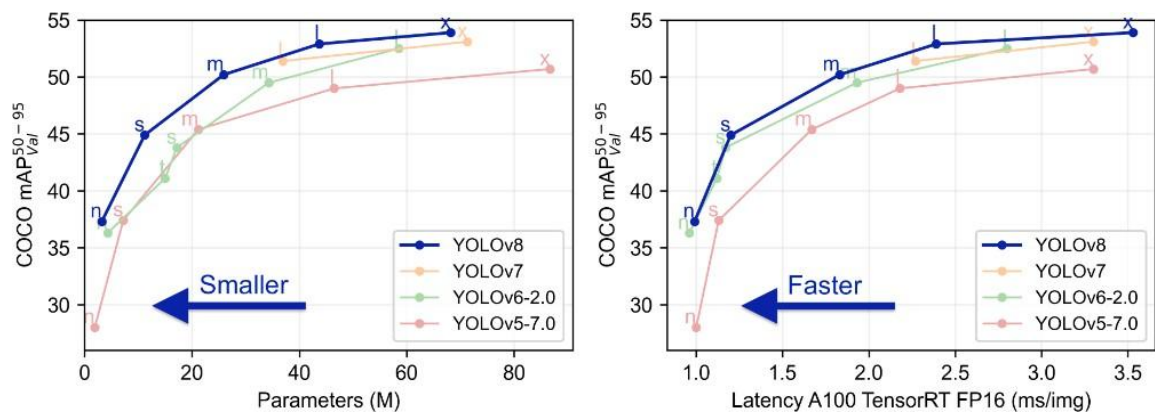


Figure 7. Performance of YOLOv8 compared with older versions

Model : All YOLOv8 pretrained models are available. Detect, Segment and Pose models are pretrained on the [COCO](#) dataset, while Classify models are pretrained on the [ImageNet](#) dataset.

▼ Detection

See [Detection Docs](#) for usage examples with these models.

Model	size (pixels)	mAP ^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

- mAP^{val} values are for single-model single-scale on [COCO val2017](#) dataset.
Reproduce by `yolo val detect data=coco.yaml device=0`
- Speed averaged over COCO val images using an [Amazon EC2 P4d](#) instance.
Reproduce by `yolo val detect data=coco128.yaml batch=1 device=0|cpu`

► Segmentation

► Classification

► Pose

Figure 8. YOLOv8 models for detection

YOLOv8 does not yet have a published paper, so we lack direct insight into the direct research methodology and ablation studies done during its creation. With that said, we analyzed the repository and information available about the model to start documenting what's new in YOLOv8.

The following image made by GitHub user RangeKing shows a detailed visualisation of the network's architecture

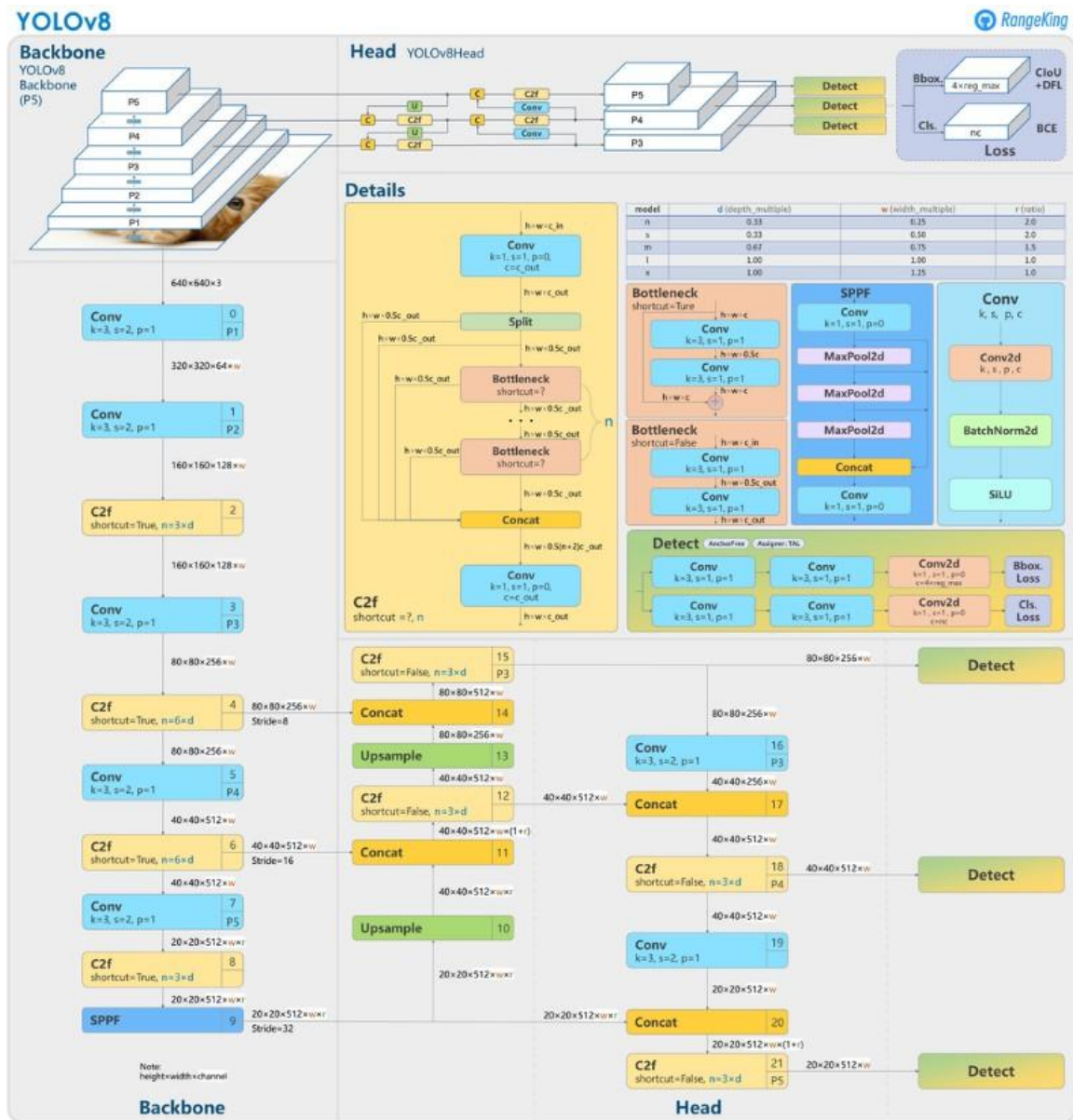


Figure 9. The architecture of YOLOv8

CHAPTER 3: SYSTEM DESIGN

3.1 Steps to be able to train YOLOv8 on Custom Dataset:

- Prepare the Dataset.
- Install necessary YOLOv8 dependencies.
- Download Custom YOLOv8 Object Detection data.
- Define configuration and architecture model YOLOv8.
- Train a custom YOLOv8 Detector.
- YOLOv8 performance evaluation.
- Visualization of YOLOv8 training data.
- Run YOLOv8 Inference on test webcam tracking.
- Export the saved YOLOv8 weights to infer the results.

3.2 Prepare the training data folder

The steps include:

- First create a folder named data's name is coco128.
- The datasets used is the coco128.zip file, which includes 2376 different image samples and has 20 classes that are the names of 20 dishes for training.

1_Banh beo	11/26/2023 10:51 AM	File folder
2_Banh chung	11/26/2023 10:51 AM	File folder
3_Banh cuon	11/26/2023 10:51 AM	File folder
4_Banh gio	11/26/2023 10:51 AM	File folder
5_Banh khot	11/26/2023 10:51 AM	File folder
6_Banh mi	11/26/2023 10:51 AM	File folder
7_Banh pia	11/26/2023 10:51 AM	File folder
8_Banh tet	11/26/2023 10:51 AM	File folder
9_Banh trang nuong	11/26/2023 10:51 AM	File folder
10_Banh xeo	11/26/2023 10:51 AM	File folder

Figure 10. The Dataset folder

In the data folder for training, there will be 2 folder structures, images and labels, the images folder to store traffic sign images and the labels folder to identify the name labels attached to the images to aid in diagnosis. or identify the exact sign. The folder structure of the images and labels files is saved as shown below

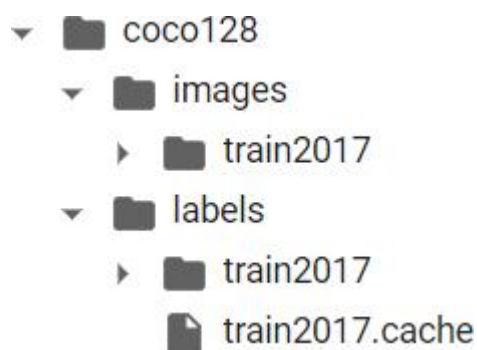


Figure 11. Dataset folder structure

If there are unlabeled images, you will need to label them first just upload the images you want to caption, caption the images and export the labels as shown below.

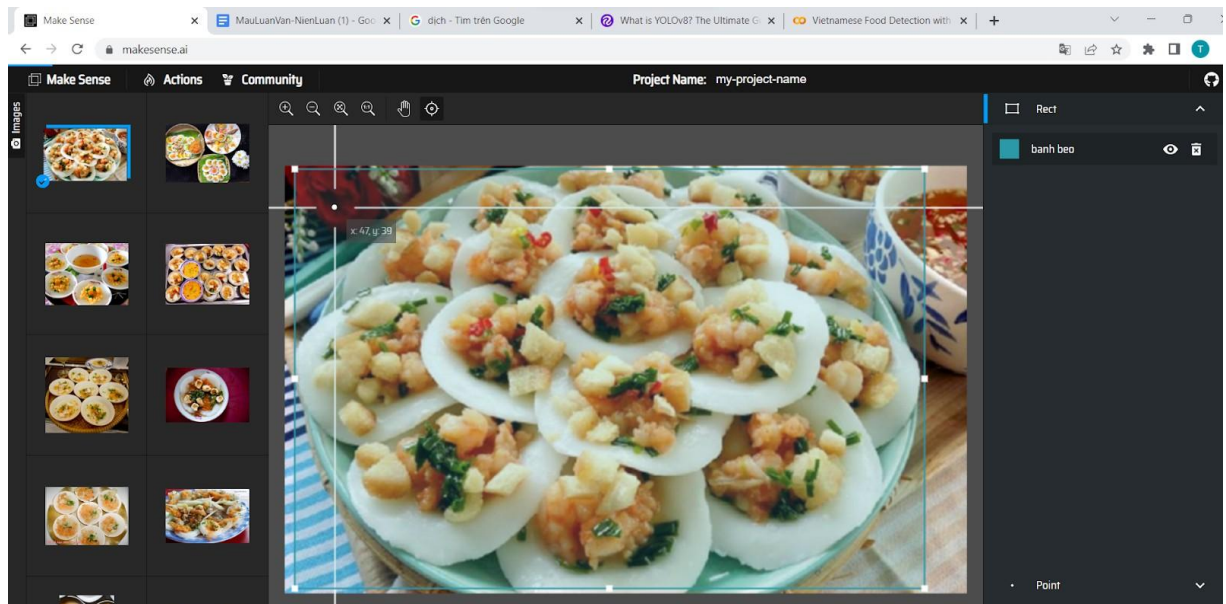


Figure 12. Labeling shapes

After labeling the signs, we will go to Actions and download the A.zip (package containing files formatted in YOLO)

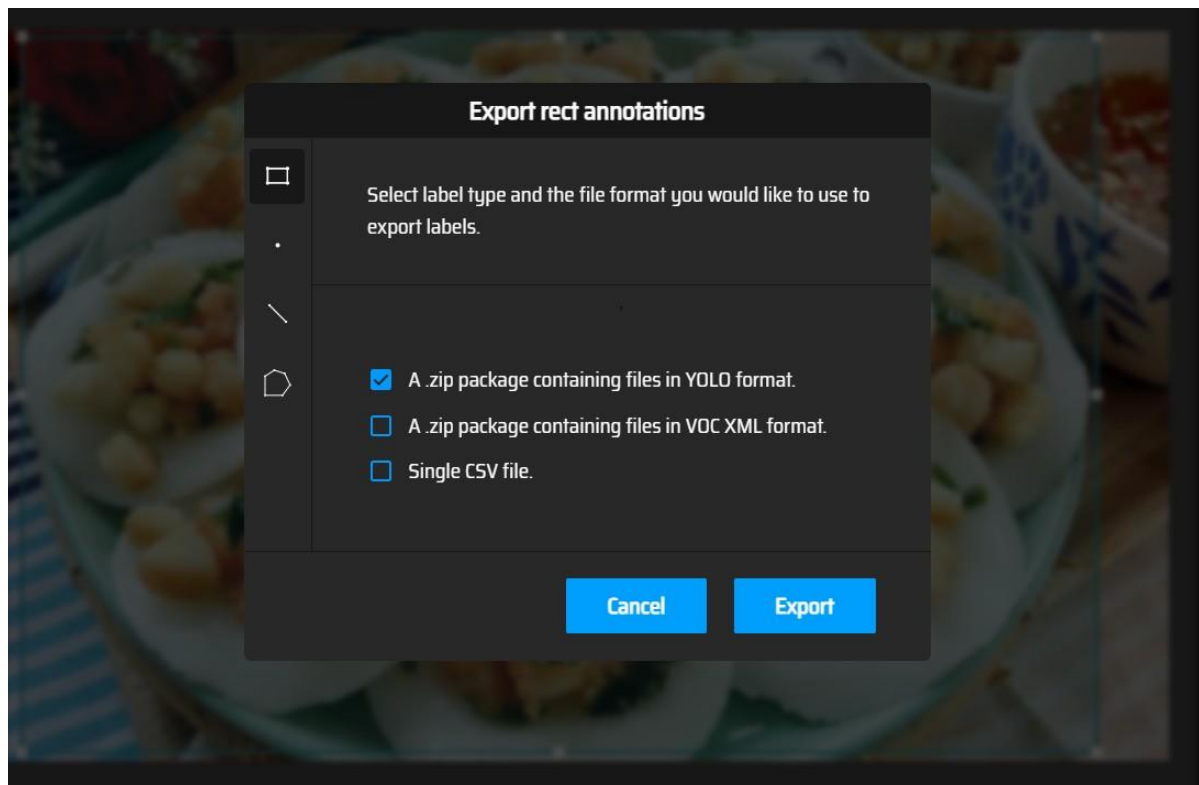
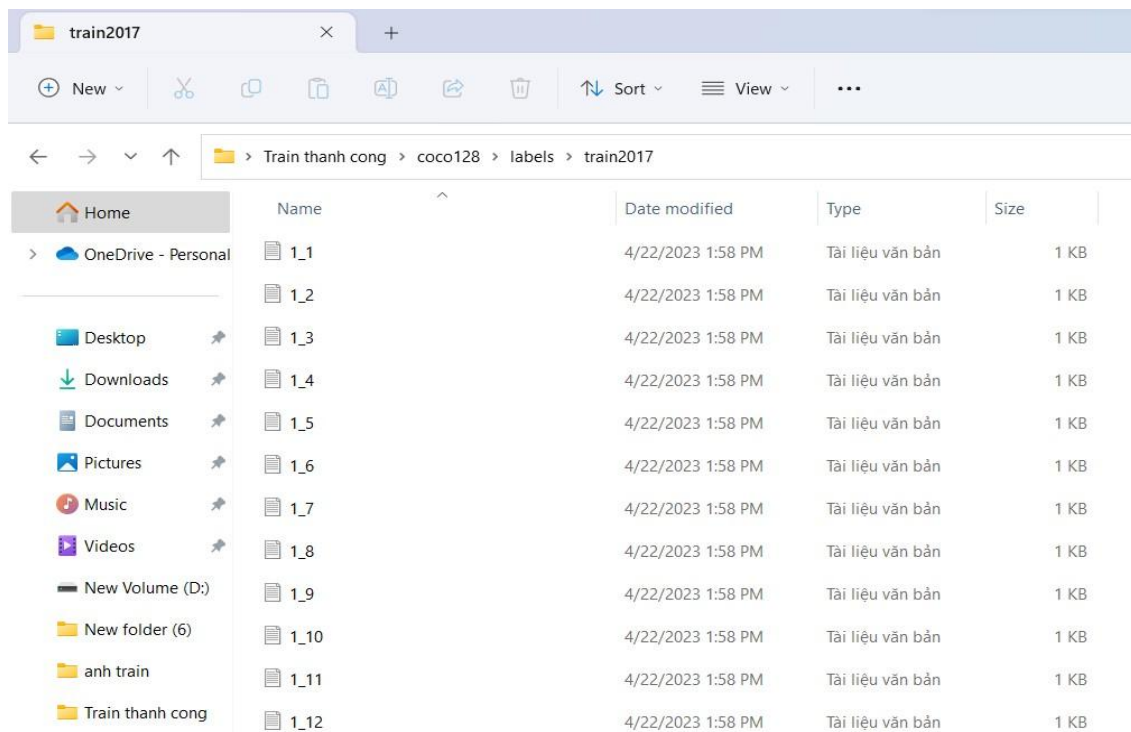


Figure 13. Downloading .zip package

Proceed to extract the file.zip to the labels folder, we get the Text Document files



The .txt file will have the following format:

- Each row will be an object.
- Each row will have format: class x_center y_center width height.
- The coordinates of the boxes will be normalized (from 0-1) in the format xywh.
- Class will start from 0 to 19 presented for 20 objects.

3.3 Setting up an environment for training and identifying Vietnamese dishes

To get started with YOLOv8, first clone the YOLOv8 repository and install the dependencies on the Google Colab environment. This will set up our programming environment ready to run training and inference commands:

▼ Setup

Pip install `ultralytics` and [dependencies](#) and check software and hardware.

```
[1] %pip install ultralytics
import ultralytics
ultralytics.checks()
```

Ultralytics YOLOv8.0.86 Python-3.9.16 torch-2.0.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Setup complete (2 CPUs, 12.7 GB RAM, 23.3/78.2 GB disk)

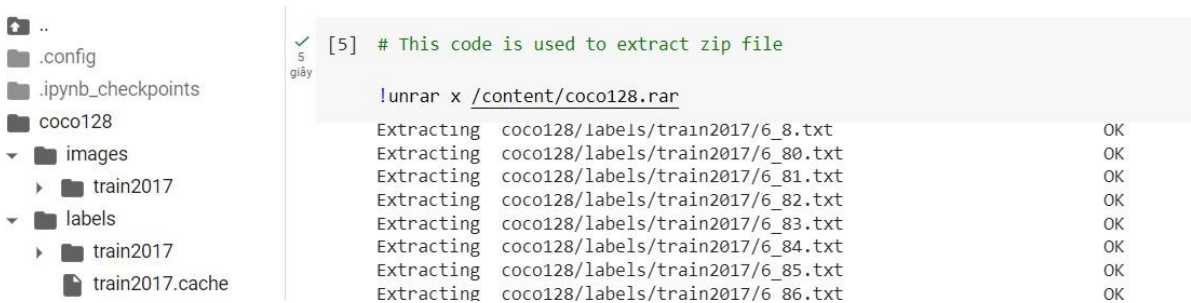
Figure 14. Set up model YOLOv8

The export produces a .yaml YOLOv8 file named data.yaml that specifies the location of the YOLOv8 images folder, the YOLOv8 label folder, and information about custom classes. Proceed to reconfigure the class label and the path to the coco128 dataset file.

```
1 # Ultralytics YOLO 🚀, AGPL-3.0 license
2 # COCO128 dataset https://www.kaggle.com/ultralytics/coco128 (first 128 images from COCO train2017) by Ultralytics
3 # Example usage: yolo train data=coco128.yaml
4 # parent
5 # └─ ultralytics
6 #   └─ datasets
7 #     └─ coco128 ← downloads here (7 MB)
8
9
10 # Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
11 path: ../coco128 # dataset root dir
12 train: images/train2017 # train images (relative to 'path') 128 images
13 val: images/train2017 # val images (relative to 'path') 128 images
14 test: # test images (optional)
15
16 # Classes
17 names:
18 0: Banh Beo
19 1: Banh Chung
20 2: Banh Cuon
21 3: Banh Gio
22 4: Banh Khot
23 5: Banh Mi
24 6: Banh Pia
25 7: Banh Tet
26 8: Banh Trang Nuong
27 9: Banh Xeo
```

Figure 15. Configuring the .yaml file

After configuring the coco128.yaml file and editing the label and class names, then upload the dataset folder coco128.rar to Colab and extract it to conduct training on the custom dataset.



The screenshot shows the Google Colab interface. On the left, the file explorer displays the directory structure: a root folder containing .config, .ipynb_checkpoints, coco128, images, and labels. The images folder contains train2017, and the labels folder contains train2017 and train2017.cache. On the right, the terminal shows the execution of a command to extract a zip file. The command is `!unrar x /content/coco128.rar`. The output shows the extraction of 12 files, each with a status of 'OK'.

```
[5] # This code is used to extract zip file
!unrar x /content/coco128.rar
Extracting coco128/labels/train2017/6_8.txt OK
Extracting coco128/labels/train2017/6_80.txt OK
Extracting coco128/labels/train2017/6_81.txt OK
Extracting coco128/labels/train2017/6_82.txt OK
Extracting coco128/labels/train2017/6_83.txt OK
Extracting coco128/labels/train2017/6_84.txt OK
Extracting coco128/labels/train2017/6_85.txt OK
Extracting coco128/labels/train2017/6_86.txt OK
```

Figure 16. Upload and extract the dataset file

3.4 Training

After preparing and uploading the dataset folder with the name coco128 , configuring the coco128.yaml file, we proceed to use the yolov8n.pt model to train with the custom dataset.

Train parameters with epoch of 17 and image size of 640.

```
# Train YOLOv8n on COCO8 for 17 epochs
!yolo train model=yolov8n.pt data=coco128.yaml epochs=17 imgsz=640
```

YOLOv8 Tutorial - Colaboratory

colab.research.google.com/github/ultralytics/ultralytics/blob/main/examples/tutorial.ipynb#scrollTo=1NcFxFdJ_O

YOLOv8 Tutorial

Tệp | Chính sửa | Xem | Chèn | Thời gian chạy | Công cụ | Trợ giúp | Không thể lưu các thay đổi

Tệp

- ..
- .config
- sample_data

```
# Train YOLOv8n on COCO8 for 3 epochs
!yolo train model=yolov8n.pt data=coco128.yaml epochs=17 imgsz=640
```

Logging results to runs/detect/train4

Starting training for 17 epochs...

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/17	2.49G	0.6994	3.694	1.306	28	640: 100% 149/149 [01:00<00:00, 2.48it/s]
Class		Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 75/75 [00:32<00:00, 2.31it/s]
all		2376	2575	0.264	0.393	0.253 0.202
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/17	2.36G	0.7077	2.601	1.295	27	640: 100% 149/149 [00:57<00:00, 2.59it/s]
Class		Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 75/75 [00:30<00:00, 2.48it/s]
all		2376	2575	0.474	0.605	0.569 0.439
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/17	2.36G	0.6935	2.085	1.279	25	640: 100% 149/149 [00:57<00:00, 2.58it/s]
Class		Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 75/75 [00:28<00:00, 2.67it/s]
all		2376	2575	0.616	0.627	0.673 0.509
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
4/17	2.36G	0.667	1.809	1.252	29	640: 100% 149/149 [00:58<00:00, 2.56it/s]
Class		Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 75/75 [00:27<00:00, 2.76it/s]
all		2376	2575	0.695	0.703	0.761 0.596
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
5/17	2.36G	0.6616	1.714	1.246	20	640: 100% 149/149 [00:57<00:00, 2.60it/s]
Class		Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 75/75 [00:29<00:00, 2.51it/s]
all		2376	2575	0.736	0.706	0.772 0.616
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
6/17	2.34G	0.6261	1.505	1.22	21	640: 100% 149/149 [00:57<00:00, 2.61it/s]

Đã kết nối với Python 3 Phần phụ trợ Google Compute Engine (GPU)

Figure 17. Training on YOLOv8.

CHAPTER 4. RESULTS

Train results are saved by Google Colab in the directory runs/detect/train

With the train result statistics file as well as the model file named best.pt

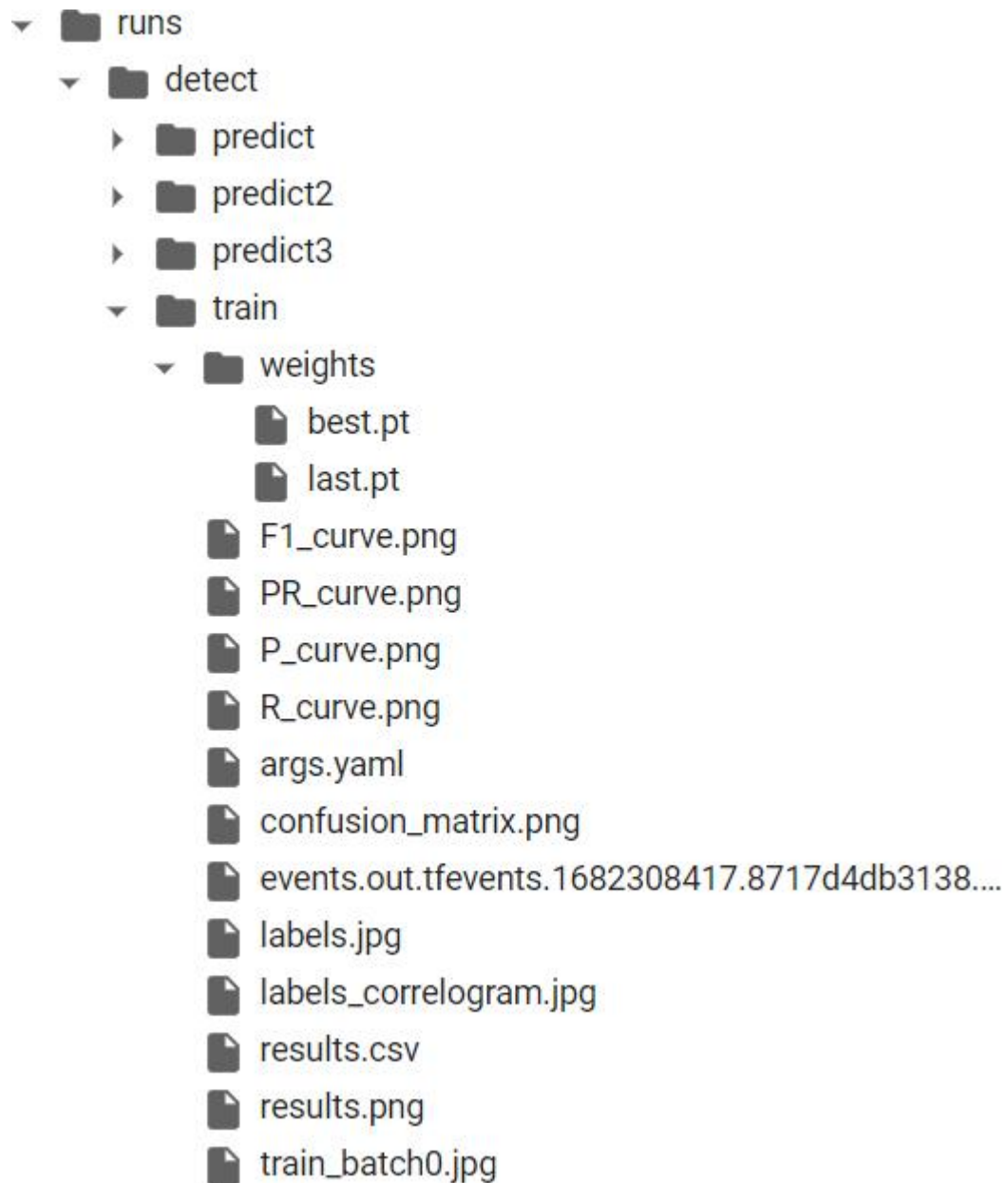


Figure 18. results directory after training.

Next step is setting up the Pycharm environment and perform random images prediction with the realtime webcam tracking.

Predicted results:

```
main.py x
1 from ultralytics import YOLO
2
3 model = YOLO("C:/Users/Tran Thien Phuc/Desktop/train17e.pt")
4
5 #results = model("0", show=True)
6 results = model("0", show=True) # predict realtime with webcam on an image
```

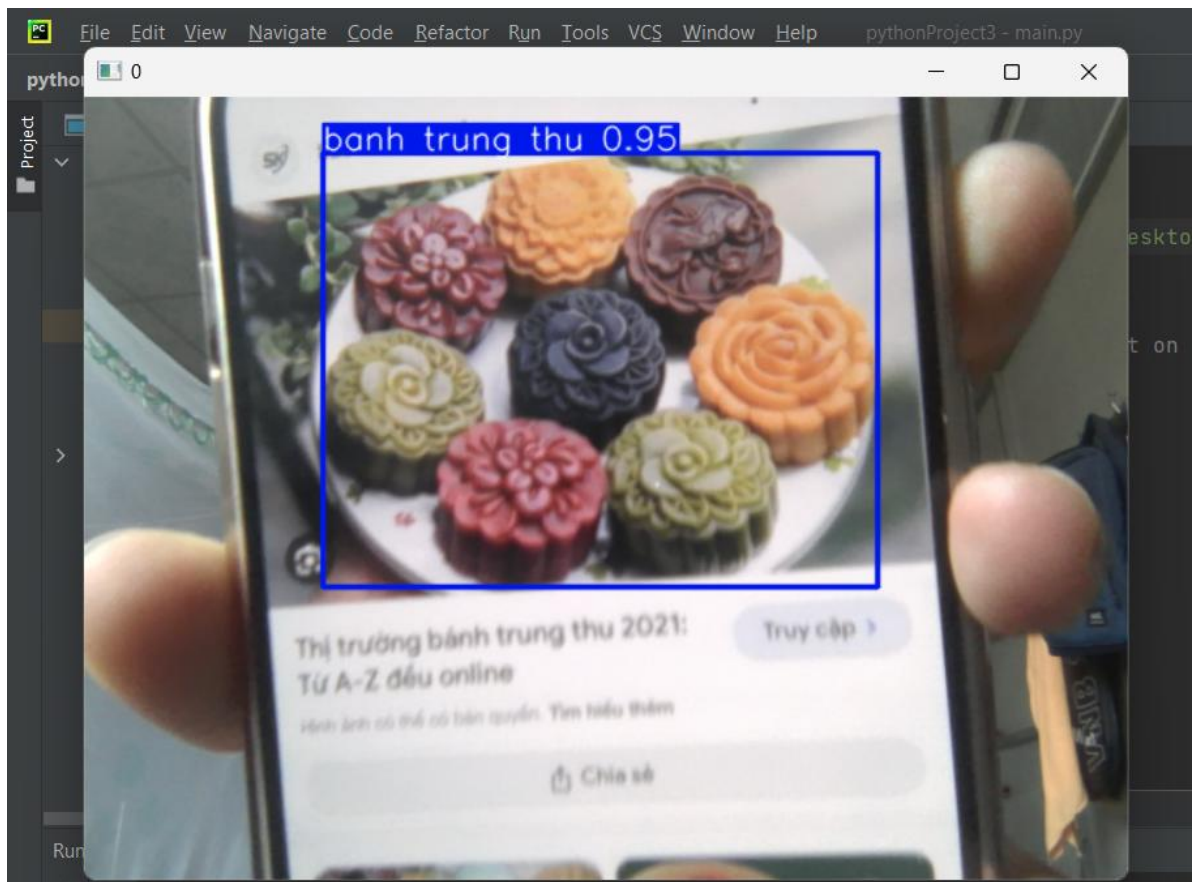


Figure 19. predicted results.

CHAPTER 5 CONCLUSIONS AND DEVELOPMENT ORIENTATIONS

5.1 Conclude

Initially, a training model was built to detect and recognize signs by using algorithms of the YOLOv8 network to assist in detecting and identifying Vietnamese dishes. Use Google Colab - Powerful Platform to build and train machine learning models, save time with GPU provided by Google. In addition, PyCharm will support realtime webcam tracking environment that Colab does not support.

5.2 Advantage

The identification and probability results of the image classification model predicting traffic signs are quite accurate and relatively stable. The average reliability of the prediction system with 20 classes is 70% or more. The outcome of this process depends heavily on the data collection and training, detection, and identification process.

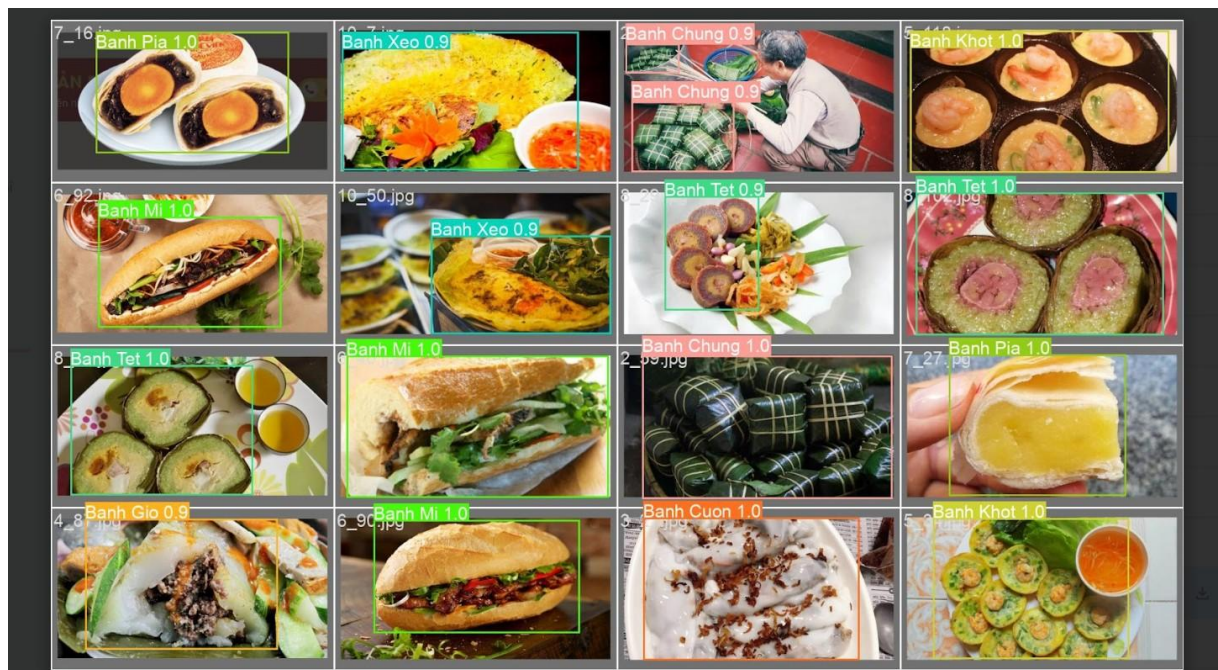


Figure 20. prediction results after training with dataset.

5.3 Disadvantage

Due to the time to learn and build the system as well as limited knowledge, only 20

different types of Vietnamese dishes have been tried, in addition to the topic being a large research area, the system has just stopped working because of hardware limit. At the research level, it is not possible to put into practical application and not fully functional to reach users.

More research time is needed to perfect the program, more image sources can be found online to train and identify many other dishes, besides, the topic can be applied in real life.

5.4 Development

Based on today's strong technology development, in the future it is possible to use the latest versions of YOLO and many other networks such as Single Shot Detector, RentiaNet, CenterNet, etc. to train and identify many groups with more accurately and quickly to:

- Improve the ability to identify dishes with similar characteristics (such as banh chung and banh tet all wrapped in the same leaf, same color, ...)
- Expand the database of Vietnamese dishes.
- Increased accuracy in a variety of lighting and weather conditions.
- Upgrade and perfect the system's ability to become a standard model that can accurately predict Vietnamese dishes.

APPENDIX

Quoting the program source code of the topic:

▼ Setup

Pip install ultralytics and [dependencies](#) and check software and hardware.

```
[ ] %pip install ultralytics
import ultralytics
ultralytics.checks()
```

Ultralytics YOLOv8.0.86 🚀 Python-3.9.16 torch-2.0.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 23.3/78.2 GB disk)

▼ Predict

YOLOv8 may be used directly in the Command Line Interface (CLI) with a `yolo` command for a variety of tasks and modes and accepts additional arguments, i.e. `imgsz=640`. See a full list of available `yolo` [arguments](#) and other details in the [YOLOv8 Predict Docs](#).

```
# Train YOLOv8n on COCO8 for 17 epochs
!yolo train model=yolov8n.pt data=coco128.yaml epochs=17 imgsz=640
```

```
1 # Ultralytics YOLO 🚀, AGPL-3.0 license
2 # COCO128 dataset https://www.kaggle.com/ultralytics/coco128 (first 128 images from COCO train2017) by Ultralytics
3 # Example usage: yolo train data=coco128.yaml
4 # parent
5 # └─ ultralytics
6 #   └─ datasets
7 #     └─ coco128 ← downloads here (7 MB)
8
9
10 # Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
11 path: ../coco128 # dataset root dir
12 train: images/train2017 # train images (relative to 'path') 128 images
13 val: images/train2017 # val images (relative to 'path') 128 images
14 test: # test images (optional)
15
16 # Classes
17 names:
18 0: Banh Beo
19 1: Banh Chung
20 2: Banh Cuon
21 3: Banh Gio
22 4: Banh Khot
23 5: Banh Mi
24 6: Banh Pia
25 7: Banh Tet
26 8: Banh Trang Nuong
27 9: Banh Xeo
```

REFERENCES

[1] WHAT IS YOLOv8 , <https://blog.roboflow.com/whats-new-in-yolov8/>

[2] CÁCH TRAIN CUSTOM DATASET TRÊN COLAB 2023
<https://www.youtube.com/watch?v=a5rwtCgVWGM>

[3] MODEL YOLOv8 , <https://github.com/ultralytics/ultralytics>

[4] HOW TO TRAIN YOLOv8 ON A CUSTOM DATASET

<https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/>

[5] Joseph Redmon, Santosh Divvalay, Ross Girshick, Ali Farhadiy, You Only Look Once: Unified, Real-Time Object Detection, University of Washington, 2016.