Milestone completion report	
Pool	New Projects
Project name	MeAI tool for healthcare system
Project ID	R3-NEW-10
Report Milestone	2
URL	https://proposals.deepfunding.ai/collaborate/f1d8eb48-62a0-4da7-b700-51323626b228
Date	13 Mar 2024

3. R&D AI models

3.1. R&D object detection models

With the deep understanding about the problem and the data, we do the R&D task for researching the state-of-the-art object detection models. They consist of, but not limited to, the following models:

a. RetinaNet

RetinaNet is a popular object detection model that introduced a novel approach to tackle the challenge of detecting objects at various scales and overcoming the foreground-background class imbalance issue that is prevalent in object detection tasks. Here are some key points about RetinaNet:

- ***Focal Loss:*** The core innovation of RetinaNet is the introduction of the focal loss function, designed to address the class imbalance problem by down-weighting the loss assigned to well-classified examples. This allows the model to focus more on hard, misclassified examples during training, improving performance on detecting objects.
- ***Feature Pyramid Network (FPN):*** RetinaNet uses a Feature Pyramid Network as its backbone for extracting features at multiple scales. The FPN improves the model's ability to detect objects at different sizes by building a multi-scale pyramid of the original image, ensuring that objects of various sizes can be detected more effectively.
- ***Two-Stage Detection in a Single Network:*** Although RetinaNet operates in a single, unified network, it conceptually integrates the benefits of both one-stage detectors (like YOLO and SSD, which are fast) and two-stage detectors (like Faster R-CNN, which are more accurate). It achieves this by efficiently detecting objects at multiple scales using the FPN and applying the focal loss to balance speed and accuracy.

The introduction of RetinaNet and its focal loss function has had a significant impact on the field of object detection, inspiring further research and development in addressing the challenges of scale and class imbalance in object detection tasks.

b. Faster R-CNN

Faster R-CNN is a highly influential model in the field of object detection, representing a significant advancement in the ability to detect objects within images with high accuracy. Here are some key features of Faster R-CNN:

- ***Region Proposal Network (RPN):*** Faster R-CNN introduces the Region Proposal Network, which efficiently generates region proposals with a wide range of scales and aspect ratios. The RPN shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals.
- ***Two-Stage Process***: The model operates in two stages. The first stage, run by the RPN, proposes candidate object bounding boxes. The second stage uses these proposals to classify the objects within the boxes and refine their positions.
- ***Shared Convolutional Features:*** Faster R-CNN efficiently shares convolutional features between the RPN and the detection network, significantly reducing computation time and making the process of detecting objects faster compared to previous models like Fast R-CNN and R-CNN.
- ***End-to-End Training:*** The model can be trained end-to-end with a multi-task loss that combines the losses of the RPN and the detection network, allowing for simultaneous training of the network to propose regions and detect object classes and box offsets.
- ***High Accuracy***: Faster R-CNN achieves high accuracy in object detection on benchmark datasets such as Pascal VOC and COCO, making it a powerful tool for applications requiring precise object localization and classification.

Faster R-CNN, with its two-stage process, is generally more accurate but slower compared to one-stage detectors like SSD and YOLO. RetinaNet aims to bridge this gap by offering a one-stage detector that approaches the accuracy of two-stage detectors like Faster R-CNN, while being faster to compute.

c. Mask R-CNN

Mask R-CNN is an extension of Faster R-CNN, a popular framework for object detection, which adds a branch for predicting segmentation masks on each Region of Interest (RoI), effectively allowing the model to perform instance segmentation. Here are some key points about Mask R-CNN:

- ***Instance Segmentation***: Unlike object detection, which involves identifying bounding boxes around objects and classifying them, instance segmentation goes further by generating pixel-wise masks for each object instance. Mask R-CNN achieves this by adding a mask branch to the Faster R-CNN framework, enabling it to predict segmentation masks for objects in addition to their bounding boxes and classifications.
- ***Architecture***: Mask R-CNN inherits the two-stage structure of Faster R-CNN: the first stage proposes candidate object bounds, and the second stage, which now includes the mask branch, classifies those objects and predicts their bounding boxes and masks. The mask prediction is parallel to the existing branch for bounding box recognition and classification.
- ***Region of Interest Align (RoIAlign)***: A key innovation in Mask R-CNN is the RoIAlign layer, which corrects the misalignment introduced by the RoIPool in Faster R-CNN. RoIAlign accurately preserves spatial locations through bilinear interpolation, significantly improving mask accuracy.
- ***Multi-Task Loss***: The training of Mask R-CNN involves a multi-task loss on each sampled RoI, which combines the losses of classification, bounding box regression, and mask prediction. This multi-task approach enables the model to learn more robust features that are beneficial for both detection and segmentation.
- ***Performance***: At its introduction, Mask R-CNN demonstrated state-of-the-art performance on several benchmarks for instance segmentation, including the challenging COCO dataset. Its effectiveness comes from both its architectural innovations and its ability to learn rich feature representations for various tasks beyond mere object detection.

Mask R-CNN represents a significant advancement in computer vision, particularly in the area of instance segmentation, by efficiently extending Faster R-CNN with minimal modifications and introducing innovations like RoIAlign that have broader implications for object recognition tasks.

d. YOLO

YOLO, which stands for "You Only Look Once," is a series of object detection models that revolutionized the field by proposing a single-stage detection algorithm that significantly speeds up the process of detecting objects while maintaining high accuracy. Here are some key points about YOLO:

- ***Single-Stage Detection:*** Unlike two-stage detectors like Faster R-CNN, which first propose regions and then classify them, YOLO frames object detection as a single regression problem, directly predicting bounding boxes and class probabilities from full images in one evaluation. This

approach significantly increases the speed of detection, making YOLO suitable for real-time applications.

- ***Architecture:*** YOLO divides the image into a grid and predicts bounding boxes and probabilities for each grid cell. The predictions are made for multiple classes simultaneously, allowing the model to detect various objects in one pass. Over the years, there have been several versions of YOLO, each improving upon the previous in terms of speed, accuracy, and efficiency.
- ***Speed and Efficiency:*** One of the hallmark features of YOLO is its speed. It can process images in real-time, making it an excellent choice for applications that require fast object detection, such as video surveillance, vehicle detection in autonomous driving, and real-time tracking.

YOLO's development and iterations showcase the rapid advancements in object detection technology, balancing speed and accuracy to cater to real-time processing needs. Its innovative approach has influenced many subsequent models and applications in the field of computer vision.

Here's a comparison of the models RetinaNet, Faster R-CNN, Mask R-CNN, and YOLO based on key features, detection type, speed vs. accuracy, and suitable applications:

Model	Key Features	Detection Type	Speed vs Accuracy	Suitable Applications
RetinaNet	Focal loss to address class imbalance, uses FPN for detecting objects at multiple scales	Object Detection	Balances speed and accuracy, faster than two-stage models but with competitive accuracy	Scenarios where both speed and accuracy are needed
Faster R-CNN	Two-stage process with RPN for region proposals, high accuracy	Object Detection	More accurate but slower than one-stage detectors	Applications requiring high accuracy, such as complex detection tasks
Mask R-CNN	Extends Faster R-CNN with a branch for pixel-wise	Object Detection and Instance Segmentation	Similar to Faster R-CNN but with additional overhead for	Detailed scene understanding, like autonomous driving, where instance

	object mask prediction, uses RoIAlign		mask prediction	segmentation is needed
YOLO	Single-stage detection, real-time processing, direct bounding box and class probability prediction	Object Detection	Designed for real-time processing, prioritizes speed	Real-time applications, such as video surveillance and autonomous vehicles

3.2. Do experiments to find the most suitable models

In object detection, there are various metrics used to evaluate the performance of a detection model. Among these metrics, MAP (Mean Average Precision) is a commonly used one. It measures how accurate and precise the detections are across different object classes.

- MAP (Mean Average Precision): It calculates the average precision for each class and then averages them across all classes. Higher MAP values indicate better performance.

Now, there are variations of MAP that are prefixed with numbers, such as MAP50 and MAP75. These variations denote different thresholds for what is considered a "correct" detection.

- MAP50: This metric calculates the average precision considering only detections where the intersection over union (IoU) between the predicted bounding box and the ground truth bounding box is greater than or equal to 0.5. IoU measures the overlap between two bounding boxes. So, MAP50 focuses on relatively accurate detections where there's at least a 50% overlap between the predicted and ground truth bounding boxes.

- MAP75: Similarly, MAP75 calculates the average precision but considering detections with an IoU greater than or equal to 0.75. This means MAP75 focuses on even more accurate detections, where there's at least a 75% overlap between the predicted and ground truth bounding boxes.

These metrics help assess the performance of object detection models at different levels of precision. They provide insights into how well the model localizes and identifies objects within images or videos. Higher values of MAP50 and MAP75 indicate better performance, especially in scenarios where accurate object localization is crucial.

Inference time in deep learning refers to the amount of time it takes for a trained neural network model to make predictions on new data. This is a critical performance metric, especially in applications requiring real-time

responses, such as voice recognition, video analysis, and autonomous driving.

We conducted experiments on the POLYP dataset, which includes the KVASIR, CVC-CLINICDB, CVC-300 and CVC-COLONDB datasets, with a train:test ratio of 8:2.

Dataset	Images	Characteristics
KVASIR	1000	The dataset comprises gastrointestinal endoscopy images captured from various parts of the gastrointestinal tract, such as the esophagus, stomach, and colon. It contains images depicting various conditions and abnormalities, including polyps, ulcers, and inflammation. Due to its diverse nature, it is likely to include a considerable number of images covering a wide range of gastrointestinal pathologies.
CVC-CLINICDB	612	This dataset primarily focuses on colonoscopy images obtained during colonoscopy procedures. It contains images representing various pathologies and abnormalities found in the colon, such as polyps, tumors, and inflammation. Given its emphasis on colonoscopy, it is expected to include a substantial number of images specifically targeting colon-related conditions.
CVC-COLONDB	380	The CVC-COLONDB dataset is designed for analyzing colonoscopy videos, including both images and video sequences obtained during colonoscopy examinations. It serves tasks related to lesion detection, tracking, and segmentation in the context of colorectal diseases. The dataset likely includes a significant number of images extracted from colonoscopy videos, providing a rich resource for studying and developing algorithms for colon-related pathologies.
CVC-300	60	CVC-300 consists of images obtained from various medical imaging modalities such as X-rays, MRIs, CT scans, or endoscopy. They may encompass a range of medical conditions, anatomical structures, and imaging variations, providing researchers with diverse data to develop and evaluate algorithms for medical image analysis tasks.

Below are the results obtained.

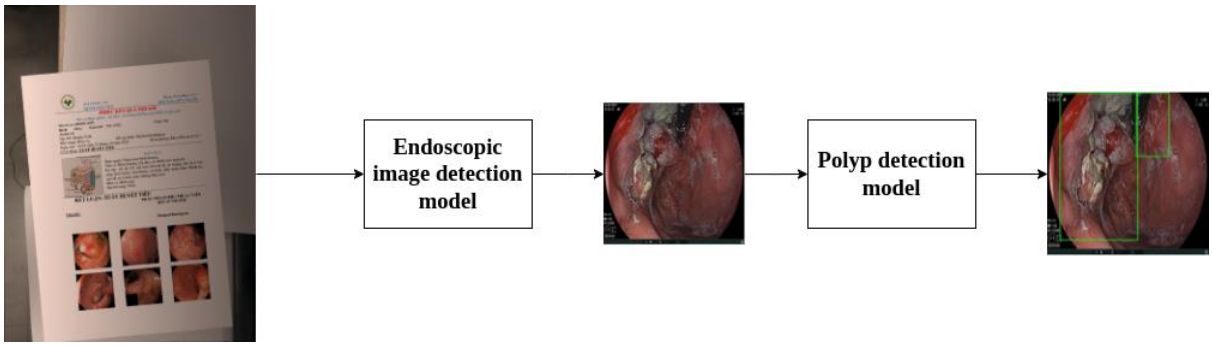
Model	mAP	mAP50	mAP75	Inference time
RentinaNet	0.7671	0.9673	0.9012	106.2 ms
Faster R-CNN	0.7200	0.9422	0.8682	103.5 ms
Mask R-CNN	0.7286	0.9409	0.8912	95.5 ms
YOLO	0.7433	0.9421	0.8608	**2.6 ms**

⇒ YOLO has superior inference time compared to other models, making it suitable for applications that require real-time processing.

4. Training AI models

4.1. Training AI models with the prepared data

We will use two models: one model will be used to crop endoscopic images from the result sheet, and another model will take the cropped endoscopic images as input. The second model will detect polyps in those images and output the bounding box coordinates enclosing the polyp.

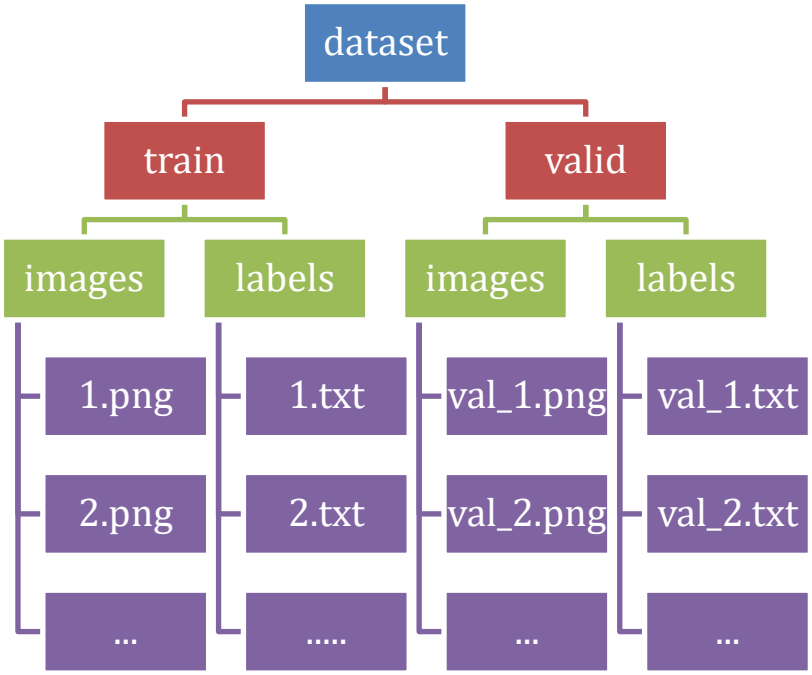


a. Endoscopic image detection model

The purpose of the Endoscopic image detection model is to detect endoscopic images in endoscopy result sheets. The output of this step is cropped endoscopic images, which will serve as input for the polyp detection model in the next step.

Step 1: Prepare dataset

Data [Here] is organized in the form as following :



With label format is:

<class-index> <x> <y> <width> <height> <px1> <py1> <px2> <py2> ... <pxn> <pyn>

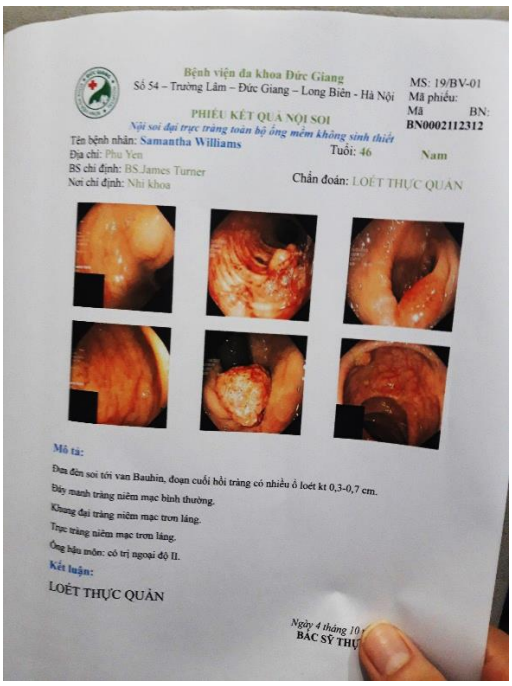
Step 2: Prepare config file

```
# 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, ..]

path: ./datasets/lesion/  # dataset root dir

train: train  # train images (relative to 'path') 4 images

val: valid  # val images (relative to 'path') 4 images

test:  # test images (optional)

# Keypoints

kpt_shape: [4, 2]  # number of keypoints, number of dims (2 for
x,y or 3 for x,y,visible)

# Classes dictionary

names:

 0: lesion
```

Step 3: Training model

Input includes images and corresponding labels.

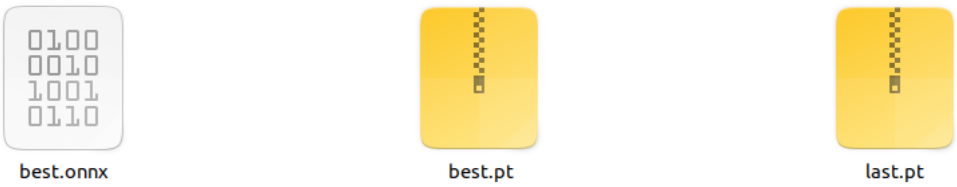
Image:



Label:

```
1 0 0.3465140478668054 0.705859375 0.14568158168574402 0.10859375 0.2736732570239334 0.66015625 0.4079084287200832 0.6515625 0.41935483870967744 0.7515625 0.2851196670135276 0.76015625
2 0 0.518210197710718 0.694921875 0.14568158168574402 0.10859375 0.445369406867846 0.64921875 0.5796045785639958 0.640625 0.5910509885535901 0.73984375 0.45681581685744016 0.74921875
3 0 0.6899063475546305 0.683203125 0.14568158168574402 0.10859375 0.6170655567117586 0.63828125 0.7513007284079084 0.62890625 0.7627471383975026 0.72890625 0.6285119667013528 0.7375
4 0 0.3340270551508845 0.597265625 0.14568158168574402 0.10859375 0.2611862643080125 0.5515625 0.39438085327783556 0.54296875 0.4068678459937565 0.64296875 0.2726326742976067 0.6515625
5 0 0.50520291363163337 0.5859375 0.14672216441207075 0.109375 0.4318418314255983 4 0.540625 0.5671175858480749 0.53125 0.578563995837669 0.63203125 0.4443288241415192 5 0.640625
6 0 0.6774193548387096 0.575 0.14568158168574402 0.109375 0.6045785639958376 0.52890625 0.7388137356919875 0.5203125 0.7502601456815817 0.6203125 0.6160249739854319 0.6296875
```

Output: Trained model




```
from ultralytics import YOLO

# Load a model

model = YOLO('yolov8n-pose.yaml')  # build a new model from YAML

model = YOLO('yolov8n-pose.pt')  # load a pretrained model
(recommended for training)

model = YOLO('yolov8n-pose.yaml').load('yolov8n-pose.pt')  # build
from YAML and transfer weights

# Train the model

results = model.train(data="<path_to_config_file>", epochs=20,
imgsz=640)
```

b. Polyp detection model

Step 1 : Prepare dataset

The training dataset folder is organized as same as that of Endoscopic image detection model. But the label format is different as described below: <object-class> <x_center> <y_center> <width> <height>

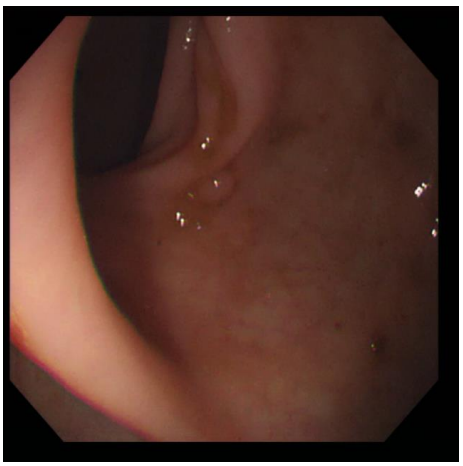
Step 2: Prepare config file

```
train: <path_to_train_folder>
val: <path_to_valid_folder>
test: <path_to_test_folder>
nc: 1  # Số lớp của đối tượng cần phát hiện (polyp). Đổi giá trị này tùy thuộc vào số lớp cần phát hiện.
names: ['polyp']  # Tên của các lớp đối tượng cần phát hiện.
img_size: 640  # Kích thước ảnh đầu vào cho mô hình YOLO.
batch: 32
```

Step 3: Training model

Input includes images and corresponding labels:

Image:



Label:

```
1 0 0.46875 0.3890625 0.1046875 0.075
```

Output: [Trained Model]

```
from ultralytics import YOLO
# Load a model
model = YOLO("yolov8n.yaml")  # build a new model from scratch
model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)
# Use the model
model.train(data="<path_to_config_file>", epochs=200)  # train the model
```

```
metrics = model.val()  # evaluate model performance on the
validation set
model.export(format="onnx")  # export the model to ONNX format
```

4.2. Evaluating trained models

The evaluation step in deep learning assesses how well a trained model performs on unseen data. It ensures the model generalizes effectively, aids in model selection and tuning, guides decision-making for deployment, and contributes to research advancements.

The input for this step is a trained model, and the path to the test folder, which is specified in the config file. The output consists of the evaluation metrics.

You can find trained models here: [Here]

a. Endoscopic image detection model

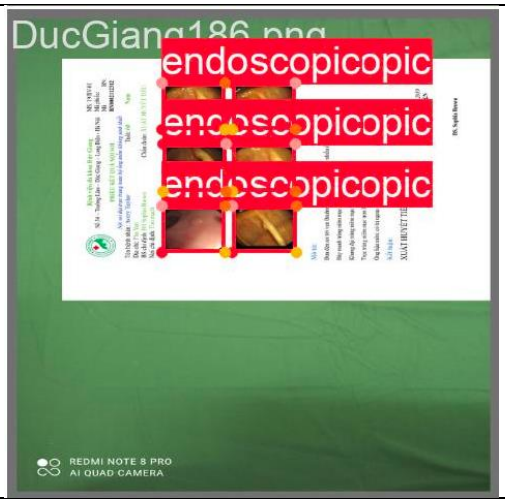
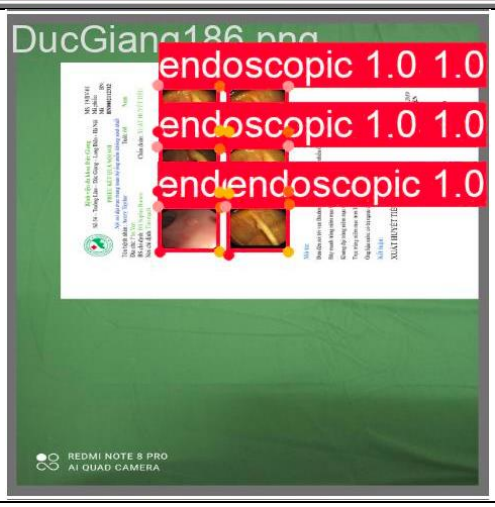
```
from ultralytics import YOLO
# Load a model
model = YOLO('yolov8n-pose.pt')  # load an official model
model = YOLO('<path_to_the_best_model>')  # load a custom model
# Validate the model
metrics = model.val()  # no arguments needed, dataset and settings
remembered
metrics.box.map    # map50-95
metrics.box.map50  # map50
metrics.box.map75  # map75
metrics.box.maps   # a list contains map50-95 of each category
```

Metrics:

Metrics	Precision (box)	Recall (Box)	Ap50 (Box)	Ap50-95 (Box)	Precisn (Pose)	Recall (Pose)	Ap50 (Pose)	Ap50-95 (Pose)
Value	1	1	0.995	0.995	0.972	0.972	0.967	0.964

These metrics indicate high precision and recall for both bounding box detection and pose estimation tasks. The average precision (AP50) and AP50-95 scores also demonstrate strong performance across both detection and pose estimation, with slightly lower values observed for pose estimation compared to bounding box detection. Overall, the model shows robust performance in accurately detecting poses within the given tasks.

Visualization of some results:

Label	Prediction
	
	

b. Polyp detection model

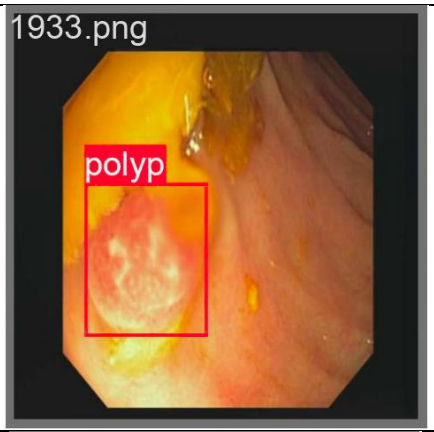
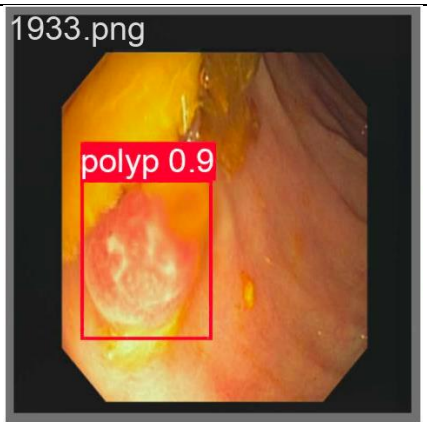
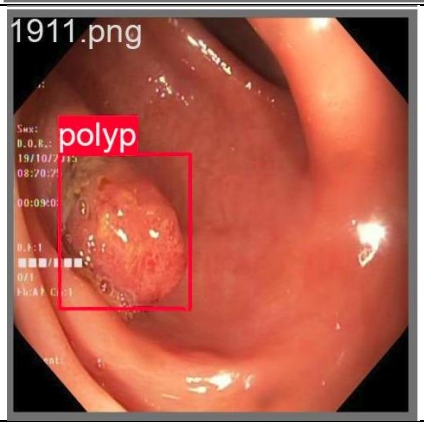
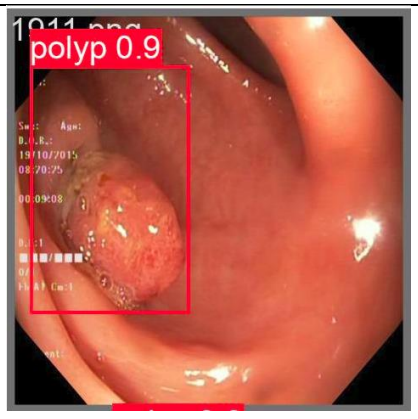
```
from ultralytics import YOLO

# Load a model
model = YOLO('yolov8n.pt')  # load an official model
model = YOLO('<path_to_the_best_model>')  # load a custom model

# Validate the model
metrics = model.val(iou=0.3)  # no arguments needed, dataset and
settings remembered
print(metrics.box.map  )  # map50-95
print(metrics.box.map50)  # map50
print(metrics.box.map75 ) # map75
print(metrics.box.mp)
print(metrics.box.mr)
```

Metrics:

Metrics	Precision (box)	Recall (Box)	Ap50 (Box)	Ap50-95 (Box)
Value	0.933	0.871	0.932	0.743

These metrics indicate a relatively high precision and recall for bounding box detection tasks. The AP50 score, which measures the average precision at IoU (Intersection over Union) threshold of 0.5, is also quite high, indicating strong performance in localizing objects. However, the AP50-95 score, which measures the average precision over a range of IoU thresholds from 0.5 to 0.95, is lower, indicating some challenges in detecting objects with higher IoU thresholds. Overall, the YOLO model demonstrates good performance in bounding box detection tasks but may have some limitations in accurately detecting objects with higher IoU thresholds.

Visualization of some results:

Labels	Predictions
	
	

4.3. Convert models for deploying

Converting models for deployment in deep learning involves optimizing them for efficient execution on target platforms, reducing size, improving speed, and adapting to hardware. This process ensures seamless integration into real-world applications.

- Remember to replace the corresponding custom trained model path

```
from ultralytics import YOLO
```

```
# Load a model
model = YOLO('yolov8n.pt')  # load an official model
model = YOLO('<path_to_best_model>')  # load a custom trained model
# Export the model
model.export(format='onnx')
```

4.4 Inference

The purpose of the inference step in deep learning is to utilize a trained model to make predictions or generate outputs on new, unseen data. This step is crucial for applying the model to real-world tasks, such as image classification, language translation, or speech recognition. Inference allows the model to process input data and produce meaningful results based on its learned patterns and parameters. It serves practical purposes, such as automating tasks, providing insights, or enabling decision-making processes, depending on the specific application domain. Overall, inference is essential for deploying deep learning models in real-world scenarios to solve various problems efficiently and effectively.

Input of inference step is

- Path to inference images folder
- The object detection threshold
- Path to result folder if you choose to save image

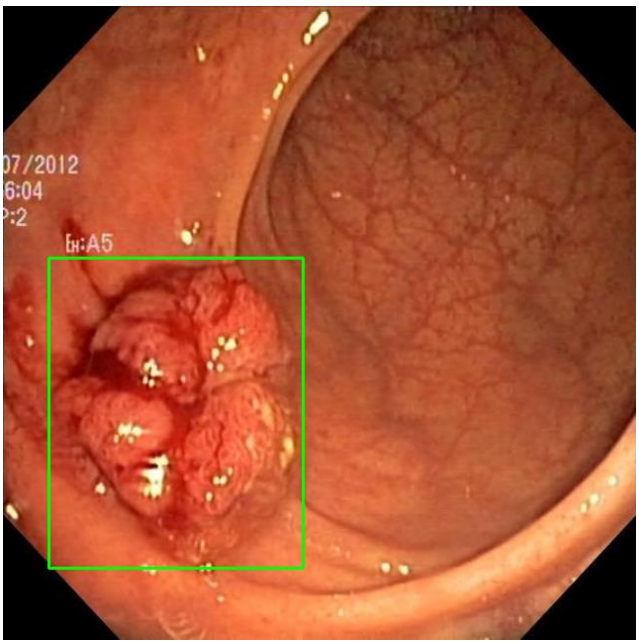
You can find full source code here: [AI-models]

```
if __name__ == '__main__':
    import argparse

    parser = argparse.ArgumentParser()
    parser.add_argument('-d', '--path', type=str, default="./datasets/polyps/valid/images", required=False, help='Path
    parser.add_argument('-t', '--threshold', type=float, default=0.5, help='Object detection threshold')
    parser.add_argument('-vf', '--save_image', type=int, default=1, help='1 for view image and 0 for not')
    parser.add_argument('-sf', '--save_xml', type=int, default=0, help='1 for save image and 0 for not')
    parser.add_argument('-sp', '--save_path', type=str, default='result',
                        help='specific save result folder when save_image=True')
    args = parser.parse_args()

    predictor = Detector('onnx')
    detect_folder(args.path, view=args.view_image, confidence=args.threshold,
                  save=args.save_image, save_path=args.save_path)
```

The output of this step is the predicted images and the coordinates of the bounding box saved in xml files.

Output of endoscopy images detection:

Output of polyps detection:



The bounding box coordinates save in XML file:

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
<annotations>
  <filename>19_8_3.png</filename>
  <size>
    <width>640</width>
    <height>640</height>
    <depth>3</depth>
  </size>
  <object>
    <name>polyps</name>
    <bndbox>
      <x1>400.9007</x1>
      <y1>334.44327</y1>
      <x2>486.725</x2>
      <y2>330.44742</y2>
      <x3>495.3101</x3>
      <y3>397.47046</y3>
      <x4>405.84775</x4>
      <y4>403.083</y4>
    </bndbox>
  </object>
  <object>
    <name>polyps</name>
    <bndbox>
      <x1>287.86755</x1>
      <y1>343.9629</y1>
      <x2>377.58868</x2>
      <y2>336.086</y2>
      <x3>383.1949</x3>
      <y3>404.16592</y3>
      <x4>296.32544</x4>
      <y4>408.63928</y4>
    </bndbox>
  </object>
  <object>
    <name>polyps</name>
    <bndbox>
      <x1>179.06372</x1>
      <y1>349.6811</y1>
      <x2>266.64307</x2>
      <y2>345.8714</y2>
      <x3>274.34085</x3>
      <y3>409.3927</y3>
      <x4>186.98665</x4>
      <y4>415.4834</y4>
```


So far, we have used the exported model to perform the inference step on the input image to achieve the desired output. The model is now ready for deployment in real-world applications.

4.5 Conclusion

In this section, we have provided detailed explanations on model selection, training, evaluation, model export for deployment in real-world applications, as well as the inference step. At each stage, we have clearly outlined the purpose, inputs, and outputs of each model, along with the methodology for training, testing, and guidance on how to convert the trained model into a suitable format for real-world applications.