Train your Own Detection Model - YOLO

Continuing from the previous blog 1 that successfully detecting objects like cats and dogs but failing to detect Saber. This blog will apply Yolo v8 to train a model from data collection & labeling, training & validating and finally predicting the object result.

Background of Yolo

YOLO: You Only Look Once

The YOLO model (Redmon et al., 2016) is the very first attempt at building a fast real-time object detector. The main feature of YOLO algorithm is that it treats the object detection problem as a regression problem, directly predicting multiple bounding boxes and category probabilities in the image through a single network forward propagation.

Basic Concept

YOLO abandons the sliding window and directly divides the original image into SS non overlapping small squares. Then, through convolution, the SS feature map is generated, where each element of the feature map corresponds to a small square in the original image.

1. **How to identify the Object**
   The training result can be viewed as a classification, and the result is $Pr(object)$. If the prediction is the same as the target then $Pr(object) = 1$, otherwise $Pr(object) = 0$.

2. **Determain the Border and Position**
   The coordinates of bounding box are defined by a tuple of 4 values, (center x-coord, center y-coord, width, height) - (x, y, w, h). x, y, w and h are normalized by the image width and height,

and thus all between (0, 1].

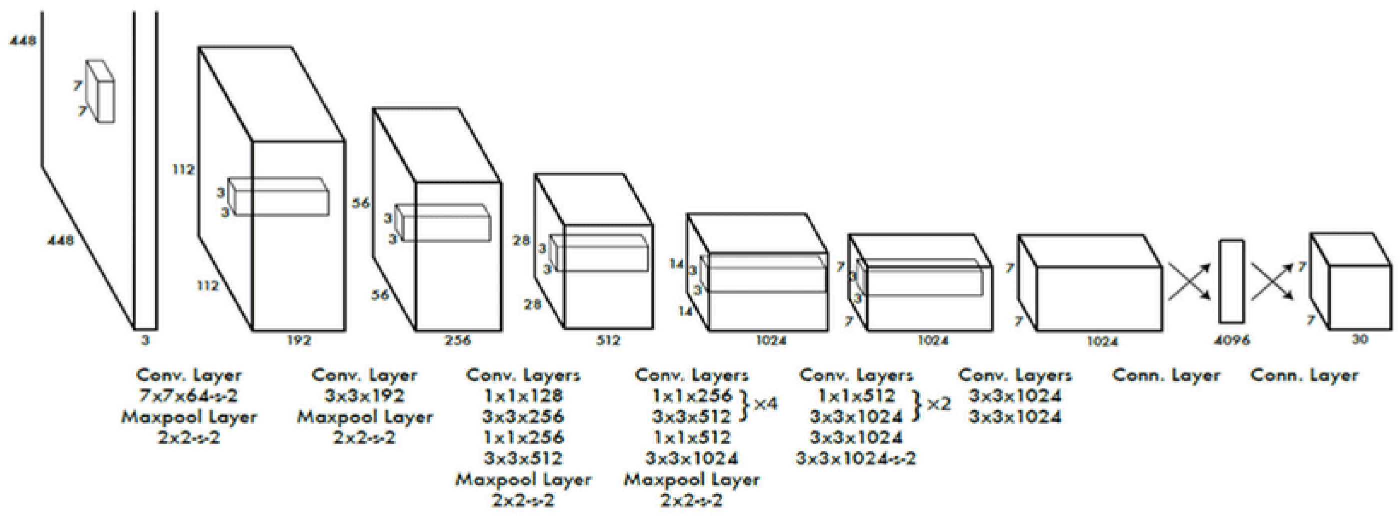
3. **Confidence Scores of Border**
   Confidence scores includes two parts: probability of target included and precision of border. The former one could be represented by $Pr(object)$, and the later one is the ratio of *prediction* and *ground truth* called **intersection over union** (IOU). IOU_{pred}^{truth}

   And the confidence scores $Pr(object) \cdot IOU_{pred}^{truth}$.

4. **Included Target or Not**
   For each cell, it is also necessary to provide predicted probability values for C categories, which represent the probabilities of the bounding boxes predicted by that cell belonging to each category. These probability values are actually *conditional probabilities* ($Pr(class_i|object)$) at the confidence levels of each bounding box. The class-specific confidence scores can be calculated: $Pr(class_i|object) \cdot Pr(object) \cdot IOU_{pred}^{truth} = Pr(class_i) \cdot IOU_{pred}^{truth}$

5. **Summary**
   Image size: $S \times S$,
   Bounding boxes numbers: B;
   In total, one image contains $S \times S \times B$ bounding boxes, each box corresponding to 4 location predictions, 1 confidence score, and C conditional probabilities for object classification. Every bounding box needs $(B \cdot 5 + C)$, so the final prediction for a image is $S \times S \times (B \cdot 5 + C)$, which is the tensor shape of the final conv layer of the model.

Network Architecture
YOLO uses convolutional networks to extract features, and then uses fully connected layers to obtain predicted values which is similar to GooLeNet.



The final prediction is produced by two fully connected layers over the whole conv feature map.

Loss Function

The loss calculation of the model includes three aspects: positioning loss, classification loss, and confidence loss.

Position: $L_{box} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{i,j}^{obj} \cdot l_{i,j}^{box}$

Classification: $L_{cls} = \lambda_{class} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{i,j}^{obj} \cdot l_{i,j}^{cls}$

Object: $L_{obj} = \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{i,j}^{noobj} \cdot l_{i,j}^{noobj} + \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{i,j}^{obj} \cdot l_{i,j}^{box}$

$1i, j^{noobj}$ indicates whether the j-th bounding box of the cell i is “responsible” for the object prediction.

The total loss: $Loss = L_{box} + L_{cls} + L_{obj}$

(The detailed explanation for yolov5 can be found here)

Training the Model

Environment

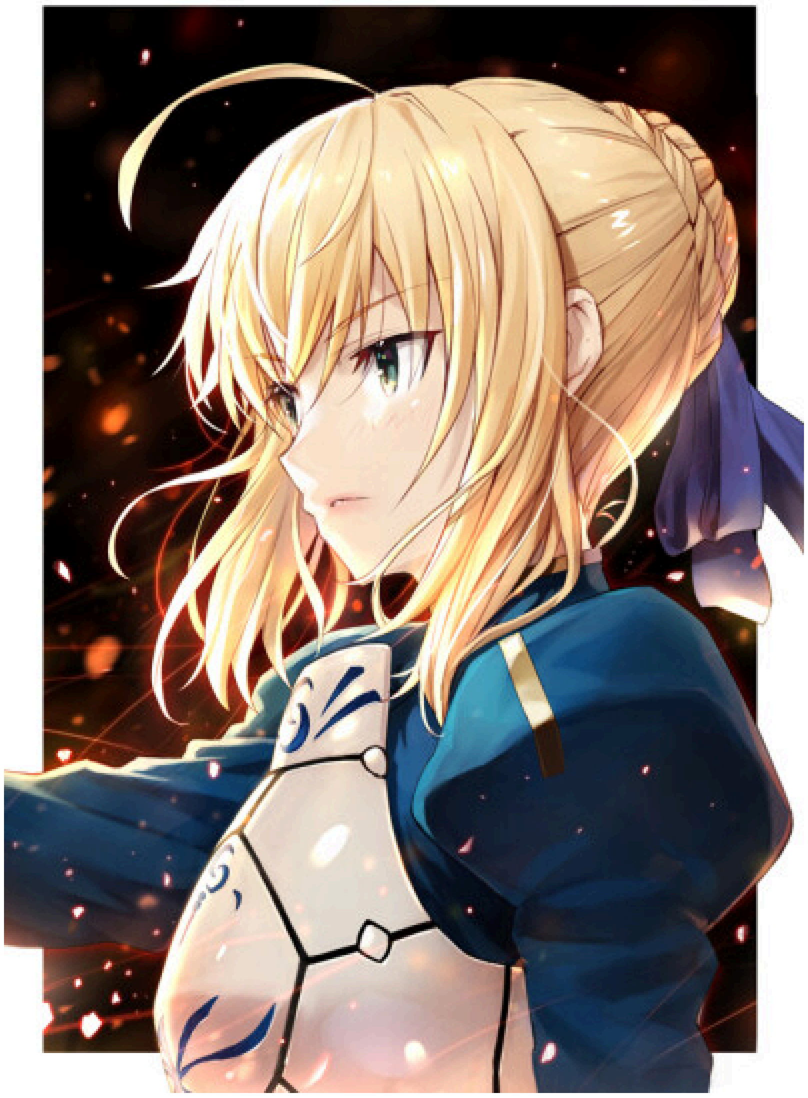
Python: 3.8

Yolo model: cv2, yolo v8

Label: labelImg for labelling the training dataset

Dataset

Here I downloaded around 150 images from google with keywords 'saber' for Yolo model training. With the help of LabelImg, images had been labeled with Yolo format. After processing, each image will pair with a txt file with format: `<class_id> <x_center> <y_center> <width> <height>`
For example:



and the txt file for this image is : `0 0.545000 0.332447 0.790000 0.491135`

There is also a calss file to save the calssification catrgories, here I only have one target so the `class` only contained `saber` .

Training

1. **Dataset Allocation**

   Allocating about 60% images to *train*, 20% to *valid* and 20% to *predict*, and put the label files as well. The class txt should be at the same level as above. The directory for the images and txts is:
   - Dataset
     - images
     - labels
     - class.txt
2. **Config the Yaml**

   The yaml tells the location for dataset and the number of class in this model training.

   ```
   train: D:/PyCharm Community Edition 2024.1.3/TechBlog/dataset/images/train
   val: D:/PyCharm Community Edition 2024.1.3/TechBlog/dataset/images/val
   test: D:/PyCharm Community Edition 2024.1.3/TechBlog/dataset/images/test

    # number of classes
    nc: 1

    # class names
    names: ['saber']
   ```

3. **Model training**

   With the downloaded `yolov8n.pt` , it's time to start to train.

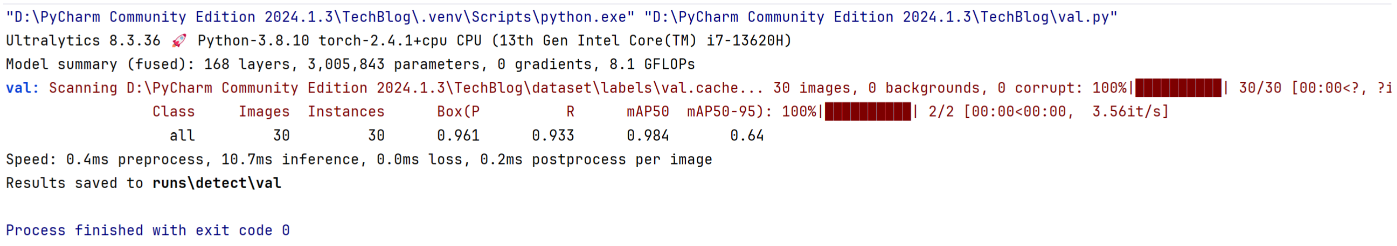
   ```
   from ultralytics import YOLO

    if __name__ == '__main__':
        # Load a model
        model = YOLO('yolov8n.pt')  # load a pretrained model

        # Train the model
        model.train(data='./dataset/data.yaml', epochs=270, imgsz=320)
   ```

   After 270 epochs training, model for detecting 'saber' has been saved.
4. **Validation**

   Code for valid dataset:

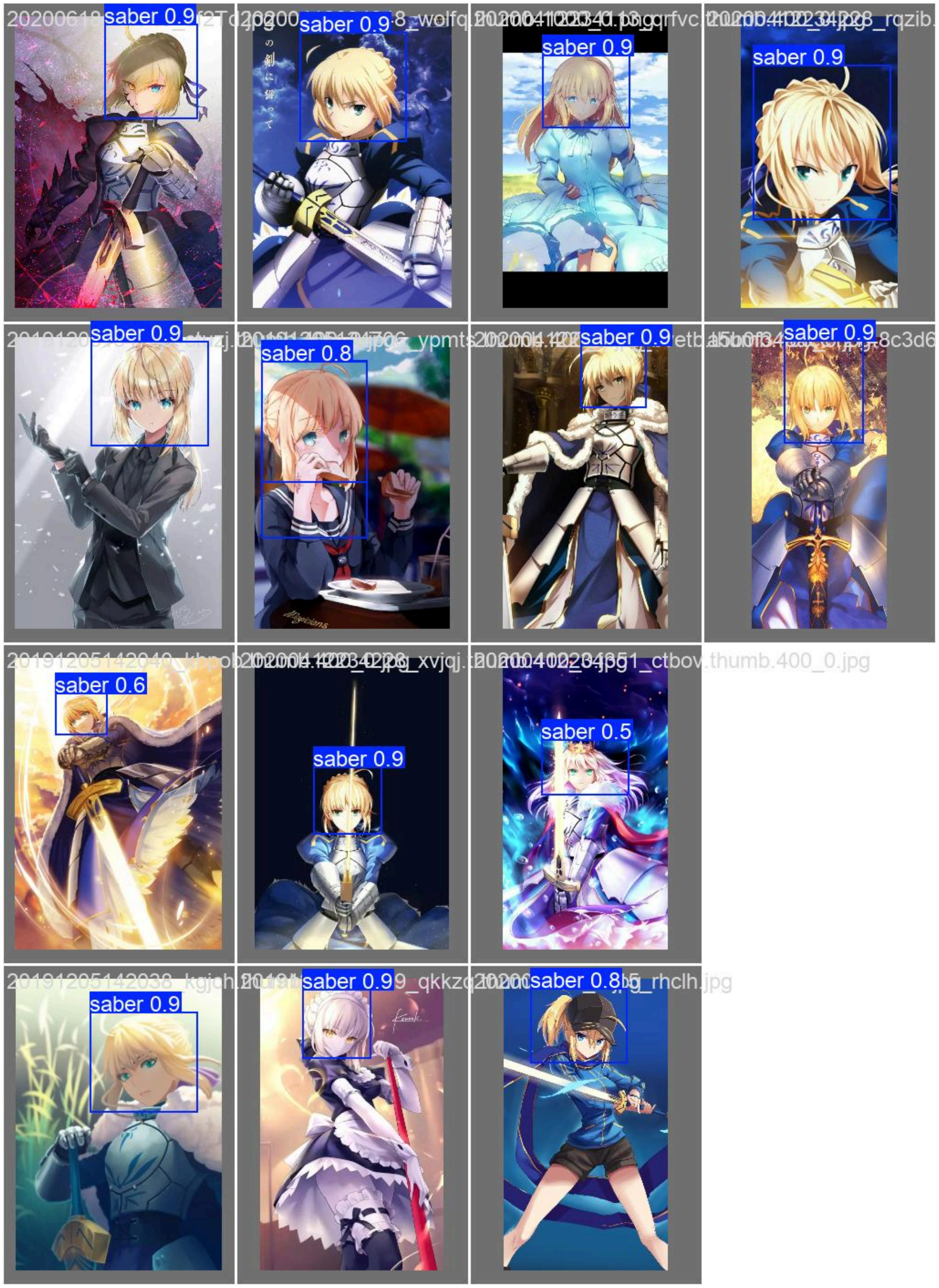
```
from ultralytics import YOLO

 if __name__ == '__main__':
    # Load a model
    model = YOLO('D:/PyCharm Community Edition 2024.1.3/TechBlog/runs/detect/train9/weight

    # Validate the model
    metrics = model.val()
```

And the result for this dataset:



```
"D:\PyCharm Community Edition 2024.1.3\TechBlog\.venv\Scripts\python.exe" "D:\PyCharm Community Edition 2024.1.3\TechBlog\val.py"
Ultralytics 8.3.36 🚀 Python-3.8.10 torch-2.4.1+cpu CPU (13th Gen Intel Core(TM) i7-13620H)
Model summary (fused): 168 layers, 3,005,843 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning D:\PyCharm Community Edition 2024.1.3\TechBlog\dataset\labels\val.cache... 30 images, 0 backgrounds, 0 corrupt: 100%|██████████| 30/30 [00:00<?, ?i
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:00<00:00,  3.56it/s]
                   all         30         30      0.961      0.933      0.984       0.64
Speed: 0.4ms preprocess, 10.7ms inference, 0.0ms loss, 0.2ms postprocess per image
Results saved to runs\detect\val

Process finished with exit code 0
```

If we go to the directory to see the predicted results in `/val` , we will find the model successfully detects all the images!

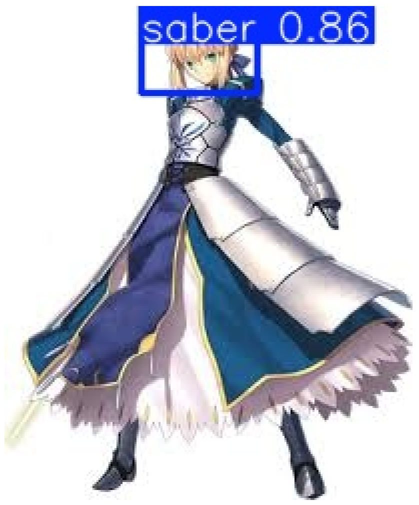

saber 0.9
saber 0.9
saber 0.9
saber 0.9
saber 0.9
saber 0.8
saber 0.9
saber 0.9
saber 0.6
saber 0.9
saber 0.5
saber 0.9
saber 0.9
saber 0.8
_ctbov.thumb.400_0.jpg
_rhclh.jpg

Resolve the Legacy issue

Now, to see how this works to solve the problem in the Blog 1. Let the model detect the input image 'saber' to see the result.

```
from PIL import Image
from ultralytics import YOLO

if __name__ == '__main__':
    # load the model
    model = YOLO("./runs/detect/train9/weights/best.pt")

    # from PIL loading img
    im1 = Image.open("saber.jpg")
    results = model.predict(source=im1, save=True)

    # using loop to print the position result of detected target
    for result in results:
        boxes = result.boxes  # bounding box

        for box in boxes:
            # position
            x_min, y_min, x_max, y_max = box.xyxy[0]
            # top-left & bottom-right
            print(f"Detected object at: x_min={x_min}, y_min={y_min}, x_max={x_max}, y_max={y_ma
```

Time to see the result 😜



Good 🎉

Reference

[1] Joseph Redmon, et al. “You only look once: Unified, real-time object detection.” CVPR 2016.

[2] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going Deeper with Convolutions. ArXiv. https://arxiv.org/abs/1409.4842

[3] Loss function detail in Yolov5 blog

[4] YOLO code source from github