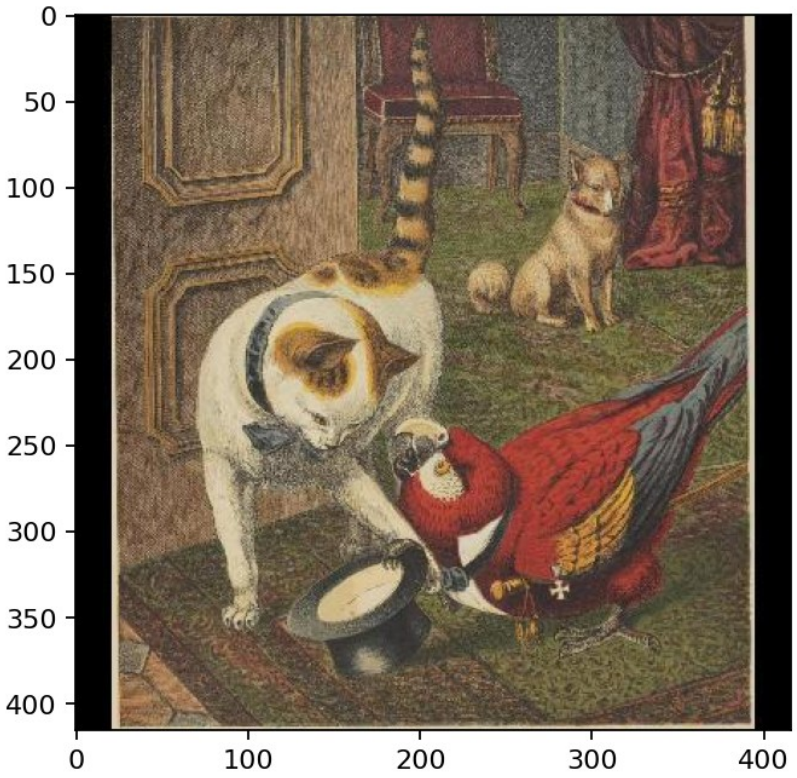
SAM and YOLO tests

```
# Download the default model from here:
!wget
https://dl.fbaipublicfiles.com/segment_anything/sam_vit_h_4b8939.pth

import numpy as np
import matplotlib.pyplot as plt
import cv2, torch, torchvision
import pandas as pd


!pip install opendatasets
import opendatasets

!pip install 'git+https://github.com/facebookresearch/segment-
anything.git'
from segment_anything import sam_model_registry,
SamAutomaticMaskGenerator, SamPredictor
```

Read in the original image

```
image = cv2.imread('/content/otsien.jpg')

# Change the format of image that is from BGR(cv2 format) to RGB
(matplotlib format)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# show the image
plt.figure(dpi=160)
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7da4124301c0>
```



Setup SAM

```
sam_checkpoint = "/content/sam_vit_h_4b8939.pth"

model_type = "vit_h"

sam = sam_model_registry[model_type](checkpoint=sam_checkpoint)
```

Check GPU

```
device = torch.device('cuda:0' if torch.cuda.is_available() else
'cpu')
sam.to(device=device)
```

Create an Instance of "SamAutomaticMaskGenerator"

```
mask_generator = SamAutomaticMaskGenerator(
    model=sam,
    points_per_side=32,
    pred_iou_thresh=0.86,
    stability_score_thresh=0.9,
    crop_n_layers=1,
    crop_n_points_downscale_factor=2,
    min_mask_region_area=100    # Requires open-cv to run post-
```

```
processing
)
```

Generate the mask

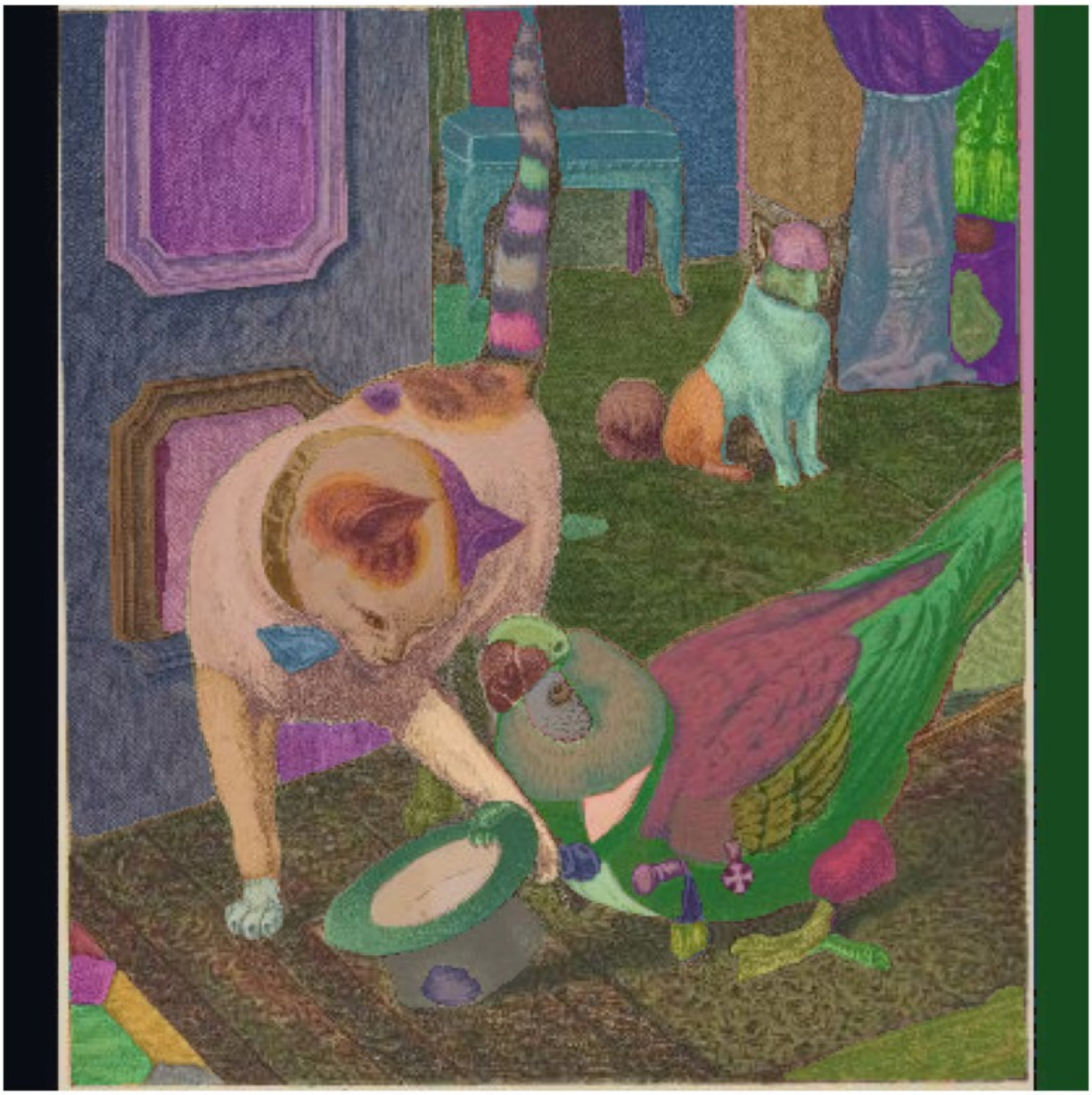
```
masks = mask_generator.generate(image)

print(len(masks))
print(masks[0].keys())
```

```
88
dict_keys(['segmentation', 'area', 'bbox', 'predicted_iou',
'point_coords', 'stability_score', 'crop_box'])
```

Define a function that Visualizes mask on an Image

```
def show_anns(anns):
    """Takes a list of masks as an input and visualizes them on an
Image"""
    if len(anns) == 0:
        return

    # Sort the masks by their area in descending order
    sorted_anns=sorted(anns, key=(lambda x: x['area']), reverse=True)
    ax=plt.gca()
    ax.set_autoscale_on(False)

    # let's create a new image that is the same size as the largest
mask
    img = np.ones((sorted_anns[0]['segmentation'].shape[0],
sorted_anns[0]['segmentation'].shape[1], 4))
    """
    1st argument: sorted_anns[0]['segmentation'].shape[0] --> Height
of the new image,
        2nd argument: sorted_anns[0]['segmentation'].shape[1] -->
Width of the new image,
        3rd argument: specifies number of channels in new array, means
each pixel in new image array will have 4 values.
    """

    img[:,:,3] = 0    # sets the alpha channel of the new image to 0.
    """A value of 0 means that the pixel is completely transparent,
and a value of 1 means that the pixel is completely opaque."""

    # Now let's iterate over a sorted masks and fills in the
corresponding pixels in new image with random color.
    for ann in sorted_anns:
        m = ann['segmentation']
        color_mask = np.concatenate([np.random.random(3), [0.35]])
```

```
# stores a random color, that will be used to fill in the pixels in
the new image.
        img[m] = color_mask

    ax.imshow(img)

plt.figure(figsize=(10, 10))
plt.imshow(image)
show_anns(masks)

plt.axis('off')
plt.savefig('masked_img.png', bbox_inches='tight', pad_inches=0)
```

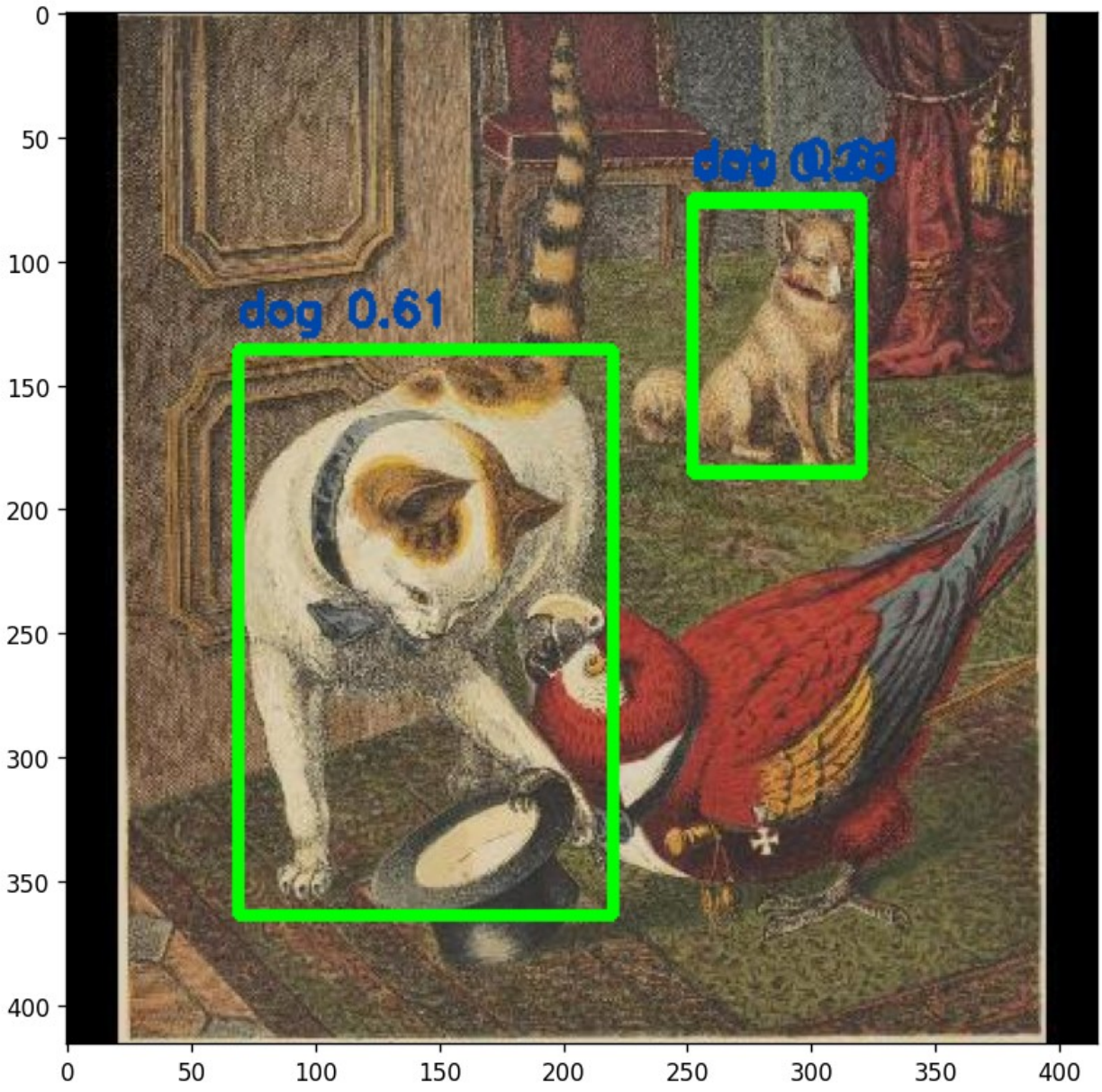


Define a function that creates the Bounding box and label it

```
!pip install ultralytics

from ultralytics import YOLO
import math

model = YOLO('yolov8n.pt')    # I am using yolo version 8 Nano
weights, but one can even use Large or Medium as weights

Downloading
https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n
.pt to 'yolov8n.pt'...

100%|██████████| 6.23M/6.23M [00:00<00:00, 78.2MB/s]

# From the COCO Dataset by Microsoft, let's create a list of classes
that YOLO could be able to detect
classnames = ["person", "bicycle", "car", "motorbike", "aeroplane",
"bus", "train", "truck", "boat", "traffic light",
              "fire hydrant", "stop sign", "parking meter", "bench",
"bird", "cat", "dog", "horse", "sheep", "cow",
              "elephant", "bear", "zebra", "giraffe", "backpack",
"umbrella", "handbag", "tie", "suitcase", "frisbee",
              "skis", "snowboard", "sports ball", "kite", "baseball
bat", "baseball glove", "skateboard", "surfboard",
              "tennis racket", "bottle", "wine glass", "cup", "fork",
"knife", "spoon", "bowl", "banana", "apple",
              "sandwich", "orange", "broccoli", "carrot", "hot dog",
"pizza", "donut", "cake", "chair", "sofa",
              "potted plant", "bed", "dining table", "toilet", "tv
monitor", "laptop", "mouse", "remote", "keyboard",
              "cell phone", "microwave", "oven", "toaster", "sink",
"refrigerator", "book", "clock", "vase", "scissors",
              "teddy bear", "hair drier", "toothbrush"]

def show_bbox_and_label(image):
    results = model(image)

    for r in results:    # for number of OBJECTS detected in the image
        boxes = r.boxes  # give me the bounding boxes of those objects
detected in the image
        for box in boxes:
            x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            w, h = x2-x1, y2-y1
            print(f"x1,y1 coordinate: {(x1,y1)}\nx2,y2 coordinate:
{x2,y2}\nWidth: {w}\nHeight:{h}")

            # Find out the confidence
            conf = math.ceil(box.conf[0]*100) / 100
```

```
            print(f"Confidence of box: {conf}\n","-"*30)

            # Find out the class name
            cls = int(box.cls[0])

            # Draw the rectangle
            cv2.rectangle(image, (x1,y1), (x2,y2), (0, 255, 0), 3)

            # Now, let's put the text on rectangle
            FONT = cv2.FONT_HERSHEY_SIMPLEX
            COLOR = (0, 71, 171)
            THICKNESS = 2
            ORG = (max(0, x1), max(35, y1 - 10))

            cv2.putText(image, str(f"{classnames[cls]} {conf}"), ORG,
FONT, 0.6, COLOR, THICKNESS)

    plt.figure(figsize=(15,8), dpi=120)
    plt.imshow(image)

image = cv2.imread('/content/otsien.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
show_bbox_and_label(image)
```

```
0: 640x640 1 cat, 2 dogs, 11.6ms
Speed: 3.0ms preprocess, 11.6ms inference, 1.9ms postprocess per image
at shape (1, 3, 640, 640)
x1,y1 coordinate: (252, 75)
x2,y2 coordinate: (320, 186)
Width: 68
Height:111
Confidence of box: 0.61
 ------------------------------
x1,y1 coordinate: (69, 136)
x2,y2 coordinate: (220, 364)
Width: 151
Height:228
Confidence of box: 0.61
 ------------------------------
x1,y1 coordinate: (252, 77)
x2,y2 coordinate: (320, 185)
Width: 68
Height:108
Confidence of box: 0.26
 ------------------------------
```



True labels: dog, hat, chair, cat, bird, curtain