COCO Dataset - use Faster RCNN + MobileNet to conduct Object Detection

Created 2023-11-03 | Updated 2023-12-10 | Code > Mechine Learning
| Word Count: 3.1k | Reading Time: 19mins | Post Views: | Comments: 0

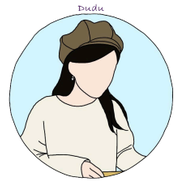
Introduction

Recently I took an AI course, the main content is the following topics:

1. Learn about Coco dataset
2. User pre-trained version of Faster R-CNN to predict the bounding box
3. Calculate IoU

Homework Requirement

1. **Download the Coco Collection** *: download the files "2017 Val images [5/1GB]" and "2017 Train/Val annotations [241MB]" from the Coco page.
   Download from Coco page. You can load them into your notebook using the pycocotools library.
2. **Randomly select ten from the dataset** : 10 images are randomly selected from this dataset.
3. **Predict the box using the pre-trained model FasterR-CNN** : use a pre-trained version of the Faster R-CNN (Resnet50 backbone) to predict the bounding box of the object on the 10 images. of the bounding box. Only regions with scores greater than 0.8 are retained.
4. **isualize the model together with the answer visualization** *: Visualize the predicted bounding



Shannon Hung

In order to avoid forgetfulness, I started to record

Articles	Tags	Categories
13	12	12

Follow Me

Announcement

I am a graduate student at the Taiwan University of Science and Technology - Institute of Information Management, planning to pursue a dual-degree program in Germany this year. The

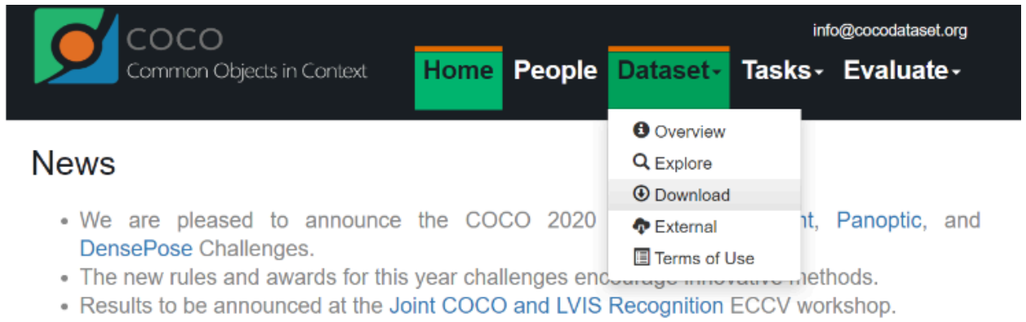
boxes and label. Show all 10 pairs of images side by side in the jupyter notebook.

5. **Use another pre-trained model Mobilnet** : Repeat the above steps using the Mobilenet backbone of the Faster R-CNN.
6. **Calculate IoU Compare Models** : Which backbone provides better results? Calculate the IoU for both methods.

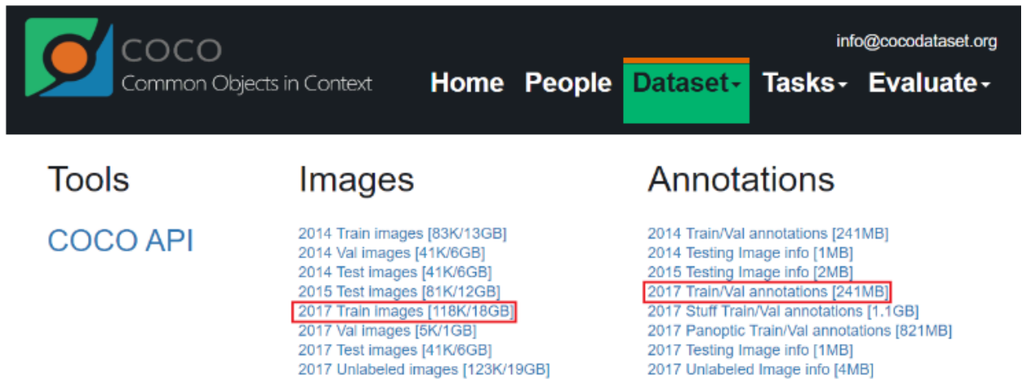
Task 1: Downloading the COCO Dataset

> **Task 1**
>
> 1. **Download the COCO Dataset** : Obtain the files "2017 Val images [5/1GB]" and "2017 Train/Val annotations [241MB]" from the Coco page. Utilize the pycocotools library to import them into your notebook.

You can follow this guide to proceed with the download: Download COCO Dataset



這部分一樣，就連到他們官網後，點 Dataset 然後選 Download，接著會進入下載頁面：




```bash
.
├── annotations # These are annotation files
```

have learned a lot, I tend to forget much of it. To combat forgetfulness, I have embarked on my note-taking journey.

Contents 24

Recent Post


Windows - A Step-by-Step Guide to Enable CUDA GPU on Ten...
2024-09-04


How to Handle Multi-Column Text Sorting with Amazon Textract
2024-03-20


Twitter Dataset - Using LSTM to predict the emotion of the arti...
2023-11-16


COCO Dataset - use Faster RCNN + MobileNet to conduct Ob...

```
│   ├── instances_val2017.json
│   ├── person_keypoints_train2017.json
│   └── person_keypoints_val2017.json
└── val2017 # This is the image set
    ├── 000000000139.jpg
    ├── 000000000285.jpg
    ├── 000000000632.jpg
    ├── 000000000724.jpg
    ├── 000000000776.jpg
    ├── 000000000785.jpg
    ├── 000000000802.jpg
    ...
```

- Download these two files as shown in the image.
- After downloading, the folder structure upon extraction will resemble the one above.

Task 2: Randomly Select Ten Images

> **Task 2**
>
> 2. **Randomly Select Ten Images from the Dataset** : Pick 10 images randomly from this dataset.

Here, we'll primarily do a few things:

- Import necessary libraries.
- Set up the COCO API to allow it to access relevant information from our dataset, such as bounding box positions, label locations, and image information.
- Visualize images and perform annotations.
- Randomly select ten images.

Let's begin by importing the necessary libraries.

PYTHON

```python

# CNN
import torch.nn.functional as F
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
```


Flower102 Dataset - Using Transfer Learning to train + Using ...
2023-10-31

```python
import matplotlib.pyplot as plt
import time
import os
from PIL import Image
from tempfile import TemporaryDirectory
import time
import random

# torchvision
import torchvision
import torchvision.transforms as transforms

# dataset
from pycocotools.coco import COCO
import cv2


cudnn.benchmark = True
plt.ion()   # interactive mode
```

Setting up the COCO API

COCO provides an API to access datasets. By providing it with a JSON file, we can easily retrieve the necessary information such as images, labels, bounding boxes, and more.

PYTHON

```python
# Specify dataset location
cocoRoot = "../../Data/Coco/"
dataType = "val2017"

# Set annotation file location
annFile = os.path.join(cocoRoot, f'annotations/instances_{dataType}.json')
print(f'Annotation file: {annFile}')
# # initialize COCO api for instance annotations
coco=COCO(annFile)
coco
```

> Result

LUA

```
Annotation file: ../../Data/Coco/annotations/instances_val2017.json
-- Indicates successful annotation file read
loading annotations into memory...
```

```
creating index...
index created!
```

Annotation Visualization

To ensure familiarity with the COCO-provided API, here's an exercise focusing on the following:

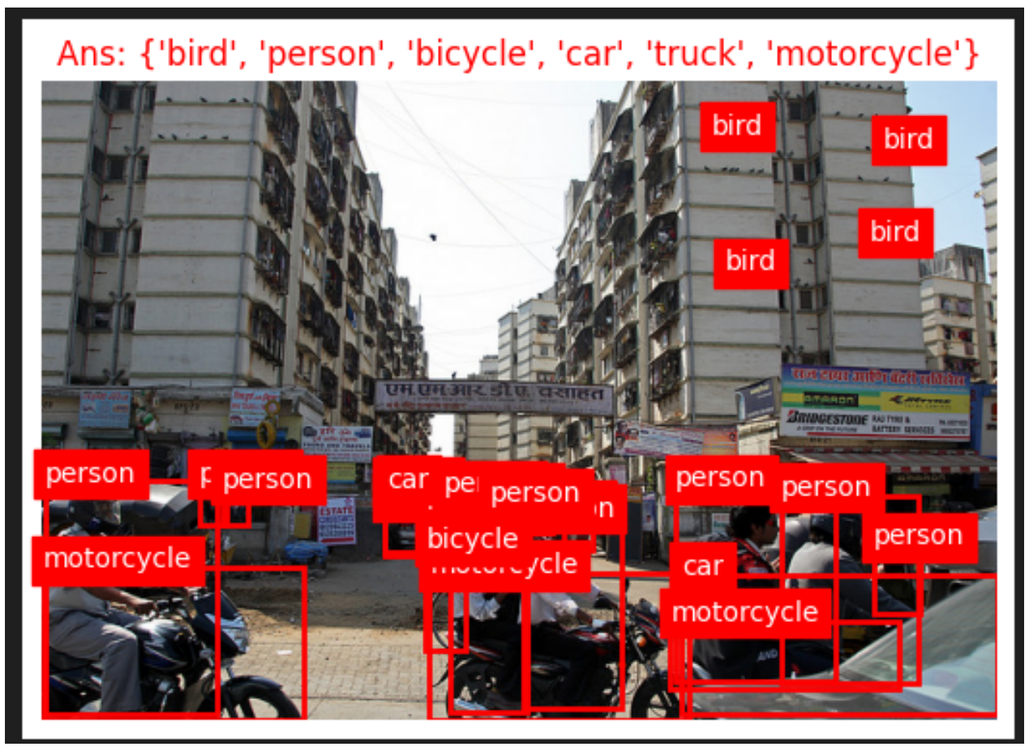
- Obtaining image info by ID
- Retrieving annotation info by ID
- Learning to draw bounding boxes and labels on images

PYTHON

```python
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

# Create a function that, given an image ID, plots the image with bounding boxes and labels
def plot_image_with_annotations(coco, cocoRoot, dataType, imgId, ax=None):
    # Get image information
    imgInfo = coco.loadImgs(imgId)[0]
    # Get image location for visualization
    imPath = os.path.join(cocoRoot, dataType, imgInfo['file_name'])
    # Read the image
    im = cv2.imread(imPath)
    # Convert color space: OpenCV defaults to BGR, but matplotlib to RGB, so conversion is ne
    im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)

    # Find all annotations (bounding boxes) for the image
    annIds = coco.getAnnIds(imgIds=imgInfo['id'])
    # Load all annotation information: bounding box coordinates, labels, accuracies
    anns = coco.loadAnns(annIds)
    all_labels = set()

    # Extract bounding box coordinates, labels, accuracies
    for ann in anns:
        # Specifically select information related to the bounding box: returns (x, y) of the
        x, y, w, h = ann['bbox']

        # Get label text information: load category name by category ID
        label = coco.loadCats(ann['category_id'])[0]["name"]
        all_labels.add(label)

        # Draw bounding boxes using provided coordinates
        rect = Rectangle((x, y), w, h, linewidth=2, edgecolor='r', facecolor='none')

        # Draw the image: if sorting of images is needed, ax parameter specifies the positior
```

```
            plt.text(x, y, f'{label}', fontsize=10, color='w', backgroundcolor='r')
        else:
            ax.add_patch(rect)
            ax.text(x, y, f'{label}', fontsize=10, color='w', backgroundcolor='r')

    # Display the image with a title
    if ax is None:
        plt.imshow(im)
        plt.axis('off')
        plt.title(f'Annotations: {all_labels}', color='r')
        plt.show()
    else:
        ax.axis('off')
        ax.set_title(f'Annotations: {all_labels}', color='r', loc='center', pad=20)
        ax.imshow(im)


# Get the tenth image
imgIds = coco.getImgIds()
imgId = imgIds[10]
# Plot the image with annotations
plot_image_with_annotations(coco, cocoRoot, dataType, imgId)
```

Result

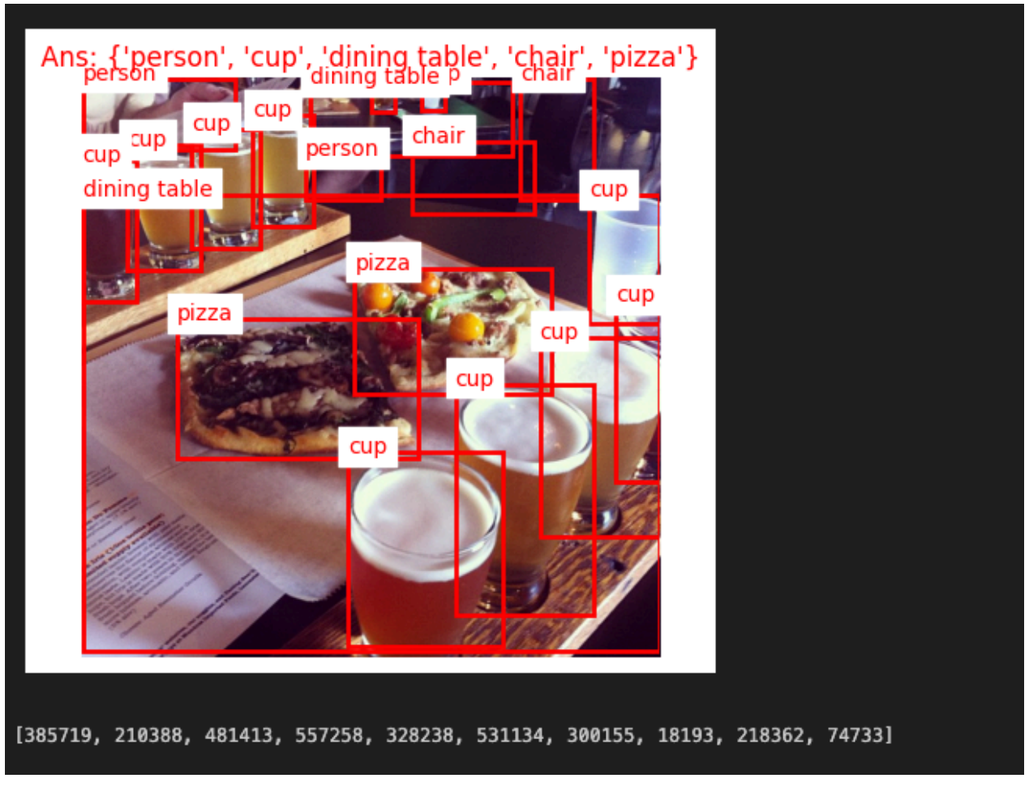


Randomly Select Ten Images

PYTHON

```python
def random_select(coco, cocoRoot, dataType, num_images=10):
    # Get IDs for all images
    imgIds = coco.getImgIds()
    # Randomly select num_images IDs from this set
    selected_imgIds = random.sample(imgIds, num_images)
    # Call the plot_image_with_annotations function for each selected ID
    for imgId in selected_imgIds:
        # Plot images based on their IDs
        plot_image_with_annotations(coco, cocoRoot, dataType, imgId)

    # Print out all selected IDs
    return selected_imgIds

valid_ids = random_select(coco, cocoRoot, dataType, num_images=10)
valid_ids
```

> Result



Task 3 : 5: FasterR-CNN v.s Mobilnet

> **Task 3 & 5**
>
> 3. **Predicting bboxes using the pre-trained model FasterR-CNN** : Use a pre-trained version of Faster R-CNN (Resnet50 backbone) to predict the bounding box of objects on the 10 images. Only keep regions that have a score > 0.8.
> 5. **Using another pre-trained model Mobilnet** : Repeat the steps from above using a Mobilenet backbone for the Faster R-CNN.

Using pre-train model

PYTHON

```python
# using pre-train model (FasterR-CNN)
model_res = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights="FasterRCNN_ResNet50_
model_res.eval()

# using pre-train model  (Mobilenet)
model_mobile = torchvision.models.detection.fasterrcnn_mobilenet_v3_large_fpn(weights=torchvis
model_mobile.eval()
```

Convert image to tensor

We need to be able to take the image out of the picture based on the position of the image. Then we need to convert the image read from the book into a tensor before we can put it into the model for prediction. So we made two functions:

- One is to read the picture
- One is to convert the picture to a tensor.

PYTHON

```python
from PIL import Image

def load_image(imgIdx):
    # Get image information
    imgInfo = coco.loadImgs(imgIdx)[0]
    # Get image location for visualization
    imPath = os.path.join(cocoRoot, dataType, imgInfo['file_name'])
    # Read the image path
    print(imPath)
    try:
        # Read the image
```

```
        raise Exception()

# Convert the picture to a tensor
def pil2tensor(pil_image):
    # Use unsqueeze(0) because the model still contains the batch size dimension, a total of
    # But the picture has only one picture without batch size, the picture is converted to te
    # /255 is because the input of the model is a number between 0 and 1, and the value of th
    return torchvision.transforms.PILToTensor()(pil_image).unsqueeze(0) / 255.0
```

Training the model

After the pre-training model is loaded, we need to train the model. The training process is as follows:

PYTHON

```
# Save the prediction result
predictions_res = []
predictions_mobile = []

# Recursively call each id, these ids are the 10 ids we randomly selected above
for i in valid_ids:
    print(i)
    # transform to tensor from PIL image
    img_as_tensor = pil2tensor(load_image(i))
    # put the tensor to resnet model
    prediction = model_res(img_as_tensor)
    # Save the prediction result: the prediction result is a dictionary, which contains the p
    predictions_res.append(prediction)
    # put the tensor to mobilenet model
    prediction = model_mobile(img_as_tensor)
    # Save the prediction result: the prediction result is a dictionary, which contains the p
    predictions_mobile.append(prediction)
```

Only select the prediction results > 0.8

After the model is trained, we need to select the prediction results that are greater than 0.8. The reason is that the model will predict a lot of bounding boxes, but we only need the bounding boxes with high accuracy. So we need to filter out the bounding boxes with low accuracy. The code is as follows:

PYTHON

```
def filter_valid_boxes(predictions, threshold=0.8):
    # Used to store the filtered prediction results
    valid_boxes_list = []
```

```
        # Recursively call each bounding box
        for box, label, score in zip(prediction[0]["boxes"], prediction[0]["labels"], predict
            # Only keep the predicted bounding box with accuracy greater than threshold
            if score >= threshold:
                # Save the predicted bounding box, label, and accuracy
                valid_boxes_for_this_prediction.append((box, label, score))
        # If none of the predicted boxes in this picture have an accuracy greater than thresh
        valid_boxes_list.append(valid_boxes_for_this_prediction)
    # Return the filtered prediction result
    return valid_boxes_list

# Set threshold to 0.8 and get the prediction results of resnet and mobilenet
valid_boxes_res = filter_valid_boxes(predictions_res, threshold=0.8)
valid_boxes_mobile = filter_valid_boxes(predictions_mobile, threshold=0.8)
```

Task 4+6: Visualization + IoU

> **Tasks 4 & 6**
>
> 1. **Visualize the model together with the solution** : Visualize the predicted bounding boxes and label together with the ground truth bounding
> 2. **CalculateIoU to compare models** : Which backbone delivers the better results? Calculate the IoU for both approaches.

There are a few important points in visual dialog, the steps are as follows:

- We need to know the id of the image first, and get the annotation information based on the id, then we can Calculate the IoU.
- We take the annotation information and the model information to conduct the IoU Calculate.
- We read the location of the image in the computer, and according to the path of the image, we draw the image through plt first.
- We read the location of the picture in the computer, and based on the path of the picture, we draw the picture through plt first, and then based on the picture, we can draw the prediction box and label on it, as well as the average value of the IoU.

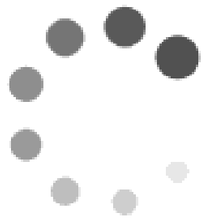
The following program is the procedure described above, we will draw the results of both models and calculate the average of the IoU.

PYTHON

```python
import matplotlib.pyplot as plt
from PIL import Image
import os


# Can put the results of different models into this function, and it will return the average
def display_annotated_results(imgId, valid_boxes, model_name, color='g', ax=None):
    # Load the image
    imgInfo = coco.loadImgs(imgId)[0]
    image_path = os.path.join(cocoRoot, dataType, imgInfo['file_name'])
    image = Image.open(image_path)

    # Get the correct bounding box results
    annIds = coco.getAnnIds(imgIds=imgInfo['id'])
    anns = coco.loadAnns(annIds)
    bbox_tlist_anns = torch.tensor([ann["bbox"] for ann in anns]) # tensor.shape[2,4]
    # because our bounding box is x,y,w,h which is the coordinate of the lower left corner of
    # But torchvision Calculate the box_iou must give the coordinates of the lower left corne
    # x,y,w,h -> x1,y1,x2,y2 = x,y,x+w,y+h
    bbox_tlist_anns[:, 2] = bbox_tlist_anns[:, 0] + bbox_tlist_anns[:, 2]
    bbox_tlist_anns[:, 3] = bbox_tlist_anns[:, 1] + bbox_tlist_anns[:, 3]

    # From resultsvalid_boxes, we only need the box part, so we use (box, _, _)
    # Use stack because we want to stack all the boxes together to become a tensor
    bbox_tlist_model = torch.stack([box for box, _, _ in valid_boxes]) # turn [4] to tensor.s
    # use box_iou 來Calculate IoU
    iou = torchvision.ops.box_iou(bbox_tlist_anns, bbox_tlist_model) # get IoU
    # Get the maximum value of each predicted box in ann, and then Calculate the average valu
    avg_iou = np.mean([t.cpu().detach().numpy().max() for t in iou]) # calculate the mean of

    # display image label
    all_labels = set()

    # Start drawing the prediction box
    for boxes in valid_boxes:
        # Get the information of the prediction box, including box, label, and accuracy
        box, label, score = boxes
        # Get the text information of the label: load category name by category ID
        label = coco.loadCats(label.item())[0]["name"]
        # Save the label for later display
        all_labels.add(label)
        # Because the results returned by the model are two coordinates, the lower left corne
        x, y, x2, y2 = box.detach().numpy() # x,y,w,h -> x,y,x2-x,y2-y
        rect = Rectangle((x, y), x2 - x, y2 - y, linewidth=2, edgecolor=color, facecolor='nor

        # Draw the picture: if you need to sort the picture, you can specify where to draw th
        if ax is None:
```

```
            # Draw the label on the prediction box
            plt.text(x, y, f'{label}', fontsize=10, color='w', backgroundcolor=color)
        else:
            # add_patch can add a patch to the current axes, and then draw the prediction box
            ax.add_patch(rect)
            # Draw the label on the prediction box
            ax.text(x, y, f'{label}', fontsize=10, color='w', backgroundcolor=color)

    # display image and give it a title, the title is the label that appears in this picture,
    if ax is None:
        plt.axis('off')
        plt.title(f'{model_name}: {all_labels} \n IoU: {avg_iou:.4f}', color=color)
        plt.imshow(image)
        plt.show()

    else:
        ax.axis('off')
        ax.set_title(f'{model_name}: {all_labels} \n I0U: {avg_iou:.4f}', color=color)
        ax.imshow(image)

    return avg_iou


res_iou = []
mobile_iou = []

# Recursively call each id, where id is one of the 10 random ids we selected above
for i in range(len(valid_ids)):
    # Create a 1x3 grid of images, each image sized 15x5
    fig, axs = plt.subplots(1, 3, figsize=(15, 5))

    # Draw the truth image and display it in the center
    plot_image_with_annotations(coco, cocoRoot, dataType, valid_ids[i], ax=axs[1])

    # Draw the predicted results from two different models on the left and right sides respec
    i_mobil_iou = display_annotated_results(valid_ids[i], valid_boxes_mobile[i], "mobile", co
    i_res_iou = display_annotated_results(valid_ids[i], valid_boxes_res[i], "ResNet", color='

    # Store the IoU of each image to assess the overall performance of the model
    mobile_iou.append(i_mobil_iou)
    res_iou.append(i_res_iou)

    # Organize the layout
    plt.tight_layout()

# Print the mean of the IoU list
print("ResNet: Avg.", np.mean(res_iou), "; each IoU:", res_iou)
print("MobileNet: Avg.", np.mean(mobile_iou), "; each IoU:", mobile_iou)
```

Result

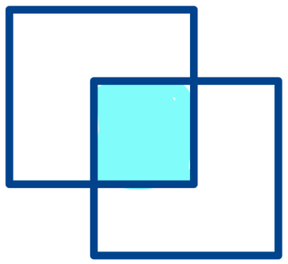


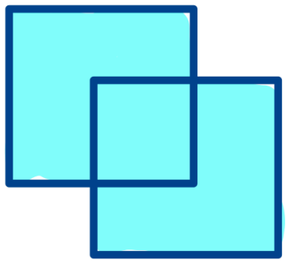
Supplement: IoU

- Ref: https://blog.csdn.net/IAMoldpan/article/details/78799857
- Ref: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

IoU (Intersection over Union) is a metric used to evaluate object detection algorithms. It is defined as the `intersection area` of the `predicted box` and the `true box` divided by their `union area`. The value ranges between 0 and 1, where a higher value indicates a greater overlap between the predicted and true boxes, signifying more accurate predictions.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



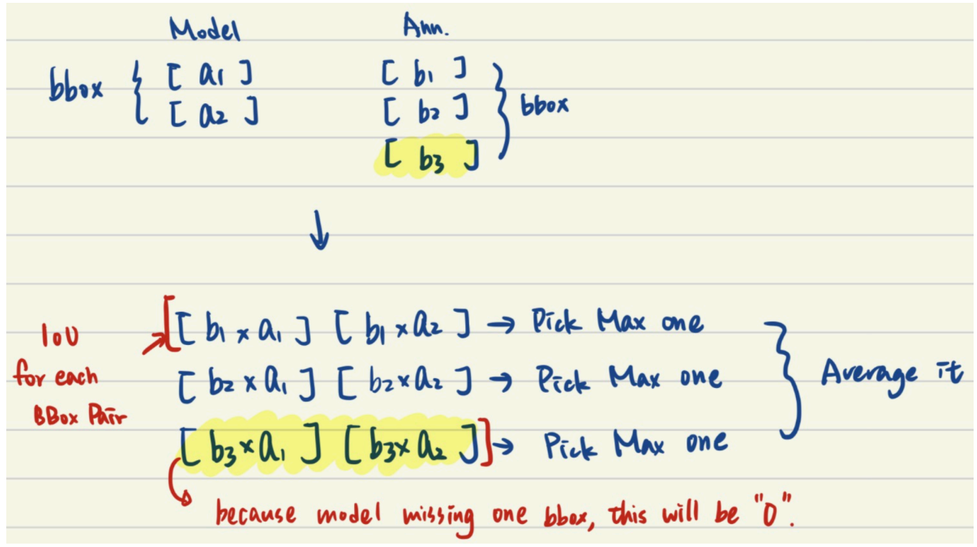


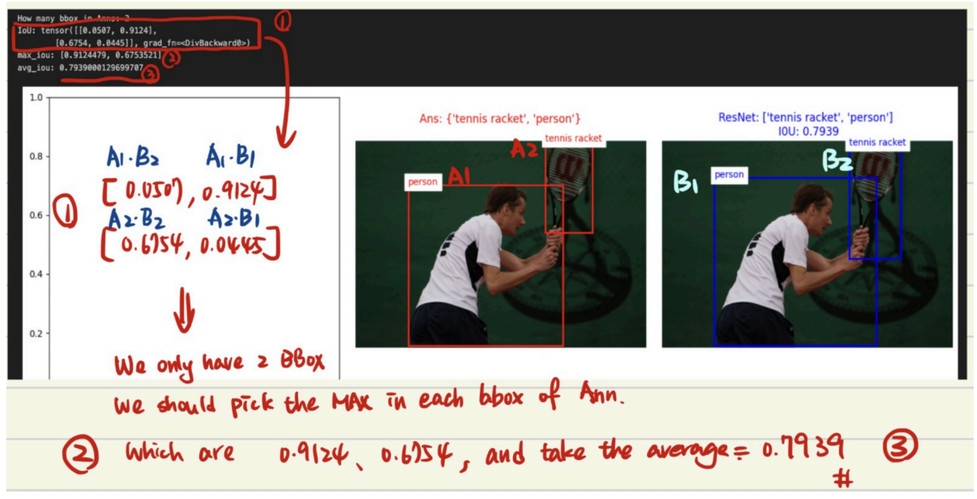
Source: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

> **From the example above, you might be wondering, what exactly does the following code segment do?**

```python
torchvision.ops.box_iou(bbox_tlist_anns, bbox_tlist_model)
```

- Essentially, it calculates the Intersection over Union (IoU) between all predicted bounding boxes of the ground truth and all predicted bounding boxes of the model. This process returns a tensor with the shape (number of ground truth bounding boxes, number of model's predicted bounding boxes). Refer to the images below.

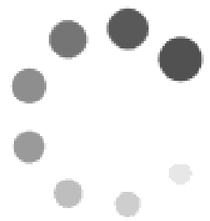




Here, we only need to obtain the maximum IoU for each ground truth bounding box and calculate the average value, so we use the following code

```python
# After obtaining the maximum value for each predicted box of the annotation (see supplementar
avg_iou = np.mean([t.cpu().detach().numpy().max() for t in iou]) # calculate the mean of IoU
```

> You might be curious whether using functions like `max()`, `mean()`, `sum()` will affect our results?



As we can see from the above image

- Using `sum()`, you may find that the value can exceed 1, which is not a reasonable range for IoU values.
- Using `max()`, it chooses, for each ground truth bounding box, the closest predicted bounding box from the model as its IoU. Then, we can obtain the maximum IoU values for `all predicted bounding boxes` of the ground truth and calculate the average to determine the overall IoU.
- Using `mean()` poses a problem as the IoU calculation will never be 1. This is because it considers the IoUs of other bounding boxes, which lowers the overall IoU. For instance, if the ground truth has two bounding boxes `[A1,A2]`, and the model also predicts two `[B1,B2]`, it's clear that B1 predicts A1, and B2 predicts A2, and the model predicts accurately. However, using mean will incorrectly consider B1 as A2 and B2 as A1, which is wrong, and these pairs have low IoUs. Thus, using `mean()` in this way will unjustly lower the IoU, making it unreasonable.

Mechine Learning

Flower102 Dataset - Using Transfer Learning to ...

Twitter Dataset - Using LSTM to predict the em...

Related Articles

2023-11-16

Twitter Dataset - Using LSTM to predict the emotion of the article

2023-09-30

MAC OS - PyTorch on Mac OS with GPU support

2023-10-31

Flower102 Dataset - Using Transfer Learning to train + Usin...

2024-09-04

Windows - A Step-by-Step Guide to Enable CUDA GPU on...

2023-10-24

CIFAR10 Dataset - Using Pytorch to build CNN + activate GPU +...

Comment

0 comments

Anonymous

Leave a comment

Markdown is supported

Login with GitHub

Preview

Be the first person to leave a comment!

Hi, welcome to Shannon blog!