DLCV_06_YOLO_colab

October 19, 2024

Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

[ ]:
```
NAME = "Ulugbek Shernazarov"
ID = "st125457"
```

1 06-YOLO

In this lab, we'll explore a fascinating use of image classification deep neural networks to peform a different task: object detection.

Credits: parts of this lab are based on other authors' code and blog posts:

- YOLO v3 in PyTorch: tutorial code, blog

1.1 Object Detection

If you could go back in time to the 1990s, there were no cameras that could find faces in a photograph, and no researcher had a way to count dogs in a video in real time. Everyone had to count the dogs manually. Times were very tough.

The Holy Grail of computer vision research at the time was real time face detection. If we could find faces in images fast enough, we could build systems that interact more naturally with human beings. But nobody had a solution.

Things changed when Viola and Jones introduced the first real time face detector, the Haar-like cascade, at the end of the 1990s. This technique swept a detection window over the input image at multiple sizes, and subjected each local patch to a cascade of simple rough classifiers. Each patch that made it to the end of the cascade of classifiers was treated as a positive detection. After a set of candidate patches were identified, there would be a cleanup stage when neighboring detections are clustered into isolated detections.

This method and one cousin, the HOG detector, which was slower but a little more accurate, dominated during the 2000s and on into the 2010s. These methods worked well enough when trained carefully on the specific environment they were used in, but usually couldn't be transfer to a new environment.

With the introduction of AlexNet and the amazing advances in image classification, we could follow the direction of R-CNN, to use a region proposal algorithm followed by a deep learning classifier to do object detection VERY slowly but much more accurately than the old real time methods.

1.2 What is YOLO?

However, it wasn't until YOLO that we had a deep learning model for object detection that could run in real time. It took some clever insight to realize that everything, from feature extraction to bounding box estimation, could actually be done with a single neural network that could be trained end-to-end to detect objects.

YOLO (You Only Look Once) uses only convolutional layers. This makes it a **"fully convolutional network" or FCN**.

YOLOv3 has 75 convolutional layers, with skip connections and upsampling layers. No pooling is used, but there is a convolutional layer with stride 2 used for downsampling. **Strided convoution** rather than pooling was used to prevent loss of fine-grained detailed information about the precise location of low-level features that would otherwise occur with pooling.

Normally, the output of a convolutional layer is a feature map. Applying convolutional layers to a possible detection window or region of interest (ROI) in the image then classifying the ROI's feature map is a reasonable method for detection prediction that is used in Fast R-CNN and Faster R-CNN. However, the innovation of YOLO was to use the feature map directly to predict bounding boxes and, for each bounding box, to predict whether or not an object is at the center of the bounding box. Finally, a classifier is used for each bounding box to indicate the content of the bounding box.

1.3 YOLO v3 from "scratch"

Early versions of YOLO were very fast but not nearly as accurate as their slower cousins. YOLO v3 included many of the tricks and techniques used by other models, such as multiscale analysis, and it achieved both high accuracy and fast inference.

Here we'll experiment with building up the YOLO v3 model in PyTorch. However, we won't train it ourselves, as that would require days of training; instead, we'll grab the weights for our PyTorch YOLO v3 from the original Darknet model by Joseph Redmon and friends.

1.3.1 Ground Truth Bounding Boxes

Here is how we present example images and corresponding object bounding boxes to the model.

The input image is divided into grid cells. The number of cells depends on the number of convolutional layers and the stride of each of those convolutional layers. For example, if we use a 416 $\times$ 416 input image size, and we apply 5 conv layers with a stride of 2 each (for a total downsampling factor of 32), we end up with a 13$\times$$13 feature map, each corresponding to a region in the original image of size 32 $\times$ 32 pixels.

A ground truth box has a center (x and y position), a width, and a height. Normally the ground truth boxes would be provided by a human annotator.

Each ground truth box's center must lie in some grid cell in the original image. Consider this example from the YOLO paper:

The grid is represented by the black lines. The ground truth bounding box for the object is the yellow rectangle. The center of this bounding box happens to be within the red-outlined grid cell.

The grid cell containing the center of a ground truth bounding box is given the responsibility during training to try to predict the presence of the object.

In order to indicate the presence of the given object, the model outputs several parameters for a given candidate object: - (t_x, t_y, t_w, t_h) indicate the box's location and size. During training, the targets for these outputs are the actual ground truth box parameters. - p_o is an "objectness" score that indicates the likelihood that an object exists in the given bounding box. This output uses a sigmoid function. During training, the target for p_o is set to 1 for the center grid cell (the red grid cell), and it is set to 0 for the the neighboring grid cells. - $(p_1, p_2, ..., p_n)$ are class confidence scores. They indicate the probability of the detected object belonging to a particular class. The targets, obviously, are set to 1 for the ground truth object class and 0 for other classes during training.

1.3.2 Anchor Boxes

One problem that would occur in YOLO if you tried to directly learn the parameters mentioned above is the problem of unstable gradients during training. In a way that is sort of analagous to how a residual block begins with an identity map and learns differences from identity, YOLO v3 uses the idea of anchor boxes originally introduced by the R-CNN team. Instead of predicting (t_x, t_y, t_w, t_h) directly, we predict how those parameters are *different from the parameters of a typical bounding box, an anchor box*. YOLO v3 uses three bounding boxes per cell. At training time, once ground truth bounding box's center is mapped to a grid cell, we find which of the anchors for that cell has the highest IoU with the ground truth box.

1.3.3 So What Does YOLO Actually Predict?

First, let's understand that all predictions are relative to the grid cell. YOLO predicts the following: - Offsets (t_x, t_y) are specified relative to the top left corner of the grid cell, as a ratio between 0 and 1, using a sigmoid to limit the values. - Height, and width (t_w, t_h) are specified relative to the dimensions of an anchor box.

Thus, YOLO does not predict absolute coordinates – it predicts values that can then be used to compute the box's position and size in absolute coordinates. This diagram gives the idea. We see that the absolute t_x is the grid cell's (c_x, c_y) plus $\sigma(t_x)$ times the grid cell width. Similarly for t_y. The absolute width of the predicted bounding box is the width of the anchor box times e^{tw}. Similarly for the height.

Hopefully you can see the difference between the YOLO v3 bounding box predictions and the Faster R-CNN bounding box predictions. The offset of the center is encoded relative to the grid cell containing the anchor box rather than the anchor box itself. The dimensions of the bounding box, however, similar to Faster R-CNN, are predicted relative to the anchor box size.

1.3.4 Multi-scale prediction

Rather than a single grid size and grid cell size, YOLO v3 detects objects at multiple sizes with downsampling factors of 32, 16, and 8. The largest objects are detected at the first, coarsest scale, whereas mid-sized objects are detected at the intermediate scale, and small objects are detected at the finest scale. The example below shows the three grid sizes relative to the image and an object:

1.3.5 YOLO dataset format

A YOLO dataset contains two sets of files, each pair with the same name: image files (in any supported format) and label files (in TXT, JSON, or XML format). A label file contains data in the format

$$i, C_x, C_y, L_x, L_y$$

- i: Label index
- C_x: Center position in the horizontal (x-axis) direction, encoded in the range 0-1, where 0 means the left edge of the image and 1 means the right edge.
- C_y: Center position in the vertical (y-axis) direction, encoded in the range 0-1, where 0 means the top edge of the image and 1 means the bottom edge.
- L_x: Object width, encoded in the range 0-1, where 1 means the width of the image.
- L_y: Object height, encoded in the range 0-1, where 1 means the height of the image.

To calculate these values for an object, suppose (W, H) is the actual size of a particular image in pixels, (O_x, O_y) is the actual position of an object in that image, in pixels, and (l_x, l_y) is the actual size of the object, again in pixels. The object label elements would be calculated as

$$C_x = \frac{O_x}{W},\ C_y = \frac{O_y}{H} L_x = \frac{l_x}{W},\ L_y = \frac{l_y}{H}$$

1.3.6 YOLOv3 Architecture

1.3.7 Preparation for Building YOLO in PyTorch

First of all, we will need OpenCV:

```
pip3 install --upgrade pip
pip install matplotlib opencv-python
```

Create a directory where the code for your detector will live.

In that directory, download util.py and darknet.py from https://github.com/ayooshkathuria/YOLO_v3_tutorial_

In Jupyter you would download thusly:

[ ]:
```
!wget https://raw.githubusercontent.com/ayooshkathuria/
 ↪YOLO_v3_tutorial_from_scratch/master/darknet.py
!wget https://raw.githubusercontent.com/ayooshkathuria/
 ↪YOLO_v3_tutorial_from_scratch/master/util.py
```

```
--2024-10-19 09:46:02--  https://raw.githubusercontent.com/ayooshkathuria/YOLO_v
3_tutorial_from_scratch/master/darknet.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.111.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 11533 (11K) [text/plain]
Saving to: ‘darknet.py.1’
```

```
darknet.py.1        100%[===================>]  11.26K  --.-KB/s    in 0s

2024-10-19 09:46:02 (124 MB/s) - ‘darknet.py.1’ saved [11533/11533]

--2024-10-19 09:46:03--  https://raw.githubusercontent.com/ayooshkathuria/YOLO_v
3_tutorial_from_scratch/master/util.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 7432 (7.3K) [text/plain]
Saving to: ‘util.py.1’

util.py.1           100%[===================>]   7.26K  --.-KB/s    in 0s

2024-10-19 09:46:03 (98.4 MB/s) - ‘util.py.1’ saved [7432/7432]
```

If you're in a Docker container, just run the `wget` commands at the command line. Make sure your proxy environment variables are set correctly.

1.3.8 Take a Look at the YOLO Darknet Configuration File

Next, let's download the `yolov3.cfg` configuration file and take a look. You could grab it from the canonical Darknet github repository or any other place it's stored.

[ ]:
```
!mkdir -p cfg
!wget https://raw.githubusercontent.com/ayooshkathuria/
 ↪YOLO_v3_tutorial_from_scratch/master/cfg/yolov3.cfg
!mv yolov3.cfg cfg/yolov3.cfg
```

```
--2024-10-19 09:46:03--  https://raw.githubusercontent.com/ayooshkathuria/YOLO_v
3_tutorial_from_scratch/master/cfg/yolov3.cfg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 8346 (8.2K) [text/plain]
Saving to: ‘yolov3.cfg’

yolov3.cfg          100%[===================>]   8.15K  --.-KB/s    in 0s

2024-10-19 09:46:03 (97.5 MB/s) - ‘yolov3.cfg’ saved [8346/8346]
```

The configuration file looks like this:

```
[net]
# Testing
batch=1
subdivisions=1
# Training
# batch=64
# subdivisions=16
width= 416

height = 416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1

[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

...

[shortcut]
from=-3
activation=linear

...

[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=80
num=9
jitter=.3
ignore_thresh = .7
```

```
truth_thresh = 1
random=1

[route]
layers = -4

[convolutional]
batch_normalize=1
filters=256
size=1
stride=1
pad=1
activation=leaky

[upsample]
stride=2

[route]
layers = -1, 61

...
```

1.3.9 Overview of the Configuration Blocks

The configuration blocks fall into a few cateogies:

- Net: the global configuration at the top of the configuration file. It declares the size of input images, batch size, learning rate, and so on.

  ```
  batch=64
  subdivisions=16
  width=608
  height=608
  channels=3
  momentum=0.9
  decay=0.0005
  angle=0
  saturation = 1.5
  exposure = 1.5
  hue=.1
  ```

- Convolutional: convolutional layer. Note that this specfication is a little more powerful than the PyTorch way of doing things, as options for batch normalization and the activation function (leaky ReLU in this case) are built in.

  ```
  [convolutional]
  batch_normalize=1
  filters=32
  size=3
  stride=1
  ```

```
pad=1
activation=leaky
```

- Shortcut: skip connections that implement residual blocks. -3 means to add the feature maps output by the previous layer to those output by the layer three layers back. Linear activation means identity (no nonlinear activation of the result).

  ```
  [shortcut]
  from=-3           # Connect the layer three layers back to here.
  activation=linear
  ```

- Upsample: Bilinear upsampling of the previous layer using a particular stride

  ```
  [upsample]
  stride=2
  ```

- Route: The route layer deserves a bit of explanation. It has an attribute `layers`, which can have either one or two values.

  ```
  [route]
  layers = -4

  [route]
  layers = -1, 61
  ```

  When the layers attribute has only one value, it outputs the feature maps of the layer indexed by the value. In our example, it is -4, so the layer will output the feature maps from the 4th layer backwards from the route layer.

  When layers has two values, it returns the concatenated feature maps of the layers indexed by its values. In our example it is -1, 61, so the layer will output feature maps from the previous layer (-1) and the 61st layer, concatenated along the channels (depth) dimension.

- YOLO:

  ```
  [yolo]
  mask = 0,1,2
  anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
  classes=80
  num=9
  jitter=.3
  ignore_thresh = .5
  truth_thresh = 1
  random=1
  ```

  Here we have a few important attributes:

  - anchors: describes the anchor boxes. The model contains 9 anchors, but only those in the `mask` are used.
  - mask: which anchor indices will be used in this YOLO layer
  - classes: number of object classes

1.3.10 Create a network from the config file

We are going to follow the general approach of some of the GitHub contributors who have developed PyTorch tools to deal with Darknet models. In the file `darknet.py`, there's a `parse_cfg` function. The function will read the Darknet configuration file and store the blocks in a dictionary.

We'll then create PyTorch NN Modules for each of the blocks in the Darknet configuration as implemented in the `create_modules` function. Take a look at this function for more understanding.

1.3.11 Convolutional block

1.3.12 Shortcut block

1.3.13 Upsample block

1.3.14 Route block

Why does it use an empty layer? The actual work will be done in the `forward()` function.

1.3.15 YOLO block

1.3.16 Using the code

OK, let's try it out. Depending on what you already have installed, you may need to run

```
# apt install libgl1-mesa-glx
```

for the next step to run.

[ ]:
```
import darknet

blocks = darknet.parse_cfg("cfg/yolov3.cfg")
print(darknet.create_modules(blocks))
```

```
({'type': 'net', 'batch': '1', 'subdivisions': '1', 'width': '416', 'height':
'416', 'channels': '3', 'momentum': '0.9', 'decay': '0.0005', 'angle': '0',
'saturation': '1.5', 'exposure': '1.5', 'hue': '.1', 'learning_rate': '0.001',
'burn_in': '1000', 'max_batches': '500200', 'policy': 'steps', 'steps':
'400000,450000', 'scales': '.1,.1'}, ModuleList(
  (0): Sequential(
    (conv_0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (batch_norm_0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_0): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (1): Sequential(
    (conv_1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
    (batch_norm_1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_1): LeakyReLU(negative_slope=0.1, inplace=True)
  )
```

```
  (2): Sequential(
    (conv_2): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_2): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (3): Sequential(
    (conv_3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (batch_norm_3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_3): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (4): Sequential(
    (shortcut_4): EmptyLayer()
  )
  (5): Sequential(
    (conv_5): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
    (batch_norm_5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_5): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (6): Sequential(
    (conv_6): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_6): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_6): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (7): Sequential(
    (conv_7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (batch_norm_7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_7): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (8): Sequential(
    (shortcut_8): EmptyLayer()
  )
  (9): Sequential(
    (conv_9): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_9): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (10): Sequential(
    (conv_10): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
```

```
    (batch_norm_10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_10): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (11): Sequential(
    (shortcut_11): EmptyLayer()
  )
  (12): Sequential(
    (conv_12): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (batch_norm_12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_12): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (13): Sequential(
    (conv_13): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_13): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_13): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (14): Sequential(
    (conv_14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_14): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (15): Sequential(
    (shortcut_15): EmptyLayer()
  )
  (16): Sequential(
    (conv_16): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_16): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_16): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (17): Sequential(
    (conv_17): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_17): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_17): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (18): Sequential(
    (shortcut_18): EmptyLayer()
  )
  (19): Sequential(
    (conv_19): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
    (batch_norm_19): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_19): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (20): Sequential(
    (conv_20): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_20): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_20): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (21): Sequential(
    (shortcut_21): EmptyLayer()
  )
  (22): Sequential(
    (conv_22): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_22): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_22): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (23): Sequential(
    (conv_23): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_23): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_23): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (24): Sequential(
    (shortcut_24): EmptyLayer()
  )
  (25): Sequential(
    (conv_25): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_25): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_25): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (26): Sequential(
    (conv_26): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_26): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_26): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (27): Sequential(
    (shortcut_27): EmptyLayer()
  )
  (28): Sequential(
    (conv_28): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
    (batch_norm_28): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_28): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (29): Sequential(
    (conv_29): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_29): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_29): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (30): Sequential(
    (shortcut_30): EmptyLayer()
  )
  (31): Sequential(
    (conv_31): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_31): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_31): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (32): Sequential(
    (conv_32): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_32): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_32): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (33): Sequential(
    (shortcut_33): EmptyLayer()
  )
  (34): Sequential(
    (conv_34): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_34): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_34): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (35): Sequential(
    (conv_35): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_35): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_35): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (36): Sequential(
    (shortcut_36): EmptyLayer()
  )
  (37): Sequential(
    (conv_37): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
```

```
1), bias=False)
    (batch_norm_37): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_37): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (38): Sequential(
    (conv_38): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_38): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_38): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (39): Sequential(
    (conv_39): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_39): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_39): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (40): Sequential(
    (shortcut_40): EmptyLayer()
  )
  (41): Sequential(
    (conv_41): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_41): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_41): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (42): Sequential(
    (conv_42): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_42): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_42): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (43): Sequential(
    (shortcut_43): EmptyLayer()
  )
  (44): Sequential(
    (conv_44): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_44): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_44): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (45): Sequential(
    (conv_45): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_45): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (leaky_45): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (46): Sequential(
    (shortcut_46): EmptyLayer()
  )
  (47): Sequential(
    (conv_47): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_47): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_47): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (48): Sequential(
    (conv_48): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_48): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_48): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (49): Sequential(
    (shortcut_49): EmptyLayer()
  )
  (50): Sequential(
    (conv_50): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_50): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_50): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (51): Sequential(
    (conv_51): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_51): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_51): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (52): Sequential(
    (shortcut_52): EmptyLayer()
  )
  (53): Sequential(
    (conv_53): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_53): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_53): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (54): Sequential(
    (conv_54): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_54): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (leaky_54): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (55): Sequential(
    (shortcut_55): EmptyLayer()
  )
  (56): Sequential(
    (conv_56): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_56): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_56): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (57): Sequential(
    (conv_57): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_57): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_57): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (58): Sequential(
    (shortcut_58): EmptyLayer()
  )
  (59): Sequential(
    (conv_59): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_59): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_59): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (60): Sequential(
    (conv_60): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_60): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_60): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (61): Sequential(
    (shortcut_61): EmptyLayer()
  )
  (62): Sequential(
    (conv_62): Conv2d(512, 1024, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (batch_norm_62): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_62): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (63): Sequential(
    (conv_63): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_63): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (leaky_63): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (64): Sequential(
    (conv_64): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_64): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_64): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (65): Sequential(
    (shortcut_65): EmptyLayer()
  )
  (66): Sequential(
    (conv_66): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_66): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_66): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (67): Sequential(
    (conv_67): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_67): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_67): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (68): Sequential(
    (shortcut_68): EmptyLayer()
  )
  (69): Sequential(
    (conv_69): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_69): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_69): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (70): Sequential(
    (conv_70): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_70): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_70): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (71): Sequential(
    (shortcut_71): EmptyLayer()
  )
  (72): Sequential(
    (conv_72): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_72): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (leaky_72): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (73): Sequential(
    (conv_73): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_73): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_73): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (74): Sequential(
    (shortcut_74): EmptyLayer()
  )
  (75): Sequential(
    (conv_75): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_75): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_75): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (76): Sequential(
    (conv_76): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_76): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_76): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (77): Sequential(
    (conv_77): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_77): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_77): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (78): Sequential(
    (conv_78): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_78): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_78): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (79): Sequential(
    (conv_79): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_79): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_79): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (80): Sequential(
    (conv_80): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_80): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
    (leaky_80): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (81): Sequential(
    (conv_81): Conv2d(1024, 255, kernel_size=(1, 1), stride=(1, 1))
  )
  (82): Sequential(
    (Detection_82): DetectionLayer()
  )
  (83): Sequential(
    (route_83): EmptyLayer()
  )
  (84): Sequential(
    (conv_84): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_84): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_84): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (85): Sequential(
    (upsample_85): Upsample(scale_factor=2.0, mode='nearest')
  )
  (86): Sequential(
    (route_86): EmptyLayer()
  )
  (87): Sequential(
    (conv_87): Conv2d(768, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_87): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_87): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (88): Sequential(
    (conv_88): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_88): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_88): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (89): Sequential(
    (conv_89): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_89): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_89): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (90): Sequential(
    (conv_90): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_90): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (leaky_90): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (91): Sequential(
    (conv_91): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_91): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_91): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (92): Sequential(
    (conv_92): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_92): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_92): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (93): Sequential(
    (conv_93): Conv2d(512, 255, kernel_size=(1, 1), stride=(1, 1))
  )
  (94): Sequential(
    (Detection_94): DetectionLayer()
  )
  (95): Sequential(
    (route_95): EmptyLayer()
  )
  (96): Sequential(
    (conv_96): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_96): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_96): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (97): Sequential(
    (upsample_97): Upsample(scale_factor=2.0, mode='nearest')
  )
  (98): Sequential(
    (route_98): EmptyLayer()
  )
  (99): Sequential(
    (conv_99): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_99): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_99): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (100): Sequential(
    (conv_100): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_100): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_100): LeakyReLU(negative_slope=0.1, inplace=True)
```

```
  )
  (101): Sequential(
    (conv_101): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_101): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_101): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (102): Sequential(
    (conv_102): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_102): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_102): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (103): Sequential(
    (conv_103): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_103): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_103): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (104): Sequential(
    (conv_104): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_104): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_104): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (105): Sequential(
    (conv_105): Conv2d(256, 255, kernel_size=(1, 1), stride=(1, 1))
  )
  (106): Sequential(
    (Detection_106): DetectionLayer()
  )
))
```

1.3.17 Darknet class

Let's make our own version of the `Darknet` class in `darknet.py`.

The class has two main functions:

1. `forward()`: forward propagation, following the instructions in the dictionary modules
2. `load_weights()`: load a set of pretrained weights into the network

[ ]:
```
from util import *

class MyDarknet(nn.Module):
    def __init__(self, cfgfile):
        super(MyDarknet, self).__init__()
```

```
        # load the config file and create our model
        self.blocks = darknet.parse_cfg(cfgfile)
        self.net_info, self.module_list = darknet.create_modules(self.blocks)

    def forward(self, x, CUDA:bool):
        modules = self.blocks[1:]
        outputs = {}   #We cache the outputs for the route layer

        write = 0
        # run forward propagation. Follow the instruction from dictionary␣
↪modules
        for i, module in enumerate(modules):
            module_type = (module["type"])

            if module_type == "convolutional" or module_type == "upsample":
                # do convolutional network
                x = self.module_list[i](x)

            elif module_type == "route":
                # concat layers
                layers = module["layers"]
                layers = [int(a) for a in layers]

                if (layers[0]) > 0:
                    layers[0] = layers[0] - i

                if len(layers) == 1:
                    x = outputs[i + (layers[0])]

                else:
                    if (layers[1]) > 0:
                        layers[1] = layers[1] - i

                    map1 = outputs[i + layers[0]]
                    map2 = outputs[i + layers[1]]
                    x = torch.cat((map1, map2), 1)


            elif  module_type == "shortcut":
                from_ = int(module["from"])
                # residual network
                x = outputs[i-1] + outputs[i+from_]

            elif module_type == 'yolo':
                anchors = self.module_list[i][0].anchors
                #Get the input dimensions
                inp_dim = int (self.net_info["height"])
```

```
            #Get the number of classes
            num_classes = int (module["classes"])

            #Transform
            # predict_transform is in util.py
            batch_size = x.size(0)
            stride =  inp_dim // x.size(2)
            grid_size = inp_dim // stride
            bbox_attrs = 5 + num_classes
            num_anchors = len(anchors)

            x = predict_transform(x, inp_dim, anchors, num_classes, CUDA)
            if not write:              #if no collector has been intialised.
                detections = x
                write = 1

            else:
                detections = torch.cat((detections, x), 1)

        outputs[i] = x

    return detections


def load_weights(self, weightfile):
    '''
    Load pretrained weight
    '''
    #Open the weights file
    fp = open(weightfile, "rb")

    #The first 5 values are header information
    # 1. Major version number
    # 2. Minor Version Number
    # 3. Subversion number
    # 4,5. Images seen by the network (during training)
    header = np.fromfile(fp, dtype = np.int32, count = 5)
    self.header = torch.from_numpy(header)
    self.seen = self.header[3]

    weights = np.fromfile(fp, dtype = np.float32)

    ptr = 0
    for i in range(len(self.module_list)):
        module_type = self.blocks[i + 1]["type"]
```

```
            #If module_type is convolutional load weights
            #Otherwise ignore.

            if module_type == "convolutional":
                model = self.module_list[i]
                try:
                    batch_normalize = int(self.blocks[i+1]["batch_normalize"])
                except:
                    batch_normalize = 0

                conv = model[0]


                if (batch_normalize):
                    bn = model[1]

                    #Get the number of weights of Batch Norm Layer
                    num_bn_biases = bn.bias.numel()

                    #Load the weights
                    bn_biases = torch.from_numpy(weights[ptr:ptr +␣
↪num_bn_biases])
                    ptr += num_bn_biases

                    bn_weights = torch.from_numpy(weights[ptr: ptr +␣
↪num_bn_biases])
                    ptr  += num_bn_biases

                    bn_running_mean = torch.from_numpy(weights[ptr: ptr +␣
↪num_bn_biases])
                    ptr  += num_bn_biases

                    bn_running_var = torch.from_numpy(weights[ptr: ptr +␣
↪num_bn_biases])
                    ptr  += num_bn_biases

                    #Cast the loaded weights into dims of model weights.
                    bn_biases = bn_biases.view_as(bn.bias.data)
                    bn_weights = bn_weights.view_as(bn.weight.data)
                    bn_running_mean = bn_running_mean.view_as(bn.running_mean)
                    bn_running_var = bn_running_var.view_as(bn.running_var)

                    #Copy the data to model
                    bn.bias.data.copy_(bn_biases)
                    bn.weight.data.copy_(bn_weights)
                    bn.running_mean.copy_(bn_running_mean)
                    bn.running_var.copy_(bn_running_var)
```

```
                else:
                    #Number of biases
                    num_biases = conv.bias.numel()

                    #Load the weights
                    conv_biases = torch.from_numpy(weights[ptr: ptr +␣
↪num_biases])
                    ptr = ptr + num_biases

                    #reshape the loaded weights according to the dims of the␣
↪model weights
                    conv_biases = conv_biases.view_as(conv.bias.data)

                    #Finally copy the data
                    conv.bias.data.copy_(conv_biases)

                #Let us load the weights for the Convolutional layers
                num_weights = conv.weight.numel()

                #Do the same as above for weights
                conv_weights = torch.from_numpy(weights[ptr:ptr+num_weights])
                ptr = ptr + num_weights

                conv_weights = conv_weights.view_as(conv.weight.data)
                conv.weight.data.copy_(conv_weights)
```

1.3.18 Test Forward Propagation

Let's propagate a single image through the network and see what we get.

[ ]:
```
!wget https://github.com/ayooshkathuria/pytorch-yolo-v3/raw/master/
↪dog-cycle-car.png
```

```
--2024-10-19 09:46:06--  https://github.com/ayooshkathuria/pytorch-
yolo-v3/raw/master/dog-cycle-car.png
Resolving github.com (github.com)… 20.205.243.166
Connecting to github.com (github.com)|20.205.243.166|:443… connected.
HTTP request sent, awaiting response… 302 Found
Location: https://raw.githubusercontent.com/ayooshkathuria/pytorch-
yolo-v3/master/dog-cycle-car.png [following]
--2024-10-19 09:46:06--
https://raw.githubusercontent.com/ayooshkathuria/pytorch-yolo-v3/master/dog-
cycle-car.png
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
```

```
HTTP request sent, awaiting response… 200 OK
Length: 347445 (339K) [image/png]
Saving to: ‘dog-cycle-car.png.1’

dog-cycle-car.png.1 100%[===================>] 339.30K  --.-KB/s    in 0.005s

2024-10-19 09:46:07 (65.0 MB/s) - ‘dog-cycle-car.png.1’ saved [347445/347445]
```

Here's code to load the image into memory and push it through the model:

[ ]:
```
import cv2
import torch

def get_test_input():
    img = cv2.imread("dog-cycle-car.png")
    img = cv2.resize(img, (416,416))          #Resize to the input dimension
    img_ =  img[:,:,::-1].transpose((2,0,1))  # BGR -> RGB | H X W C -> C X H X␣
 ↪W
    img_ = img_[np.newaxis,:,:,:]/255.0       #Add a channel at 0 (for batch) |␣
 ↪Normalise
    img_ = torch.from_numpy(img_).float()     #Convert to float
    img_ = Variable(img_)                     # Convert to Variable
    return img_
```

Go ahead and try it (noting that the model hasn't been trained so we don't expect any correct result):

[ ]:
```
from util import *

model = MyDarknet("cfg/yolov3.cfg")
inp = get_test_input()
pred = model(inp, False)
print (pred)
```

```
tensor([[[1.3927e+01, 1.8489e+01, 1.1397e+02,  …, 4.8350e-01,
          5.6331e-01, 5.1292e-01],
         [1.7388e+01, 1.5664e+01, 1.2931e+02,  …, 5.3444e-01,
          5.0698e-01, 5.6196e-01],
         [1.3788e+01, 1.6116e+01, 3.4173e+02,  …, 4.0188e-01,
          5.7769e-01, 4.2387e-01],
         …,
         [4.1091e+02, 4.1140e+02, 1.6161e+01,  …, 4.9643e-01,
          5.2532e-01, 5.4997e-01],
         [4.1225e+02, 4.1185e+02, 2.2585e+01,  …, 4.9896e-01,
          5.1473e-01, 5.5245e-01],
         [4.1195e+02, 4.1205e+02, 3.0434e+01,  …, 4.3810e-01,
          4.7809e-01, 4.7495e-01]]], grad_fn=<CatBackward0>)
```

1.3.19 Understanding the output result

The result from prediction model will be $B(13 \cdot 13 + 26 \cdot 26 + 52 \cdot 52)3 \cdot 85$. Why? We have - B: the number of images in the batch - $13 \cdot 13$: number of elements (grid cells) in the coarsest feature map - $26 \cdot 16$: number of elements (grid cells) in the medium scale feature map - $52 \cdot 52$: number of elements (grid cells) in the finest cale feature map - 3: the number of anchor boxes per grid cell - 85: number of bounding box attributes (4 for bounding box, 1 for objectness, 80 for the COCO classes)

1.3.20 Download a pretrained weight file

Darknet stores weights as in this diagram:

[ ]:
```
!wget https://pjreddie.com/media/files/yolov3.weights
```

```
--2024-10-19 09:46:08--  https://pjreddie.com/media/files/yolov3.weights
Resolving pjreddie.com (pjreddie.com)… 162.0.215.52
Connecting to pjreddie.com (pjreddie.com)|162.0.215.52|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 248007048 (237M) [application/octet-stream]
Saving to: ‘yolov3.weights.1’

yolov3.weights.1    100%[===================>] 236.52M  16.4MB/s    in 15s

2024-10-19 09:46:25 (15.5 MB/s) - ‘yolov3.weights.1’ saved [248007048/248007048]
```

You can download the yolov3 weights from here

[ ]:
```
model.load_weights("yolov3.weights")
```

1.3.21 Test with the sample image again

[ ]:
```
inp = get_test_input()
pred = model(inp, False)
print (pred)
```

```
tensor([[[8.5426e+00, 1.9015e+01, 1.1130e+02,  …, 1.7305e-03,
          1.3874e-03, 9.2979e-04],
         [1.4105e+01, 1.8868e+01, 9.4014e+01,  …, 5.9498e-04,
          9.2469e-04, 1.3084e-03],
         [2.1125e+01, 1.5269e+01, 3.5793e+02,  …, 8.3604e-03,
          5.1065e-03, 5.8559e-03],
         …,
         [4.1268e+02, 4.1069e+02, 3.7159e+00,  …, 1.7188e-06,
          4.0959e-06, 6.5904e-07],
         [4.1132e+02, 4.1023e+02, 8.0354e+00,  …, 1.3928e-05,
          3.2255e-05, 1.2078e-05],
         [4.1076e+02, 4.1318e+02, 4.9634e+01,  …, 4.2181e-06,
          1.0795e-05, 1.8107e-05]]], grad_fn=<CatBackward0>)
```

1.3.22 From YOLO output tensor to *true* detections

In the prediction result, there are many results. We need to threshold them using the objectness score output for each bounding box prediction. The `write_results` function in `util.py` does just that.

`def write_results(prediction, confidence, num_classes, nms_conf = 0.4)`

- prediction: prediction result tensor returned from the YOLO model
- confidence: objectness score threshold to apply to the set of detections
- num_classes: number of classes to expect
- nms_conf: NMS IoU threshold

NMS stands for "non-maxima suppression." The basic idea is that if you have two predicted bounding boxes that overlap each other significantly, you should throw away the box with the lower confidence score. Overlap is measured by IoU (Intersection over Union), wich is just the ratio of the the area of intersection of the two regions with the area of the union of the two regions:

$$IoU(R_1, R_2) = \frac{|R_1 \cap R_2|}{|R_1 \cup R_2|}.$$

The default of 0.4 means if the intersection is 40% or more of the union, the two bounding boxes are overlapping enough that only one of the detections should survive.

[ ]:
```
write_results(pred.detach(), 0.5, 80, nms_conf = 0.4)
```

[ ]:
```
tensor([[  0.0000,  61.5397, 100.8604, 307.2721, 303.1128,   0.9469,   0.9985,
           1.0000],
        [  0.0000, 253.8480,  66.1098, 378.0398, 118.0090,   0.9992,   0.8164,
           7.0000],
        [  0.0000,  71.0339, 163.2243, 175.7470, 382.2701,   0.9999,   0.9936,
          16.0000]])
```

1.3.23 Show the resulting detections on top of an image

The model was trained on the COCO dataset, so download the class label file `coco.names`:

[ ]:
```
!wget https://raw.githubusercontent.com/ayooshkathuria/
 ↪YOLO_v3_tutorial_from_scratch/master/data/coco.names
!mkdir data
!mv coco.names data/coco.names
```

```
--2024-10-19 09:46:26--  https://raw.githubusercontent.com/ayooshkathuria/YOLO_v
3_tutorial_from_scratch/master/data/coco.names
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.109.133, 185.199.111.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 625 [text/plain]
```

```
Saving to: ‘coco.names’

coco.names          100%[===================>]     625  --.-KB/s    in 0s

2024-10-19 09:46:26 (46.3 MB/s) - ‘coco.names’ saved [625/625]

mkdir: cannot create directory ‘data’: File exists
```

[ ]:
```
def load_classes(namesfile):
    fp = open(namesfile, "r")
    names = fp.read().split("\n")[:-1]
    return names
```

[ ]:
```
num_classes = 80
classes = load_classes("data/coco.names")
print(classes)
```

```
['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',
'pottedplant', 'bed', 'diningtable', 'toilet', 'tvmonitor', 'laptop', 'mouse',
'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink',
'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier',
'toothbrush']
```

So we see that the three surviving bounding boxes above, outputting object types 1, 7, and 16, indicate a bicycle, a truck, and a dog. Let's draw the detections on top of the input image for better visualization.

We'll use some code based on Kathuria's `detect.py`. You can download the original as

[ ]:
```
!wget https://raw.githubusercontent.com/ayooshkathuria/
↪YOLO_v3_tutorial_from_scratch/master/detect.py
```

```
--2024-10-19 09:46:27--  https://raw.githubusercontent.com/ayooshkathuria/YOLO_v
3_tutorial_from_scratch/master/detect.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.109.133, 185.199.111.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.109.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 7273 (7.1K) [text/plain]
Saving to: ‘detect.py.1’

detect.py.1         100%[===================>]   7.10K  --.-KB/s    in 0s
```

```
2024-10-19 09:46:27 (93.1 MB/s) - ‘detect.py.1’ saved [7273/7273]
```

Here's our version. It will process the images in subdirectory `cocoimages` so let's make it and put our sample there:

[ ]:
```
!mkdir -p cocoimages
!cp dog-cycle-car.png cocoimages/
```

[ ]:
```
from __future__ import division
import time
import torch
import torch.nn as nn
from torch.autograd import Variable
import numpy as np
import cv2
from util import *
import argparse
import os
import os.path as osp
from darknet import Darknet
import pickle as pkl
import pandas as pd
import random

images = "cocoimages"
batch_size = 4
confidence = 0.5
nms_thesh = 0.4
start = 0
CUDA = torch.cuda.is_available()

num_classes = 80
classes = load_classes("data/coco.names")

#Set up the neural network

print("Loading network.....")
model = MyDarknet("cfg/yolov3.cfg")
model.load_weights("yolov3.weights")
print("Network successfully loaded")

model.net_info["height"] = 416
inp_dim = int(model.net_info["height"])
assert inp_dim % 32 == 0
assert inp_dim > 32
```

```
#If there's a GPU availible, put the model on GPU

if CUDA:
    model.cuda()

# Set the model in evaluation mode

model.eval()

read_dir = time.time()

# Detection phase

try:
    imlist = [osp.join(osp.realpath('.'), images, img) for img in os.
 ↪listdir(images)]
except NotADirectoryError:
    imlist = []
    imlist.append(osp.join(osp.realpath('.'), images))
except FileNotFoundError:
    print ("No file or directory with the name {}".format(images))
    exit()

if not os.path.exists("des"):
    os.makedirs("des")

load_batch = time.time()
loaded_ims = [cv2.imread(x) for x in imlist]

im_batches = list(map(prep_image, loaded_ims, [inp_dim for x in␣
 ↪range(len(imlist))]))
im_dim_list = [(x.shape[1], x.shape[0]) for x in loaded_ims]
im_dim_list = torch.FloatTensor(im_dim_list).repeat(1,2)


leftover = 0
if (len(im_dim_list) % batch_size):
    leftover = 1

if batch_size != 1:
    num_batches = len(imlist) // batch_size + leftover
    im_batches = [torch.cat((im_batches[i*batch_size : min((i +  1)*batch_size,
                         len(im_batches))]))  for i in range(num_batches)]

write = 0

if CUDA:
```

```
    im_dim_list = im_dim_list.cuda()

start_det_loop = time.time()
for i, batch in enumerate(im_batches):
    # Load the image
    start = time.time()
    if CUDA:
        batch = batch.cuda()
    with torch.no_grad():
        prediction = model(Variable(batch), CUDA)

    prediction = write_results(prediction, confidence, num_classes, nms_conf =␣
 ↪nms_thesh)

    end = time.time()

    if type(prediction) == int:

        for im_num, image in enumerate(imlist[i*batch_size: min((i + ␣
 ↪1)*batch_size, len(imlist))]):
            im_id = i*batch_size + im_num
            print("{0:20s} predicted in {1:6.3f} seconds".format(image.split("/
 ↪")[-1], (end - start)/batch_size))
            print("{0:20s} {1:s}".format("Objects Detected:", ""))
            print("----------------------------------------------------------")
        continue

    prediction[:,0] += i*batch_size    #transform the atribute from index in␣
 ↪batch to index in imlist

    if not write:                      #If we have't initialised output
        output = prediction
        write = 1
    else:
        output = torch.cat((output,prediction))

    for im_num, image in enumerate(imlist[i*batch_size: min((i + ␣
 ↪1)*batch_size, len(imlist))]):
        im_id = i*batch_size + im_num
        objs = [classes[int(x[-1])] for x in output if int(x[0]) == im_id]
        print("{0:20s} predicted in {1:6.3f} seconds".format(image.split("/
 ↪")[-1], (end - start)/batch_size))
        print("{0:20s} {1:s}".format("Objects Detected:", " ".join(objs)))
        print("----------------------------------------------------------")

    if CUDA:
```

```
        torch.cuda.synchronize()
try:
    output
except NameError:
    print ("No detections were made")
    exit()

im_dim_list = torch.index_select(im_dim_list, 0, output[:,0].long())

scaling_factor = torch.min(416/im_dim_list,1)[0].view(-1,1)

output[:,[1,3]] -= (inp_dim - scaling_factor*im_dim_list[:,0].view(-1,1))/2
output[:,[2,4]] -= (inp_dim - scaling_factor*im_dim_list[:,1].view(-1,1))/2

output[:,1:5] /= scaling_factor

for i in range(output.shape[0]):
    output[i, [1,3]] = torch.clamp(output[i, [1,3]], 0.0, im_dim_list[i,0])
    output[i, [2,4]] = torch.clamp(output[i, [2,4]], 0.0, im_dim_list[i,1])

output_recast = time.time()
class_load = time.time()
colors = [[255, 0, 0], [255, 0, 0], [255, 255, 0], [0, 255, 0], [0, 255, 255],␣
 ↪[0, 0, 255], [255, 0, 255]]

draw = time.time()

def write(x, results):
    c1 = tuple(x[1:3].int())
    c2 = tuple(x[3:5].int())
    img = results[int(x[0])]
    cls = int(x[-1])
    color = random.choice(colors)
    label = "{0}".format(classes[cls])
    cv2.rectangle(img, c1, c2,color, 1)
    t_size = cv2.getTextSize(label, cv2.FONT_HERSHEY_PLAIN, 1 , 1)[0]
    c2 = c1[0] + t_size[0] + 3, c1[1] + t_size[1] + 4
    cv2.rectangle(img, c1, c2,color, -1)
    cv2.putText(img, label, (c1[0], c1[1] + t_size[1] + 4), cv2.
 ↪FONT_HERSHEY_PLAIN, 1, [225,255,255], 1);
    return img


# list(map(lambda x: write(x, loaded_ims), output))

det_names = pd.Series(imlist).apply(lambda x: "{}/det_{}".format("des",x.
 ↪split("/")[-1]))
```

```
list(map(cv2.imwrite, det_names, loaded_ims))

end = time.time()

print("SUMMARY")
print("----------------------------------------------------------")
print("{:25s}: {}".format("Task", "Time Taken (in seconds)"))
print()
print("{:25s}: {:2.3f}".format("Reading addresses", load_batch - read_dir))
print("{:25s}: {:2.3f}".format("Loading batch", start_det_loop - load_batch))
print("{:25s}: {:2.3f}".format("Detection (" + str(len(imlist)) +  " images)",␣
 ↪output_recast - start_det_loop))
print("{:25s}: {:2.3f}".format("Output Processing", class_load - output_recast))
print("{:25s}: {:2.3f}".format("Drawing Boxes", end - draw))
print("{:25s}: {:2.3f}".format("Average time_per_img", (end - load_batch)/
 ↪len(imlist)))
print("----------------------------------------------------------")


torch.cuda.empty_cache()
```

```
Loading network…
Network successfully loaded
dog-cycle-car.png    predicted in  0.087 seconds
Objects Detected:    bicycle truck dog
----------------------------------------------------------
SUMMARY
----------------------------------------------------------
Task                      : Time Taken (in seconds)

Reading addresses         : 0.000
Loading batch             : 0.016
Detection (1 images)      : 0.809
Output Processing         : 0.000
Drawing Boxes             : 0.016
Average time_per_img      : 0.841
----------------------------------------------------------
```

Voila! You got the YOLO result in folder “des”

1.3.24 Training in YOLOv3

Now let’s create training function. There are 2 parts: - Dataset - Training

Dataset You can download COCO dataset from COCO dataset websit. If you want to create dataset, you must create an images folder and put annotation file in json coco format in another position.

For your customdataset, there are many application can create the annotation file in coco format. For example: makesense

Put the folder name in code:

[ ]:
```
path2data_train="/path/to/train/images/"
path2json_train="/path/to/train/coco_annotation.json"

path2data_val="/path/to/validate/images/"
path2json_val="/path/to/validate/coco_annotation.json"
```

[ ]:
```
# path2data_train="/home/alisa/fiftyone/coco-2017/train2017/data/"
# path2json_train="/home/alisa/fiftyone/coco-2017/annotations/
 ↪instances_train2017.json"

# path2data_val="/home/alisa/fiftyone/coco-2017/val2017/data"
# path2json_val="/home/alisa/fiftyone/coco-2017/annotations/instances_val2017.
 ↪json"


path2data_train="/home/alisa/fiftyone/coco-2017/train2017/data/"
path2json_train="/home/alisa/fiftyone/coco-2017/annotations/instances_train2017.
 ↪json"

path2data_val="/home/alisa/fiftyone/coco-2017/val2017/data"
path2json_val="/home/alisa/fiftyone/coco-2017/annotations/instances_val2017.
 ↪json"

img_size = 416
```

Create custom dataset in coco format. The important thing of the dataset is annotation position. You need to create annotation in each image into yolo format. Thus, in class CustomCoco, it contains *create_label* function for convert bounding box in each class into yolo label

[ ]:
```
# coding=utf-8
import os
import sys
from PIL import Image
import torch
import math
import cv2
import numpy as np
import torch
import random
from torchvision.datasets import CocoDetection

from typing import Any, Callable, Optional, Tuple
import json
```

```
ANCHORS = [
            [[12, 16], [19, 36], [40, 28]],
            [[36, 75], [76, 55], [72, 146]],
            [[142, 110], [192, 243], [459, 401]]
]

STRIDES = [8, 16, 32]

IP_SIZE = img_size
NUM_ANCHORS = 3
NUM_CLASSES = 80

with open('coco_cats.json') as js:
    data = json.load(js)["categories"]

cats_dict = {}
for i in range(0, 80):
    cats_dict[str(data[i]['id'])] = i



class CustomCoco(CocoDetection):
    def __init__(
            self,
            root: str,
            annFile: str,
            transform: Optional[Callable] = None,
            target_transform: Optional[Callable] = None,
            transforms: Optional[Callable] = None,
    ) -> None:
        super(CocoDetection, self).__init__(root, transforms, transform,␣
 ↪target_transform)
        from pycocotools.coco import COCO
        self.coco = COCO(annFile)
        self.ids = list(sorted(self.coco.imgs.keys()))


    def __getitem__(self, index: int) -> Tuple[Any, Any]:
        """
        Args:
            index (int): Index

        Returns:
            tuple: Tuple (image, target). target is the object returned by␣
 ↪``coco.loadAnns``.
        """
```

```
        coco = self.coco
        img_id = self.ids[index]
        ann_ids = coco.getAnnIds(imgIds=img_id)
        target = coco.loadAnns(ann_ids)

        path = coco.loadImgs(img_id)[0]['file_name']

        img = Image.open(os.path.join(self.root, path)).convert('RGB')
        img = np.array(img)

        labels = list(obj['category_id'] for obj in target)
        bboxes = list(obj['bbox'] for obj in target)

        if self.transform is not None:
            bboxes = list(obj['bbox'] for obj in target)
            category_ids = list(obj['category_id'] for obj in target)
            transformed = self.transform(image=img, bboxes=bboxes,␣
↪category_ids=category_ids)
            img = transformed['image'],
            bboxes = torch.Tensor(transformed['bboxes'])
            cat_ids = torch.Tensor(transformed['category_ids'])
            labels, bboxes = self.__create_label(bboxes, cat_ids.type(torch.
↪IntTensor))

        return img, labels, bboxes

    def __len__(self) -> int:
        return len(self.ids)

    def __create_label(self, bboxes, class_inds):
        """
        Label assignment. For a single picture all GT box bboxes are assigned␣
↪anchor.
        1 Select a bbox in order, convert its coordinates("xyxy") to "xywh";␣
↪and scale bbox'
            xywh by the strides.
        2 Calculate the iou between the each detection layer'anchors and the␣
↪bbox in turn, and select the largest
            anchor to predict the bbox.If the ious of all detection layers are␣
↪smaller than 0.3, select the largest
            of all detection layers' anchors to predict the bbox.
        Note :
        1 The same GT may be assigned to multiple anchors. And the anchors may␣
↪be on the same or different layer.
        2 The total number of bboxes may be more than it is, because the same␣
↪GT may be assigned to multiple layers
```

```
        of detection.
        """
        # print("Class indices: ", class_inds)
        bboxes = np.array(bboxes)
        class_inds = np.array(class_inds)
        anchors = ANCHORS # all the anchors
        strides = np.array(STRIDES) # list of strides
        train_output_size = IP_SIZE / strides # image with different scales
        anchors_per_scale = NUM_ANCHORS # anchor per scale

        label = [
            np.zeros(
                (
                    int(train_output_size[i]),
                    int(train_output_size[i]),
                    anchors_per_scale,
                    5 + NUM_CLASSES,
                )
            )
            for i in range(3)
        ]
        # 150 bounding box ground truths per scale
        bboxes_xywh = [
            np.zeros((150, 4)) for _ in range(3)
        ]  # Darknet the max_num is 30
        bbox_count = np.zeros((3,))

        for i in range(len(bboxes)):
            bbox_coor = bboxes[i][:4]
            bbox_class_ind = cats_dict[str(class_inds[i])]

            # onehot
            one_hot = np.zeros(NUM_CLASSES, dtype=np.float32)
            one_hot[bbox_class_ind] = 1.0
            # one_hot_smooth = dataAug.LabelSmooth()(one_hot, self.num_classes)

            # convert "xyxy" to "xywh"
            bbox_xywh = np.concatenate(
                [
                    (0.5 * bbox_coor[2:] + bbox_coor[:2]) ,
                    bbox_coor[2:],
                ],
                axis=-1,
            )

            bbox_xywh_scaled = (
                1.0 * bbox_xywh[np.newaxis, :] / strides[:, np.newaxis]
```

```
            )

            iou = []
            exist_positive = False
            for i in range(3):
                anchors_xywh = np.zeros((anchors_per_scale, 4))
                anchors_xywh[:, 0:2] = (
                    np.floor(bbox_xywh_scaled[i, 0:2]).astype(np.int32) + 0.5
                )  # 0.5 for compensation

                # assign all anchors
                anchors_xywh[:, 2:4] = anchors[i]

                iou_scale = iou_xywh_numpy(
                    bbox_xywh_scaled[i][np.newaxis, :], anchors_xywh
                )
                iou.append(iou_scale)
                iou_mask = iou_scale > 0.3

                if np.any(iou_mask):
                    xind, yind = np.floor(bbox_xywh_scaled[i, 0:2]).astype(
                        np.int32
                    )

                    label[i][yind, xind, iou_mask, 0:4] = bbox_xywh * strides[i]
                    label[i][yind, xind, iou_mask, 4:5] = 1.0
                    label[i][yind, xind, iou_mask, 5:] = one_hot

                    bbox_ind = int(bbox_count[i] % 150)  # BUG : 150    ,
                    bboxes_xywh[i][bbox_ind, :4] = bbox_xywh * strides[i]
                    bbox_count[i] += 1

                    exist_positive = True

            if not exist_positive:
                # check if a ground truth bb have the best anchor with any scale
                best_anchor_ind = np.argmax(np.array(iou).reshape(-1), axis=-1)
                best_detect = int(best_anchor_ind / anchors_per_scale)
                best_anchor = int(best_anchor_ind % anchors_per_scale)

                xind, yind = np.floor(
                    bbox_xywh_scaled[best_detect, 0:2]
                ).astype(np.int32)

                label[best_detect][yind, xind, best_anchor, 0:4] = bbox_xywh *␣
↪strides[best_detect]
                label[best_detect][yind, xind, best_anchor, 4:5] = 1.0
```

```
                # label[best_detect][yind, xind, best_anchor, 5:6] = bbox_mix
                label[best_detect][yind, xind, best_anchor, 5:] = one_hot

                bbox_ind = int(bbox_count[best_detect] % 150)
                bboxes_xywh[best_detect][bbox_ind, :4] = bbox_xywh *␣
↪strides[best_detect]
                bbox_count[best_detect] += 1

        flatten_size_s = int(train_output_size[2]) * int(train_output_size[2])␣
↪* anchors_per_scale
        flatten_size_m = int(train_output_size[1]) * int(train_output_size[1])␣
↪* anchors_per_scale
        flatten_size_l = int(train_output_size[0]) * int(train_output_size[0])␣
↪* anchors_per_scale

        label_s = torch.Tensor(label[2]).view(1, flatten_size_s, 5 +␣
↪NUM_CLASSES).squeeze(0)
        label_m = torch.Tensor(label[1]).view(1, flatten_size_m, 5 +␣
↪NUM_CLASSES).squeeze(0)
        label_l = torch.Tensor(label[0]).view(1, flatten_size_l, 5 +␣
↪NUM_CLASSES).squeeze(0)

        bboxes_s = torch.Tensor(bboxes_xywh[2])
        bboxes_m = torch.Tensor(bboxes_xywh[1])
        bboxes_l = torch.Tensor(bboxes_xywh[0])

        # label_sbbox, label_mbbox, label_lbbox = label
        sbboxes, mbboxes, lbboxes = bboxes_xywh
        # print("label")
        labels = torch.cat([label_l, label_m, label_s], 0)
        bboxes = torch.cat([bboxes_l, bboxes_m, bboxes_s], 0)
        return labels, bboxes
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-19-d6c8814dbc83> in <cell line: 24>()
     22 STRIDES = [8, 16, 32]
     23
---> 24 IP_SIZE = img_size
     25 NUM_ANCHORS = 3
     26 NUM_CLASSES = 80

NameError: name 'img_size' is not defined
```

1.3.25 There still are two important functions for training:

- iou_xywh_numpy: calculate the iou (intersect over union) boxes between boxes1 and boxes2. The function has been used in CustomCoco dataset
- CIOU_xywh_torch: calculate IOU from prediction.

[ ]:
```
def iou_xywh_numpy(boxes1, boxes2):
    boxes1 = np.array(boxes1)
    boxes2 = np.array(boxes2)
    # print(boxes1, boxes2)

    boxes1_area = boxes1[..., 2] * boxes1[..., 3]
    boxes2_area = boxes2[..., 2] * boxes2[..., 3]

    boxes1 = np.concatenate([boxes1[..., :2] - boxes1[..., 2:] * 0.5,
                             boxes1[..., :2] + boxes1[..., 2:] * 0.5],␣
 ↪axis=-1)
    boxes2 = np.concatenate([boxes2[..., :2] - boxes2[..., 2:] * 0.5,
                             boxes2[..., :2] + boxes2[..., 2:] * 0.5],␣
 ↪axis=-1)

    left_up = np.maximum(boxes1[..., :2], boxes2[..., :2])
    right_down = np.minimum(boxes1[..., 2:], boxes2[..., 2:])

    inter_section = np.maximum(right_down - left_up, 0.0)
    inter_area = inter_section[..., 0] * inter_section[..., 1]
    union_area = boxes1_area + boxes2_area - inter_area
    IOU = 1.0 * inter_area / union_area
    return IOU


def CIOU_xywh_torch(boxes1,boxes2):
    '''
    cal CIOU of two boxes or batch boxes
    :param boxes1:[xmin,ymin,xmax,ymax] or
                [[xmin,ymin,xmax,ymax],[xmin,ymin,xmax,ymax],...]
    :param boxes2:[xmin,ymin,xmax,ymax]
    :return:
    '''
    # cx cy w h->xyxy
    boxes1 = torch.cat([boxes1[..., :2] - boxes1[..., 2:] * 0.5,
                        boxes1[..., :2] + boxes1[..., 2:] * 0.5], dim=-1)
    boxes2 = torch.cat([boxes2[..., :2] - boxes2[..., 2:] * 0.5,
                        boxes2[..., :2] + boxes2[..., 2:] * 0.5], dim=-1)

    boxes1 = torch.cat([torch.min(boxes1[..., :2], boxes1[..., 2:]),
                        torch.max(boxes1[..., :2], boxes1[..., 2:])], dim=-1)
    boxes2 = torch.cat([torch.min(boxes2[..., :2], boxes2[..., 2:]),
```

```
                    torch.max(boxes2[..., :2], boxes2[..., 2:])], dim=-1)

    # (x2 minus x1 = width)  * (y2 - y1 = height)
    boxes1_area = (boxes1[..., 2] - boxes1[..., 0]) * (boxes1[..., 3] - boxes1[.
↪.., 1])
    boxes2_area = (boxes2[..., 2] - boxes2[..., 0]) * (boxes2[..., 3] - boxes2[.
↪.., 1])

    # upper left of the intersection region (x,y)
    inter_left_up = torch.max(boxes1[..., :2], boxes2[..., :2])

    # bottom right of the intersection region (x,y)
    inter_right_down = torch.min(boxes1[..., 2:], boxes2[..., 2:])

    # if there is overlapping we will get (w,h) else set to (0,0) because it␣
↪could be negative if no overlapping
    inter_section = torch.max(inter_right_down - inter_left_up, torch.
↪zeros_like(inter_right_down))
    inter_area = inter_section[..., 0] * inter_section[..., 1]
    union_area = boxes1_area + boxes2_area - inter_area
    ious = 1.0 * inter_area / union_area

    # cal outer boxes
    outer_left_up = torch.min(boxes1[..., :2], boxes2[..., :2])
    outer_right_down = torch.max(boxes1[..., 2:], boxes2[..., 2:])
    outer = torch.max(outer_right_down - outer_left_up, torch.
↪zeros_like(inter_right_down))
    outer_diagonal_line = torch.pow(outer[..., 0], 2) + torch.pow(outer[...,␣
↪1], 2)

    # cal center distance
    # center x center y
    boxes1_center = (boxes1[..., :2] +  boxes1[...,2:]) * 0.5
    boxes2_center = (boxes2[..., :2] +  boxes2[...,2:]) * 0.5

    # euclidean distance
    # x1-x2 square
    center_dis = torch.pow(boxes1_center[...,0]-boxes2_center[...,0], 2) +\
                 torch.pow(boxes1_center[...,1]-boxes2_center[...,1], 2)

    # cal penalty term
    # cal width,height
    boxes1_size = torch.max(boxes1[..., 2:] - boxes1[..., :2], torch.
↪zeros_like(inter_right_down))
    boxes2_size = torch.max(boxes2[..., 2:] - boxes2[..., :2], torch.
↪zeros_like(inter_right_down))
```

```
    v = (4 / (math.pi ** 2)) * torch.pow(
            torch.atan((boxes1_size[...,0]/torch.clamp(boxes1_size[...,1],min =␣
↪1e-6))) -
            torch.atan((boxes2_size[..., 0] / torch.clamp(boxes2_size[...,␣
↪1],min = 1e-6))), 2)

    alpha = v / (1-ious+v)

    #cal ciou
    cious = ious - (center_dis / outer_diagonal_line + alpha*v)

    return cious
```

Training After finished your custom dataset code, let's train your data.

[ ]:
```
from __future__ import division
import time
import os
import os.path as osp
import numpy as np
import cv2
import pickle as pkl
import pandas as pd
import random
from copy import copy, deepcopy
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
from torch.autograd import Variable
from torch.utils.data import Subset
import torch.optim as optim
import torch.nn.functional as F

import torchvision
from torchvision import datasets, models, transforms

from util import *

import albumentations as A
```

```
/usr/local/lib/python3.10/dist-packages/albumentations/__init__.py:13:
UserWarning: A new version of Albumentations is available: 1.4.18 (you have
1.4.15). Upgrade using: pip install -U albumentations. To disable automatic
update checks, set the environment variable NO_ALBUMENTATIONS_UPDATE to 1.
  check_for_updates()
```

Select the device

[ ]:
```
# Set device to GPU or CPU
gpu = "0"
device = torch.device("cuda:{}".format(gpu) if torch.cuda.is_available() else␣
 ↪"cpu")
```

Load dataset Make image transform and load train dataset. We have used the albumentations class which can transform box parameters automatically, but because we want to test the training coco in yolov3, the transform of training is only resize to 416×416.

[ ]:
```
train_transform = A.Compose([
    #A.SmallestMaxSize(256),
    A.Resize(img_size, img_size),
    # A.RandomCrop(width=224, height=224),
    # A.HorizontalFlip(p=0.5),
    # A.RandomBrightnessContrast(p=0.2),
], bbox_params=A.BboxParams(format='coco', label_fields=['category_ids']),
)

eval_transform = A.Compose([
    A.Resize(img_size, img_size),
    #A.SmallestMaxSize(256),
    #A.CenterCrop(width=224, height=224),
], bbox_params=A.BboxParams(format='coco', label_fields=['category_ids']),
)
```

[ ]:
```
BATCH_SIZE = 10
def collate_fn(batch):
    return tuple(zip(*batch))

train_dataset = Subset(CustomCoco(root = path2data_train,
                                  annFile = path2json_train,␣
 ↪transform=train_transform), list(range(0,20)))
train_dataloader = torch.utils.data.DataLoader(train_dataset,␣
 ↪batch_size=BATCH_SIZE,
                                               shuffle=True, num_workers=0,␣
 ↪collate_fn=collate_fn)
```

Load Yolov3 model If you want to train the model with pretrained network, load weights from the example above.

[ ]:
```
print("Loading network.....")
model = MyDarknet("cfg/yolov3.cfg")
# load pretrained
# model.load_weights("yolov3.weights")
print("Network successfully loaded")
```

Training code Create the training code. The important part of yolo training is you must convert and calculate the iou correctly.

```
[ ]: def run_training(model, optimizer, dataloader, device, img_size, n_epoch,␣
     ↪every_n_batch, every_n_epoch, ckpt_dir):
        for epoch_i in range(n_epoch):
            running_loss = 0.0
            for inputs, labels, bboxes in dataloader:
                inputs = torch.from_numpy(np.array(inputs)).squeeze(1).float()
                inputs = inputs.to(device)

                # Initialize lists to hold the tensor labels and bounding boxes for␣
     ↪the current batch
                label_tensors = []
                box_tensors = []

                for label in labels:
                    # Convert each label into a tensor and move to device
                    label_tensor = torch.tensor(label).to(device)
                    label_tensors.append(label_tensor)

                for bbox in bboxes:
                    # Convert each bounding box into a tensor and move to device
                    bbox_tensor = torch.tensor(bbox).to(device)
                    box_tensors.append(bbox_tensor)

                # At this point, label_tensors is a list of tensors
                # You can process these lists as needed

                # Perform the model forward pass and loss calculation
                optimizer.zero_grad()
                with torch.set_grad_enabled(True):
                    outputs = model(inputs, True)

                    pred_xywh = outputs[..., 0:4] / img_size
                    pred_conf = outputs[..., 4:5]
                    pred_cls = outputs[..., 5:]

                    # Prepare the target values
                    label_xywh = [label[:, :4] / img_size for label in␣
     ↪label_tensors]  # Normalize the xywh
                    label_obj_mask = [label[:, 4:5] for label in label_tensors]
                    label_cls = [label[:, 5:] for label in label_tensors]

                    # Compute the losses for each image in the batch
                    loss = 0
                    for i in range(len(label_tensors)):
```

```
                    lambda_coord = 0.001
                    lambda_noobj = 0.05
                    # Calculate losses for the i-th image
                    loss_coord = lambda_coord * label_obj_mask[i] * nn.
↪MSELoss()(pred_xywh[i], label_xywh[i])
                    loss_conf = (label_obj_mask[i] * nn.BCELoss()(pred_conf[i],␣
↪label_obj_mask[i])) + \
                                (lambda_noobj * (1.0 - label_obj_mask[i]) * nn.
↪BCELoss()(pred_conf[i], label_obj_mask[i]))
                    loss_cls = label_obj_mask[i] * nn.BCELoss()(pred_cls[i],␣
↪label_cls[i])

                    loss += loss_coord + loss_conf + loss_cls

                loss.backward()
                optimizer.step()

                # Statistics
                running_loss += loss.item() * inputs.size(0)

        epoch_loss = running_loss / len(dataloader.dataset)
        print(epoch_loss)
        print('End Epoch')
```

Let's train it!

[ ]:
```
model.to(device)

optimizer = optim.Adam(model.parameters(), lr=0.001)

n_epoch = 5
img_size = 416
save_every_batch = False
save_every_epoch = True
ckpt_dir = "../checkpoints"


run_training(model, optimizer, train_dataloader, device,
                 img_size,
                 n_epoch,
                 save_every_batch,
                 save_every_epoch,
                 ckpt_dir)
```

1.4 Exercises

1. Train on Pascal VOC dataset (2007 and above) with YOLOv3 model. Show your results into your own test images.

2. Please see YOLOv8. This is the latest version of YOLO. Follow the instruction on the github and perform the training on any dataset.
3. Please write a summary report on your training process and the result. Submit the .pdf file of your work.

[ ]:
```
!mkdir -p /content/VOC
!wget http://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar -P /content/
 ↪VOC/
!wget http://pjreddie.com/media/files/VOCtest_06-Nov-2007.tar -P /content/VOC/
```

```
--2024-10-19 09:32:33--
http://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar
Resolving pjreddie.com (pjreddie.com)… 162.0.215.52
Connecting to pjreddie.com (pjreddie.com)|162.0.215.52|:80… connected.
HTTP request sent, awaiting response… 301 Moved Permanently
Location: https://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar
[following]
--2024-10-19 09:32:34--
https://pjreddie.com/media/files/VOCtrainval_06-Nov-2007.tar
Connecting to pjreddie.com (pjreddie.com)|162.0.215.52|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 460032000 (439M) [application/x-tar]
Saving to: '/content/VOC/VOCtrainval_06-Nov-2007.tar'

VOCtrainval_06-Nov- 100%[===================>] 438.72M  13.2MB/s    in 34s

2024-10-19 09:33:08 (13.0 MB/s) - '/content/VOC/VOCtrainval_06-Nov-2007.tar'
saved [460032000/460032000]

--2024-10-19 09:33:08--  http://pjreddie.com/media/files/VOCtest_06-Nov-2007.tar
Resolving pjreddie.com (pjreddie.com)… 162.0.215.52
Connecting to pjreddie.com (pjreddie.com)|162.0.215.52|:80… connected.
HTTP request sent, awaiting response… 301 Moved Permanently
Location: https://pjreddie.com/media/files/VOCtest_06-Nov-2007.tar [following]
--2024-10-19 09:33:09--
https://pjreddie.com/media/files/VOCtest_06-Nov-2007.tar
Connecting to pjreddie.com (pjreddie.com)|162.0.215.52|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 451020800 (430M) [application/x-tar]
Saving to: '/content/VOC/VOCtest_06-Nov-2007.tar'

VOCtest_06-Nov-2007 100%[===================>] 430.13M  10.7MB/s    in 42s

2024-10-19 09:33:51 (10.3 MB/s) - '/content/VOC/VOCtest_06-Nov-2007.tar' saved
[451020800/451020800]
```

[ ]:
```
!tar -xf /content/VOC/VOCtrainval_06-Nov-2007.tar -C /content/VOC/
!tar -xf /content/VOC/VOCtest_06-Nov-2007.tar -C /content/VOC/
```

[ ]:
```
voc_dataset_path = '/content/VOC/VOCdevkit/VOC2007'
voc_labels_path = '/content/VOC/VOCdevkit/VOC2007/labels/'
```

[ ]:
```
import xml.etree.ElementTree as ET
import os

def convert_to_yolo_format(xml_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    size = root.find('size')
    width = int(size.find('width').text)
    height = int(size.find('height').text)

    yolo_annotation = []

    for obj in root.iter('object'):
        difficult = obj.find('difficult').text
        if int(difficult) == 1:
            continue
        class_name = obj.find('name').text
        if class_name not in class_names:
            continue
        class_id = class_names.index(class_name)

        xmlbox = obj.find('bndbox')
        b = (float(xmlbox.find('xmin').text), float(xmlbox.find('ymin').text),
             float(xmlbox.find('xmax').text), float(xmlbox.find('ymax').text))
        bb = convert_bbox((width, height), b)
        yolo_annotation.append(f"{class_id} " + " ".join(map(str, bb)) + "\n")

    return yolo_annotation

def convert_bbox(size, box):
    dw = 1. / size[0]
    dh = 1. / size[1]
    x = (box[0] + box[2]) / 2.0 - 1
    y = (box[1] + box[3]) / 2.0 - 1
    w = box[2] - box[0]
    h = box[3] - box[1]
    return x * dw, y * dh, w * dw, h * dh

# Replace with your Pascal VOC classes (e.g., 'person', 'car', 'dog', etc.)
```

```
class_names = ["aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car",␣
 ↪"cat",
               "chair", "cow", "diningtable", "dog", "horse", "motorbike",␣
 ↪"person",
               "pottedplant", "sheep", "sofa", "train", "tvmonitor"]

# Parse and convert the annotations
for xml_file in os.listdir(voc_dataset_path + '/Annotations'):
    if not xml_file.endswith(".xml"):
        continue
    annotations = convert_to_yolo_format(os.path.join(voc_dataset_path,␣
 ↪'Annotations', xml_file))

    # Write converted annotations to txt file in YOLO format
    with open(os.path.join(voc_labels_path, xml_file.replace('.xml', '.txt')),␣
 ↪'w') as f:
        f.writelines(annotations)
```

[ ]:
```
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import os


class VOCDataset(Dataset):
    def __init__(self, voc_dataset_path, transform=None):
        self.voc_dataset_path = voc_dataset_path
        self.transform = transform
        self.image_folder = os.path.join(voc_dataset_path, 'JPEGImages')
        self.label_folder = os.path.join(voc_dataset_path, 'labels')

        self.image_files = [f for f in os.listdir(self.image_folder) if f.
 ↪endswith('.jpg')]

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        img_name = os.path.join(self.image_folder, self.image_files[idx])
        image = Image.open(img_name).convert("RGB")

        # Load YOLO annotations
        label_name = self.image_files[idx].replace('.jpg', '.txt')
        labels_path = os.path.join(self.label_folder, label_name)

        boxes = []
```

```
        class_ids = []
        with open(labels_path, 'r') as f:
            for line in f.readlines():
                values = line.strip().split()
                class_id = int(values[0])
                x_center, y_center, width, height = map(float, values[1:5])

                # Convert YOLO format back to bounding box coordinates
                x1 = (x_center - width / 2) * image.size[0]
                y1 = (y_center - height / 2) * image.size[1]
                x2 = (x_center + width / 2) * image.size[0]
                y2 = (y_center + height / 2) * image.size[1]

                boxes.append([x1, y1, x2, y2])
                class_ids.append(class_id)

        if self.transform:
            image = self.transform(image)

        return image, torch.tensor(class_ids), np.array(boxes)

        if self.transform is not None:
            bboxes = list(obj['bbox'] for obj in target)
            category_ids = list(obj['category_id'] for obj in target)
            transformed = self.transform(image=img, bboxes=bboxes,␣
↪category_ids=category_ids)
            img = transformed['image'],
            bboxes = torch.Tensor(transformed['bboxes'])
            cat_ids = torch.Tensor(transformed['category_ids'])
            labels, bboxes = self.__create_label(bboxes, cat_ids.type(torch.
↪IntTensor))

        return img, labels, bboxes

    def __len__(self) -> int:
        return len(self.ids)

    def __create_label(self, bboxes, class_inds):
        """
        Label assignment. For a single picture all GT box bboxes are assigned␣
↪anchor.
        1 Select a bbox in order, convert its coordinates("xyxy") to "xywh";␣
↪and scale bbox'
            xywh by the strides.
        2 Calculate the iou between the each detection layer'anchors and the␣
↪bbox in turn, and select the largest
```

```
            anchor to predict the bbox.If the ious of all detection layers are␣
↪smaller than 0.3, select the largest
            of all detection layers' anchors to predict the bbox.
        Note :
        1 The same GT may be assigned to multiple anchors. And the anchors may␣
↪be on the same or different layer.
        2 The total number of bboxes may be more than it is, because the same␣
↪GT may be assigned to multiple layers
        of detection.
        """
        # print("Class indices: ", class_inds)
        bboxes = np.array(bboxes)
        class_inds = np.array(class_inds)
        anchors = ANCHORS # all the anchors
        strides = np.array(STRIDES) # list of strides
        train_output_size = IP_SIZE / strides # image with different scales
        anchors_per_scale = NUM_ANCHORS # anchor per scale

        label = [
            np.zeros(
                (
                    int(train_output_size[i]),
                    int(train_output_size[i]),
                    anchors_per_scale,
                    5 + NUM_CLASSES,
                )
            )
            for i in range(3)
        ]
        # 150 bounding box ground truths per scale
        bboxes_xywh = [
            np.zeros((150, 4)) for _ in range(3)
        ]  # Darknet the max_num is 30
        bbox_count = np.zeros((3,))

        for i in range(len(bboxes)):
            bbox_coor = bboxes[i][:4]
            bbox_class_ind = cats_dict[str(class_inds[i])]

            # onehot
            one_hot = np.zeros(NUM_CLASSES, dtype=np.float32)
            one_hot[bbox_class_ind] = 1.0
            # one_hot_smooth = dataAug.LabelSmooth()(one_hot, self.num_classes)

            # convert "xyxy" to "xywh"
            bbox_xywh = np.concatenate(
                [
```

```
                (0.5 * bbox_coor[2:] + bbox_coor[:2]) ,
                bbox_coor[2:],
            ],
            axis=-1,
        )

        bbox_xywh_scaled = (
            1.0 * bbox_xywh[np.newaxis, :] / strides[:, np.newaxis]
        )

        iou = []
        exist_positive = False
        for i in range(3):
            anchors_xywh = np.zeros((anchors_per_scale, 4))
            anchors_xywh[:, 0:2] = (
                np.floor(bbox_xywh_scaled[i, 0:2]).astype(np.int32) + 0.5
            )  # 0.5 for compensation

            # assign all anchors
            anchors_xywh[:, 2:4] = anchors[i]

            iou_scale = iou_xywh_numpy(
                bbox_xywh_scaled[i][np.newaxis, :], anchors_xywh
            )
            iou.append(iou_scale)
            iou_mask = iou_scale > 0.3

            if np.any(iou_mask):
                xind, yind = np.floor(bbox_xywh_scaled[i, 0:2]).astype(
                    np.int32
                )

                label[i][yind, xind, iou_mask, 0:4] = bbox_xywh * strides[i]
                label[i][yind, xind, iou_mask, 4:5] = 1.0
                label[i][yind, xind, iou_mask, 5:] = one_hot

                bbox_ind = int(bbox_count[i] % 150)  # BUG : 150    ,
                bboxes_xywh[i][bbox_ind, :4] = bbox_xywh * strides[i]
                bbox_count[i] += 1

                exist_positive = True

        if not exist_positive:
            # check if a ground truth bb have the best anchor with any scale
            best_anchor_ind = np.argmax(np.array(iou).reshape(-1), axis=-1)
            best_detect = int(best_anchor_ind / anchors_per_scale)
            best_anchor = int(best_anchor_ind % anchors_per_scale)
```

```
                xind, yind = np.floor(
                    bbox_xywh_scaled[best_detect, 0:2]
                ).astype(np.int32)

                label[best_detect][yind, xind, best_anchor, 0:4] = bbox_xywh *␣
↪strides[best_detect]
                label[best_detect][yind, xind, best_anchor, 4:5] = 1.0
                # label[best_detect][yind, xind, best_anchor, 5:6] = bbox_mix
                label[best_detect][yind, xind, best_anchor, 5:] = one_hot

                bbox_ind = int(bbox_count[best_detect] % 150)
                bboxes_xywh[best_detect][bbox_ind, :4] = bbox_xywh *␣
↪strides[best_detect]
                bbox_count[best_detect] += 1

        flatten_size_s = int(train_output_size[2]) * int(train_output_size[2])␣
↪* anchors_per_scale
        flatten_size_m = int(train_output_size[1]) * int(train_output_size[1])␣
↪* anchors_per_scale
        flatten_size_l = int(train_output_size[0]) * int(train_output_size[0])␣
↪* anchors_per_scale

        label_s = torch.Tensor(label[2]).view(1, flatten_size_s, 5 +␣
↪NUM_CLASSES).squeeze(0)
        label_m = torch.Tensor(label[1]).view(1, flatten_size_m, 5 +␣
↪NUM_CLASSES).squeeze(0)
        label_l = torch.Tensor(label[0]).view(1, flatten_size_l, 5 +␣
↪NUM_CLASSES).squeeze(0)

        bboxes_s = torch.Tensor(bboxes_xywh[2])
        bboxes_m = torch.Tensor(bboxes_xywh[1])
        bboxes_l = torch.Tensor(bboxes_xywh[0])

        # label_sbbox, label_mbbox, label_lbbox = label
        sbboxes, mbboxes, lbboxes = bboxes_xywh
        # print("label")
        labels = torch.cat([label_l, label_m, label_s], 0)
        bboxes = torch.cat([bboxes_l, bboxes_m, bboxes_s], 0)
        return labels, bboxes
```

[ ]:
```
from __future__ import division
import time
import os
import os.path as osp
import numpy as np
```

```
import cv2
import pickle as pkl
import pandas as pd
import random
from copy import copy, deepcopy
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
from torch.autograd import Variable
from torch.utils.data import Subset
import torch.optim as optim
import torch.nn.functional as F

import torchvision
from torchvision import datasets, models, transforms

from util import *

import albumentations as A
```

```
/usr/local/lib/python3.10/dist-packages/albumentations/__init__.py:13:
UserWarning: A new version of Albumentations is available: 1.4.18 (you have
1.4.15). Upgrade using: pip install -U albumentations. To disable automatic
update checks, set the environment variable NO_ALBUMENTATIONS_UPDATE to 1.
  check_for_updates()
```

[ ]:
```
from util import *
import darknet

blocks = darknet.parse_cfg("cfg/yolov3.cfg")
print(darknet.create_modules(blocks))

class MyDarknet(nn.Module):
    def __init__(self, cfgfile):
        super(MyDarknet, self).__init__()
        # load the config file and create our model
        self.blocks = darknet.parse_cfg(cfgfile)
        self.net_info, self.module_list = darknet.create_modules(self.blocks)

    def forward(self, x, CUDA:bool):
        modules = self.blocks[1:]
        outputs = {}   #We cache the outputs for the route layer

        write = 0
        # run forward propagation. Follow the instruction from dictionary␣
 ↪modules
```

```
        for i, module in enumerate(modules):
            module_type = (module["type"])

            if module_type == "convolutional" or module_type == "upsample":
                # do convolutional network
                x = self.module_list[i](x)

            elif module_type == "route":
                # concat layers
                layers = module["layers"]
                layers = [int(a) for a in layers]

                if (layers[0]) > 0:
                    layers[0] = layers[0] - i

                if len(layers) == 1:
                    x = outputs[i + (layers[0])]

                else:
                    if (layers[1]) > 0:
                        layers[1] = layers[1] - i

                    map1 = outputs[i + layers[0]]
                    map2 = outputs[i + layers[1]]
                    x = torch.cat((map1, map2), 1)


            elif  module_type == "shortcut":
                from_ = int(module["from"])
                # residual network
                x = outputs[i-1] + outputs[i+from_]

            elif module_type == 'yolo':
                anchors = self.module_list[i][0].anchors
                #Get the input dimensions
                inp_dim = int (self.net_info["height"])

                #Get the number of classes
                num_classes = int (module["classes"])

                #Transform
                # predict_transform is in util.py
                batch_size = x.size(0)
                stride =  inp_dim // x.size(2)
                grid_size = inp_dim // stride
                bbox_attrs = 5 + num_classes
                num_anchors = len(anchors)
```

```
            x = predict_transform(x, inp_dim, anchors, num_classes, CUDA)
            if not write:              #if no collector has been intialised.
                detections = x
                write = 1

            else:
                detections = torch.cat((detections, x), 1)

        outputs[i] = x

    return detections


def load_weights(self, weightfile):
    '''
    Load pretrained weight
    '''
    #Open the weights file
    fp = open(weightfile, "rb")

    #The first 5 values are header information
    # 1. Major version number
    # 2. Minor Version Number
    # 3. Subversion number
    # 4,5. Images seen by the network (during training)
    header = np.fromfile(fp, dtype = np.int32, count = 5)
    self.header = torch.from_numpy(header)
    self.seen = self.header[3]

    weights = np.fromfile(fp, dtype = np.float32)

    ptr = 0
    for i in range(len(self.module_list)):
        module_type = self.blocks[i + 1]["type"]

        #If module_type is convolutional load weights
        #Otherwise ignore.

        if module_type == "convolutional":
            model = self.module_list[i]
            try:
                batch_normalize = int(self.blocks[i+1]["batch_normalize"])
            except:
                batch_normalize = 0

            conv = model[0]
```

```
            if (batch_normalize):
                bn = model[1]

                #Get the number of weights of Batch Norm Layer
                num_bn_biases = bn.bias.numel()

                #Load the weights
                bn_biases = torch.from_numpy(weights[ptr:ptr +␣
↪num_bn_biases])
                ptr += num_bn_biases

                bn_weights = torch.from_numpy(weights[ptr: ptr +␣
↪num_bn_biases])
                ptr  += num_bn_biases

                bn_running_mean = torch.from_numpy(weights[ptr: ptr +␣
↪num_bn_biases])
                ptr  += num_bn_biases

                bn_running_var = torch.from_numpy(weights[ptr: ptr +␣
↪num_bn_biases])
                ptr  += num_bn_biases

                #Cast the loaded weights into dims of model weights.
                bn_biases = bn_biases.view_as(bn.bias.data)
                bn_weights = bn_weights.view_as(bn.weight.data)
                bn_running_mean = bn_running_mean.view_as(bn.running_mean)
                bn_running_var = bn_running_var.view_as(bn.running_var)

                #Copy the data to model
                bn.bias.data.copy_(bn_biases)
                bn.weight.data.copy_(bn_weights)
                bn.running_mean.copy_(bn_running_mean)
                bn.running_var.copy_(bn_running_var)

            else:
                #Number of biases
                num_biases = conv.bias.numel()

                #Load the weights
                conv_biases = torch.from_numpy(weights[ptr: ptr +␣
↪num_biases])
                ptr = ptr + num_biases
```

```
                    #reshape the loaded weights according to the dims of the␣
↪model weights
                    conv_biases = conv_biases.view_as(conv.bias.data)

                    #Finally copy the data
                    conv.bias.data.copy_(conv_biases)

                #Let us load the weights for the Convolutional layers
                num_weights = conv.weight.numel()

                #Do the same as above for weights
                conv_weights = torch.from_numpy(weights[ptr:ptr+num_weights])
                ptr = ptr + num_weights

                conv_weights = conv_weights.view_as(conv.weight.data)
                conv.weight.data.copy_(conv_weights)
```

```
({'type': 'net', 'batch': '1', 'subdivisions': '1', 'width': '416', 'height':
'416', 'channels': '3', 'momentum': '0.9', 'decay': '0.0005', 'angle': '0',
'saturation': '1.5', 'exposure': '1.5', 'hue': '.1', 'learning_rate': '0.001',
'burn_in': '1000', 'max_batches': '500200', 'policy': 'steps', 'steps':
'400000,450000', 'scales': '.1,.1'}, ModuleList(
  (0): Sequential(
    (conv_0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (batch_norm_0): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_0): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (1): Sequential(
    (conv_1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
    (batch_norm_1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_1): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (2): Sequential(
    (conv_2): Conv2d(64, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_2): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (3): Sequential(
    (conv_3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (batch_norm_3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_3): LeakyReLU(negative_slope=0.1, inplace=True)
```

```
  )
  (4): Sequential(
    (shortcut_4): EmptyLayer()
  )
  (5): Sequential(
    (conv_5): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
    (batch_norm_5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_5): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (6): Sequential(
    (conv_6): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_6): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_6): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (7): Sequential(
    (conv_7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
    (batch_norm_7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_7): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (8): Sequential(
    (shortcut_8): EmptyLayer()
  )
  (9): Sequential(
    (conv_9): Conv2d(128, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_9): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (10): Sequential(
    (conv_10): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_10): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (11): Sequential(
    (shortcut_11): EmptyLayer()
  )
  (12): Sequential(
    (conv_12): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (batch_norm_12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (leaky_12): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (13): Sequential(
    (conv_13): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_13): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_13): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (14): Sequential(
    (conv_14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_14): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (15): Sequential(
    (shortcut_15): EmptyLayer()
  )
  (16): Sequential(
    (conv_16): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_16): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_16): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (17): Sequential(
    (conv_17): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_17): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_17): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (18): Sequential(
    (shortcut_18): EmptyLayer()
  )
  (19): Sequential(
    (conv_19): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_19): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_19): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (20): Sequential(
    (conv_20): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_20): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_20): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (21): Sequential(
```

```
    (shortcut_21): EmptyLayer()
  )
  (22): Sequential(
    (conv_22): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_22): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_22): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (23): Sequential(
    (conv_23): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_23): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_23): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (24): Sequential(
    (shortcut_24): EmptyLayer()
  )
  (25): Sequential(
    (conv_25): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_25): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_25): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (26): Sequential(
    (conv_26): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_26): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_26): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (27): Sequential(
    (shortcut_27): EmptyLayer()
  )
  (28): Sequential(
    (conv_28): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_28): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_28): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (29): Sequential(
    (conv_29): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_29): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_29): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (30): Sequential(
```

```
    (shortcut_30): EmptyLayer()
  )
  (31): Sequential(
    (conv_31): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_31): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_31): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (32): Sequential(
    (conv_32): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_32): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_32): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (33): Sequential(
    (shortcut_33): EmptyLayer()
  )
  (34): Sequential(
    (conv_34): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_34): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_34): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (35): Sequential(
    (conv_35): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_35): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_35): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (36): Sequential(
    (shortcut_36): EmptyLayer()
  )
  (37): Sequential(
    (conv_37): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (batch_norm_37): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_37): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (38): Sequential(
    (conv_38): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_38): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_38): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (39): Sequential(
```

```
    (conv_39): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_39): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_39): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (40): Sequential(
    (shortcut_40): EmptyLayer()
  )
  (41): Sequential(
    (conv_41): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_41): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_41): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (42): Sequential(
    (conv_42): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_42): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_42): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (43): Sequential(
    (shortcut_43): EmptyLayer()
  )
  (44): Sequential(
    (conv_44): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_44): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_44): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (45): Sequential(
    (conv_45): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_45): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_45): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (46): Sequential(
    (shortcut_46): EmptyLayer()
  )
  (47): Sequential(
    (conv_47): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_47): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_47): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (48): Sequential(
```

```
    (conv_48): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_48): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_48): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (49): Sequential(
    (shortcut_49): EmptyLayer()
  )
  (50): Sequential(
    (conv_50): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_50): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_50): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (51): Sequential(
    (conv_51): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_51): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_51): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (52): Sequential(
    (shortcut_52): EmptyLayer()
  )
  (53): Sequential(
    (conv_53): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_53): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_53): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (54): Sequential(
    (conv_54): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_54): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_54): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (55): Sequential(
    (shortcut_55): EmptyLayer()
  )
  (56): Sequential(
    (conv_56): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_56): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_56): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (57): Sequential(
```

```
    (conv_57): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_57): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_57): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (58): Sequential(
    (shortcut_58): EmptyLayer()
  )
  (59): Sequential(
    (conv_59): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_59): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_59): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (60): Sequential(
    (conv_60): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_60): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_60): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (61): Sequential(
    (shortcut_61): EmptyLayer()
  )
  (62): Sequential(
    (conv_62): Conv2d(512, 1024, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
    (batch_norm_62): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_62): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (63): Sequential(
    (conv_63): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_63): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_63): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (64): Sequential(
    (conv_64): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_64): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_64): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (65): Sequential(
    (shortcut_65): EmptyLayer()
  )
```

```
  (66): Sequential(
    (conv_66): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_66): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_66): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (67): Sequential(
    (conv_67): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_67): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_67): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (68): Sequential(
    (shortcut_68): EmptyLayer()
  )
  (69): Sequential(
    (conv_69): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_69): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_69): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (70): Sequential(
    (conv_70): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_70): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_70): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (71): Sequential(
    (shortcut_71): EmptyLayer()
  )
  (72): Sequential(
    (conv_72): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_72): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_72): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (73): Sequential(
    (conv_73): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_73): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_73): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (74): Sequential(
    (shortcut_74): EmptyLayer()
  )
```

```
  (75): Sequential(
    (conv_75): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_75): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_75): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (76): Sequential(
    (conv_76): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_76): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_76): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (77): Sequential(
    (conv_77): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_77): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_77): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (78): Sequential(
    (conv_78): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_78): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_78): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (79): Sequential(
    (conv_79): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_79): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_79): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (80): Sequential(
    (conv_80): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_80): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_80): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (81): Sequential(
    (conv_81): Conv2d(1024, 255, kernel_size=(1, 1), stride=(1, 1))
  )
  (82): Sequential(
    (Detection_82): DetectionLayer()
  )
  (83): Sequential(
    (route_83): EmptyLayer()
  )
```

```
  (84): Sequential(
    (conv_84): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_84): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_84): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (85): Sequential(
    (upsample_85): Upsample(scale_factor=2.0, mode='nearest')
  )
  (86): Sequential(
    (route_86): EmptyLayer()
  )
  (87): Sequential(
    (conv_87): Conv2d(768, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_87): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_87): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (88): Sequential(
    (conv_88): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_88): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_88): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (89): Sequential(
    (conv_89): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_89): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_89): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (90): Sequential(
    (conv_90): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_90): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_90): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (91): Sequential(
    (conv_91): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_91): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_91): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (92): Sequential(
    (conv_92): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_92): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
    (leaky_92): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (93): Sequential(
    (conv_93): Conv2d(512, 255, kernel_size=(1, 1), stride=(1, 1))
  )
  (94): Sequential(
    (Detection_94): DetectionLayer()
  )
  (95): Sequential(
    (route_95): EmptyLayer()
  )
  (96): Sequential(
    (conv_96): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_96): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_96): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (97): Sequential(
    (upsample_97): Upsample(scale_factor=2.0, mode='nearest')
  )
  (98): Sequential(
    (route_98): EmptyLayer()
  )
  (99): Sequential(
    (conv_99): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_99): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_99): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (100): Sequential(
    (conv_100): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_100): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_100): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (101): Sequential(
    (conv_101): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_101): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_101): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (102): Sequential(
    (conv_102): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_102): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (leaky_102): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (103): Sequential(
    (conv_103): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (batch_norm_103): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_103): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (104): Sequential(
    (conv_104): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (batch_norm_104): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (leaky_104): LeakyReLU(negative_slope=0.1, inplace=True)
  )
  (105): Sequential(
    (conv_105): Conv2d(256, 255, kernel_size=(1, 1), stride=(1, 1))
  )
  (106): Sequential(
    (Detection_106): DetectionLayer()
  )
))
```

[ ]:
```python
transform = transforms.Compose([
    transforms.Resize((416, 416)),  # Resize for YOLO
    transforms.ToTensor()
])

def collate_fn(batch):
    images, labels, boxes = zip(*batch)

    # Stack images
    images = torch.stack(images)

    # Pad labels and boxes
    labels = [torch.tensor(label) for label in labels]
    boxes = [torch.tensor(box) for box in boxes]

    # Pad boxes to have a fixed size
    max_boxes = 50  # Adjust as needed
    padded_boxes = []
    for box in boxes:
        padded_box = torch.zeros((max_boxes, 4))  # 4 coordinates
        padded_box[:box.size(0)] = box
        padded_boxes.append(padded_box)

    padded_boxes = torch.stack(padded_boxes)
```

```
    return images, labels, padded_boxes

BATCH_SIZE = 10
# def collate_fn(batch):
#     return zip(*batch)

voc_dataset_path = '/content/VOC/VOCdevkit/VOC2007'  # Replace with your␣
↪dataset path

train_dataset = Subset(VOCDataset(voc_dataset_path, transform=transform),␣
↪list(range(0,20)))
train_dataloader = torch.utils.data.DataLoader(train_dataset,␣
↪batch_size=BATCH_SIZE,
                                               shuffle=True, num_workers=4,␣
↪collate_fn=collate_fn)

# dataloader = DataLoader(dataset, batch_size=16, shuffle=True, num_workers=4,␣
↪collate_fn=collate_fn)  # Adjust batch_size and num_workers as needed

# Example usage
for images, labels, boxes in train_dataloader:
    print(images.shape)  # Shape of the image batch
    print(labels)       # Bounding boxes for the images
    print(boxes)
    break
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557:
UserWarning: This DataLoader will create 4 worker processes in total. Our
suggested max number of worker in current system is 2, which is smaller than
what this DataLoader is going to create. Please be aware that excessive worker
creation might get DataLoader running slow or even freeze, lower the worker
number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]
<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]
```

```
torch.Size([10, 3, 416, 416])
[tensor([13]), tensor([12, 12, 14, 14, 14]), tensor([12, 12]), tensor([8]),
tensor([ 4, 14, 14, 14, 14]), tensor([19, 14]), tensor([3, 3]), tensor([1]),
```

```
tensor([ 1, 14]), tensor([14, 14])]
tensor([[[ 69.,  90., 457., 265.],
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.],
         …,
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.]],

        [[  0.,  62., 324., 317.],
         [286.,  74., 361., 237.],
         [395.,   0., 468., 315.],
         …,
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.]],

        [[ 71., 131., 256., 308.],
         [256., 132., 448., 267.],
         [  0.,   0.,   0.,   0.],
         …,
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.]],

        …,

        [[ 14.,  50., 453., 317.],
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.],
         …,
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.]],

        [[ 31., 180., 215., 495.],
         [ 24.,  26., 109., 355.],
         [  0.,   0.,   0.,   0.],
         …,
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.],
         [  0.,   0.,   0.,   0.]],

        [[ 64.,   1., 499., 374.],
         [  0.,   0., 118., 246.],
         [  0.,   0.,   0.,   0.],
         …,
         [  0.,   0.,   0.,   0.],
```

```
        [ 0.,  0.,  0.,  0.],
        [ 0.,  0.,  0.,  0.]]])
```

[ ]:
```python
def iou_xywh_numpy(boxes1, boxes2):
    boxes1 = np.array(boxes1)
    boxes2 = np.array(boxes2)
    # print(boxes1, boxes2)

    boxes1_area = boxes1[..., 2] * boxes1[..., 3]
    boxes2_area = boxes2[..., 2] * boxes2[..., 3]

    boxes1 = np.concatenate([boxes1[..., :2] - boxes1[..., 2:] * 0.5,
                             boxes1[..., :2] + boxes1[..., 2:] * 0.5],␣
 ↪axis=-1)
    boxes2 = np.concatenate([boxes2[..., :2] - boxes2[..., 2:] * 0.5,
                             boxes2[..., :2] + boxes2[..., 2:] * 0.5],␣
 ↪axis=-1)

    left_up = np.maximum(boxes1[..., :2], boxes2[..., :2])
    right_down = np.minimum(boxes1[..., 2:], boxes2[..., 2:])

    inter_section = np.maximum(right_down - left_up, 0.0)
    inter_area = inter_section[..., 0] * inter_section[..., 1]
    union_area = boxes1_area + boxes2_area - inter_area
    IOU = 1.0 * inter_area / union_area
    return IOU


def CIOU_xywh_torch(boxes1,boxes2):
    '''
    cal CIOU of two boxes or batch boxes
    :param boxes1:[xmin,ymin,xmax,ymax] or
                [[xmin,ymin,xmax,ymax],[xmin,ymin,xmax,ymax],...]
    :param boxes2:[xmin,ymin,xmax,ymax]
    :return:
    '''
    # cx cy w h->xyxy
    boxes1 = torch.cat([boxes1[..., :2] - boxes1[..., 2:] * 0.5,
                        boxes1[..., :2] + boxes1[..., 2:] * 0.5], dim=-1)
    boxes2 = torch.cat([boxes2[..., :2] - boxes2[..., 2:] * 0.5,
                        boxes2[..., :2] + boxes2[..., 2:] * 0.5], dim=-1)

    boxes1 = torch.cat([torch.min(boxes1[..., :2], boxes1[..., 2:]),
                        torch.max(boxes1[..., :2], boxes1[..., 2:])], dim=-1)
    boxes2 = torch.cat([torch.min(boxes2[..., :2], boxes2[..., 2:]),
                        torch.max(boxes2[..., :2], boxes2[..., 2:])], dim=-1)
```

```
    # (x2 minus x1 = width)  * (y2 - y1 = height)
    boxes1_area = (boxes1[..., 2] - boxes1[..., 0]) * (boxes1[..., 3] - boxes1[.
↪.., 1])
    boxes2_area = (boxes2[..., 2] - boxes2[..., 0]) * (boxes2[..., 3] - boxes2[.
↪.., 1])

    # upper left of the intersection region (x,y)
    inter_left_up = torch.max(boxes1[..., :2], boxes2[..., :2])

    # bottom right of the intersection region (x,y)
    inter_right_down = torch.min(boxes1[..., 2:], boxes2[..., 2:])

    # if there is overlapping we will get (w,h) else set to (0,0) because it␣
↪could be negative if no overlapping
    inter_section = torch.max(inter_right_down - inter_left_up, torch.
↪zeros_like(inter_right_down))
    inter_area = inter_section[..., 0] * inter_section[..., 1]
    union_area = boxes1_area + boxes2_area - inter_area
    ious = 1.0 * inter_area / union_area

    # cal outer boxes
    outer_left_up = torch.min(boxes1[..., :2], boxes2[..., :2])
    outer_right_down = torch.max(boxes1[..., 2:], boxes2[..., 2:])
    outer = torch.max(outer_right_down - outer_left_up, torch.
↪zeros_like(inter_right_down))
    outer_diagonal_line = torch.pow(outer[..., 0], 2) + torch.pow(outer[...,␣
↪1], 2)

    # cal center distance
    # center x center y
    boxes1_center = (boxes1[..., :2] +  boxes1[...,2:]) * 0.5
    boxes2_center = (boxes2[..., :2] +  boxes2[...,2:]) * 0.5

    # euclidean distance
    # x1-x2 square
    center_dis = torch.pow(boxes1_center[...,0]-boxes2_center[...,0], 2) +\
                 torch.pow(boxes1_center[...,1]-boxes2_center[...,1], 2)

    # cal penalty term
    # cal width,height
    boxes1_size = torch.max(boxes1[..., 2:] - boxes1[..., :2], torch.
↪zeros_like(inter_right_down))
    boxes2_size = torch.max(boxes2[..., 2:] - boxes2[..., :2], torch.
↪zeros_like(inter_right_down))
    v = (4 / (math.pi ** 2)) * torch.pow(
```

```
            torch.atan((boxes1_size[...,0]/torch.clamp(boxes1_size[...,1],min =␣
 ↪1e-6))) -
            torch.atan((boxes2_size[..., 0] / torch.clamp(boxes2_size[...,␣
 ↪1],min = 1e-6))), 2)

    alpha = v / (1-ious+v)

    #cal ciou
    cious = ious - (center_dis / outer_diagonal_line + alpha*v)

    return cious




# def run_training(model, optimizer, dataloader, device, img_size, n_epoch,␣
 ↪every_n_batch, every_n_epoch, ckpt_dir):
#     for epoch_i in range(n_epoch):
#         running_loss = 0.0
#         for inputs, labels, bboxes in dataloader:
#             inputs = torch.from_numpy(np.array(inputs)).squeeze(1).float()
#             inputs = inputs.to(device)

#             # Initialize lists to hold the tensor labels and bounding boxes␣
 ↪for the current batch
#             label_tensors = []
#             box_tensors = []

#             for label in labels:
#                 # Convert each label into a tensor and move to device
#                 label_tensor = torch.tensor(label).to(device)
#                 label_tensors.append(label_tensor)

#             print(label_tensors)

#             for bbox in bboxes:
#                 # Convert each bounding box into a tensor and move to device
#                 bbox_tensor = torch.tensor(bbox).to(device)
#                 box_tensors.append(bbox_tensor)

#             # At this point, label_tensors is a list of tensors
#             # You can process these lists as needed

#             # Perform the model forward pass and loss calculation
#             optimizer.zero_grad()
#             with torch.set_grad_enabled(True):
#                 outputs = model(inputs, True)
```

```
#                 pred_xywh = outputs[..., 0:4] / img_size
#                 pred_conf = outputs[..., 4:5]
#                 pred_cls = outputs[..., 5:]

#                 # Prepare the target values
#                 # label_xywh = [label[:, :4] for label in label_tensors]  #␣
 ↪Normalize the xywh
#                 label_obj_mask = [label[:, 4:5] for label in label_tensors]
#                 label_cls = [label[:, 5:] for label in label_tensors]

#                 # Compute the losses for each image in the batch
#                 loss = 0
#                 for i in range(len(label_tensors)):
#                     lambda_coord = 0.001
#                     lambda_noobj = 0.05
#                     # Calculate losses for the i-th image
#                     loss_coord = lambda_coord * label_obj_mask[i] * nn.
 ↪MSELoss()(pred_xywh[i], label_xywh[i])
#                     loss_conf = (label_obj_mask[i] * nn.
 ↪BCELoss()(pred_conf[i], label_obj_mask[i])) + \
#                                 (lambda_noobj * (1.0 - label_obj_mask[i]) *␣
 ↪nn.BCELoss()(pred_conf[i], label_obj_mask[i]))
#                     loss_cls = label_obj_mask[i] * nn.BCELoss()(pred_cls[i],␣
 ↪label_cls[i])

#                     loss += loss_coord + loss_conf + loss_cls

#                 loss.backward()
#                 optimizer.step()

#                 # Statistics
#                 running_loss += loss.item() * inputs.size(0)

#         epoch_loss = running_loss / len(dataloader.dataset)
#         print(epoch_loss)
#         print('End Epoch')

import torch
import torch.nn as nn
import numpy as np

def run_training(model, optimizer, dataloader, device, img_size, n_epoch,␣
 ↪every_n_batch, every_n_epoch, ckpt_dir):
    for epoch_i in range(n_epoch):
        running_loss = 0.0
```

```
        for inputs, labels, bboxes in dataloader:
            # Assume inputs, labels, and bboxes are provided as batches of size␣
↪1
            inputs = torch.from_numpy(np.array(inputs)).squeeze(1).float().
↪to(device)  # Ensure inputs are on device

            # Extract single label and bounding box
            label = torch.tensor(labels[0], dtype=torch.float32,␣
↪requires_grad=False).to(device)
            bbox = torch.tensor(bboxes[0], dtype=torch.float32,␣
↪requires_grad=False).to(device)

            # Ensure label tensor is 2D: shape (1, 6)
            if label.dim() == 1:  # If it's 1D, we can add a dimension
                label = label.unsqueeze(0)  # Add a new dimension

            # Ensure bounding box tensor is 2D: shape (1, 4)
            if bbox.dim() == 1:
                bbox = bbox.unsqueeze(0)  # Add a new dimension

            # Forward pass
            optimizer.zero_grad()
            with torch.set_grad_enabled(True):
                outputs = model(inputs, device)  # Ensure model outputs require␣
↪grad

                pred_xywh = outputs[..., :4]  # Assuming the first four are x,␣
↪y, width, height
                pred_conf = outputs[..., 4:5]
                pred_cls = outputs[..., 5:]

                # Initialize loss variables
                loss = torch.tensor(0.0, device=device, requires_grad=True)  #␣
↪Initialize loss as a tensor with requires_grad=True

                lambda_coord = 0.001
                lambda_noobj = 0.05

                # Compute losses
                if label.size(1) == 6:  # Ensure label has the correct shape
                    label_obj_mask = label[:, 4:5]  # Object mask
                    label_cls = label[:, 5:]  # Class labels

                    # Compute losses for the current image
```

```
                    loss_coord = lambda_coord * label_obj_mask * nn.
↪MSELoss()(pred_xywh, label[:, :4])
                    loss_conf = (label_obj_mask * nn.BCELoss()(pred_conf,␣
↪label_obj_mask)) + \
                                (lambda_noobj * (1.0 - label_obj_mask) * nn.
↪BCELoss()(pred_conf, label_obj_mask))
                    loss_cls = label_obj_mask * nn.BCELoss()(pred_cls,␣
↪label_cls)

                    # Accumulate the loss
                    loss += loss_coord.sum() + loss_conf.sum() + loss_cls.sum()␣
↪ # Ensure loss is accumulated correctly

                # Check if loss requires grad
                print(f"Loss requires_grad: {loss.requires_grad}, Loss value:␣
↪{loss.item()}")

                loss.backward()  # Ensure loss is a tensor
                optimizer.step()

                # Statistics
                running_loss += loss.item()

        epoch_loss = running_loss / len(dataloader)
        print(f'Epoch [{epoch_i + 1}/{n_epoch}], Loss: {epoch_loss:.4f}')
        print('End Epoch')

        # Optional: Save checkpoint every n epoch
        if (epoch_i + 1) % every_n_epoch == 0:
            torch.save(model.state_dict(), f'{ckpt_dir}/model_epoch_{epoch_i +␣
↪1}.pth')
```

[ ]:
```
# Set device to GPU or CPU
gpu = "0"
device = torch.device("cuda:{}".format(gpu) if torch.cuda.is_available() else␣
 ↪"cpu")

print("Loading network.....")
model = MyDarknet("cfg/yolov3.cfg")
# load pretrained
# model.load_weights("yolov3.weights")
print("Network successfully loaded")

model.to(device)

optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
n_epoch = 5
img_size = 416
save_every_batch = False
save_every_epoch = True
ckpt_dir = "./checkpoints"


run_training(model, optimizer, train_dataloader, device,
                 img_size,
                 n_epoch,
                 save_every_batch,
                 save_every_epoch,
                 ckpt_dir)
```

```
Loading network…
Network successfully loaded
```

```
<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]
<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]
<ipython-input-7-2251ccd1d1c8>:169: UserWarning: To copy construct from a
tensor, it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  label = torch.tensor(labels[0], dtype=torch.float32,
requires_grad=False).to(device)
<ipython-input-7-2251ccd1d1c8>:170: UserWarning: To copy construct from a
tensor, it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  bbox = torch.tensor(bboxes[0], dtype=torch.float32,
requires_grad=False).to(device)
```

```
Loss requires_grad: True, Loss value: 0.0
Loss requires_grad: True, Loss value: 0.0
Epoch [1/5], Loss: 0.0000
End Epoch
```

```
<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
```

```
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]
<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]

Loss requires_grad: True, Loss value: 0.0
Loss requires_grad: True, Loss value: 0.0
Epoch [2/5], Loss: 0.0000
End Epoch

<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]
<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]

Loss requires_grad: True, Loss value: 0.0
Loss requires_grad: True, Loss value: 0.0
Epoch [3/5], Loss: 0.0000
End Epoch

<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]
<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]

Loss requires_grad: True, Loss value: 0.0
Loss requires_grad: True, Loss value: 0.0
Epoch [4/5], Loss: 0.0000
End Epoch

<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]
<ipython-input-6-cc9cf5029238>:13: UserWarning: To copy construct from a tensor,
```

```
it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  labels = [torch.tensor(label) for label in labels]

Loss requires_grad: True, Loss value: 0.0
Loss requires_grad: True, Loss value: 0.0
Epoch [5/5], Loss: 0.0000
End Epoch
```

2. Please see YOLOv8. This is the latest version of YOLO. Follow the instruction on the github and perform the training on any dataset.

[ ]:
```
!pip install ultralytics
```

```
Collecting ultralytics
  Downloading ultralytics-8.3.17-py3-none-any.whl.metadata (34 kB)
Requirement already satisfied: numpy>=1.23.0 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (1.26.4)
Requirement already satisfied: matplotlib>=3.3.0 in
/usr/local/lib/python3.10/dist-packages (from ultralytics) (3.7.1)
Requirement already satisfied: opencv-python>=4.6.0 in
/usr/local/lib/python3.10/dist-packages (from ultralytics) (4.10.0.84)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (10.4.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (6.0.2)
Requirement already satisfied: requests>=2.23.0 in
/usr/local/lib/python3.10/dist-packages (from ultralytics) (2.32.3)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (1.13.1)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (2.4.1+cu121)
Requirement already satisfied: torchvision>=0.9.0 in
/usr/local/lib/python3.10/dist-packages (from ultralytics) (0.19.1+cu121)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (4.66.5)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages
(from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (9.0.0)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-
packages (from ultralytics) (2.2.2)
Requirement already satisfied: seaborn>=0.11.0 in
/usr/local/lib/python3.10/dist-packages (from ultralytics) (0.13.2)
Collecting ultralytics-thop>=2.0.0 (from ultralytics)
  Downloading ultralytics_thop-2.0.9-py3-none-any.whl.metadata (9.3 kB)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics)
```

```
(1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=3.3.0->ultralytics) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics)
(4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics)
(1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics)
(24.1)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics)
(3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->ultralytics)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.1.4->ultralytics) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.1.4->ultralytics) (2024.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics)
(3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests>=2.23.0->ultralytics) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics)
(2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.23.0->ultralytics)
(2024.8.30)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch>=1.8.0->ultralytics) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch>=1.8.0->ultralytics)
(4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch>=1.8.0->ultralytics) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch>=1.8.0->ultralytics) (3.4.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch>=1.8.0->ultralytics) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch>=1.8.0->ultralytics) (2024.6.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib>=3.3.0->ultralytics) (1.16.0)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.8.0->ultralytics)
(3.0.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.8.0->ultralytics)
(1.3.0)
Downloading ultralytics-8.3.17-py3-none-any.whl (876 kB)
                         876.6/876.6 kB
26.3 MB/s eta 0:00:00
Downloading ultralytics_thop-2.0.9-py3-none-any.whl (26 kB)
Installing collected packages: ultralytics-thop, ultralytics
Successfully installed ultralytics-8.3.17 ultralytics-thop-2.0.9
```

[ ]:
```
!wget https://storage.googleapis.com/openimages/v6/oidv6-train-annotations-bbox.
↪csv
```

```
--2024-10-19 10:44:36--
https://storage.googleapis.com/openimages/v6/oidv6-train-annotations-bbox.csv
Resolving storage.googleapis.com (storage.googleapis.com)… 64.233.170.207,
142.251.175.207, 74.125.24.207, …
Connecting to storage.googleapis.com
(storage.googleapis.com)|64.233.170.207|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 2258447590 (2.1G) [text/csv]
Saving to: ‘oidv6-train-annotations-bbox.csv’

oidv6-train-annotat 100%[===================>]   2.10G  22.9MB/s    in 97s

2024-10-19 10:46:14 (22.2 MB/s) - ‘oidv6-train-annotations-bbox.csv’ saved
[2258447590/2258447590]
```

[ ]:
```
!wget https://storage.googleapis.com/openimages/v5/validation-annotations-bbox.
↪csv
```

```
--2024-10-19 10:46:14--
https://storage.googleapis.com/openimages/v5/validation-annotations-bbox.csv
Resolving storage.googleapis.com (storage.googleapis.com)… 74.125.200.207,
74.125.130.207, 74.125.68.207, …
Connecting to storage.googleapis.com
(storage.googleapis.com)|74.125.200.207|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 25105048 (24M) [text/csv]
Saving to: ‘validation-annotations-bbox.csv’

validation-annotati 100%[===================>]  23.94M  9.96MB/s    in 2.4s

2024-10-19 10:46:17 (9.96 MB/s) - ‘validation-annotations-bbox.csv’ saved
[25105048/25105048]
```

[ ]:
```
!wget https://storage.googleapis.com/openimages/v5/test-annotations-bbox.csv
```

```
--2024-10-19 10:47:46--  https://storage.googleapis.com/openimages/v5/test-
annotations-bbox.csv
Resolving storage.googleapis.com (storage.googleapis.com)… 74.125.68.207,
64.233.170.207, 142.251.175.207, …
Connecting to storage.googleapis.com
(storage.googleapis.com)|74.125.68.207|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 77484237 (74M) [text/csv]
Saving to: ‘test-annotations-bbox.csv’

test-annotations-bb 100%[===================>]  73.89M  19.0MB/s    in 5.3s

2024-10-19 10:47:51 (14.0 MB/s) - ‘test-annotations-bbox.csv’ saved
[77484237/77484237]
```

[ ]:
```
!wget https://raw.githubusercontent.com/openimages/dataset/master/downloader.py
!pip install boto3
```

```
--2024-10-19 10:47:51--
https://raw.githubusercontent.com/openimages/dataset/master/downloader.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 4244 (4.1K) [text/plain]
Saving to: ‘downloader.py’

downloader.py       100%[===================>]   4.14K  --.-KB/s    in 0s

2024-10-19 10:47:52 (50.1 MB/s) - ‘downloader.py’ saved [4244/4244]

Collecting boto3
  Downloading boto3-1.35.44-py3-none-any.whl.metadata (6.7 kB)
Collecting botocore<1.36.0,>=1.35.44 (from boto3)
  Downloading botocore-1.35.44-py3-none-any.whl.metadata (5.7 kB)
Collecting jmespath<2.0.0,>=0.7.1 (from boto3)
  Downloading jmespath-1.0.1-py3-none-any.whl.metadata (7.6 kB)
Collecting s3transfer<0.11.0,>=0.10.0 (from boto3)
  Downloading s3transfer-0.10.3-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
/usr/local/lib/python3.10/dist-packages (from botocore<1.36.0,>=1.35.44->boto3)
(2.8.2)
Requirement already satisfied: urllib3!=2.2.0,<3,>=1.25.4 in
```

```
/usr/local/lib/python3.10/dist-packages (from botocore<1.36.0,>=1.35.44->boto3)
(2.2.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil<3.0.0,>=2.1->botocore<1.36.0,>=1.35.44->boto3)
(1.16.0)
Downloading boto3-1.35.44-py3-none-any.whl (139 kB)
                         139.2/139.2 kB
11.7 MB/s eta 0:00:00
Downloading botocore-1.35.44-py3-none-any.whl (12.6 MB)
                         12.6/12.6 MB
64.8 MB/s eta 0:00:00
Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Downloading s3transfer-0.10.3-py3-none-any.whl (82 kB)
                         82.6/82.6 kB
7.5 MB/s eta 0:00:00
Installing collected packages: jmespath, botocore, s3transfer, boto3
Successfully installed boto3-1.35.44 botocore-1.35.44 jmespath-1.0.1
s3transfer-0.10.3
```

[ ]:
```
import os
import shutil

# install and import pyyaml used to create custom config files for training
%pip install pyyaml
import yaml
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages
(6.0.2)
```

[ ]:
```
def create_yolo_yaml_config(yaml_filepath, dataset_path, dataset_labels):

    data = {'path':dataset_path,
            'train': os.path.join('images', 'train'),
            'val': os.path.join('images', 'validation'),
            'names':{i:label for i, label in enumerate(dataset_labels)}
            }

    # Save the changes to the file
    with open(yaml_filepath, 'w') as fp:
    # set sort_keys = False to preserve the order of keys
        yaml.dump(data, fp, sort_keys=False)
```

[ ]:
```
def get_class_id(classes_file, class_names):
    id_name_dict = {}
    with open(classes_file, 'r') as f:
        for line in f:
            print(line)
            id, label = line.split(',')
```

```
            label = label.strip()
            #print(label)
            if label in class_names:
                print(label)
                id_name_dict[label] = id

    return id_name_dict
```

[ ]:
```
names = ['Ant', 'Insect']

# path to the dataset directory (recommended to use absolute path)
dataset_path = os.path.abspath(os.path.join('.', 'data'))

# path to the YAML file that contains training configuration
yaml_filepath = os.path.join('.', 'config.yaml')

# Create a custom YAML config file based on the above selected target objects␣
 ↪and dataset path
create_yolo_yaml_config(yaml_filepath, dataset_path, names)
```

[ ]:
```
!wget https://github.com/mohamedamine99/YOLOv8-custom-object-detection/blob/
 ↪main/Custom-object-detection-with-YOLOv8/class-descriptions-boxable.csv
```

```
--2024-10-19 10:50:32--  https://github.com/mohamedamine99/YOLOv8-custom-object-
detection/blob/main/Custom-object-detection-with-YOLOv8/class-descriptions-
boxable.csv
Resolving github.com (github.com)… 20.205.243.166
Connecting to github.com (github.com)|20.205.243.166|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: unspecified [text/html]
Saving to: ‘class-descriptions-boxable.csv’

class-descriptions-     [ <=>                ] 182.18K  --.-KB/s    in 0.01s

2024-10-19 10:50:32 (12.7 MB/s) - ‘class-descriptions-boxable.csv’ saved
[186554]
```

[ ]:
```
class_ids = get_class_id('./class-descriptions-boxable.csv', names)
print(class_ids)
print(names)
```

```
/m/011k07,Tortoise

/m/011q46kg,Container

/m/012074,Magpie
```

/m/0120dh,Sea turtle

/m/01226z,Football

/m/012n7d,Ambulance

/m/012w5l,Ladder

/m/012xff,Toothbrush

/m/012ysf,Syringe

/m/0130jx,Sink

/m/0138tl,Toy

/m/013y1f,Organ (Musical Instrument)

/m/01432t,Cassette deck

/m/014j1m,Apple

/m/014sv8,Human eye

/m/014trl,Cosmetics

/m/014y4n,Paddle

/m/0152hh,Snowman

/m/01599,Beer

/m/01_5g,Chopsticks

/m/015h_t,Human beard

/m/015p6,Bird

/m/015qbp,Parking meter

/m/015qff,Traffic light

/m/015wgc,Croissant

/m/015x4r,Cucumber

/m/015x5n,Radish

/m/0162_1,Towel

/m/0167gd,Doll

/m/016m2d,Skull

/m/0174k2,Washing machine

/m/0174n1,Glove

/m/0175cv,Tick

/m/0176mf,Belt

/m/017ftj,Sunglasses

/m/018j2,Banjo

/m/018p4k,Cart

/m/018xm,Ball

/m/01940j,Backpack

/m/0199g,Bicycle

/m/019dx1,Home appliance

/m/019h78,Centipede

/m/019jd,Boat

/m/019w40,Surfboard

/m/01b638,Boot

/m/01b7fy,Headphones

/m/01b9xk,Hot dog

/m/01bfm9,Shorts

/m/01_bhs,Fast food

/m/01bjv,Bus

/m/01bl7v,Boy

/m/01bms0,Screwdriver

/m/01bqk0,Bicycle wheel

/m/01btn,Barge

/m/01c648,Laptop

/m/01cmb2,Miniskirt

/m/01d380,Drill (Tool)

/m/01d40f,Dress

/m/01dws,Bear

/m/01dwsz,Waffle

/m/01dwwc,Pancake

/m/01dxs,Brown bear

/m/01dy8n,Woodpecker

/m/01f8m5,Blue jay

/m/01f91_,Pretzel

/m/01fb_0,Bagel

/m/01fdzj,Tower

/m/01fh4r,Teapot

/m/01g317,Person

/m/01g3x7,Bow and arrow

/m/01gkx_,Swimwear

/m/01gllr,Beehive

/m/01gmv2,Brassiere

/m/01h3n,Bee

/m/01h44,Bat (Animal)

/m/01h8tj,Starfish

/m/01hrv5,Popcorn

/m/01j3zr,Burrito

/m/01j4z9,Chainsaw

/m/01j51,Balloon

/m/01j5ks,Wrench

/m/01j61q,Tent

/m/01jfm_,Vehicle registration plate

/m/01jfsr,Lantern

/m/01k6s3,Toaster

/m/01kb5b,Flashlight

/m/01knjb,Billboard

/m/01krhy,Tiara

/m/01lcw4,Limousine

/m/01llwg,Necklace

/m/01lrl,Carnivore

/m/01lsmm,Scissors

/m/01lynh,Stairs

/m/01m2v,Computer keyboard

/m/01m4t,Printer

/m/01mqdt,Traffic sign

/m/01mzpv,Chair

/m/01n4qj,Shirt

/m/01n5jq,Poster

/m/01nkt,Cheese

/m/01nq26,Sock

/m/01pns0,Fire hydrant

/m/01prls,Land vehicle

/m/01r546,Earrings

/m/01rkbr,Tie

/m/01rzcn,Watercraft

/m/01s105,Cabinetry

/m/01s55n,Suitcase

/m/01tcjp,Muffin

/m/01vbnl,Bidet

/m/01ww8y,Snack

/m/01x3jk,Snowmobile

/m/01x3z,Clock

/m/01xgg_,Medical equipment

/m/01xq0k1,Cattle

/m/01xqw,Cello

/m/01xs3r,Jet ski

/m/01x_v,Camel

/m/01xygc,Coat

/m/01xyhv,Suit

/m/01y9k5,Desk

/m/01yrx,Cat

/m/01yx86,Bronze sculpture

/m/01z1kdw,Juice

/m/02068x,Gondola

/m/020jm,Beetle

/m/020kz,Cannon

/m/020lf,Computer mouse

/m/021mn,Cookie

/m/021sj1,Office building

/m/0220r2,Fountain

/m/0242l,Coin

/m/024d2,Calculator

/m/024g6,Cocktail

/m/02522,Computer monitor

/m/025dyy,Box

/m/025fsf,Stapler

/m/025nd,Christmas tree

/m/025rp__,Cowboy hat

/m/0268lbt,Hiking equipment

/m/026qbn5,Studio couch

/m/026t6,Drum

/m/0270h,Dessert

/m/0271qf7,Wine rack

/m/0271t,Drink

/m/027pcv,Zucchini

/m/027rl48,Ladle

/m/0283dt1,Human mouth

/m/0284d,Dairy Product

/m/029b3,Dice

/m/029bxz,Oven

/m/029tx,Dinosaur

/m/02bm9n,Ratchet (Device)

/m/02crq1,Couch

/m/02ctlc,Cricket ball

/m/02cvgx,Winter melon

/m/02d1br,Spatula

/m/02d9qx,Whiteboard

/m/02ddwp,Pencil sharpener

/m/02dgv,Door

/m/02dl1y,Hat

/m/02f9f_,Shower

/m/02fh7f,Eraser

/m/02fq_6,Fedora

/m/02g30s,Guacamole

/m/02gzp,Dagger

/m/02h19r,Scarf

/m/02hj4,Dolphin

/m/02jfl0,Sombrero

/m/02jnhm,Tin can

/m/02jvh9,Mug

/m/02jz0l,Tap

/m/02l8p9,Harbor seal

/m/02lbcq,Stretcher

/m/02mqfb,Can opener

/m/02_n6y,Goggles

/m/02p0tk3,Human body

/m/02p3w7d,Roller skates

/m/02p5f1q,Coffee cup

/m/02pdsw,Cutting board

/m/02pjr4,Blender

/m/02pkr5,Plumbing fixture

/m/02pv19,Stop sign

/m/02rdsp,Office supplies

/m/02rgn06,Volleyball (Ball)

/m/02s195,Vase

/m/02tsc9,Slow cooker

/m/02vkqh8,Wardrobe

/m/02vqfm,Coffee

/m/02vwcm,Whisk

/m/02w3r3,Paper towel

/m/02w3_ws,Personal care

/m/02wbm,Food

/m/02wbtzl,Sun hat

/m/02wg_p,Tree house

/m/02wmf,Flying disc

/m/02wv6h6,Skirt

/m/02wv84t,Gas stove

/m/02x8cch,Salt and pepper shakers

/m/02x984l,Mechanical fan

/m/02xb7qb,Face powder

/m/02xqq,Fax

/m/02xwb,Fruit

/m/02y6n,French fries

/m/02z51p,Nightstand

/m/02zn6n,Barrel

/m/02zt3,Kite

/m/02zvsm,Tart

/m/030610,Treadmill

/m/0306r,Fox

/m/03120,Flag

/m/0319l,French horn

/m/031b6r,Window blind

/m/031n1,Human foot

/m/0323sq,Golf cart

/m/032b3c,Jacket

/m/033cnk,Egg (Food)

/m/033rq4,Street light

/m/0342h,Guitar

/m/034c16,Pillow

/m/035r7c,Human leg

/m/035vxb,Isopod

/m/0388q,Grape

/m/039xj_,Human ear

/m/03bbps,Power plugs and sockets

/m/03bj1,Panda

/m/03bk1,Giraffe

/m/03bt1vf,Woman

/m/03c7gz,Door handle

/m/03d443,Rhinoceros

/m/03dnzn,Bathtub

/m/03fj2,Goldfish

/m/03fp41,Houseplant

/m/03fwl,Goat

/m/03g8mr,Baseball bat

/m/03grzl,Baseball glove

/m/03hj559,Mixing bowl

/m/03hl4l9,Marine invertebrates

/m/03hlz0c,Kitchen utensil

/m/03jbxj,Light switch

/m/03jm5,House

/m/03k3r,Horse

/m/03kt2w,Stationary bicycle

/m/03l9g,Hammer

/m/03ldnb,Ceiling fan

/m/03m3pdh,Sofa bed

/m/03m3vtv,Adhesive tape

/m/03m5k,Harp

/m/03nfch,Sandal

/m/03p3bw,Bicycle helmet

/m/03q5c7,Saucer

/m/03q5t,Harpsichord

/m/03q69,Human hair

/m/03qhv5,Heater

/m/03qjg,Harmonica

/m/03qrc,Hamster

/m/03rszm,Curtain

/m/03ssj5,Bed

/m/03s_tn,Kettle

/m/03tw93,Fireplace

/m/03txqz,Scale

/m/03v5tg,Drinking straw

/m/03vt0,Insect

Insect
/m/03wvsk,Hair dryer

/m/03_wxk,Kitchenware

/m/03wym,Indoor rower

/m/03xxp,Invertebrate

/m/03y6mg,Food processor

/m/03__z0,Bookcase

/m/040b_t,Refrigerator

/m/04169hn,Wood-burning stove

/m/0420v5,Punching bag

/m/043nyj,Common fig

/m/0440zs,Cocktail shaker

/m/0449p,Jaguar (Animal)

/m/044r5d,Golf ball

/m/0463sg,Fashion accessory

/m/046dlr,Alarm clock

/m/047j0r,Filing cabinet

/m/047v4b,Artichoke

/m/04bcr3,Table

/m/04brg2,Tableware

/m/04c0y,Kangaroo

/m/04cp_,Koala

/m/04ctx,Knife

/m/04dr76w,Bottle

/m/04f5ws,Bottle opener

/m/04g2r,Lynx

/m/04gth,Lavender (Plant)

/m/04h7h,Lighthouse

/m/04h8sr,Dumbbell

/m/04hgtk,Human head

/m/04kkgm,Bowl

/m/04lvq_,Humidifier

/m/04m6gz,Porch

/m/04m9y,Lizard

/m/04p0qw,Billiard table

/m/04rky,Mammal

/m/04rmv,Mouse

/m/04_sv,Motorcycle

/m/04szw,Musical instrument

/m/04tn4x,Swim cap

/m/04v6l4,Frying pan

/m/04vv5k,Snowplow

/m/04y4h8h,Bathroom cabinet

/m/04ylt,Missile

/m/04yqq2,Bust

/m/04yx4,Man

/m/04z4wx,Waffle iron

/m/04zpv,Milk

/m/04zwwv,Ring binder

/m/050gv4,Plate

/m/050k8,Mobile phone

/m/052lwg6,Baked goods

/m/052sf,Mushroom

/m/05441v,Crutch

/m/054fyh,Pitcher (Container)

/m/054_l,Mirror

/m/054xkw,Personal flotation device

/m/05_5p_0,Table tennis racket

/m/05676x,Pencil case

/m/057cc,Musical keyboard

/m/057p5t,Scoreboard

/m/0584n8,Briefcase

/m/058qzx,Kitchen knife

/m/05bm6,Nail (Construction)

/m/05ctyq,Tennis ball

/m/05gqfk,Plastic bag

/m/05kms,Oboe

/m/05kyg_,Chest of drawers

/m/05n4y,Ostrich

/m/05r5c,Piano

/m/05r655,Girl

/m/05s2s,Plant

/m/05vtc,Potato

/m/05w9t9,Hair spray

/m/05y5lj,Sports equipment

/m/05z55,Pasta

/m/05z6w,Penguin

/m/05zsy,Pumpkin

/m/061_f,Pear

/m/061hd_,Infant bed

/m/0633h,Polar bear

/m/063rgb,Mixer

/m/0642b4,Cupboard

/m/065h6l,Jacuzzi

/m/0663v,Pizza

/m/06_72j,Digital clock

/m/068zj,Pig

/m/06bt6,Reptile

/m/06c54,Rifle

/m/06c7f7,Lipstick

/m/06_fw,Skateboard

/m/06j2d,Raven

/m/06k2mb,High heels

/m/06l9r,Red panda

/m/06m11,Rose

/m/06mf6,Rabbit

/m/06msq,Sculpture

/m/06ncr,Saxophone

/m/06nrc,Shotgun

/m/06nwz,Seafood

/m/06pcq,Submarine sandwich

/m/06__v,Snowboard

/m/06y5r,Sword

/m/06z37_,Picture frame

/m/07030,Sushi

/m/0703r8,Loveseat

/m/071p9,Ski

/m/071qp,Squirrel

/m/073bxn,Tripod

/m/073g6,Stethoscope

/m/074d1,Submarine

/m/0755b,Scorpion

/m/076bq,Segway

/m/076lb9,Training bench

/m/078jl,Snake

/m/078n6m,Coffee table

/m/079cl,Skyscraper

/m/07bgp,Sheep

/m/07c52,Television

/m/07c6l,Trombone

/m/07clx,Tea

/m/07cmd,Tank

/m/07crc,Taco

/m/07cx4,Telephone

/m/07dd4,Torch

/m/07dm6,Tiger

/m/07fbm7,Strawberry

/m/07gql,Trumpet

/m/07j7r,Tree

/m/07j87,Tomato

/m/07jdr,Train

/m/07k1x,Tool

/m/07kng9,Picnic basket

/m/07mcwg,Cooking spray

/m/07mhn,Trousers

/m/07pj7bq,Bowling equipment

/m/07qxg_,Football helmet

/m/07r04,Truck

/m/07v9_z,Measuring cup

/m/07xyvk,Coffeemaker

/m/07y_7,Violin

/m/07yv9,Vehicle

/m/080hkjn,Handbag

/m/080n7g,Paper cutter

/m/081qc,Wine

/m/083kb,Weapon

/m/083wq,Wheel

/m/084hf,Worm

/m/084rd,Wok

/m/084zz,Whale

/m/0898b,Zebra

/m/08dz3q,Auto part

/m/08hvt4,Jug

/m/08ks85,Pizza cutter

/m/08p92x,Cream

/m/08pbxl,Monkey

/m/096mb,Lion

/m/09728,Bread

/m/099ssp,Platter

/m/09b5t,Chicken

/m/09csl,Eagle

/m/09ct_,Helicopter

/m/09d5_,Owl

/m/09ddx,Duck

/m/09dzg,Turtle

/m/09f20,Hippopotamus

/m/09f_2,Crocodile

/m/09g1w,Toilet

/m/09gtd,Toilet paper

/m/09gys,Squid

/m/09j2d,Clothing

/m/09j5n,Footwear

/m/09k_b,Lemon

/m/09kmb,Spider

/m/09kx5,Deer

/m/09ld4,Frog

/m/09qck,Banana

/m/09rvcxw,Rocket

/m/09tvcd,Wine glass

/m/0b3fp9,Countertop

/m/0bh9flk,Tablet computer

/m/0bjyj5,Waste container

/m/0b_rs,Swimming pool

/m/0bt9lr,Dog

/m/0bt_c3,Book

/m/0bwd_0j,Elephant

/m/0by6g,Shark

/m/0c06p,Candle

/m/0c29q,Leopard

/m/0c2jj,Axe

/m/0c3m8g,Hand dryer

/m/0c3mkw,Soap dispenser

/m/0c568,Porcupine

/m/0c9ph5,Flower

/m/0ccs93,Canary

/m/0cd4d,Cheetah

/m/0cdl1,Palm tree

/m/0cdn1,Hamburger

/m/0cffdh,Maple

/m/0cgh4,Building

/m/0ch_cf,Fish

/m/0cjq5,Lobster

/m/0cjs7,Garden Asparagus

/m/0c_jw,Furniture

/m/0cl4p,Hedgehog

/m/0cmf2,Airplane

/m/0cmx8,Spoon

/m/0cn6p,Otter

/m/0cnyhnx,Bull

/m/0_cp5,Oyster

/m/0cqn2,Horizontal bar

/m/0crjs,Convenience store

/m/0ct4f,Bomb

/m/0cvnqh,Bench

/m/0cxn2,Ice cream

/m/0cydv,Caterpillar

/m/0cyf8,Butterfly

/m/0cyfs,Parachute

/m/0cyhj_,Orange

/m/0czz2,Antelope

/m/0d20w4,Beaker

/m/0d_2m,Moths and butterflies

/m/0d4v4,Window

/m/0d4w1,Closet

/m/0d5gx,Castle

/m/0d8zb,Jellyfish

/m/0dbvp,Goose

/m/0dbzx,Mule

/m/0dftk,Swan

/m/0dj6p,Peach

/m/0djtd,Coconut

/m/0dkzw,Seat belt

/m/0dq75,Raccoon

/m/0_dqb,Chisel

/m/0dt3t,Fork

/m/0dtln,Lamp

/m/0dv5r,Camera

/m/0dv77,Squash (Plant)

/m/0dv9c,Racket

/m/0dzct,Human face

/m/0dzf4,Human arm

/m/0f4s2w,Vegetable

/m/0f571,Diaper

/m/0f6nr,Unicycle

/m/0f6wt,Falcon

/m/0f8s22,Chime

/m/0f9_l,Snail

/m/0fbdv,Shellfish

/m/0fbw6,Cabbage

/m/0fj52s,Carrot

/m/0fldg,Mango

/m/0fly7,Jeans

/m/0fm3zh,Flowerpot

/m/0fp6w,Pineapple

/m/0fqfqc,Drawer

/m/0fqt361,Stool

/m/0frqm,Envelope

/m/0fszt,Cake

/m/0ft9s,Dragonfly

/m/0ftb8,Common sunflower

/m/0fx9l,Microwave oven

/m/0fz0h,Honeycomb

/m/0gd2v,Marine mammal

/m/0gd36,Sea lion

/m/0gj37,Ladybug

/m/0gjbg72,Shelf

/m/0gjkl,Watch

/m/0gm28,Candy

/m/0grw1,Salad

/m/0gv1x,Parrot

/m/0gxl3,Handgun

/m/0h23m,Sparrow

/m/0h2r6,Van

/m/0h8jyh6,Grinder

/m/0h8kx63,Spice rack

/m/0h8l4fh,Light bulb

/m/0h8lkj8,Corded phone

/m/0h8mhzd,Sports uniform

/m/0h8my_4,Tennis racket

/m/0h8mzrc,Wall clock

/m/0h8n27j,Serving tray

/m/0h8n5zk,Kitchen & dining room table

/m/0h8n6f9,Dog bed

/m/0h8n6ft,Cake stand

/m/0h8nm9j,Cat furniture

/m/0h8nr_l,Bathroom accessory

/m/0h8nsvg,Facial tissue holder

/m/0h8ntjv,Pressure cooker

/m/0h99cwc,Kitchen appliance

/m/0h9mv,Tire

/m/0hdln,Ruler

/m/0hf58v5,Luggage and bags

/m/0hg7b,Microphone

/m/0hkxq,Broccoli

/m/0hnnb,Umbrella

/m/0hnyx,Pastry

/m/0hqkz,Grapefruit

/m/0j496,Band-aid

/m/0jbk,Animal

/m/0jg57,Bell pepper

/m/0jly1,Turkey

/m/0jqgx,Lily

/m/0jwn_,Pomegranate

/m/0jy4k,Doughnut

/m/0jyfg,Glasses

/m/0k0pj,Human nose

/m/0k1tl,Pen

/m/0_k2,Ant

Ant
/m/0k4j,Car

/m/0k5j,Aircraft

/m/0k65p,Human hand

/m/0km7z,Skunk

/m/0kmg4,Teddy bear

/m/0kpqd,Watermelon

/m/0kpt_,Cantaloupe

/m/0ky7b,Dishwasher

```
/m/0l14j_,Flute

/m/0l3ms,Balance beam

/m/0l515,Sandwich

/m/0ll1f78,Shrimp

/m/0llzx,Sewing machine

/m/0lt4_,Binoculars

/m/0m53l,Rays and skates

/m/0mcx2,Ipod

/m/0mkg,Accordion

/m/0mw_6,Willow

/m/0n28_,Crab

/m/0nl46,Crown

/m/0nybt,Seahorse

/m/0p833,Perfume

/m/0pcr,Alpaca

/m/0pg52,Taxi

/m/0ph39,Canoe

/m/0qjjc,Remote control

/m/0qmmr,Wheelchair

/m/0wdt60w,Rugby ball

/m/0xfy,Armadillo

/m/0xzly,Maracas

/m/0zvk5,Helmet

{'Insect': '/m/03vt0', 'Ant': '/m/0_k2'}
['Ant', 'Insect']
```

[ ]:
```
train_bboxes_filename = os.path.join('.', 'oidv6-train-annotations-bbox.csv')
validation_bboxes_filename = os.path.join('.', 'validation-annotations-bbox.
 ↪csv')
test_bboxes_filename = os.path.join('.', 'test-annotations-bbox.csv')

image_list_file_path = os.path.join('.', 'image_list_file.txt')

image_list_file_list = []
for j, filename in enumerate([train_bboxes_filename,␣
 ↪validation_bboxes_filename, test_bboxes_filename]):
    print(filename)
    with open(filename, 'r') as f:
        line = f.readline()
        while len(line) != 0:
            id, _, class_name, _, x1, x2, y1, y2, _, _, _, _, _ = line.
 ↪split(',')[:13]
            if class_name in list(class_ids.values()) and id not in␣
 ↪image_list_file_list:
                image_list_file_list.append(id)
                with open(image_list_file_path, 'a') as fw:
                    fw.write('{}/{}\n'.format(['train', 'validation',␣
 ↪'test'][j], id))
            line = f.readline()

        f.close()
```

```
./oidv6-train-annotations-bbox.csv
./validation-annotations-bbox.csv
./test-annotations-bbox.csv
```

[ ]:
```
DATA_ALL_DIR = os.path.join('.', 'data_all') # directory that contains all␣
 ↪downloaded data selected from the list
DATA_OUT_DIR = os.path.join('.', 'data_out') # directory that contains data␣
 ↪reorganized in YOLO format
os.makedirs(DATA_ALL_DIR)
os.makedirs(DATA_OUT_DIR)
```

[ ]:
```
!python downloader.py ./image_list_file.txt --download_folder=./data_all␣
 ↪--num_processes=5
```

```
Downloading images: 100% 7430/7430 [15:10<00:00,  8.16it/s]
```

[ ]:
```
for dir_ in ['images', 'labels']:
    for set_ in ['train', 'validation', 'test']:
        new_dir = os.path.join(DATA_OUT_DIR, dir_, set_)
        if os.path.exists(new_dir):
            shutil.rmtree(new_dir)
        os.makedirs(new_dir)
```

```
for j, filename in enumerate([train_bboxes_filename,␣
↪validation_bboxes_filename, test_bboxes_filename]):
    set_ = ['train', 'validation', 'test'][j]
    print(filename)
    with open(filename, 'r') as f:
        line = f.readline()
        while len(line) != 0:
            id, _, class_name, _, x1, x2, y1, y2, _, _, _, _, _ = line.
↪split(',')[:13]
            if class_name in list(class_ids.values()):

                if not os.path.exists(os.path.join(DATA_OUT_DIR, 'images',␣
↪set_, '{}.jpg'.format(id))):

                    shutil.copy(os.path.join(DATA_ALL_DIR, '{}.jpg'.format(id)),
                                os.path.join(DATA_OUT_DIR, 'images', set_, '{}.
↪jpg'.format(id)))

                with open(os.path.join(DATA_OUT_DIR, 'labels', set_, '{}.txt'.
↪format(id)), 'a') as f_ann:
                    # class_id, xc, yx, w, h
                    #
                    x1, x2, y1, y2 = [float(j) for j in [x1, x2, y1, y2]]
                    xc = (x1 + x2) / 2
                    yc = (y1 + y2) / 2
                    w = x2 - x1
                    h = y2 - y1

                    # class id = 0 if 'Ant' and 1 if 'Insect'
                    name = [k for k, v in class_ids.items() if v ==␣
↪class_name][0]
                    class_id = names.index(name)

                    #*****
                    f_ann.write('{} {} {} {} {}\n'.format(class_id, xc, yc, w,␣
↪h))
                    f_ann.close()

            line = f.readline()
```

```
./oidv6-train-annotations-bbox.csv
./validation-annotations-bbox.csv
./test-annotations-bbox.csv
```

[ ]:
```
from ultralytics import YOLO
```

```
os.environ["WANDB_MODE"] = "dryrun"

model = YOLO('yolov8n.pt')

results = model.train(data='config.yaml', epochs=5)

model.save('yolov8_trained.pt')
```

```
Ultralytics 8.3.17  Python-3.10.12 torch-2.4.1+cu121 CUDA:0 (Tesla T4,
15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8n.pt,
data=config.yaml, epochs=5, time=None, patience=100, batch=16, imgsz=640,
save=True, save_period=-1, cache=False, device=None, workers=8, project=None,
name=train4, exist_ok=False, pretrained=True, optimizer=auto, verbose=True,
seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=False,
close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False,
freeze=None, multi_scale=False, overlap_mask=True, mask_ratio=4, dropout=0.0,
val=True, split=val, save_json=False, save_hybrid=False, conf=None, iou=0.7,
max_det=300, half=False, dnn=False, plots=True, source=None, vid_stride=1,
stream_buffer=False, visualize=False, augment=False, agnostic_nms=False,
classes=None, retina_masks=False, embed=None, show=False, save_frames=False,
save_txt=False, save_conf=False, save_crop=False, show_labels=True,
show_conf=True, show_boxes=True, line_width=None, format=torchscript,
keras=False, optimize=False, int8=False, dynamic=False, simplify=True,
opset=None, workspace=4, nms=False, lr0=0.01, lrf=0.01, momentum=0.937,
weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1,
box=7.5, cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64,
hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5,
shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, bgr=0.0, mosaic=1.0,
mixup=0.0, copy_paste=0.0, copy_paste_mode=flip, auto_augment=randaugment,
erasing=0.4, crop_fraction=1.0, cfg=None, tracker=botsort.yaml,
save_dir=runs/detect/train4
Overriding model.yaml nc=80 with nc=2

                   from  n    params  module
arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv
[3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv
[16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f
[32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv
[32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f
[64, 64, 2, True]
  5                  -1  1     73984  ultralytics.nn.modules.conv.Conv
[64, 128, 3, 2]
```

```
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f
[128, 128, 2, True]
  7                  -1  1    295424  ultralytics.nn.modules.conv.Conv
[128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f
[256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF
[256, 256, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat
[1]
 12                  -1  1    148224  ultralytics.nn.modules.block.C2f
[384, 128, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat
[1]
 15                  -1  1     37248  ultralytics.nn.modules.block.C2f
[192, 64, 1]
 16                  -1  1     36992  ultralytics.nn.modules.conv.Conv
[64, 64, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat
[1]
 18                  -1  1    123648  ultralytics.nn.modules.block.C2f
[192, 128, 1]
 19                  -1  1    147712  ultralytics.nn.modules.conv.Conv
[128, 128, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat
[1]
 21                  -1  1    493056  ultralytics.nn.modules.block.C2f
[384, 256, 1]
 22        [15, 18, 21]  1    751702  ultralytics.nn.modules.head.Detect
[2, [64, 128, 256]]
Model summary: 225 layers, 3,011,238 parameters, 3,011,222 gradients, 8.2 GFLOPs

Transferred 319/355 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train4',
view at http://localhost:6006/

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLO11n…
Downloading
https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo11n.pt to
```

```
'yolo11n.pt'…

100%|     | 5.35M/5.35M [00:00<00:00, 212MB/s]

AMP: checks passed

train: Scanning /content/data_out/labels/train… 6461 images, 0
backgrounds, 0 corrupt: 100%|     | 6461/6461 [00:07<00:00, 857.75it/s]

train: New cache created: /content/data_out/labels/train.cache
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01,
blur_limit=(3, 7)), ToGray(p=0.01, num_output_channels=3,
method='weighted_average'), CLAHE(p=0.01, clip_limit=(1, 4.0),
tile_grid_size=(8, 8))

val: Scanning /content/data_out/labels/validation… 210 images, 0
backgrounds, 0 corrupt: 100%|     | 210/210 [00:00<00:00, 462.87it/s]

val: New cache created: /content/data_out/labels/validation.cache


Plotting labels to runs/detect/train4/labels.jpg…
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and
'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum'
automatically…
optimizer: AdamW(lr=0.001667, momentum=0.9) with parameter groups
57 weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
TensorBoard: model graph visualization added
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/detect/train4
Starting training for 5 epochs…

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size

        1/5      8.55G      1.326      2.077      1.566         50        640:
100%|     | 404/404 [03:28<00:00,  1.94it/s]
                 Class     Images  Instances      Box(P          R      mAP50
mAP50-95): 100%|     | 7/7 [00:04<00:00,  1.71it/s]

                   all        210        280      0.781       0.21      0.273
0.148



      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size

        2/5      2.82G      1.385      1.756      1.606         43        640:
100%|     | 404/404 [03:21<00:00,  2.00it/s]
                 Class     Images  Instances      Box(P          R      mAP50
mAP50-95): 100%|     | 7/7 [00:02<00:00,  2.49it/s]
```

```
                   all        210        280       0.34      0.322      0.282
0.137

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
        3/5      3.95G      1.347      1.615      1.585         49        640:
100%|      | 404/404 [03:09<00:00,  2.14it/s]
                 Class     Images  Instances      Box(P          R      mAP50
mAP50-95): 100%|      | 7/7 [00:03<00:00,  1.97it/s]
                   all        210        280       0.47      0.397       0.36
0.214



      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
        4/5      2.96G       1.28      1.496       1.53         34        640:
100%|      | 404/404 [03:15<00:00,  2.07it/s]
                 Class     Images  Instances      Box(P          R      mAP50
mAP50-95): 100%|      | 7/7 [00:05<00:00,  1.34it/s]
                   all        210        280      0.479      0.441      0.411
0.269



      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
        5/5      2.55G       1.19      1.342       1.46         37        640:
100%|      | 404/404 [03:08<00:00,  2.14it/s]
                 Class     Images  Instances      Box(P          R      mAP50
mAP50-95): 100%|      | 7/7 [00:02<00:00,  2.52it/s]
                   all        210        280      0.579      0.404      0.436
0.284



5 epochs completed in 0.281 hours.
Optimizer stripped from runs/detect/train4/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train4/weights/best.pt, 6.2MB

Validating runs/detect/train4/weights/best.pt…
Ultralytics 8.3.17  Python-3.10.12 torch-2.4.1+cu121 CUDA:0 (Tesla T4,
15102MiB)
Model summary (fused): 168 layers, 3,006,038 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50
mAP50-95): 100%|      | 7/7 [00:07<00:00,  1.00s/it]
```

```
                   all        210        280      0.579      0.404      0.436
0.286
                   Ant         12         30      0.406        0.2      0.164
0.105
                Insect        198        250      0.752      0.608      0.709
0.466
Speed: 0.3ms preprocess, 3.9ms inference, 0.0ms loss, 7.0ms postprocess per
image
Results saved to runs/detect/train4
```

```
VBox(children=(Label(value='0.000 MB of 0.000 MB uploaded\r'),␣
 ↪FloatProgress(value=1.0, max=1.0)))
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

The process of training was long. Pascal VOC dataset was causing errors. Understanding those errors took a lot of time. Yolov8 is training process is much easier with the help of ultralytics. I needed to only bring the dataset into yolov8 format.