

ECE60146 HW7

Yuchen Song

Part 1 Extract Multi-instance image from coco

1 3x3 grid image

The images from coco dataset are filtered by categories ['pizza', 'cat', 'bus'], and the foreground object are filtered to have more than 40000 pixels. The sample images are in Figure 1.

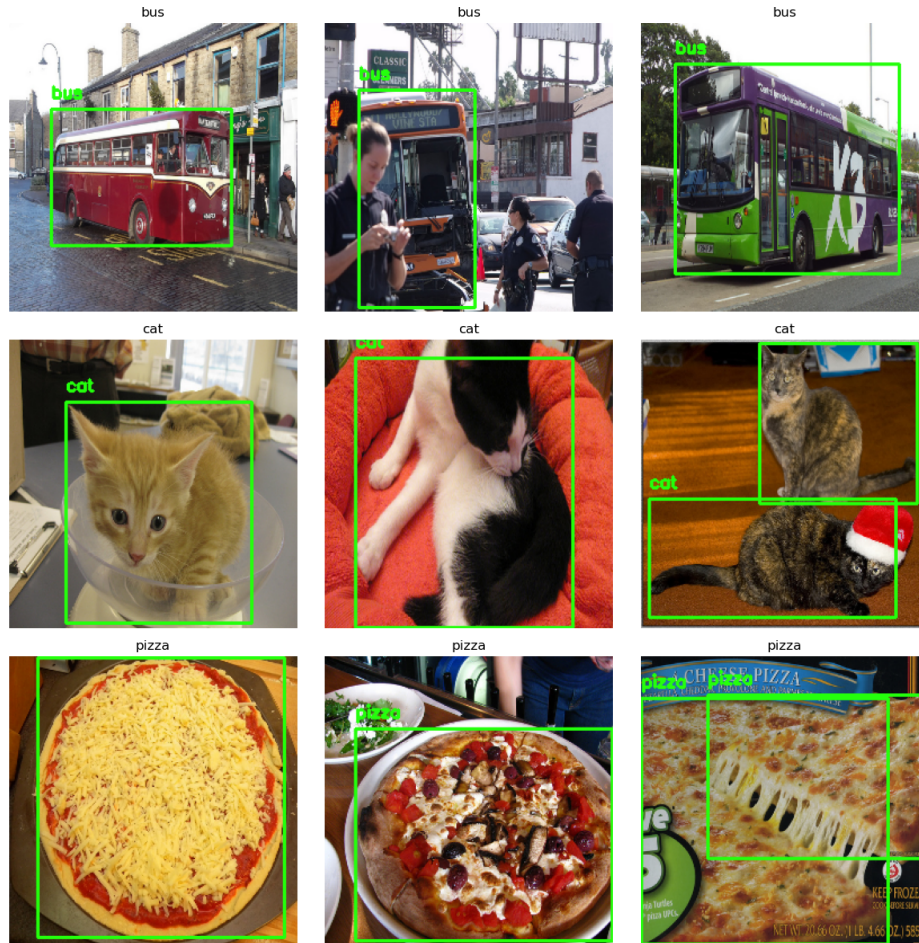


Figure 1: 3x3 grid image

Part 2 Dataloader

1 code block showing all parameters for yolo vector are generated

To generate the yolo vector explained in the lecture slides, a costumed yoloDataset class is being created to generate the image tensor, the yolo tensor and an additional augment yolo tensor, which refereed from the YOLOLogic, with a nil element to add extra loss for the yolo cell that does not have a object. The class is :

```
1  #-----YoloDataset
   ↳ class-----
2  class YoloDataset(torch.utils.data.Dataset):
3      def __init__(self, train_root, val_root, transform=None,
   ↳ train_or_val='train', image_size=256, S=8, B=5, C=3, classes=['bus',
   ↳ 'cat', 'pizza']):
4          self.transform = transform
5          self.train_or_val = train_or_val
6          self.train_root = train_root
7          self.val_root = val_root
8          self.image_size = image_size
9          self.B = B # number of anchor boxes
10         self.S = S # number of yolo cells on each axle
11         self.num_of_cells = S * S # number of yolo cells
12         self.yolo_interval = image_size / S
13         self.C = C # number of classes
14         self.classes = classes
15         if self.train_or_val == 'train':
16             self.database =
   ↳ torch.load("/home/syc/Desktop/ECE60146/coco_subset_train.pt")
17         else:
18             self.database =
   ↳ torch.load("/home/syc/Desktop/ECE60146/coco_subset_val.pt")
19         self.dataset_size = len(self.database)
20
21     def __len__(self):
22         return self.dataset_size
23
24     def convert_bbox_to_yolo_format(self, bbox):
25         x1, y1, x2, y2 = bbox
26         x_center = (x1 + x2) / 2
27         y_center = (y1 + y2) / 2
28         w = (x2 - x1)
29         h = (y2 - y1)
30         return [x_center, y_center, w, h]
31
32     def label_to_onehot(self, label):
33         onehot = [0,0,0]
```



```

34         onehot[label] = 1
35     return onehot
36
37     def __getitem__(self, idx):
38         image_path = self.database[idx]['image_path']
39         annotations = self.database[idx]['annotations']
40
41         im = Image.open(image_path).convert('RGB')
42         im = im.resize((self.image_size, self.image_size))
43         im_tensor = tvn.ToTensor()(im)
44
45         if self.transform:
46             im_tensor = self.transform(im_tensor)
47
48         yolo_tensor = torch.zeros(self.num_of_cells, self.B, 5 + self.C)
49         yolo_tensor_aug = torch.zeros(self.num_of_cells, self.B, 5 + self.C +
50             ↪ 1)
51         bbox_tensor = torch.zeros(5,4,dtype=torch.uint8)
52         bbox_label_tensor = torch.zeros(5,dtype=torch.uint8) + 13
53         num_objects = len(annotations)
54         for i in range(num_objects):
55             bbox = annotations[i]['bbox']
56             x1, y1, w, h = bbox
57             x2, y2 = x1 + w, y1 + h
58             bbox = [x1, y1, x2, y2]
59             label = annotations[i]['category_id']
60             bbox_tensor[i] = torch.LongTensor(bbox)
61             bbox_label_tensor[i] = label
62         anchor_boxes = [[1, 5],[1,3],[1,1],[3,1],[5,1]]
63         for annotation in annotations:
64             bbox = annotation['bbox']
65             class_id = annotation['category_id']
66             x1, y1, w, h = bbox
67             x2, y2 = x1 + w, y1 + h
68             bbox_converted = self.convert_bbox_to_yolo_format([x1, y1, x2,
69                 ↪ y2])
70             grid_x = int(bbox_converted[0] /self.yolo_interval)
71             grid_y = int(bbox_converted[1] /self.yolo_interval)
72             '''This anchor box selection logic is borrowed from
73             the DLStudio module
74             ↪ https://engineering.purdue.edu/kak/distYOLO/'''
75             AR = h / w
76             if AR <=0.2:
77                 anch_box_idx = 0
78             elif 0.2 < AR <=0.5:
79                 anch_box_idx = 1
80             elif 0.5 < AR <=1.5:

```

```

78         anch_box_idx = 2
79     elif 1.5 < AR <=4:
80         anch_box_idx = 3
81     elif 4 < AR:
82         anch_box_idx = 4
83     anch_box = anchor_boxes[anch_box_idx]
84
85     del_x = bbox_converted[0] - (grid_x * self.yolo_interval +
86         ↪ self.yolo_interval / 2)
87     del_x = del_x / self.yolo_interval
88     del_y = bbox_converted[1] - (grid_y * self.yolo_interval +
89         ↪ self.yolo_interval / 2)
90     del_y = del_y / self.yolo_interval
91
92     delta_w = w / (anch_box[0] * self.yolo_interval)
93     delta_h = h / (anch_box[1] * self.yolo_interval)
94     onehot = self.label_to_onehot(class_id)
95     yolo_vector = torch.FloatTensor([1, del_x, del_y, delta_w,
96         ↪ delta_h, *onehot])
97     yolo_tensor[grid_x * self.S + grid_y, anch_box_idx] = yolo_vector
98     yolo_tensor_aug[grid_x * self.S + grid_y, anch_box_idx, :-1] =
99         ↪ yolo_vector
100     cell_num = grid_x * self.S + grid_y
101     grid_x = int(cell_num // self.S)
102     grid_y = int(cell_num % self.S)
103     h = delta_h * self.yolo_interval * anch_box[1]
104     w = delta_w * self.yolo_interval * anch_box[0]
105     for icx in range(yolo_tensor_aug.shape[0]):
106         for iax in range(yolo_tensor_aug.shape[1]):
107             if yolo_tensor_aug[icx, iax, 0] == 0:
108                 yolo_tensor_aug[icx, iax, -1] = 1
109     return im_tensor, yolo_tensor, yolo_tensor_aug, bbox_tensor,
110         ↪ bbox_label_tensor

```

2 explanation showing all parameters for yolo vector are generated

The YOLO vector consists of 8 elements

$$[p_o, \delta x, \delta y, \delta w, \delta h, c_1, c_2, c_3]$$

and augmented YOLO vector has 9 elements.

$$[p_o, \delta x, \delta y, \delta w, \delta h, c_1, c_2, c_3, bg]$$

where: p_o , the objectness score, which indicates whether an object exists in the anchor box ($p_o = 1$ means an object is present); δx and δy , which are offsets for the bounding box center

relative to the anchor box center; δw and δh , which represent scaling factors for the width and height of the bounding box relative to the anchor box dimensions; c_1, c_2, c_3 , which are the one-hot encoded class probabilities; and bg , an additional indicator where $bg = 1$ means no object exists in the given anchor box.

To verify that the YOLO vectors are correctly generated, I implemented a check dataset function. This function takes the image tensor and the YOLO tensor and decodes the bounding box coordinates using the same method as during inference. The decoded bounding box should match the ground truth bounding box.

The bounding box decoding process follows:

$$x_{center} = (i + 0.5 + \delta x) yolo_interval, \quad y_{center} = (j + 0.5 + \delta y) yolo_interval$$

where i, j are the grid cell row and column indices and $yolo_interval$ is the cell size.

$$w = w_a \cdot \delta w, \quad h = h_a \cdot \delta h$$

where w_a, h_a are the anchor box dimensions.

$$x_1 = x_{center} - \frac{w}{2}, \quad y_1 = y_{center} - \frac{h}{2}$$

This ensures that the bounding boxes are properly computed. The following code was used to test the data loader for both training and validation datasets:

```

1  #-----Train and Test Data
   ↳ Loader-----
2  dataset = YoloDataset("/home/syc/Desktop/ECdelta_xE60146/data/train2014",
3                      "/home/syc/Desktop/ECE60146/data/val2014",
   ↳ train_or_val='train')
4  train_loader = torch.utils.data.DataLoader(dataset, batch_size=4,
   ↳ shuffle=True)
5  dataset_val = YoloDataset("/home/syc/Desktop/ECE60146/data/train2014",
6                          "/home/syc/Desktop/ECE60146/data/val2014",
   ↳ train_or_val='val')
7  test_loader = torch.utils.data.DataLoader(dataset_val, batch_size=1,
   ↳ shuffle=True)
8
9  #-----Check Data-----
10 def check_data(loader, class_list=['bus', 'cat', 'pizza']):
11     for batch in loader:
12         im, yolo_tensor, yolo_tensor_aug, bbox_tensor, bbox_label_tensor =
   ↳ batch
13         batch_size = im.shape[0]
14         fig, axes = plt.subplots(1, batch_size, figsize=(5 * batch_size, 5))
15         if batch_size == 1:
16             axes = [axes]
17         anchors = [[1, 5], [1, 3], [1, 1], [3, 1], [5, 1]] #predefined anchor
   ↳ boxes size

```

```

18     yolo_interval = 32
19     cell_per_axle = 8
20     for idx in range(batch_size):
21         ax = axes[idx]
22         ax.imshow(im[idx].permute(1, 2, 0).cpu().numpy())
23         num_cells = yolo_tensor.shape[1]      # 64 cells
24         num_anchors = yolo_tensor.shape[2]    # 5 anchors
25         for cell in range(num_cells):
26             grid_x = cell // cell_per_axle
27             grid_y = cell % cell_per_axle
28             for anch in range(num_anchors):
29                 if yolo_tensor[idx, cell, anch, 0] == 1:
30                     del_x = yolo_tensor[idx, cell, anch, 1].item()
31                     del_y = yolo_tensor[idx, cell, anch, 2].item()
32                     del_w = yolo_tensor[idx, cell, anch, 3].item()
33                     del_h = yolo_tensor[idx, cell, anch, 4].item()
34                     anchor_box = anchors[anch]
35                     x_center = (grid_x + 0.5 + del_x) * yolo_interval
36                     y_center = (grid_y + 0.5 + del_y) * yolo_interval
37                     w = del_w * yolo_interval * anchor_box[0]
38                     h = del_h * yolo_interval * anchor_box[1]
39                     x1 = x_center - w/2
40                     y1 = y_center - h/2
41                     rect = Rectangle((x1, y1), w, h, edgecolor='g',
42                                     ↪ facecolor='none', lw=2)
43                     ax.add_patch(rect)
44                     label = torch.argmax(yolo_tensor[idx, cell, anch,
45                                     ↪ 5:]).item()
46                     ax.text(x1, y1, class_list[label], color='g',
47                             ↪ fontsize=12)
48                 ax.axis("off")
49             plt.tight_layout()
50             plt.show()
51             break # Only process one batch
52
53 check_data(train_loader)
54 check_data(test_loader)

```

The plot of the check for the train loader is Figure 2.

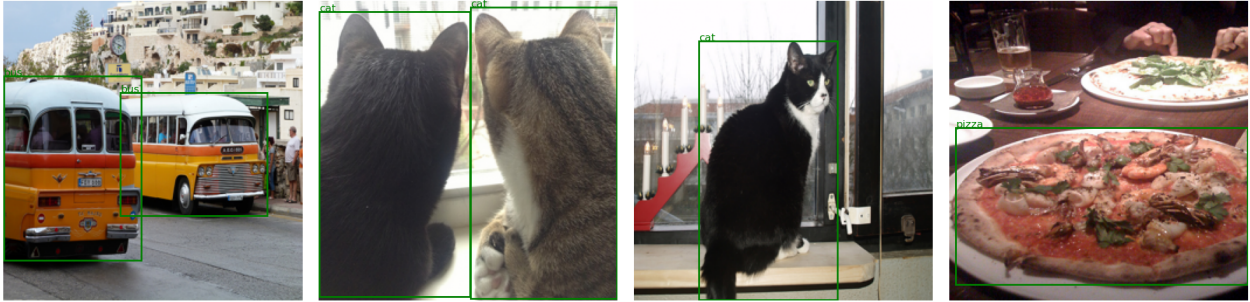


Figure 2: Train Loader validation

The plot of the check for the val loader is Figure 3.

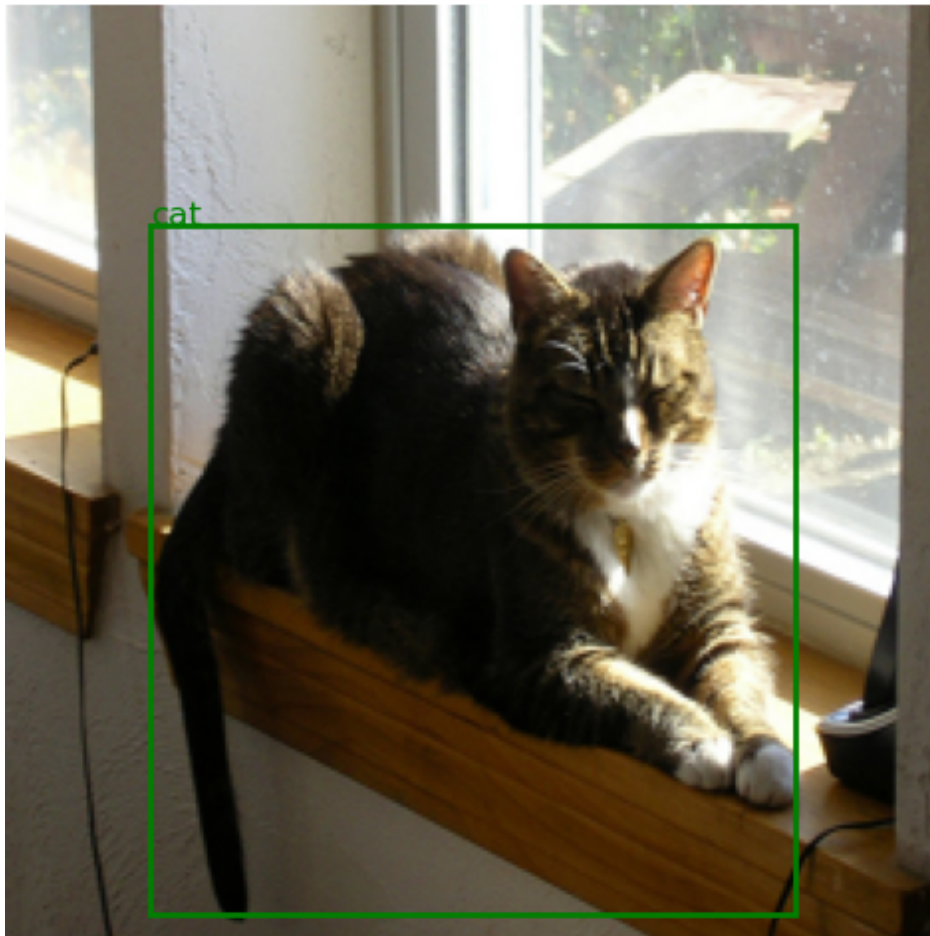


Figure 3: Val Loader validation

Part 3 Training

1 code block showing how yolo tensor is built

The yolo tensors are being generated in the Yolodataset, and pass to a dataloader, when do the training, it will iterate through the data loader and get the needed tensor and use three loss criterion for training

```
1 def run_code_for_training(net, train_loader, device="cuda", epochnum=20,
  ↳ save_path="./yolo_model.pth",plt_save_path="./training_loss.png"):
2     '''this function is borrowed from the DLStudio module
  ↳ https://engineering.purdue.edu/kak/distYOLO/,
3     the function is modified to fit the 256 image size and 3 classes, and 8
  ↳ yolo cells on each axle. the yolo vector is modified, the dw and dh
  ↳ are
4     ratio of the bbox to the anchor box size, instead of the yolo interval,
  ↳ the other code are mostly the same, the plotting code are removed'''
5     device = torch.device(device if torch.cuda.is_available() else "cpu")
6     net = net.to(device)
7     class_list = ['bus', 'cat', 'pizza']
8     # Define Loss Functions
9     criterion1 = nn.BCELoss(reduction='sum')
10    criterion2 = nn.MSELoss(reduction='sum')
11    criterion3 = nn.CrossEntropyLoss(reduction='sum')
12
13    optimizer = optim.Adam(net.parameters(), lr=1e-5)
14
15    print("\n Starting Training \n")
16    start_time = time.time()
17
18    loss_history = [] # Track loss over iterations
19    Loss_totally = []
20    BCE_loss = []
21    MSE_loss = []
22    CrossEntropy_loss = []
23
24    # Training Loop
25    for epoch in range(epochnum):
26        running_loss = 0.0
27        running_bce = 0.0
28        running_mse = 0.0
29        running_ce = 0.0
30
31        for iter,data in enumerate(train_loader):
32            im_tensor, yolo_tensor, yolo_tensor_aug, bbox_tensor,
  ↳ bbox_label_tensor = data
33            im_tensor, yolo_tensor, yolo_tensor_aug = im_tensor.to(device),
  ↳ yolo_tensor.to(device), yolo_tensor_aug.to(device)
```

```

34
35     optimizer.zero_grad() # Reset gradients
36
37     predictions = net(im_tensor)
38
39     num_yolo_cells = 64
40     num_anchor_boxes = 5
41     predictions = predictions.view(-1, num_yolo_cells,
42     ↪ num_anchor_boxes, 9)
43
44     # Loss Calculation
45     loss = torch.tensor(0.0, requires_grad=True).float().to(device)
46
47     # Object Presence Loss
48     loss_obj = criterion1(nn.Sigmoid()(predictions[:, :, :, 0]),
49     ↪ yolo_tensor_aug[:, :, :, 0])
50     loss += loss_obj
51
52     # Bounding Box Loss
53     loss_bbox = criterion2(predictions[:, :, :, 1:5],
54     ↪ yolo_tensor_aug[:, :, :, 1:5])
55     loss += loss_bbox
56
57     # Class Prediction Loss
58     targets = yolo_tensor_aug[:, :, :, 5:].view(-1, 4)
59     targets = torch.argmax(targets, dim=1)
60     probs = predictions[:, :, :, 5:].view(-1, 4)
61     loss_class = criterion3(probs, targets)
62     loss += loss_class
63
64     loss.backward()
65     optimizer.step()
66
67
68     running_loss += loss.item()
69     running_bce += loss_obj.item()
70     running_mse += loss_bbox.item()
71     running_ce += loss_class.item()
72
73     if iter % 300 == 299:
74         elapsed_time = time.time() - start_time
75         avg_loss = running_loss / 300
76         loss_history.append(avg_loss)
77         Loss_totoally.append(running_loss)
78         BCE_loss.append(running_bce/300)
79         MSE_loss.append(running_mse/300)
80         CrossEntropy_loss.append(running_ce/300)
81
82         print(f"\n[Epoch {epoch+1}/{epochnum}, Batch {iter+1}] ")

```

```

78         f"Elapsed Time: {int(elapsed_time)} sec | Loss:
          ↳ {avg_loss:.4f}")
79     print(f"   BCE Loss: {running_bce/300:.4f} | MSE Loss:
          ↳ {running_mse/300:.4f} | CrossEntropy Loss:
          ↳ {running_ce/300:.4f}")
80     running_loss = 0.0
81     running_bce = 0.0
82     running_mse = 0.0
83     running_ce = 0.0
84
85     # Save Model Every Epoch
86     torch.save(net.state_dict(), save_path)
87     print(f"\nModel saved at {save_path} (Epoch {epoch+1})")
88
89     # Training Finished
90     print("\nTraining Complete!\n")
91
92     # Plot Loss Curve
93     plt.figure(figsize=(10,5))
94     plt.plot(loss_history, label="Loss vs. Iterations")
95     plt.plot(BCE_loss, label="BCE Loss")
96     plt.plot(MSE_loss, label="MSE Loss")
97     plt.plot(CrossEntropy_loss, label="CrossEntropy Loss")
98     plt.title("Training Loss Curve")
99     plt.xlabel("Iterations (every 300 steps)")
100    plt.ylabel("Loss")
101    plt.legend()
102    plt.savefig(plt_save_path)
103    # plt.show()
104
105    return net
106

```

2 Explanation of How YOLO Tensor is Built

The YOLO tensor is constructed within the YoloDataset class, which processes raw image data and annotations into a structured format suitable for training the YOLO model. The output augment yolo tensor used for training follows the format:

$$B \times C \times A \times 9$$

where:

- B is the batch size,
- C is the number of YOLO grid cells,

- A is the number of anchor boxes per cell,
- The 9 elements per anchor box encode object information explained in previous yolo vector section.

Each image's ground-truth bounding boxes are mapped into the corresponding YOLO grid cell.

The YOLO tensor is used with loss functions for object presence, bounding box regression, and class prediction:

- Binary Cross-Entropy (BCE) loss for object presence.
- Mean Squared Error (MSE) loss for bounding box regression.
- Cross-Entropy loss for class predictions.

These losses are then backpropagated to update the model parameters.

3 BCE,CE,MSE loss curves

After the 20 epoches training, the average loss of each 300 iteration for all three loss and total is shown in Figure 4.

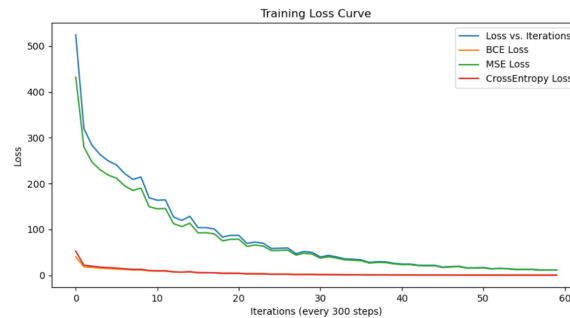


Figure 4: Loss Curve

Part 4 Evaluation

1 Code block translate yolo tensor to BB pred and class label

```

1 anchor_boxes = [[1, 5], [1, 3], [1, 1], [3, 1], [5, 1]]
2 num_of_validation_images = len(test_loader)
3 print("num of test_loader", len(test_loader))
4 with torch.no_grad():
5     for iter, data in enumerate(test_loader):
6         im_tensor, yolo_tensor, yolo_tensor_aug, bbox_tensor,
          ↳ bbox_label_tensor = data
7         im_tensor, yolo_tensor, yolo_tensor_aug = im_tensor.to(device),
          ↳ yolo_tensor.to(device), yolo_tensor_aug.to(device)

```

```

8         predictions = net(im_tensor)
9         predictions = predictions.view(-1, num_yolo_cells,
    ↪     num_anchor_boxes, 9)
10        # print("predictions", predictions.shape)
11        for ibx in range(predictions.shape[0]):
12            icx_2_best_anchor = {ic: None for ic in
    ↪     range(predictions.shape[1])}
13            for icx in range(predictions.shape[1]):
14                cell_pred = predictions[ibx, icx]
15                previous_best = 0
16                for anchor in range(cell_pred.shape[0]):
17                    if cell_pred[anchor][0] > previous_best:
18                        previous_best = anchor
19                best_anchor_icx = previous_best
20                icx_2_best_anchor[icx] = best_anchor_icx
21            sorted_icx = sorted(icx_2_best_anchor, key=lambda x:
    ↪     predictions[ibx, x, icx_2_best_anchor[x]][0],
    ↪     reverse=True)
22
23            retained_cells = sorted_icx[:5]
24            objects_detected = []
25            predicted_bboxes = []
26            predicted_labels_for_bboxes = []
27            predicted_label_index_vals = []
28            for icx in retained_cells:
29                pred_vec = predictions[ibx, icx, icx_2_best_anchor[icx]]
30                class_label_prdict = pred_vec[-4:]
31                class_label_prob = torch.nn.Softmax(dim=0)(class_label_prdict)
32                class_label_prob = class_label_prob[:-1]
33                if torch.all(class_label_prob < 0.2):
34                    predicted_class_label = None
35                else:
36                    best_predicted_class_index = (class_label_prob ==
    ↪     class_label_prob.max())
37                    best_predicted_class_index =
    ↪     torch.nonzero(best_predicted_class_index,
    ↪     as_tuple=True)
38
    ↪     predicted_label_index_vals.append(best_predicted_class_index[0].item())
39            predicted_class_label =
    ↪     class_list[best_predicted_class_index[0].item()]
40            predicted_labels_for_bboxes.append(predicted_class_label)
41
42            pred_regression_vec = pred_vec[1:5].cpu()
43            del_x, del_y, delta_w, delta_h = pred_regression_vec
44            grid_x = int(icx // 8)
45            grid_y = int(icx % 8)

```

```

46         bb_center_x = (grid_x + 0.5 + del_x) * yolo_interval
47         bb_center_y = (grid_y + 0.5 + del_y) * yolo_interval
48         w = delta_w * yolo_interval *
         ↪ anchor_boxes[icx_2_best_anchor[icx]][0]
49         h = delta_h * yolo_interval *
         ↪ anchor_boxes[icx_2_best_anchor[icx]][1]
50         bb_top_left_x = bb_center_x - w/2
51         bb_top_left_y = bb_center_y - h/2
52         predicted_bboxes.append([bb_top_left_x, bb_top_left_y,
         ↪ int(w), int(h)])
53         # print("predicted_bboxes", predicted_bboxes)
54     saved_predicted_bboxes = [predicted_bboxes[i][:] for i in
         ↪ range(len(predicted_bboxes))]
55     gt_bboxes = torch.squeeze(bbox_tensor).cpu().numpy()

```

2 Explanation Translate yolo tensor to BB pred and class label

From the prediction, the dimension of the prediction is

$$B \times C \times A \times 9$$

where:

- B is the batch size,
- C is the number of YOLO grid cells,
- A is the number of anchor boxes per cell,
- The 9 elements per anchor box encode object information predict by the model.

In the code block, it loops through the batches, cells, and anchor boxes to extract the 9-element YOLO vectors. A threshold is applied to select the cells with the highest probability of containing an object. The YOLO vectors values that satisfy the threshold are then used to decode the predicted bounding box.

Since the cell index provides the center of the anchor box, it is used to get the row and column index of the yolo cell i and j , and then use the indexes to compute the center of the predicted bounding box. The second and third elements in the tensor (δx and δy) are used to refine the bounding box center:

$$x_{center} = (i + 0.5 + \delta x) yolo_interval, \quad y_{center} = (j + 0.5 + \delta y) yolo_interval$$

The fourth and fifth elements (δw and δh) along with the anchor box dimensions (w_a, h_a) are used to compute the bounding box width and height:

$$w = w_a \cdot \delta w, \quad h = h_a \cdot \delta h$$

Finally, the top-left corner of the bounding box is computed as:

$$x_1 = x_{center} - \frac{w}{2}, \quad y_1 = y_{center} - \frac{h}{2}$$

This method converts the YOLO tensor predictions into bounding box coordinates for visualization and IOU calculation

3 24 images

After training the model, the testing phase identifies the best predicted bounding boxes for evaluation. To analyze the performance, the plots has:

- 18 well-predicted bounding boxes (6 per class: *bus*, *cat*, *pizza*), showing good detection quality.
- 6 examples of poor predictions (6 per class: *bus*, *cat*, *pizza*), illustrating cases where the model struggles with accurate localization.

In the plots:

- Green bounding boxes represent the ground truth (GT) annotations extracted from the YOLO tensor.
- Red bounding boxes represent the model's predicted bounding boxes obtained from the YOLO output tensor.

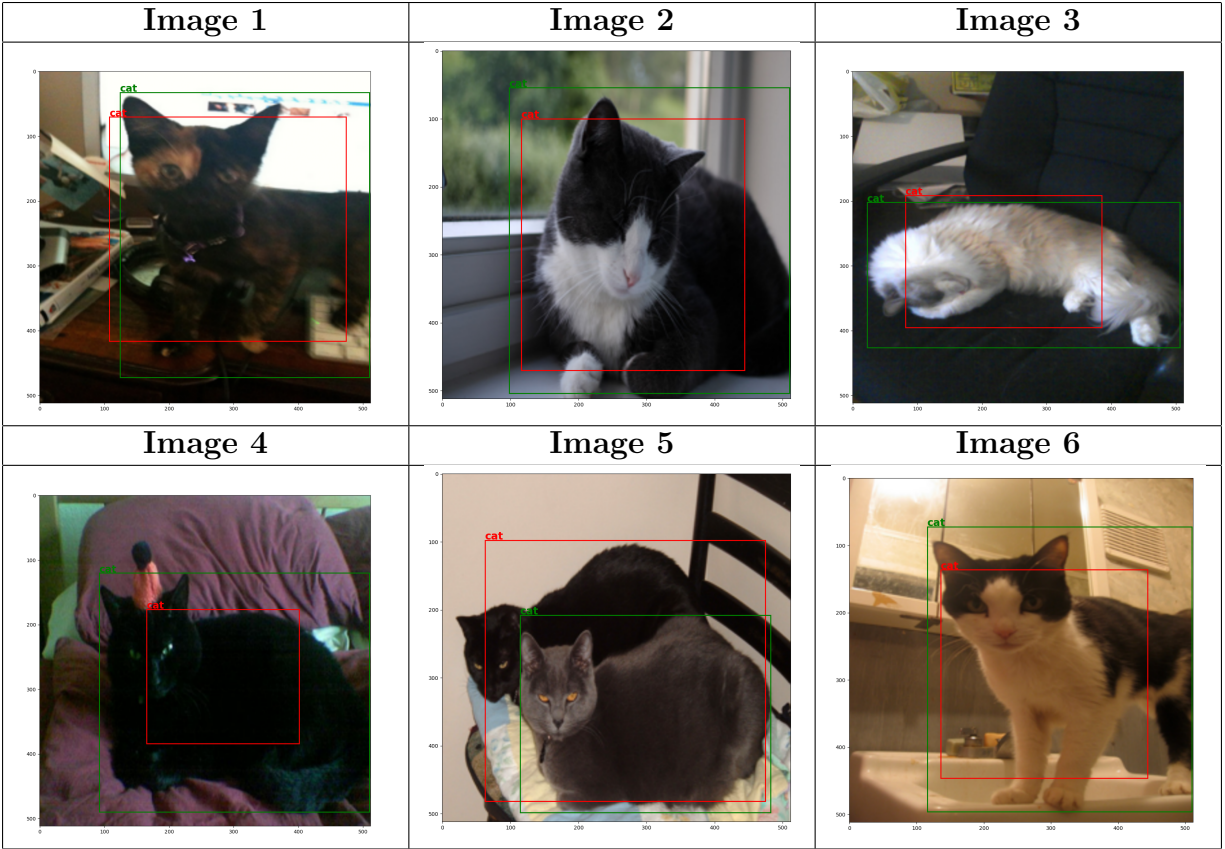


Table 1: Good cat prediction images

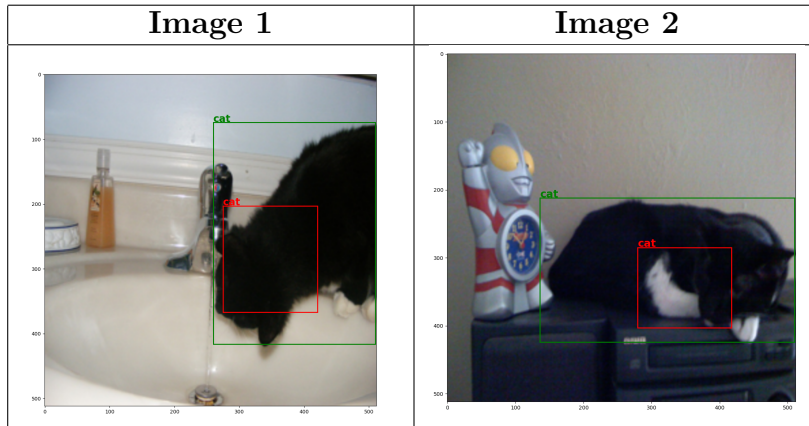


Table 2: Bad cat prediction images

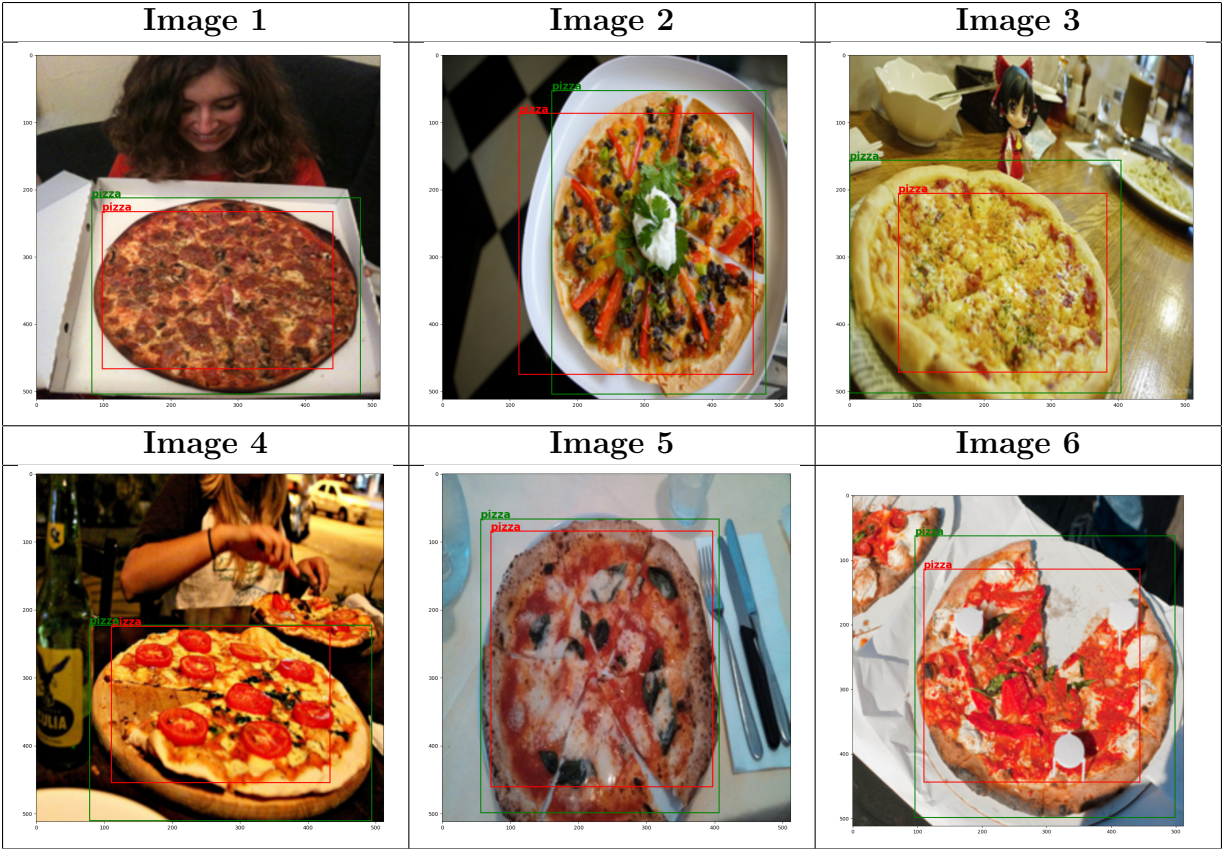


Table 3: Good pizza prediction images

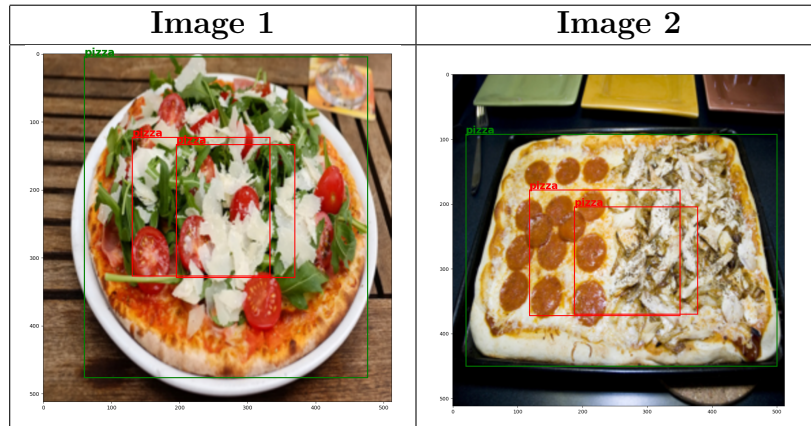


Table 4: Bad pizza prediction images

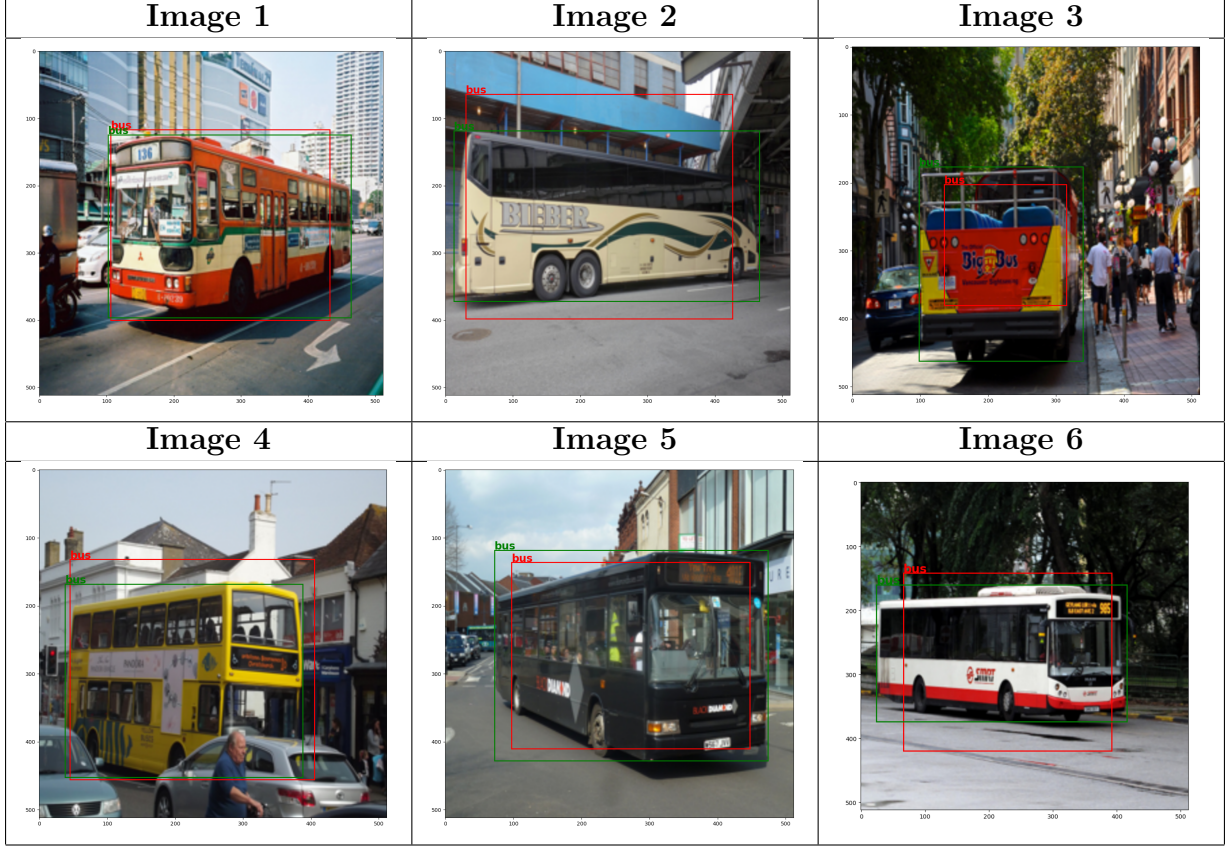


Table 5: Good bus prediction images

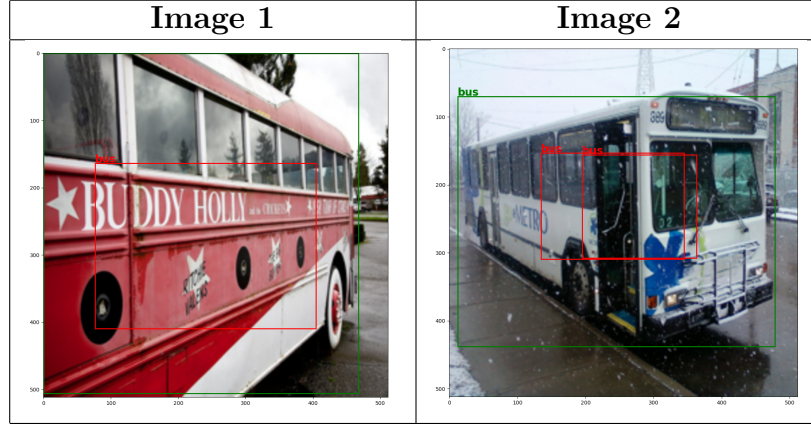


Table 6: Bad bus prediction images

In the code, I tried several different network, including the YOLOnet in YOLOLogic module with different depth, as well as the Resnet with ResNet50 model, the results are different, in the above plots, the results are from the ResNet.

In the good detection cases, the model successfully identifies and classifies the objects, with a significant overlap between the predicted and ground truth bounding boxes. The

predicted class labels match the ground truth labels with high confidence. Although the bounding box center and size are close the ground truth, slight misalignment in position and dimensions are still observed.

In the bad detection cases, the model struggles to localize objects accurately. The predicted bounding box is significantly smaller than the ground truth. Multiple predicted boxes are generated for the same object that have similar position and size. The predicted center is off from the ground truth sometime, leading to incorrect localization.

Part 5 code

1 code for filtering data

```
1 from pycocotools.coco import COCO
2 import numpy as np
3 import skimage.io as io
4 import skimage.color
5 import cv2
6 import torch
7 import os
8
9 # Define paths
10 data_dir = "/home/syc/Desktop/ECE60146"
11 train_json = os.path.join(data_dir, "annotations/instances_train2014.json")
12 val_json = os.path.join(data_dir, "annotations/instances_val2014.json")
13 train_image_dir = os.path.join(data_dir, "train2014")
14 val_image_dir = os.path.join(data_dir, "val2014")
15 train_output_dataset_file = os.path.join(data_dir, "coco_subset_train.pt")
16 val_output_dataset_file = os.path.join(data_dir, "coco_subset_val.pt")
17
18 class_list = ['bus', 'cat', 'pizza']
19 image_size = 256
20
21 coco_train = COCO(train_json)
22 coco_val = COCO(val_json)
23
24 catIds = coco_train.getCatIds(catNms=class_list)
25 categories = coco_train.loadCats(catIds)
26 categories.sort(key=lambda x: x['id'])
27 coco_labels_inverse = {c['id']: idx for idx, c in enumerate(categories)}
28
29 train_imgIds = []
30 val_imgIds = []
31
32 for cat_id in catIds:
33     train_imgIds.extend(coco_train.getImgIds(catIds=[cat_id]))
34     val_imgIds.extend(coco_val.getImgIds(catIds=[cat_id]))
35
36 train_imgIds = list(set(train_imgIds))
37 val_imgIds = list(set(val_imgIds))
38
39 def process_dataset(image_ids, coco, image_dir, max_images):
40     dataset = {}
41     for img_id in image_ids:
42         img_info = coco.loadImgs(img_id)[0]
```

```

43     image_path = os.path.join(image_dir, img_info['file_name'])
44
45     I = io.imread(image_path)
46     if len(I.shape) == 2: #convert to RGB
47         I = skimage.color.gray2rgb(I)
48
49     # Get annotations
50     annIds = coco.getAnnIds(imgIds=img_id, catIds=catIds, iscrowd=False)
51     anns = coco.loadAnns(annIds)
52     valid_annotations = []
53     valid_bbox = []
54     valid_labels = []
55     for ann in anns:
56         if ann['area'] >= 40000: # filter by area
57             x, y, w, h = ann['bbox']
58             label = coco_labels_inverse[ann['category_id']]
59             valid_bbox.append([x, y, w, h])
60             valid_labels.append(label)
61             valid_annotations.append({'bbox': [x, y, w, h], 'category_id':
62                                     ↪ label})
63
64     # print("valid_annotations", len(valid_annotations))
65     if len(valid_annotations) < 1:
66         continue
67     # resize image and adjust bounding boxes
68     orig_h, orig_w, _ = I.shape
69     scale_x = image_size / orig_w
70     scale_y = image_size / orig_h
71     I_resized = cv2.resize(I, (image_size, image_size))
72     adjusted_annotations = []
73     adjusted_bbox = []
74     for x, y, w, h in valid_bbox:
75         new_x = int(x * scale_x)
76         new_y = int(y * scale_y)
77         new_w = int(w * scale_x)
78         new_h = int(h * scale_y)
79         adjusted_annotations.append({'bbox': [new_x, new_y, new_w, new_h],
80                                     ↪ 'category_id': label, "true_class": class_list[label]})
81         adjusted_bbox.append([new_x, new_y, new_w, new_h])
82     dataset[len(dataset)] = {
83         "image_path": image_path,
84         "annotations": adjusted_annotations
85     }
86
87     if len(dataset) >= max_images:
88         break

```



```

88     return dataset
89
90     # Train Dataset (Max 4000 Images)
91     filtered_dataset_train = process_dataset(train_imgIds, coco_train,
92     ↪     train_image_dir, max_images=4000)
93     torch.save(filtered_dataset_train, train_output_dataset_file)
94     print(f"Training subset saved to {train_output_dataset_file} with
95     ↪     {len(filtered_dataset_train)} images.")
96
97     # Validation Dataset (Max 2000 Images)
98     filtered_dataset_val = process_dataset(val_imgIds, coco_val, val_image_dir,
99     ↪     max_images=5000)
100    torch.save(filtered_dataset_val, val_output_dataset_file)
101    print(f"Validation subset saved to {val_output_dataset_file} with
102    ↪     {len(filtered_dataset_val)} images.")

```

2 code for plot dataset with 3x3 grid

```

1  import torch
2  import matplotlib.pyplot as plt
3  import cv2
4  import random
5  import numpy as np
6  import skimage.io as io
7  import os
8
9  data_dir = "/home/syc/Desktop/ECE60146"
10 output_dataset_file = os.path.join(data_dir, "coco_subset_train.pt")
11 image_dir = os.path.join(data_dir, "train2014")
12
13 subset = torch.load(output_dataset_file)
14 num_images = len(subset)
15 print(f"Loaded {num_images} images")
16
17 class_list = ['bus', 'cat', 'pizza']
18
19 fig, axes = plt.subplots(3, 3, figsize=(12, 12))
20
21 class_images = {class_name: [] for class_name in class_list}
22
23 for key, data in subset.items():
24     for ann in data["annotations"]:
25         label_idx = ann["category_id"]
26         class_name = class_list[label_idx]

```

```

27         class_images[class_name].append((key, data))
28
29     for row, class_name in enumerate(class_list):
30         if len(class_images[class_name]) == 0:
31             print(f"No images found for class {class_name}")
32             continue
33
34         selected_images = random.sample(class_images[class_name], min(3,
35         ↪ len(class_images[class_name]))) # pick 3 images
36
37         for col, (key, data) in enumerate(selected_images):
38             image_path = data["image_path"]
39             annotations = data["annotations"]
40
41             image = io.imread(image_path)
42             image_resized = cv2.resize(image, (256, 256))
43
44             for ann in annotations:
45                 x, y, w, h = ann["bbox"]
46                 label_idx = ann["category_id"]
47
48                 image_resized = cv2.rectangle(image_resized, (int(x), int(y)),
49                 ↪ (int(x + w), int(y + h)), (36, 255, 12), 2)
50                 image_resized = cv2.putText(image_resized, class_list[label_idx],
51                 ↪ (int(x), int(y - 10)),
52                 ↪ cv2.FONT_HERSHEY_SIMPLEX, 0.5, (36,
53                 ↪ 255, 12), 2)
54
55             ax = axes[row, col]
56             ax.imshow(image_resized)
57             ax.set_title(f"{class_name}")
58             ax.set_axis_off()
59
60 plt.tight_layout()
61 plt.show()

```

3 code for dataset, dataloader, net, training and testing

```

1 import random
2 import numpy
3 import torch
4 import os, sys
5 import torchvision.models as models
6

```

```

7 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
↪ '/home/syc/Desktop/ECE60146/HW7/YOLOLogic-2.1.4')))
8 sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
↪ '/home/syc/Desktop/ECE60146/DLStudio-2.5.1')))
9 from YOLOLogic import *
10 from DLStudio import *
11 import torch
12 import torch.optim as optim
13 import torch.nn as nn
14 import time
15 import matplotlib.pyplot as plt
16 from matplotlib.patches import Rectangle
17
18 # seed = 0
19 # random.seed(seed)
20 # torch.manual_seed(seed)
21 # torch.cuda.manual_seed(seed)
22 # numpy.random.seed(seed)
23 # torch.backends.cudnn.deterministic=True
24 # torch.backends.cudnn.benchmark=False
25 # os.environ['PYTHONHASHSEED'] = str(seed)
26 # from YOLOLogic import *
27 import torch
28 import torch.nn as nn
29 import torchvision
30 import torchvision.transforms as tvf
31 import torchvision.transforms.functional as F
32 import torchvision.utils as tvutils
33 import torch.optim as optim
34 import numpy as np
35 import time
36
37 from PIL import Image
38 from PIL import ImageDraw
39 from PIL import ImageTk
40 from PIL import ImageFont
41 import sys,os,os.path,glob,signal
42 import re
43 import functools
44 import math
45 import random
46 import copy
47 import pickle
48 if sys.version_info[0] == 3:
49     import tkinter as Tkinter
50     from tkinter.constants import *
51 else:

```

```

52     import Tkinter
53     from Tkconstants import *
54
55     import matplotlib.pyplot as plt
56     from matplotlib.patches import Rectangle
57     import logging
58     import torch
59     import torchvision.transforms as tvf
60     from PIL import Image
61     import numpy as np
62
63
64
65     #-----YoloDataset
66     ↪ class-----
67     class YoloDataset(torch.utils.data.Dataset):
68         def __init__(self, train_root, val_root, transform=None,
69             ↪ train_or_val='train', image_size=256, S=8, B=5, C=3, classes=['bus',
70             ↪ 'cat', 'pizza']):
71             self.transform = transform
72             self.train_or_val = train_or_val
73             self.train_root = train_root
74             self.val_root = val_root
75             self.image_size = image_size
76             self.B = B # number of anchor boxes
77             self.S = S # number of yolo cells on each axle
78             self.num_of_cells = S * S # number of yolo cells
79             self.yolo_interval = image_size / S
80             self.C = C # number of classes
81             self.classes = classes
82             self.grid_size = self.image_size / S
83             if self.train_or_val == 'train':
84                 self.database =
85                 ↪ torch.load("/home/syc/Desktop/ECE60146/coco_subset_train.pt")
86             else:
87                 self.database =
88                 ↪ torch.load("/home/syc/Desktop/ECE60146/coco_subset_val.pt")
89             self.dataset_size = len(self.database)
90
91     def __len__(self):
92         return self.dataset_size
93
94     def convert_bbox_to_yolo_format(self, bbox):
95         x1, y1, x2, y2 = bbox
96         x_center = (x1 + x2) / 2
97         y_center = (y1 + y2) / 2
98         w = (x2 - x1)

```

```

94         h = (y2 - y1)
95         return [x_center, y_center, w, h]
96
97     def label_to_onehot(self, label):
98         onehot = [0,0,0]
99         onehot[label] = 1
100         return onehot
101
102     def __getitem__(self, idx):
103         image_path = self.database[idx]['image_path']
104         annotations = self.database[idx]['annotations']
105
106         im = Image.open(image_path).convert('RGB')
107         im = im.resize((self.image_size, self.image_size))
108         im_tensor = tvn.ToTensor()(im)
109
110         if self.transform:
111             im_tensor = self.transform(im_tensor)
112
113         yolo_tensor = torch.zeros(self.num_of_cells, self.B, 5 + self.C)
114         yolo_tensor_aug = torch.zeros(self.num_of_cells, self.B, 5 + self.C +
115 ↪ 1)
116         bbox_tensor = torch.zeros(5,4,dtype=torch.uint8)
117         bbox_label_tensor = torch.zeros(5,dtype=torch.uint8) + 13
118         num_objects = len(annotations)
119         for i in range(num_objects):
120             bbox = annotations[i]['bbox']
121             x1, y1, w, h = bbox
122             x2, y2 = x1 + w, y1 + h
123             bbox = [x1, y1, x2, y2]
124             label = annotations[i]['category_id']
125             bbox_tensor[i] = torch.LongTensor(bbox)
126             bbox_label_tensor[i] = label
127         anchor_boxes = [[1, 5],[1,3],[1,1],[3,1],[5,1]]
128         for annotation in annotations:
129             bbox = annotation['bbox']
130             class_id = annotation['category_id']
131             x1, y1, w, h = bbox
132             x2, y2 = x1 + w, y1 + h
133             bbox_converted = self.convert_bbox_to_yolo_format([x1, y1, x2,
134 ↪ y2])
135             grid_x = int(bbox_converted[0] /self.grid_size)
136             grid_y = int(bbox_converted[1] /self.grid_size)
137             '''This anchor box selection logic is borrowed from
the DLStudio module
↪ https://engineering.purdue.edu/kak/distYOLO/'''
AR = h / w

```

```

138         if AR <=0.2:
139             anch_box_idx = 0
140         elif 0.2 < AR <=0.5:
141             anch_box_idx = 1
142         elif 0.5 < AR <=1.5:
143             anch_box_idx = 2
144         elif 1.5 < AR <=4:
145             anch_box_idx = 3
146         elif 4 < AR:
147             anch_box_idx = 4
148         anch_box = anchor_boxes[anch_box_idx]
149
150         del_x = bbox_converted[0] - (grid_x * self.grid_size +
151             ↪ self.grid_size / 2)
152         del_x = del_x / self.grid_size
153         del_y = bbox_converted[1] - (grid_y * self.grid_size +
154             ↪ self.grid_size / 2)
155         del_y = del_y / self.grid_size
156
157         delta_w = w / (anch_box[0] * self.yolo_interval)
158         delta_h = h / (anch_box[1] * self.yolo_interval)
159         onehot = self.label_to_onehot(class_id)
160         yolo_vector = torch.FloatTensor([1, del_x, del_y, delta_w,
161             ↪ delta_h, *onehot])
162         yolo_tensor[grid_x * self.S + grid_y, anch_box_idx] = yolo_vector
163         yolo_tensor_aug[grid_x * self.S + grid_y, anch_box_idx, :-1] =
164             ↪ yolo_vector
165         cell_num = grid_x * self.S + grid_y
166         grid_x = int(cell_num // self.S)
167         grid_y = int(cell_num %self.S)
168         center_x = grid_x * self.grid_size + self.grid_size / 2 + del_x *
169             ↪ self.grid_size
170         center_y = grid_y * self.grid_size + self.grid_size / 2 + del_y *
171             ↪ self.grid_size
172         h = delta_h * self.yolo_interval * anch_box[1]
173         w = delta_w * self.yolo_interval * anch_box[0]
174         for icx in range(yolo_tensor_aug.shape[0]):
175             for iax in range(yolo_tensor_aug.shape[1]):
176                 if yolo_tensor_aug[icx, iax, 0] == 0:
177                     yolo_tensor_aug[icx, iax, -1] = 1
178         return im_tensor, yolo_tensor, yolo_tensor_aug, bbox_tensor,
179             ↪ bbox_label_tensor
180
181 #-----Train and Test Data
182 ↪ Loader-----
183 dataset = YoloDataset("/home/syc/Desktop/ECdelta_xE60146/data/train2014",
184     "/home/syc/Desktop/ECE60146/data/val2014",
185     ↪ train_or_val='train')

```



```

177 train_loader = torch.utils.data.DataLoader(dataset, batch_size=4,
    ↪ shuffle=True)
178 dataset_val = YoloDataset("/home/syc/Desktop/ECE60146/data/train2014",
179                             "/home/syc/Desktop/ECE60146/data/val2014",
    ↪ train_or_val='val')
180 test_loader = torch.utils.data.DataLoader(dataset_val, batch_size=1,
    ↪ shuffle=True)
181
182 #-----Check Data-----
183 def check_data(loader, class_list=['bus', 'cat', 'pizza']):
184     for batch in loader:
185         im, yolo_tensor, yolo_tensor_aug, bbox_tensor, bbox_label_tensor =
    ↪ batch
186         batch_size = im.shape[0]
187         fig, axes = plt.subplots(1, batch_size, figsize=(5 * batch_size, 5))
188         if batch_size == 1:
189             axes = [axes]
190         anchors = [[1, 5], [1, 3], [1, 1], [3, 1], [5, 1]] #predefined anchor
    ↪ boxes size
191         grid_size1 = 32
192         cell_per_axle = 8
193         for idx in range(batch_size):
194             ax = axes[idx]
195             ax.imshow(im[idx].permute(1, 2, 0).cpu().numpy())
196             num_cells = yolo_tensor.shape[1] # 64 cells
197             num_anchors = yolo_tensor.shape[2] # 5 anchors
198             for cell in range(num_cells):
199                 grid_x = cell // cell_per_axle
200                 grid_y = cell % cell_per_axle
201                 for anch in range(num_anchors):
202                     if yolo_tensor[idx, cell, anch, 0] == 1:
203                         del_x = yolo_tensor[idx, cell, anch, 1].item()
204                         del_y = yolo_tensor[idx, cell, anch, 2].item()
205                         del_w = yolo_tensor[idx, cell, anch, 3].item()
206                         del_h = yolo_tensor[idx, cell, anch, 4].item()
207                         anchor_box = anchors[anch]
208                         x_center = (grid_x + 0.5 + del_x) * grid_size1
209                         y_center = (grid_y + 0.5 + del_y) * grid_size1
210                         w = del_w * grid_size1 * anchor_box[0]
211                         h = del_h * grid_size1 * anchor_box[1]
212                         x1 = x_center - w/2
213                         y1 = y_center - h/2
214                         rect = Rectangle((x1, y1), w, h, edgecolor='g',
    ↪ facecolor='none', lw=2)
215                         ax.add_patch(rect)
216                         label = torch.argmax(yolo_tensor[idx, cell, anch,
    ↪ 5:]).item()

```

```

217         ax.text(x1, y1, class_list[label], color='g',
                ↪     fontsize=12)
218         ax.axis("off")
219     plt.tight_layout()
220     plt.show()
221     break # Only process one batch
222
223 check_data(train_loader)
224 check_data(test_loader)
225
226 #-----SkipBlock
227 ↪ class-----
228 '''This skip block class is borrowed from the DLStudio module
229 ↪ https://engineering.purdue.edu/kak/distYOLO/'''
230 class SkipBlock(nn.Module):
231     def __init__(self, in_ch, out_ch, downsample=False,
232     ↪ skip_connections=True):
233         super(SkipBlock, self).__init__()
234         self.downsample = downsample
235         self.skip_connections = skip_connections
236         self.in_ch = in_ch
237         self.out_ch = out_ch
238         self.conv1 = nn.Conv2d(in_ch, in_ch, 3, stride=1, padding=1)
239         self.conv2 = nn.Conv2d(in_ch, out_ch, 3, stride=1, padding=1)
240         self.bn1 = nn.BatchNorm2d(in_ch)
241         self.bn2 = nn.BatchNorm2d(out_ch)
242         self.in2out = nn.Conv2d(in_ch, out_ch, 1)
243         if downsample:
244             self.downsampler1 = nn.Conv2d(in_ch, in_ch, 1, stride=2)
245             self.downsampler2 = nn.Conv2d(out_ch, out_ch, 1, stride=2)
246
247     def forward(self, x):
248         identity = x
249         out = self.conv1(x)
250         out = self.bn1(out)
251         out = nn.functional.relu(out)
252         out = self.conv2(out)
253         out = self.bn2(out)
254         out = nn.functional.relu(out)
255         if self.downsample:
256             identity = self.downsampler1(identity)
257             out = self.downsampler2(out)
258         if self.skip_connections:
259             if (self.in_ch == self.out_ch) and (self.downsample is False):
260                 out = out + identity
261             elif (self.in_ch != self.out_ch) and (self.downsample is False):
262                 identity = self.in2out( identity )

```

```

260         out = out + identity
261         elif (self.in_ch != self.out_ch) and (self.downsample is True):
262             out = out + torch.cat((identity, identity), dim=1)
263     return out
264
265
266     ##-----NetForYolo
267     ↪ class-----
268     '''This class is borrowed from the DLStudio module
269     ↪ https://engineering.purdue.edu/kak/distYOLO/'''
270     class NetForYolo(nn.Module):
271         '''this class is borrowed from the DLStudio module
272         ↪ https://engineering.purdue.edu/kak/distYOLO/,
273         the network is modified to fit the 256 image size and 3 classes, and 8
274         ↪ yolo cells'''
275         def __init__(self, skip_connections=True, depth=8):
276             super(NetForYolo, self).__init__()
277             # if depth not in [8, 10, 12, 14, 16]:
278             #     raise ValueError("Depth must be one of 8, 10, 12, 14, or 16")
279             self.depth = depth // 2
280             self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
281             self.conv2 = nn.Conv2d(64, 64, 3, padding=1)
282             self.pool = nn.MaxPool2d(2, 2)
283             self.bn1 = nn.BatchNorm2d(64)
284
285             self.bn2 = nn.BatchNorm2d(128)
286             self.bn3 = nn.BatchNorm2d(256)
287             self.skip64_arr = nn.ModuleList()
288             for i in range(self.depth):
289                 self.skip64_arr.append(SkipBlock(64, 64,
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

297     self.skip128to256 = SkipBlock(128, 256,
298
                                                    ↪ skip_connect
                                                    ↪ )
299
300     self.skip256_arr = nn.ModuleList()
301     for i in range(self.depth):
302         self.skip256_arr.append(SkipBlock(256,256,
                                                    ↪ skip_connect
303
304         self.skip256ds = SkipBlock(256,256,
                                                    ↪ skip_connections=skip_connections,
                                                    ↪ skip_connections=skip_connections,
305
306         self.fc_seqn = nn.Sequential(
307             nn.Linear(32768, 16384),
308             nn.ReLU(),
309             nn.Linear(16384, 8192),
310             nn.ReLU(),
311             nn.Linear(8192, 4096),
312             nn.ReLU(),
313             nn.Linear(4096, 2880),
314         )
315
316     def forward(self, x):
317         x = self.pool(torch.nn.functional.relu(self.conv1(x)))
318         x = nn.MaxPool2d(2,2)(torch.nn.functional.relu(self.conv2(x)))
319
320         for i, skip64 in enumerate(self.skip64_arr[:self.depth//4]):
321             x = skip64(x)
322         x = self.skip64ds(x)
323
324         for i, skip64 in enumerate(self.skip64_arr[self.depth//4:]):
325             x = skip64(x)
326         x = self.bn1(x)
327
328         x = self.skip64to128(x)
329         for i, skip128 in enumerate(self.skip128_arr[:self.depth//4]):
330             x = skip128(x)
331         x = self.bn2(x)
332         x = self.skip128ds(x)
333         x = x.view(-1, 32768)
334         x = self.fc_seqn(x)
335
336     return x
337
338
339

```

```

340
341 ##-----ResNetForYolo
    ↪ class-----
342 '''This class is from
    ↪ https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py'''
343 class ResNetForYolo(nn.Module):
344     '''This class is using the ResNet50 model from the torchvision module,
345     the model is to fit the 256 image size and 3 classes, and 9 yolo cells
    ↪ on each axle, 8x8x5x9 = 2880'''
346     def __init__(self, resnet_type="resnet50", pretrained=True):
347         super(ResNetForYolo, self).__init__()
348
349         resnet = getattr(models, resnet_type)(pretrained=pretrained)
350         self.backbone = nn.Sequential(*list(resnet.children())[:-2])
351
352         # Define the YOLO prediction head
353         self.conv = nn.Conv2d(2048, 1024, kernel_size=3, stride=1, padding=1)
    ↪ #
354         self.fc = nn.Linear(1024 * 8 * 8, 2880)
355
356     def forward(self, x):
357         x = self.backbone(x)
358         x = self.conv(x)
359         x = x.view(x.shape[0], -1)
360         x = self.fc(x)
361         return x
362
363 ##-----Training
    ↪ functions-----
364 def run_code_for_training(net, train_loader, device="cuda", epochnum=20,
    ↪ save_path="./yolo_model.pth",plt_save_path="./training_loss.png"):
365     '''this function is borrowed from the DLStudio module
    ↪ https://engineering.purdue.edu/kak/distYOLO/,
366     the function is modified to fit the 256 image size and 3 classes, and 8
    ↪ yolo cells on each axle. the yolo vector is modified, the dw and dh
    ↪ are
367     ratio of the bbox to the anchor box size, instead of the yolo interval,
    ↪ the other code are mostly the same, the plotting code are removed'''
368     device = torch.device(device if torch.cuda.is_available() else "cpu")
369     net = net.to(device)
370     class_list = ['bus', 'cat', 'pizza']
371     # Define Loss Functions
372     criterion1 = nn.BCELoss(reduction='sum')
373     criterion2 = nn.MSELoss(reduction='sum')
374     criterion3 = nn.CrossEntropyLoss(reduction='sum')
375
376     optimizer = optim.Adam(net.parameters(), lr=1e-5)

```

```

377
378 print("\n Starting Training \n")
379 start_time = time.time()
380
381 loss_history = [] # Track loss over iterations
382 Loss_totoally = []
383 BCE_loss = []
384 MSE_loss = []
385 CrossEntropy_loss = []
386
387 # Training Loop
388 for epoch in range(epochnum):
389     running_loss = 0.0
390     running_bce = 0.0
391     running_mse = 0.0
392     running_ce = 0.0
393
394     for iter,data in enumerate(train_loader):
395         im_tensor, yolo_tensor, yolo_tensor_aug, bbox_tensor,
396         ↪ bbox_label_tensor = data
397         im_tensor, yolo_tensor, yolo_tensor_aug = im_tensor.to(device),
398         ↪ yolo_tensor.to(device), yolo_tensor_aug.to(device)
399
400         optimizer.zero_grad() # Reset gradients
401
402         predictions = net(im_tensor)
403
404         num_yolo_cells = 64
405         num_anchor_boxes = 5
406         predictions = predictions.view(-1, num_yolo_cells,
407         ↪ num_anchor_boxes, 9)
408
409         # Loss Calculation
410         loss = torch.tensor(0.0, requires_grad=True).float().to(device)
411
412         # Object Presence Loss
413         loss_obj = criterion1(nn.Sigmoid()(predictions[:, :, :, 0]),
414         ↪ yolo_tensor_aug[:, :, :, 0])
415         loss += loss_obj
416
417         # Bounding Box Loss
418         loss_bbox = criterion2(predictions[:, :, :, 1:5],
419         ↪ yolo_tensor_aug[:, :, :, 1:5])
420         loss += loss_bbox
421
422         # Class Prediction Loss
423         targets = yolo_tensor_aug[:, :, :, 5:].view(-1, 4)
424         targets = torch.argmax(targets, dim=1)

```

```

419         probs = predictions[:, :, :, 5:].view(-1, 4)
420         loss_class = criterion3(probs, targets)
421         loss += loss_class
422
423         loss.backward()
424         optimizer.step()
425
426         running_loss += loss.item()
427         running_bce += loss_obj.item()
428         running_mse += loss_bbox.item()
429         running_ce += loss_class.item()
430
431         if iter % 300 == 299:
432             elapsed_time = time.time() - start_time
433             avg_loss = running_loss / 300
434             loss_history.append(avg_loss)
435             Loss_totoally.append(running_loss)
436             BCE_loss.append(running_bce/300)
437             MSE_loss.append(running_mse/300)
438             CrossEntropy_loss.append(running_ce/300)
439
440             print(f"\n[Epoch {epoch+1}/{epochnum}, Batch {iter+1}] "
441                   f"Elapsed Time: {int(elapsed_time)} sec | Loss:
442                   ↳ {avg_loss:.4f}")
443             print(f"   BCE Loss: {running_bce/300:.4f} | MSE Loss:
444                   ↳ {running_mse/300:.4f} | CrossEntropy Loss:
445                   ↳ {running_ce/300:.4f}")
446             running_loss = 0.0
447             running_bce = 0.0
448             running_mse = 0.0
449             running_ce = 0.0
450
451             # Save Model Every Epoch
452             torch.save(net.state_dict(), save_path)
453             print(f"\nModel saved at {save_path} (Epoch {epoch+1})")
454
455             # Training Finished
456             print("\nTraining Complete!\n")
457
458             # Plot Loss Curve
459             plt.figure(figsize=(10,5))
460             plt.plot(loss_history, label="Loss vs. Iterations")
461             plt.plot(BCE_loss, label="BCE Loss")
462             plt.plot(MSE_loss, label="MSE Loss")
463             plt.plot(CrossEntropy_loss, label="CrossEntropy Loss")
464             plt.title("Training Loss Curve")
465             plt.xlabel("Iterations (every 300 steps)")

```



```

463     plt.ylabel("Loss")
464     plt.legend()
465     plt.savefig(plt_save_path)
466     # plt.show()
467
468     return net
469
470 def IoU_calculator(box1, box2):
471     '''this function is borrowed from the DLStudio module
472     ↪ https://engineering.purdue.edu/kak/distYOLO/, '''
473     """
474     Computes Intersection over Union (IoU) between two boxes.
475
476     Args:
477     box1, box2: Lists or arrays in [x1, y1, x2, y2] format.
478
479     Returns:
480     iou (float): Intersection over union score.
481     """
482
483     x1 = max(box1[0], box2[0])
484     y1 = max(box1[1], box2[1])
485     x2 = min(box1[2], box2[2])
486     y2 = min(box1[3], box2[3])
487
488     inter_w = max(0, x2 - x1)
489     inter_h = max(0, y2 - y1)
490     intersection = inter_w * inter_h
491
492     area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
493     area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])
494     union = area1 + area2 - intersection
495
496     if union == 0:
497         return 0.0
498     return intersection / union
499
500 def test_yolo_model(net, test_loader, device="cuda",
501     ↪ model_path="./yolo_model_new.pth", display_images=True):
502     '''this function is borrowed from the DLStudio module
503     ↪ https://engineering.purdue.edu/kak/distYOLO/,
504     the function is modified to fit the 256 image size and 3 classes, the yolo
505     ↪ vector is modified, the dw and dh are
506     ratio of the bbox to the anchor box size, instead of the yolo interval,
507     ↪ the other code are mostly the same'''
508
509     yolo_debug = False
510     device = device
511     net.load_state_dict(torch.load(model_path))

```

```

505 net = net.to(device)
506 yolo_interval = 256 / 8
507 num_yolo_cells = 64
508 num_anchor_boxes = 5
509 confusion_matrix = torch.zeros(3, 3)
510 class_list = ['bus', 'cat', 'pizza']
511 class_correct = [0] * 3
512 class_total = [0] * 3
513 totals_for_conf_mat = 0
514 totals_correct = 0
515 iou_scores = [0.0] * 3
516 anchor_boxes = [[1, 5], [1, 3], [1, 1], [3, 1], [5, 1]]
517 num_of_validation_images = len(test_loader)
518 print("num of test_loader", len(test_loader))
519 with torch.no_grad():
520     for iter, data in enumerate(test_loader):
521         im_tensor, yolo_tensor, yolo_tensor_aug, bbox_tensor,
522         ↪ bbox_label_tensor = data
523         im_tensor, yolo_tensor, yolo_tensor_aug = im_tensor.to(device),
524         ↪ yolo_tensor.to(device), yolo_tensor_aug.to(device)
525         predictions = net(im_tensor)
526         predictions = predictions.view(-1, num_yolo_cells,
527         ↪ num_anchor_boxes, 9)
528         # print("predictions", predictions.shape)
529         for ibx in range(predictions.shape[0]):
530             icx_2_best_anchor = {ic: None for ic in
531             ↪ range(predictions.shape[1])}
532             for icx in range(predictions.shape[1]):
533                 cell_pred = predictions[ibx, icx]
534                 previous_best = 0
535                 for anchor in range(cell_pred.shape[0]):
536                     if cell_pred[anchor][0] > previous_best:
537                         previous_best = anchor
538                     best_anchor_icx = previous_best
539                     icx_2_best_anchor[icx] = best_anchor_icx
540             sorted_icx = sorted(icx_2_best_anchor, key=lambda x:
541             ↪ predictions[ibx, x, icx_2_best_anchor[x]][0],
542             ↪ reverse=True)
543
544             retained_cells = sorted_icx[:5]
545             objects_detected = []
546             predicted_bboxes = []
547             predicted_labels_for_bboxes = []
548             predicted_label_index_vals = []
549             for icx in retained_cells:
550                 pred_vec = predictions[ibx, icx, icx_2_best_anchor[icx]]
551                 class_label_predict = pred_vec[-4:]

```

```

546         class_label_prob = torch.nn.Softmax(dim=0)(class_label_pdict)
547         class_label_prob = class_label_prob[:-1]
548         if torch.all(class_label_prob < 0.2):
549             predicted_class_label = None
550         else:
551             best_predicted_class_index = (class_label_prob ==
552             ↪ class_label_prob.max())
553             best_predicted_class_index =
554             ↪ torch.nonzero(best_predicted_class_index,
555             ↪ as_tuple=True)
556
557             ↪ predicted_label_index_vals.append(best_predicted_class_index[0].item())
558         predicted_class_label =
559         ↪ class_list[best_predicted_class_index[0].item()]
560         predicted_labels_for_bboxes.append(predicted_class_label)
561
562         pred_regression_vec = pred_vec[1:5].cpu()
563         del_x, del_y, delta_w, delta_h = pred_regression_vec
564         grid_x = int(icx // 8)
565         grid_y = int(icx % 8)
566         bb_center_x = (grid_x + 0.5 + del_x) * yolo_interval
567         bb_center_y = (grid_y + 0.5 + del_y) * yolo_interval
568         w = delta_w * yolo_interval *
569         ↪ anchor_boxes[icx_2_best_anchor[icx]][0]
570         h = delta_h * yolo_interval *
571         ↪ anchor_boxes[icx_2_best_anchor[icx]][1]
572         bb_top_left_x = bb_center_x - w/2
573         bb_top_left_y = bb_center_y - h/2
574         predicted_bboxes.append([bb_top_left_x, bb_top_left_y,
575         ↪ int(w), int(h)])
576         # print("predicted_bboxes",predicted_bboxes)
577         saved_predicted_bboxes = [predicted_bboxes[i][:] for i in
578         ↪ range(len(predicted_bboxes))]
579         gt_bboxes = torch.squeeze(bbox_tensor).cpu().numpy()
580         for pred_bbox in predicted_bboxes:
581             w, h = pred_bbox[2], pred_bbox[3]
582             pred_bbox[2] = pred_bbox[0] + w
583             pred_bbox[3] = pred_bbox[1] + h
584         mapping_from_pred_to_gt = {i: None for i in
585         ↪ range(len(predicted_bboxes))}
586         for i in range(len(predicted_bboxes)):
587             gt_possibles = {k : 0.0 for k in range(5)}      ## 0.0 for
588             ↪ IoU
589             for j in range(len(gt_bboxes)):
590                 if all(gt_bboxes[j][x] == 0 for x in range(4)): continue
591                 ↪ ## 4 is for the four coords of a bbox
592                 if (gt_bboxes[j].all() == 0): continue      ## 4 is for
593                 ↪ the four coords of a bbox

```

```

581         gt_possibles[j] = IoU_calculator(predicted_bboxes[i],
        ↪ gt_bboxes[j])
582     sorted_gt_possibles = sorted(gt_possibles, key=lambda x:
        ↪ gt_possibles[x], reverse=True)
583     # if sorted_gt_possibles[0] != 0:
584         # print("sorted_gt_possibles",sorted_gt_possibles)
585     # if display_images:
586         # print("For predicted bbox %d: the best gt bbox is: %d" %
        ↪ (i, sorted_gt_possibles[0]))
587     mapping_from_pred_to_gt[i] = (sorted_gt_possibles[0],
        ↪ gt_possibles[sorted_gt_possibles[0]])
588     gt_labels = torch.squeeze(bbox_label_tensor).cpu().numpy()
589     # print("\ngt_labels: ", gt_labels)
590     ## These are the predicted numeric class labels for the predicted
        ↪ bboxes in the image
591     pred_labels_ints = predicted_label_index_vals
592     for i,bbox_pred in enumerate(predicted_bboxes):
593         if display_images:
594             print("for i=%d, the predicted label: %s    the
        ↪ ground_truth label: %s" % (i,
        ↪ predicted_labels_for_bboxes[i],
595
600
601
602
603
604
605
606
607
608
609
610
611
        ↪ class_list[gt_labels[pred_labels_ints[i]]] != 13:

        ↪ confusion_matrix[gt_labels[mapping_from_pred_to_gt[i][0]]][pred_labels_ints[i]]
        ↪ += 1

    totals_for_conf_mat += 1
    class_total[gt_labels[mapping_from_pred_to_gt[i][0]]] += 1
    if gt_labels[mapping_from_pred_to_gt[i][0]] ==
        ↪ pred_labels_ints[i]:
        totals_correct += 1
        class_correct[gt_labels[mapping_from_pred_to_gt[i][0]]] +=
            ↪ 1

    iou_scores[gt_labels[mapping_from_pred_to_gt[i][0]]] +=
        ↪ mapping_from_pred_to_gt[i][1]
    ## If the user wants to see the image with the predicted bboxes
        ↪ and also the predicted labels:
    if display_images:
        predicted_bboxes = saved_predicted_bboxes
        if yolo_debug:
            print("[batch image=%d] objects found in descending
        ↪ probability order: " % ibx,
            ↪ predicted_labels_for_bboxes)
        logger = logging.getLogger()
        old_level = logger.level
        logger.setLevel(100)

```

```

612         fig = plt.figure(figsize=[12,12])
613         ax = fig.add_subplot(111)
614         display_scale = 2
615         new_im_tensor = torch.nn.functional.interpolate(im_tensor,
        ↪ scale_factor=display_scale, mode='bilinear',
        ↪ align_corners=False)
616
        ↪ ax.imshow(np.transpose(torchvision.utils.make_grid(new_im_tensor,
        ↪ normalize=True, padding=3, pad_value=255).cpu(), (1,2,0)))
617     for i, bbox_pred in enumerate(predicted_bboxes):
618         x,y,w,h = np.array(bbox_pred)
619         x,y,w,h = [item * display_scale for item in (x,y,w,h)]
620         rect = Rectangle((x,y),w,h,angle=0.0,edgecolor='r',fill =
        ↪ False,lw=2)
621         ax.add_patch(rect)
622         ax.annotate(predicted_labels_for_bboxes[i], (x,y-1),
        ↪ color='red', weight='bold', fontsize=10*display_scale)
623         gt_box_index = mapping_from_pred_to_gt[i][0]
        ↪ ## '[0]' because mapping returns (index,prob) pair
624         x1,y1,x2,y2 = np.array(gt_bboxes[gt_box_index])
625         x,y,w,h = x1,y1,x2-x1,y2-y1
626         x,y,w,h = [item * display_scale for item in (x,y,w,h)]
627         label = class_list[gt_labels[gt_box_index]]
628         ax.annotate(label, (x,y-1), color='g', weight='bold',
        ↪ fontsize=10*display_scale)
629         rect = Rectangle((x,y),w,h,angle=0.0,edgecolor='g',fill =
        ↪ False,lw=2)
630         ax.add_patch(rect)
631         # plt.savefig(dir_name_for_results + "/" + str(iter) +
        ↪ ".png")
632     plt.show()
633     logger.setLevel(old_level)
634     ## Our next job is to present to the user the information collected for the
    ↪ confusion matrix for the validation dataset:
635     if True:
636         print("\nConfusion Matrix: ", confusion_matrix)
637         print("\nclass_correct: ", class_correct)
638         print("\nclass_total: ", class_total)
639         print("\ntotals_for_conf_mat: ", totals_for_conf_mat)
640         print("\ntotals_correct: ", totals_correct)
641     for j in range(len(class_list)):
642         print('Prediction accuracy for %5s : %2d %%' % (class_list[j], 100 *
        ↪ class_correct[j] / class_total[j]))
643     print("\n\nOverall accuracy of multi-instance detection on %d test
    ↪ images: %d %%" % (num_of_validation_images,

```

```

    ↪ 100
    ↪ *
    ↪ sum
    ↪ /
    ↪ floo

```

```

645     print("""\nNOTE 1: This accuracy does not factor in the missed detection.
        ↳ This number is related to just the
646     mis-labeling errors for the detected instances. Percentage of the missed
        ↳ detections are shown in
647     the last column of the table shown below.""")
648     print("\n\nDisplaying the confusion matrix:\n")
649     out_str = " "
650     for j in range(len(class_list)): out_str += "%15s" % class_list[j]
651     out_str += "%15s" % "missing"
652     print(out_str + "\n")
653     for i,label in enumerate(class_list):
654         out_percents = [100 * confusion_matrix[i,j] / float(class_total[i])
        ↳ for j in range(len(class_list))]
655         missing_percent = 100 - sum(out_percents)
656         out_percents.append(missing_percent)
657         out_percents = ["%.2f" % item.item() for item in out_percents]
658         out_str = "%10s: " % class_list[i]
659         for j in range(len(class_list)+1): out_str += "%15s" %
        ↳ out_percents[j]
660         print(out_str)
661     print("\n\nNOTE 2: 'missing' means that an object instance of that label
        ↳ was NOT extracted from the image.")
662     print("\nNOTE 3: 'prediction accuracy' means the labeling accuracy for the
        ↳ extracted objects.")
663     print("\nNOTE 4: True labels are in the left-most column and the predicted
        ↳ labels at the top of the table.")
664
665     ## Finally, we present to the user the IoU scores for each of the object
        ↳ types:
666     iou_score_by_label = {class_list[i] : 0.0 for i in range(len(class_list))}
667     for i,label in enumerate(class_list):
668         iou_score_by_label[class_list[i]] =
        ↳ iou_scores[i]/float(class_total[i])
669     print("\n\nIoU scores for the different types of objects: ")
670     for obj_type in iou_score_by_label:
671         print("\n %10s: %.4f" % (obj_type,
        ↳ iou_score_by_label[obj_type]))
672
673     #-----Define the
        ↳ models-----
674     net0 = ResNetForYolo() # net using ResNet50
675     net1 = NetForYolo(skip_connections=True, depth=4) # net borrowing from the
        ↳ YOLOLogic with depth 4
676     net2 = NetForYolo(skip_connections=False, depth=2) # net borrowing from the
        ↳ YOLOLogic with depth 2
677     net3 = NetForYolo(skip_connections=True, depth=8) # net borrowing from the
        ↳ YOLOLogic with depth 8

```

```

678 #-----Train and Test the
    ↪ models-----
679 net0 = run_code_for_training(net0, train_loader, device="cuda", epochnum=20,
    ↪ save_path="./yolo_model_resnet.pth",
    ↪ plt_save_path="training_loss_resnet.png")
680 net1 = run_code_for_training(net1, train_loader, device="cuda", epochnum=20,
    ↪ save_path="./yolo_model_skip4.pth",
    ↪ plt_save_path="./training_loss_skip4.png")
681 net2 = run_code_for_training(net2, train_loader, device="cuda", epochnum=20,
    ↪ save_path="./yolo_model_skip2.pth",
    ↪ plt_save_path="./training_loss_skip2.png")
682 net3 = run_code_for_training(net3, train_loader, device="cuda", epochnum=20,
    ↪ save_path="./yolo_model_skip8.pth",
    ↪ plt_save_path="./training_loss_skip8.png")
683
684 test_yolo_model(net0, test_loader, device="cpu",
    ↪ model_path="./yolo_model_resnet.pth", display_images=False)
685 test_yolo_model(net1, test_loader, device="cuda",
    ↪ model_path="./yolo_model_skip4.pth", display_images=False)
686 test_yolo_model(net2, test_loader, device="cuda",
    ↪ model_path="./yolo_model_skip2.pth", display_images=False)
687 test_yolo_model(net3, test_loader, device="cuda",
    ↪ model_path="./yolo_model_skip8.pth", display_images=False)
688
689
690

```
