computervision-final-prep / lab / Lab 08 (YOLO)-20251128 / utils.py

limhpone YOLO fd90b91 · 2 hours ago

349 lines (278 loc) · 11.6 KB

Code Blame Raw

```python
import torch
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from collections import Counter

def intersection_over_union(boxes_preds, boxes_labels, box_format="midpoint"):
    """
    Calculates intersection over union

    Parameters:
        boxes_preds (tensor): Predictions of Bounding Boxes (BATCH_SIZE, 4)
        boxes_labels (tensor): Correct labels of Bounding Boxes (BATCH_SIZE, 4)
        box_format (str): midpoint/corners, if boxes (x,y,w,h) or (x1,y1,x2,y2)

    Returns:
        tensor: Intersection over union for all examples
    """

    if box_format == "midpoint":
        box1_x1 = boxes_preds[..., 0:1] - boxes_preds[..., 2:3] / 2
        box1_y1 = boxes_preds[..., 1:2] - boxes_preds[..., 3:4] / 2
        box1_x2 = boxes_preds[..., 0:1] + boxes_preds[..., 2:3] / 2
        box1_y2 = boxes_preds[..., 1:2] + boxes_preds[..., 3:4] / 2
        box2_x1 = boxes_labels[..., 0:1] - boxes_labels[..., 2:3] / 2
        box2_y1 = boxes_labels[..., 1:2] - boxes_labels[..., 3:4] / 2
        box2_x2 = boxes_labels[..., 0:1] + boxes_labels[..., 2:3] / 2
        box2_y2 = boxes_labels[..., 1:2] + boxes_labels[..., 3:4] / 2

    if box_format == "corners":
        box1_x1 = boxes_preds[..., 0:1]
        box1_y1 = boxes_preds[..., 1:2]
        box1_x2 = boxes_preds[..., 2:3]
```

```
            box1_y2 = boxes_preds[..., 3:4]  # (N, 1)
            box2_x1 = boxes_labels[..., 0:1]
            box2_y1 = boxes_labels[..., 1:2]
            box2_x2 = boxes_labels[..., 2:3]
            box2_y2 = boxes_labels[..., 3:4]

        x1 = torch.max(box1_x1, box2_x1)
        y1 = torch.max(box1_y1, box2_y1)
        x2 = torch.min(box1_x2, box2_x2)
        y2 = torch.min(box1_y2, box2_y2)

        # .clamp(0) is for the case when they do not intersect
        intersection = (x2 - x1).clamp(0) * (y2 - y1).clamp(0)

        box1_area = abs((box1_x2 - box1_x1) * (box1_y2 - box1_y1))
        box2_area = abs((box2_x2 - box2_x1) * (box2_y2 - box2_y1))

        return intersection / (box1_area + box2_area - intersection + 1e-6)


    def non_max_suppression(bboxes, iou_threshold, threshold, box_format="corners"):
        """
        Does Non Max Suppression given bboxes

        Parameters:
            bboxes (list): list of lists containing all bboxes with each bboxes
            specified as [class_pred, prob_score, x1, y1, x2, y2]
            iou_threshold (float): threshold where predicted bboxes is correct
            threshold (float): threshold to remove predicted bboxes (independent of IoU)
            box_format (str): "midpoint" or "corners" used to specify bboxes

        Returns:
            list: bboxes after performing NMS given a specific IoU threshold
        """

        assert type(bboxes) == list

        bboxes = [box for box in bboxes if box[1] > threshold]
        bboxes = sorted(bboxes, key=lambda x: x[1], reverse=True)
        bboxes_after_nms = []

        while bboxes:
            chosen_box = bboxes.pop(0)

            bboxes = [
                box
                for box in bboxes
                if box[0] != chosen_box[0]
                or intersection_over_union(
```

```
                    torch.tensor(chosen_box[2:]),
                    torch.tensor(box[2:]),
                    box_format=box_format,
                )
                < iou_threshold
            ]

            bboxes_after_nms.append(chosen_box)

        return bboxes_after_nms


def mean_average_precision(
    pred_boxes, true_boxes, iou_threshold=0.5, box_format="midpoint", num_classes=20
):
    """
    Calculates mean average precision

    Parameters:
        pred_boxes (list): list of lists containing all bboxes with each bboxes
        specified as [train_idx, class_prediction, prob_score, x1, y1, x2, y2]
        true_boxes (list): Similar as pred_boxes except all the correct ones
        iou_threshold (float): threshold where predicted bboxes is correct
        box_format (str): "midpoint" or "corners" used to specify bboxes
        num_classes (int): number of classes

    Returns:
        float: mAP value across all classes given a specific IoU threshold
    """

    # list storing all AP for respective classes
    average_precisions = []

    # used for numerical stability later on
    epsilon = 1e-6

    for c in range(num_classes):
        detections = []
        ground_truths = []

        # Go through all predictions and targets,
        # and only add the ones that belong to the
        # current class c
        for detection in pred_boxes:
            if detection[1] == c:
                detections.append(detection)

        for true_box in true_boxes:
            if true_box[1] == c:
```

```
                    ground_truths.append(true_box)

            # find the amount of bboxes for each training example
            # Counter here finds how many ground truth bboxes we get
            # for each training example, so let's say img 0 has 3,
            # img 1 has 5 then we will obtain a dictionary with:
            # amount_bboxes = {0:3, 1:5}
            amount_bboxes = Counter([gt[0] for gt in ground_truths])

            # We then go through each key, val in this dictionary
            # and convert to the following (w.r.t same example):
            # ammount_bboxes = {0:torch.tensor[0,0,0], 1:torch.tensor[0,0,0,0,0]}
            for key, val in amount_bboxes.items():
                amount_bboxes[key] = torch.zeros(val)

            # sort by box probabilities which is index 2
            detections.sort(key=lambda x: x[2], reverse=True)
            TP = torch.zeros((len(detections)))
            FP = torch.zeros((len(detections)))
            total_true_bboxes = len(ground_truths)

            # If none exists for this class then we can safely skip
            if total_true_bboxes == 0:
                continue

            for detection_idx, detection in enumerate(detections):
                # Only take out the ground_truths that have the same
                # training idx as detection
                ground_truth_img = [
                    bbox for bbox in ground_truths if bbox[0] == detection[0]
                ]

                num_gts = len(ground_truth_img)
                best_iou = 0

                for idx, gt in enumerate(ground_truth_img):
                    iou = intersection_over_union(
                        torch.tensor(detection[3:]),
                        torch.tensor(gt[3:]),
                        box_format=box_format,
                    )

                    if iou > best_iou:
                        best_iou = iou
                        best_gt_idx = idx

                if best_iou > iou_threshold:
                    # only detect ground truth detection once
                    if amount_bboxes[detection[0]][best_gt_idx] == 0:
```

```
                        # true positive and add this bounding box to seen
                        TP[detection_idx] = 1
                        amount_bboxes[detection[0]][best_gt_idx] = 1
                    else:
                        FP[detection_idx] = 1

                # if IOU is lower then the detection is a false positive
                else:
                    FP[detection_idx] = 1

            TP_cumsum = torch.cumsum(TP, dim=0)
            FP_cumsum = torch.cumsum(FP, dim=0)
            recalls = TP_cumsum / (total_true_bboxes + epsilon)
            precisions = torch.divide(TP_cumsum, (TP_cumsum + FP_cumsum + epsilon))
            precisions = torch.cat((torch.tensor([1]), precisions))
            recalls = torch.cat((torch.tensor([0]), recalls))
            # torch.trapz for numerical integration
            average_precisions.append(torch.trapz(precisions, recalls))

        return sum(average_precisions) / len(average_precisions)


def plot_image(image, boxes):
    """Plots predicted bounding boxes on the image"""
    im = np.array(image)
    height, width, _ = im.shape

    # Create figure and axes
    fig, ax = plt.subplots(1)
    # Display the image
    ax.imshow(im)

    # box[0] is x midpoint, box[2] is width
    # box[1] is y midpoint, box[3] is height

    # Create a Rectangle potch
    for box in boxes:
        box = box[2:]
        assert len(box) == 4, "Got more values than in x, y, w, h, in a box!"
        upper_left_x = box[0] - box[2] / 2
        upper_left_y = box[1] - box[3] / 2
        rect = patches.Rectangle(
            (upper_left_x * width, upper_left_y * height),
            box[2] * width,
            box[3] * height,
            linewidth=1,
            edgecolor="r",
            facecolor="none",
        )
```

```
            # Add the patch to the Axes
            ax.add_patch(rect)

        plt.show()

    def get_bboxes(
        loader,
        model,
        iou_threshold,
        threshold,
        pred_format="cells",
        box_format="midpoint",
        device="cuda",
    ):
        all_pred_boxes = []
        all_true_boxes = []

        # make sure model is in eval before get bboxes
        model.eval()
        train_idx = 0

        for batch_idx, (x, labels) in enumerate(loader):
            x = x.to(device)
            labels = labels.to(device)

            with torch.no_grad():
                predictions = model(x)

            batch_size = x.shape[0]
            true_bboxes = cellboxes_to_boxes(labels)
            bboxes = cellboxes_to_boxes(predictions)

            for idx in range(batch_size):
                nms_boxes = non_max_suppression(
                    bboxes[idx],
                    iou_threshold=iou_threshold,
                    threshold=threshold,
                    box_format=box_format,
                )


                #if batch_idx == 0 and idx == 0:
                #    plot_image(x[idx].permute(1,2,0).to("cpu"), nms_boxes)
                #    print(nms_boxes)

                for nms_box in nms_boxes:
                    all_pred_boxes.append([train_idx] + nms_box)

                for box in true_bboxes[idx]:
```

```
279                     # many will get converted to 0 pred
280                     if box[1] > threshold:
281                         all_true_boxes.append([train_idx] + box)
282
283                 train_idx += 1
284
285         model.train()
286         return all_pred_boxes, all_true_boxes
287
288
289
290     def convert_cellboxes(predictions, S=7):
291         """
292         Converts bounding boxes output from Yolo with
293         an image split size of S into entire image ratios
294         rather than relative to cell ratios. Tried to do this
295         vectorized, but this resulted in quite difficult to read
296         code... Use as a black box? Or implement a more intuitive,
297         using 2 for loops iterating range(S) and convert them one
298         by one, resulting in a slower but more readable implementation.
299         """
300
301         predictions = predictions.to("cpu")
302         batch_size = predictions.shape[0]
303         predictions = predictions.reshape(batch_size, 7, 7, 30)
304         bboxes1 = predictions[..., 21:25]
305         bboxes2 = predictions[..., 26:30]
306         scores = torch.cat(
307             (predictions[..., 20].unsqueeze(0), predictions[..., 25].unsqueeze(0)), dim=0
308         )
309         best_box = scores.argmax(0).unsqueeze(-1)
310         best_boxes = bboxes1 * (1 - best_box) + best_box * bboxes2
311         cell_indices = torch.arange(7).repeat(batch_size, 7, 1).unsqueeze(-1)
312         x = 1 / S * (best_boxes[..., :1] + cell_indices)
313         y = 1 / S * (best_boxes[..., 1:2] + cell_indices.permute(0, 2, 1, 3))
314         w_y = 1 / S * best_boxes[..., 2:4]
315         converted_bboxes = torch.cat((x, y, w_y), dim=-1)
316         predicted_class = predictions[..., :20].argmax(-1).unsqueeze(-1)
317         best_confidence = torch.max(predictions[..., 20], predictions[..., 25]).unsqueeze(
318             -1
319         )
320         converted_preds = torch.cat(
321             (predicted_class, best_confidence, converted_bboxes), dim=-1
322         )
323
324         return converted_preds
325
326
327     def cellboxes_to_boxes(out, S=7):
```

```python
        converted_pred = convert_cellboxes(out).reshape(out.shape[0], S * S, -1)
        converted_pred[..., 0] = converted_pred[..., 0].long()
        all_bboxes = []

        for ex_idx in range(out.shape[0]):
            bboxes = []

            for bbox_idx in range(S * S):
                bboxes.append([x.item() for x in converted_pred[ex_idx, bbox_idx, :]])
            all_bboxes.append(bboxes)

        return all_bboxes

    def save_checkpoint(state, filename="my_checkpoint.pth.tar"):
        print("=> Saving checkpoint")
        torch.save(state, filename)


    def load_checkpoint(checkpoint, model, optimizer):
        print("=> Loading checkpoint")
        model.load_state_dict(checkpoint["state_dict"])
        optimizer.load_state_dict(checkpoint["optimizer"])
```