16824 Homework 1

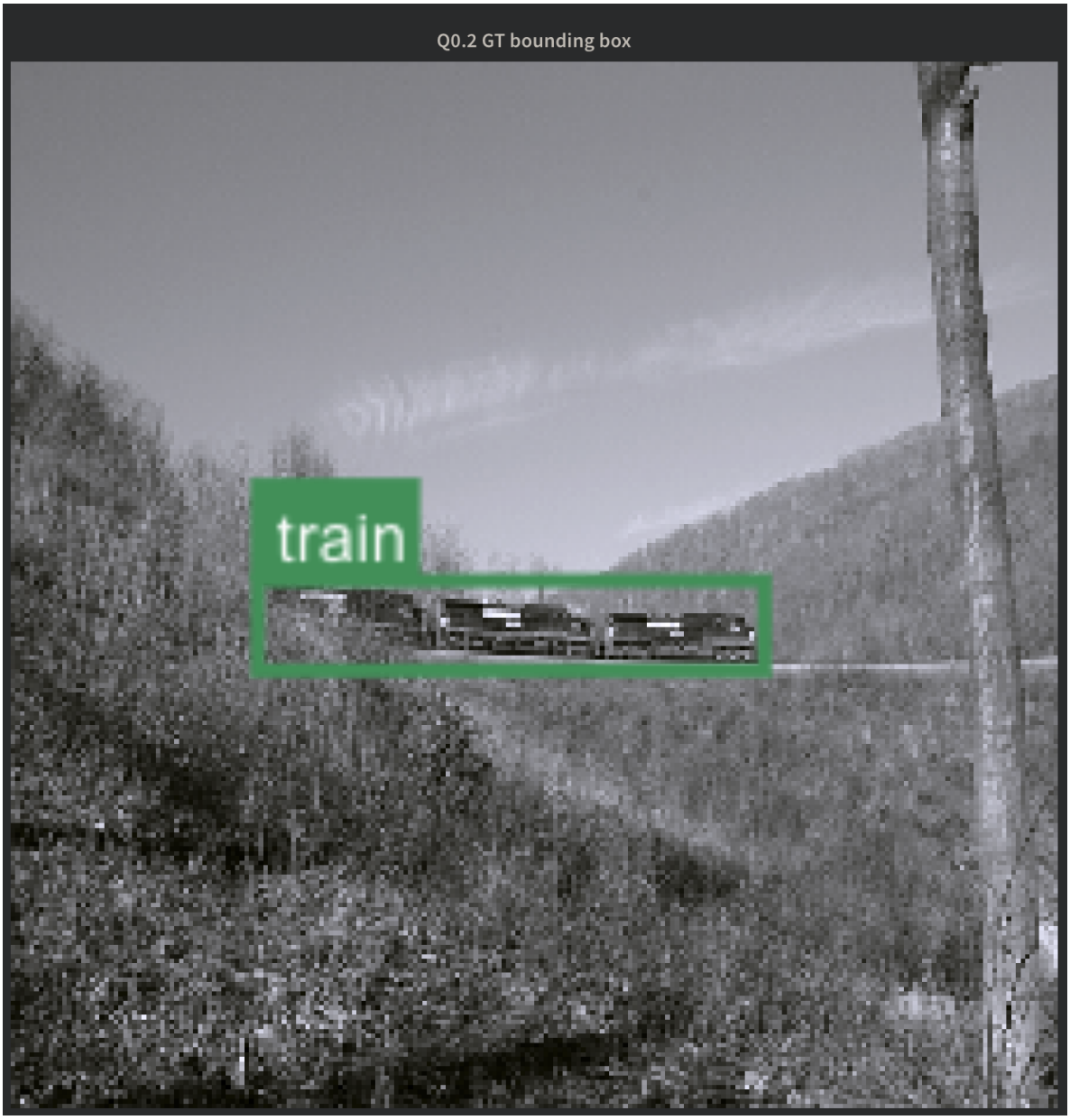
Jiajun Wan
Andrew ID: jiajunw2

Task 0: Visualization and Understanding the Data Structures

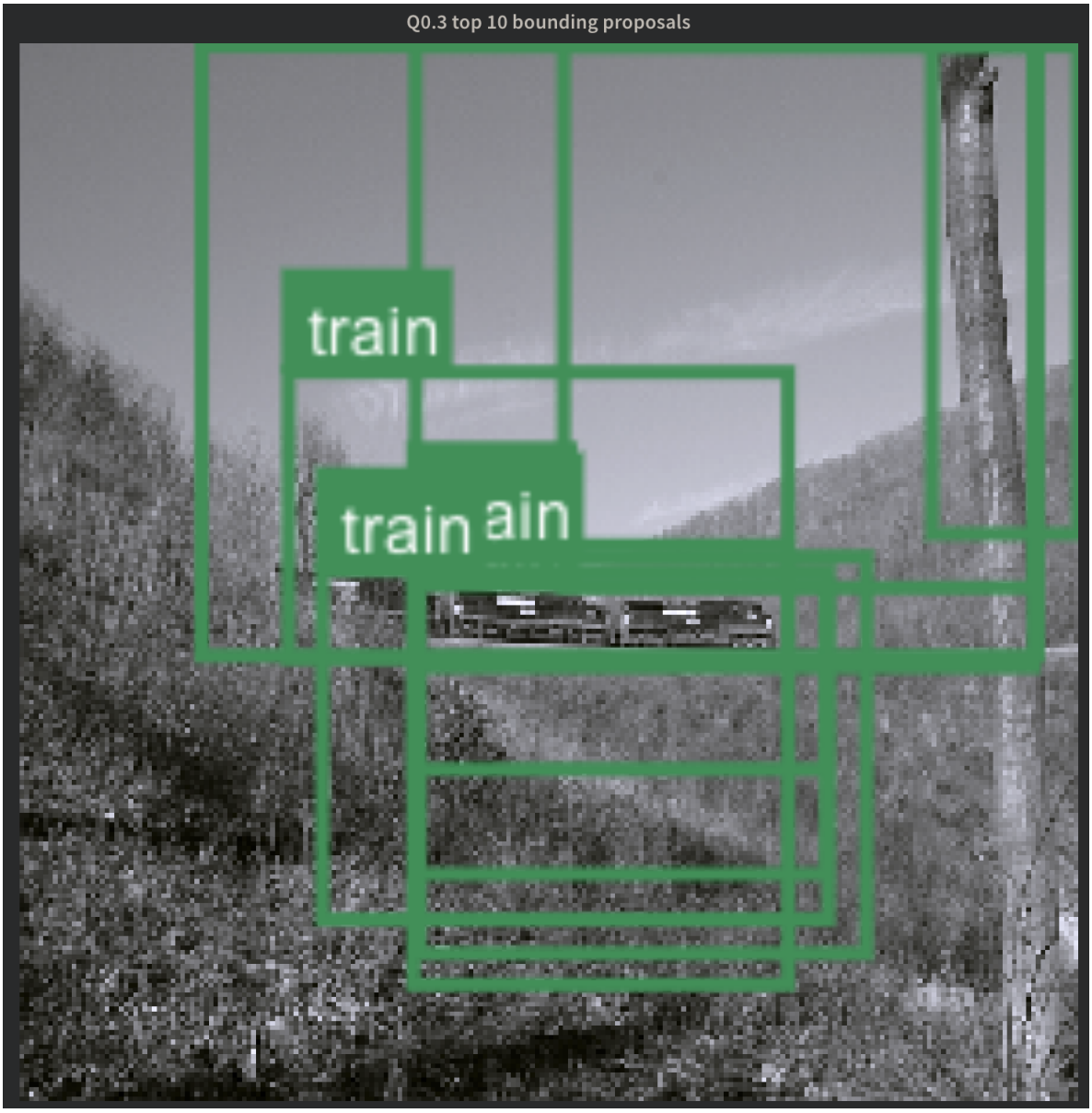
Q 0.1: What classes does the image at index 2020 contain (index 2020 is the 2021-th image due to 0-based numbering)?

It contains train.

Q 0.2 Use Wandb to visualize the ground-truth bounding box and the class for the image at index 2020.



Q 0.3 Use Wandb to visualize the top ten bounding box proposals for the image at index 2020.



Q1.1 describe functionality of the completed TODO blocks with comments

```
# TODO (Q1.1): define loss function (criterion) and optimizer from [1]
# also use an LR scheduler to decay LR by 10 every 30 epochs
# Binary Cross Entropy Loss
criterion = nn.BCEWithLogitsLoss()
# SGD
optimizer = torch.optim.SGD(model.parameters(),
                            lr=args.lr,
                            momentum=args.momentum,
                            weight_decay=args.weight_decay)
# LR Step scheduler
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                            step_size=30,
                                            gamma=0.1)

# TODO (Q1.1): Create Datasets and Dataloaders using VOCDataset
# Ensure that the sizes are 512x512
# Also ensure that data directories are correct
# The ones use for testing by TAs might be different
# create train and validation dataset
train_dataset = VOCDataset(split='trainval',
                           image_size=512)
val_dataset = VOCDataset(split='test',
                         image_size=512)
train_sampler = None

# create train and validation dataloader with custom collate function
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=args.batch_size,
    shuffle=False,
    num_workers=args.workers,
    collate_fn=train_dataset.collate_fn, # use custom collate function
    pin_memory=True,
    sampler=train_sampler,
    drop_last=True)

val_loader = torch.utils.data.DataLoader(
    val_dataset,
    batch_size=args.batch_size,
    shuffle=False,
    num_workers=args.workers,
    collate_fn=val_dataset.collate_fn, # use custom collate function
    pin_memory=True,
    drop_last=True)

# TODO (Q1.1): Get inputs from the data dict
# Convert inputs to cuda if training on GPU
# clear optimizer gradient
optimizer.zero_grad()
```

```
input = data[0].cuda()
target = data[1].cuda()

# TODO (Q1.1): Get output from model
output = model(input)

# TODO (Q1.1): Perform any necessary operations on the output

# TODO (Q1.1): Compute loss using ``criterion``
loss = criterion(output, target)
# sigmoid on each element to enforce probability from 0 to 1
output = torch.sigmoid(output)

# TODO (Q1.1): compute gradient and perform optimizer step
# backprop
loss.backward()
optimizer.step()

# TODO (Q1.1): Get inputs from the data dict
# Convert inputs to cuda if training on GPU
input = data[0].cuda()
target = data[1].cuda()

# TODO (Q1.1): Get output from model
output = model(input)

# TODO (Q1.1): Perform any necessary functions on the output

# TODO (Q1.1): Compute loss using ``criterion``
loss = criterion(output, target)
# sigmoid on each element to enforce probability from 0 to 1
output = torch.sigmoid(output)

# TODO (Q1.1): Define model
# AlexNet fueature extractor layers
self.features = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2)),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2), dilation=(1, 1), ceil_mode=False
    ↪ ),
    nn.Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2)),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2), dilation=(1, 1), ceil_mode=False
    ↪ ),
    nn.Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
    nn.ReLU(),
    nn.Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
    nn.ReLU(),
    nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
    nn.ReLU(),
)
```

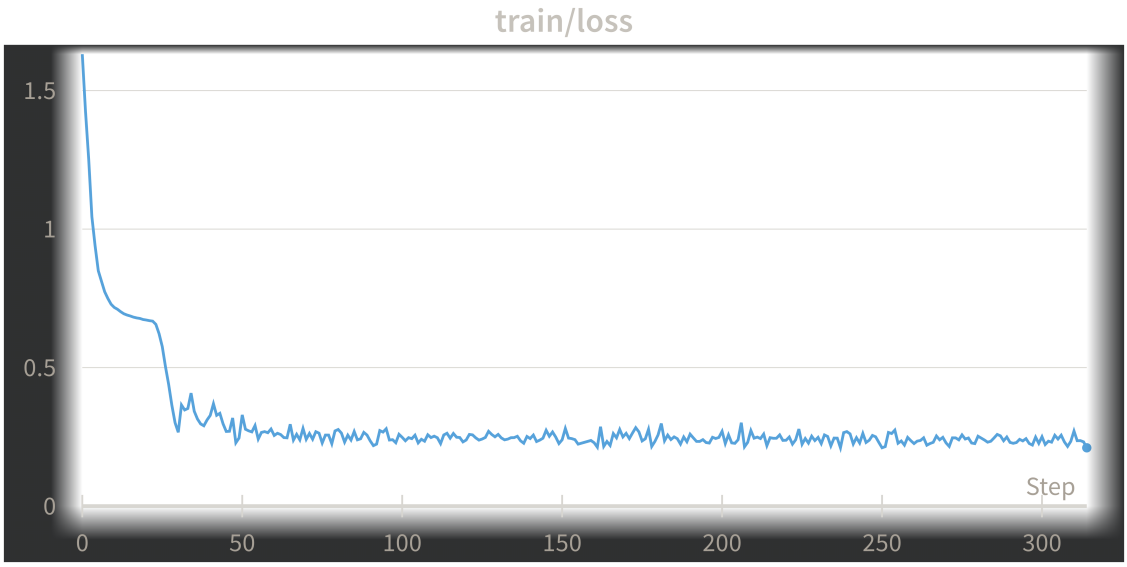
```
# our own classifier layers
self.classifier = nn.Sequential(
    nn.Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1)),
    nn.ReLU(),
    nn.Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1)),
    nn.ReLU(),
    nn.Conv2d(256, num_classes, kernel_size=(1, 1), stride=(1, 1)),
)
# final global max pooling layers
self.pool = nn.Sequential(
    nn.AdaptiveMaxPool2d((1, 1)),
    nn.Flatten(),
)


def forward(self, x):
# TODO (Q1.1): Define forward pass
# forward propagation
x = self.features(x)
x = self.classifier(x)
x = self.pool(x)
return x

"""
TODO:
    1. Load bounding box proposals for the index from self.roi_data. The proposals are of the format:
    [y_min, x_min, y_max, x_max] or [top left row, top left col, bottom right row, bottom right col]
    2. Normalize in the range (0, 1) according to image size (be careful of width/height and x/y correspondences)
    3. Make sure to return only the top_n proposals based on proposal confidence ("boxScores")!
    4. You may have to write a custom collate_fn since some of the attributes below may be variable in number for each data point
"""
# get array of (boxScore, boxes) tuples and sort by boxScor
score_bbox_pairs = sorted(list(zip(self.roi_data['boxScores'][0][index][:, 0], self.roi_data['boxes'][0][index])), key=lambda x: x[0], reverse=True)[:self.top_n]
# convert proposal to [xmin, ymin, xmax, ymax], same as gt_boxes
proposals = np.asarray(list(map(lambda x: np.array([x[1][1] / width, x[1][0] / height, x[1][3] / width, x[1][2] / height]), score_bbox_pairs))).astype(np.float32)

def collate_fn(self, dict):
    """
    :param dict: a batch list of ret dictionary from __getitem__
    :return: pytorch tensor of x and y
    """
    # extract input image and target label from dictionary and convert to pytorch single precision float tensor
```

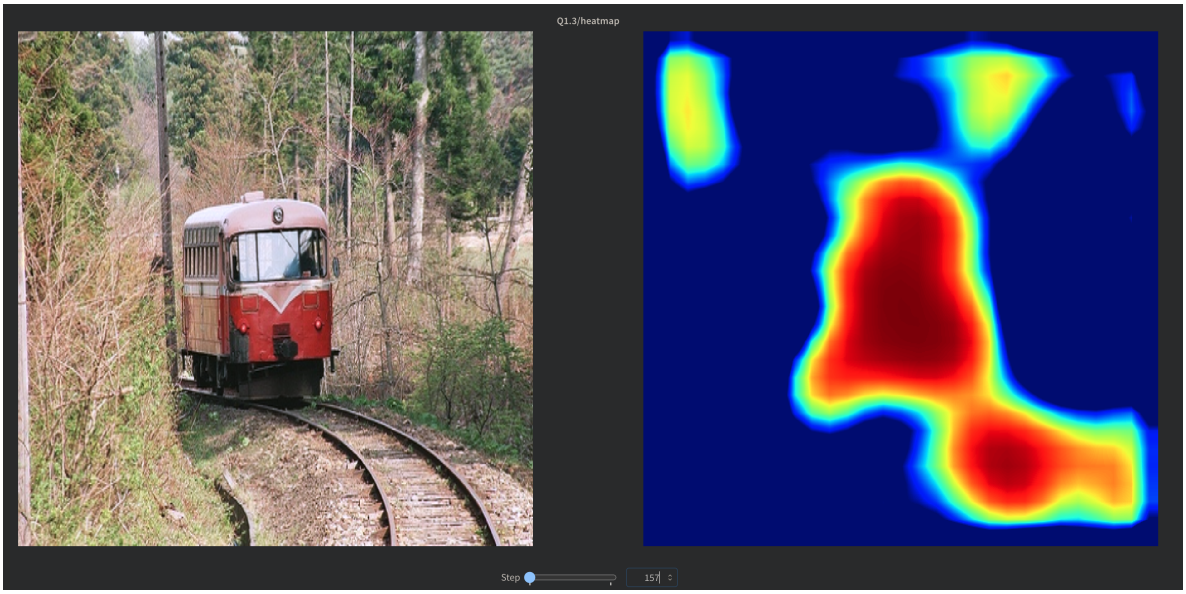
```
    X = [x['image'] for x in dict]
    Y = [y['label'] for y in dict]
    X = torch.stack(X).to(torch.float32)
    Y = torch.stack(Y).to(torch.float32)
    return X, Y
```

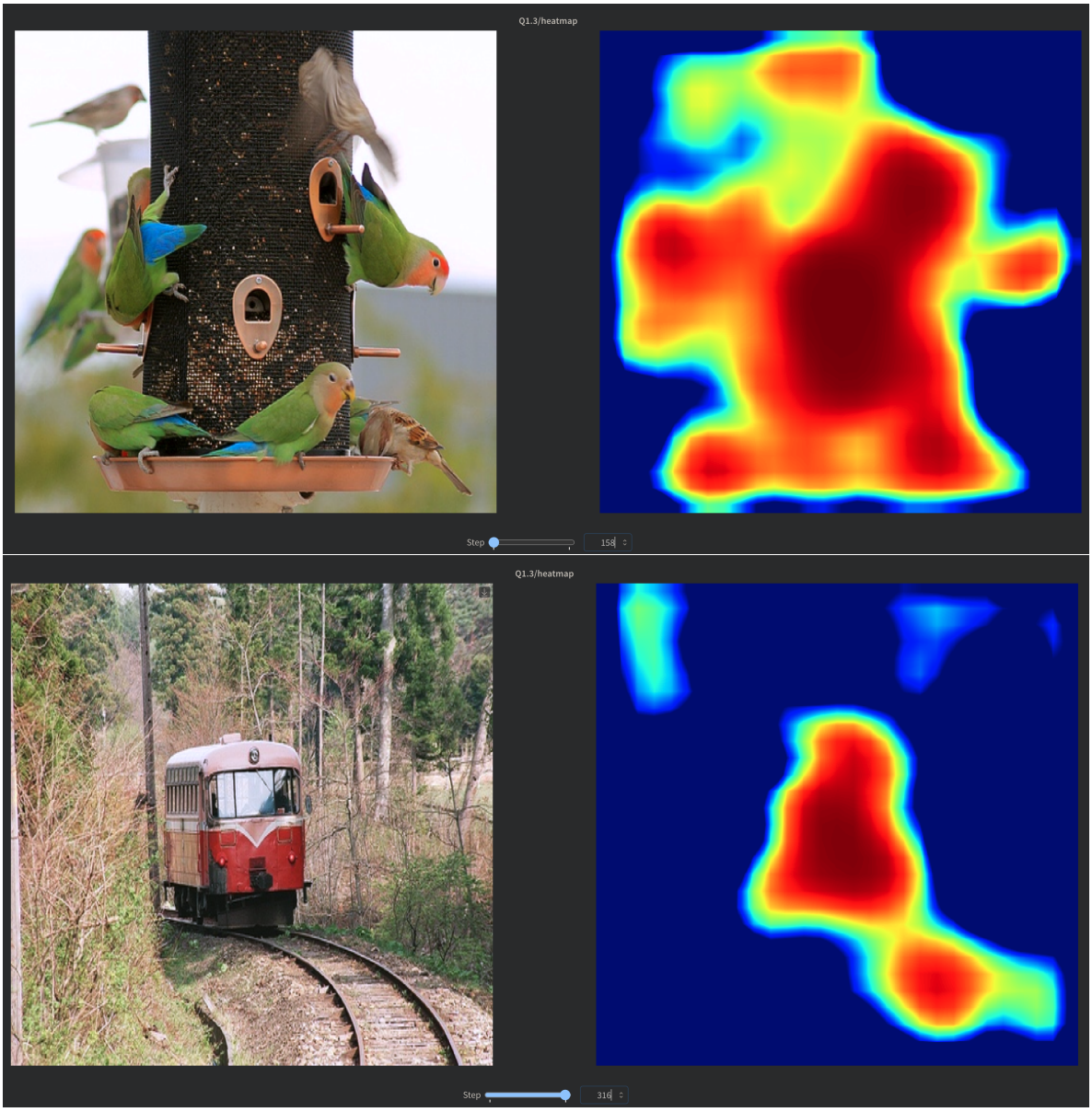
Q 1.2 What is the shape (NxCxHxW) of the output of the model?

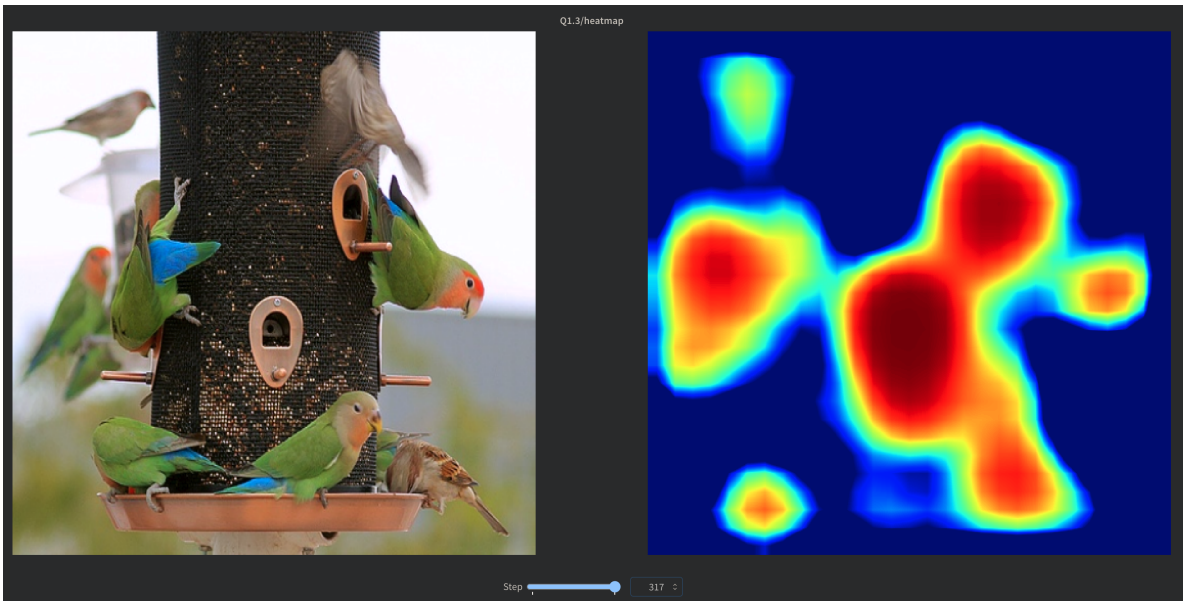
[Batch Size, 20, 1, 1]. After flatten: [Batch Size, 20]

Q 1.3






Q1.3/heatmap
Step
158
Q1.3/heatmap
Step
316



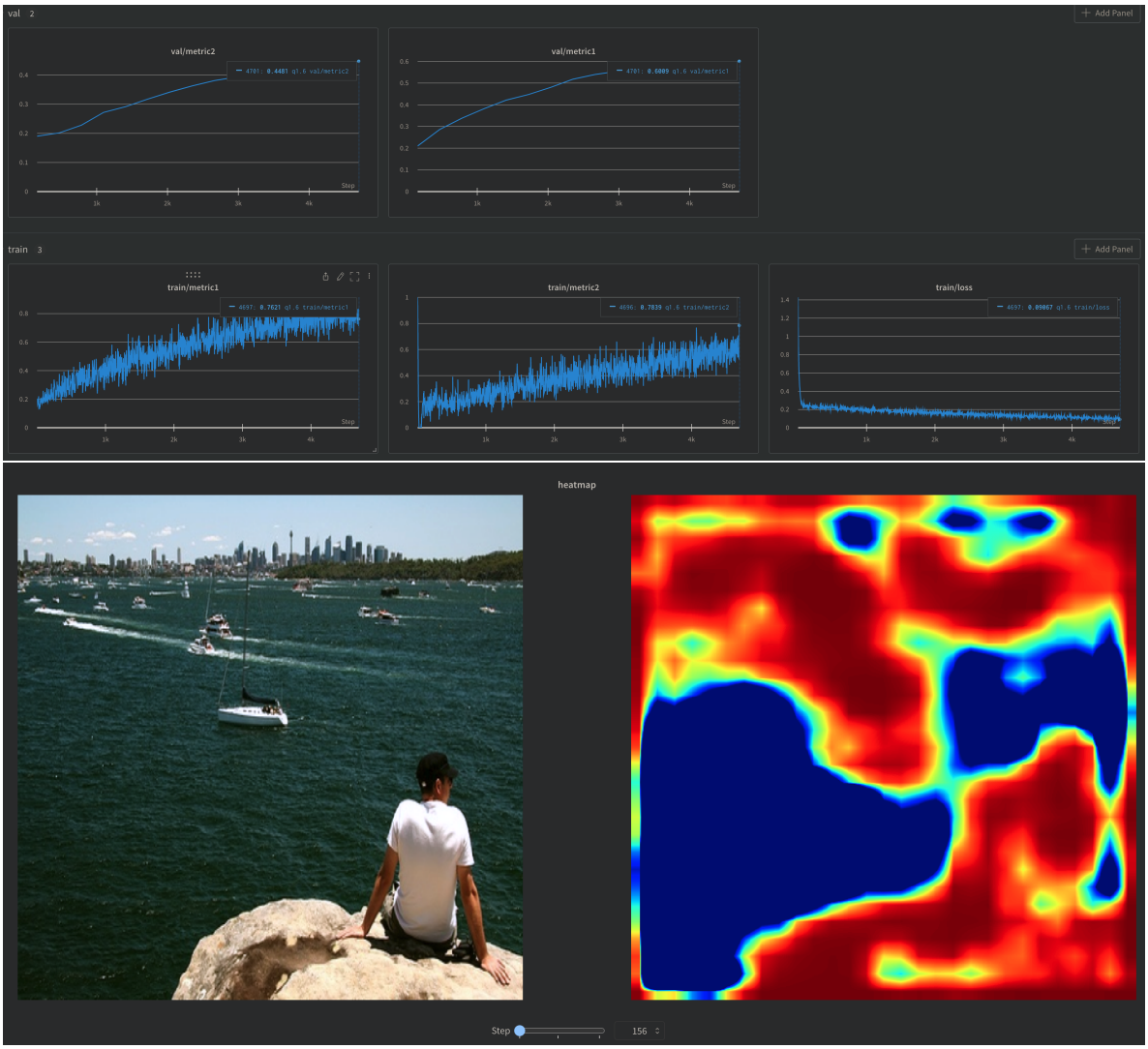
Q 1.4 In the first few iterations, you should observe a steep drop in the loss value. Why does this happen?

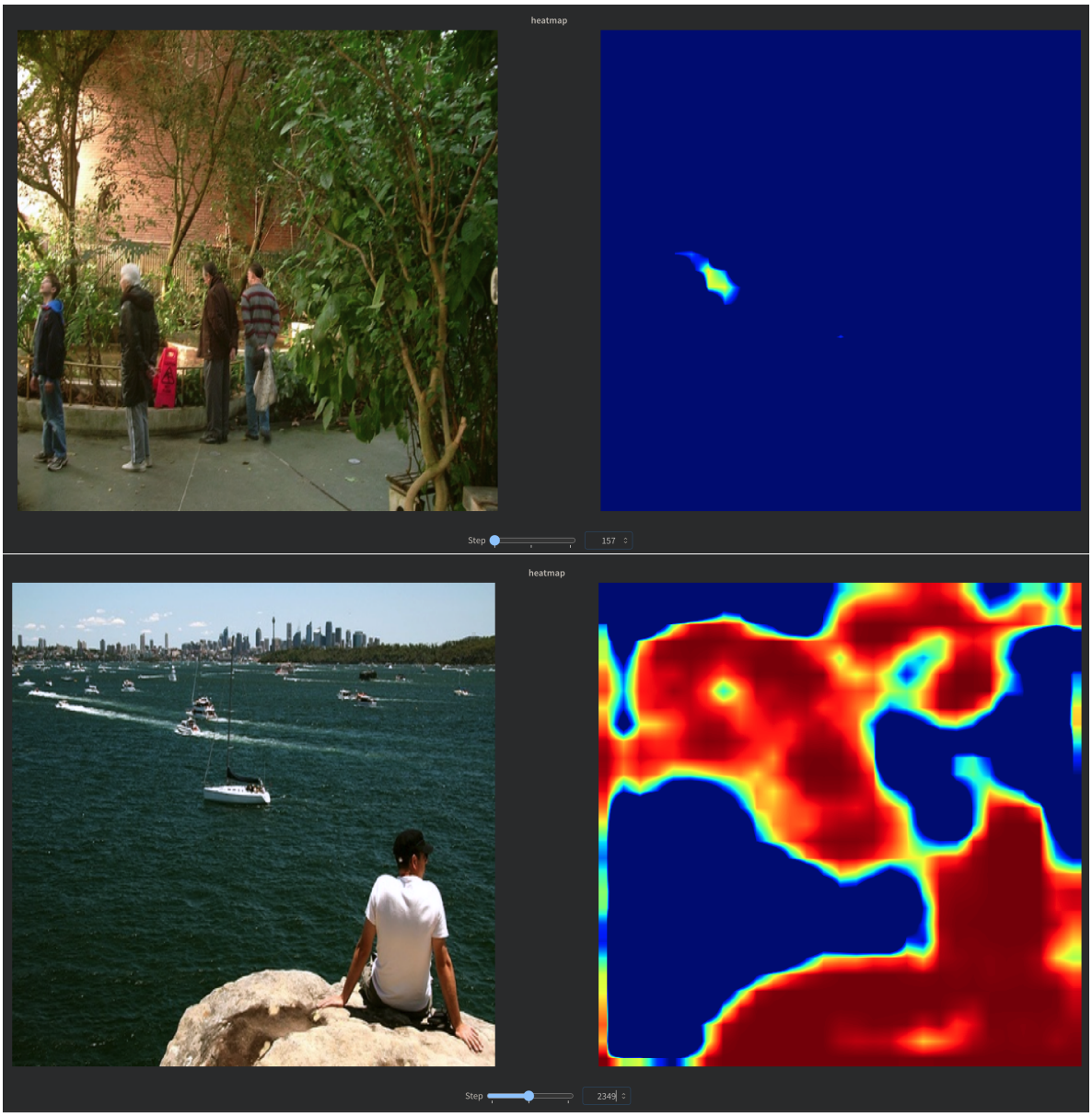
Because at first the network is initialized randomly, and there are multiple labels associated with an image, the loss is going to be large. Also the labels across a batch are not balanced with some labels in most images and some labels not appearing in any image. After the network is learning how to classify based on the activation heat maps, the loss decreases steeply. The loss gradually saturates after the network learns the most in the first few iterations.

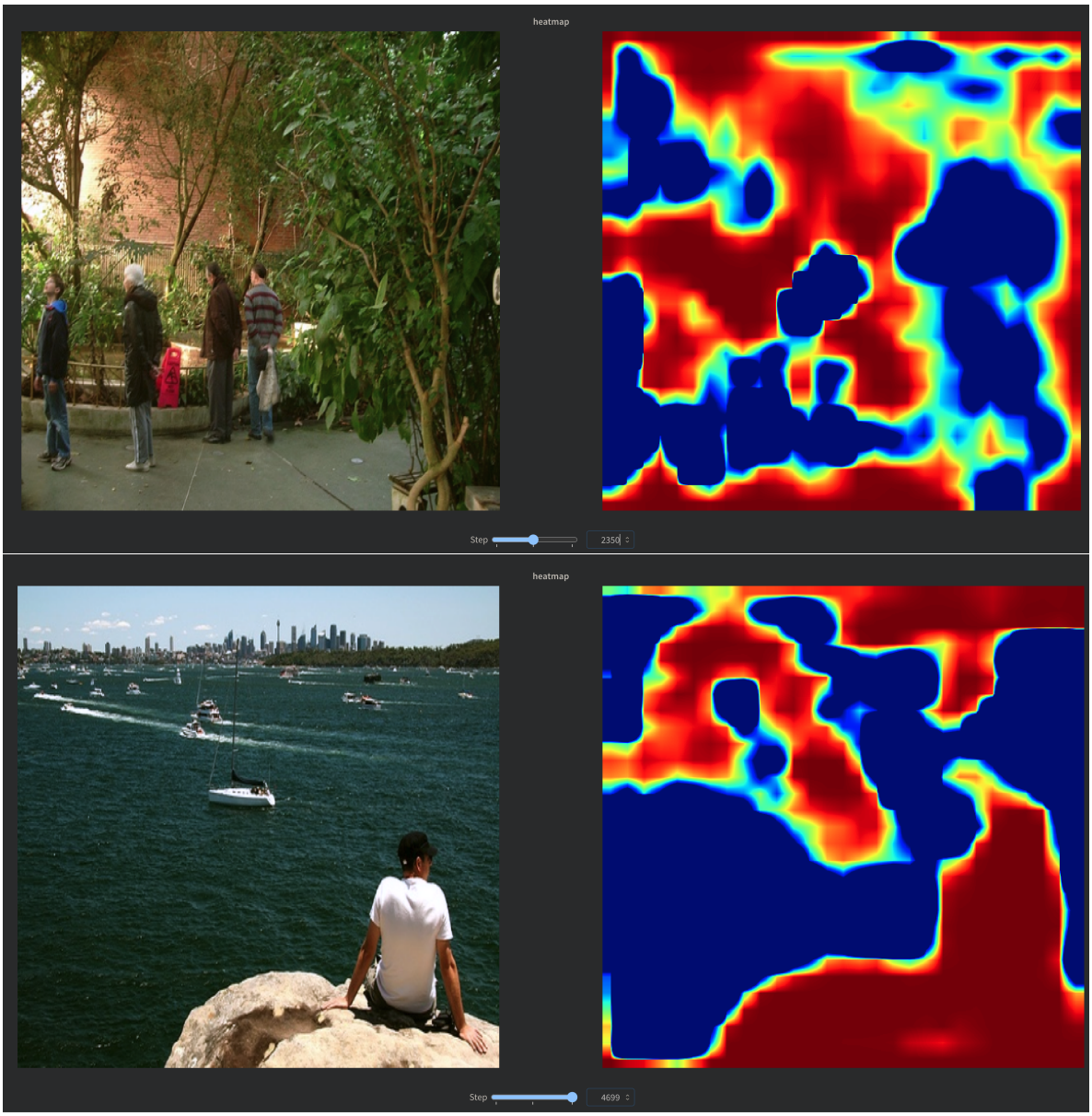
Q 1.5

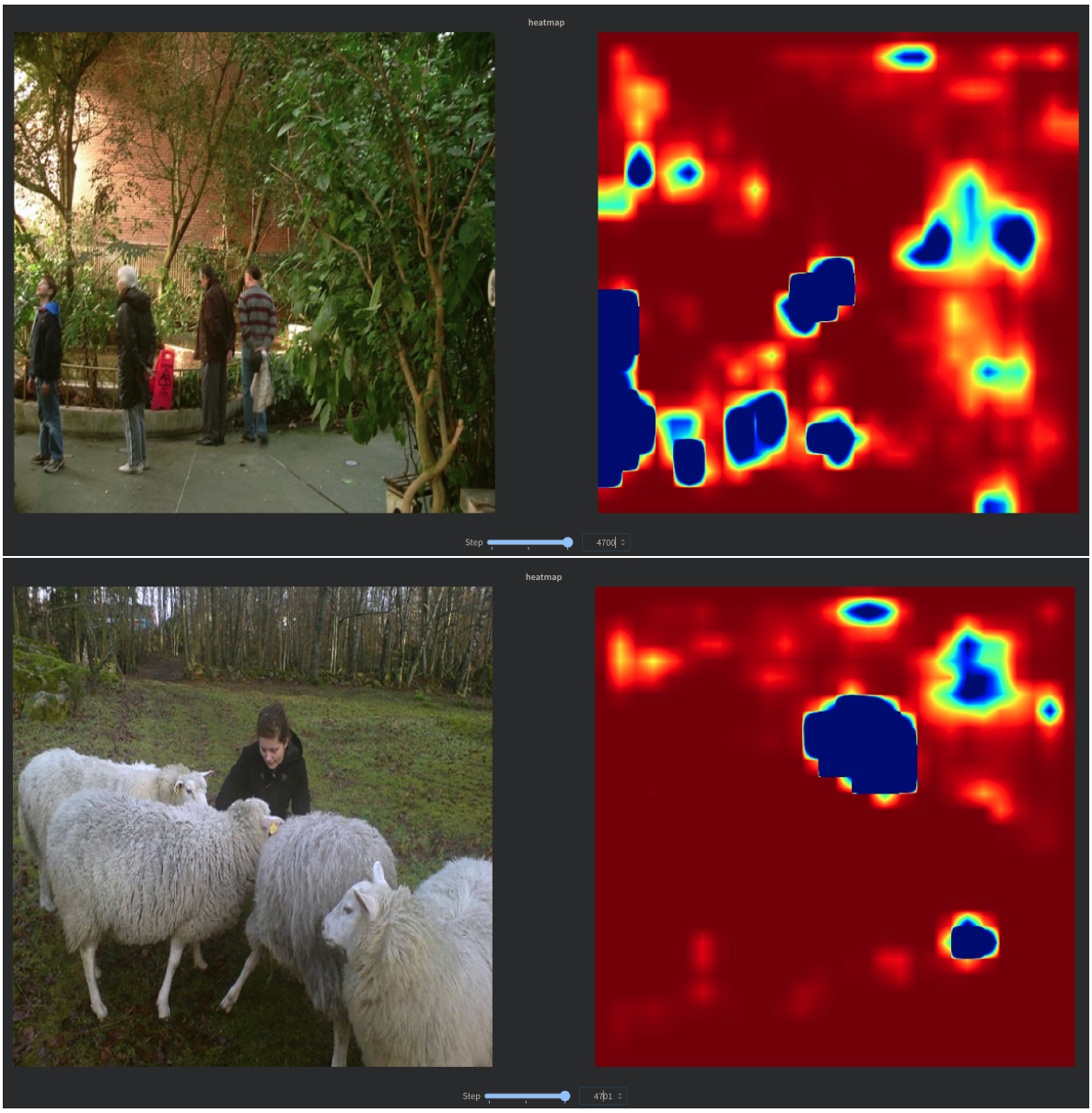
Filled out TODOs in the code files. The assumption is threshold for deciding positive prediction is 0.5.

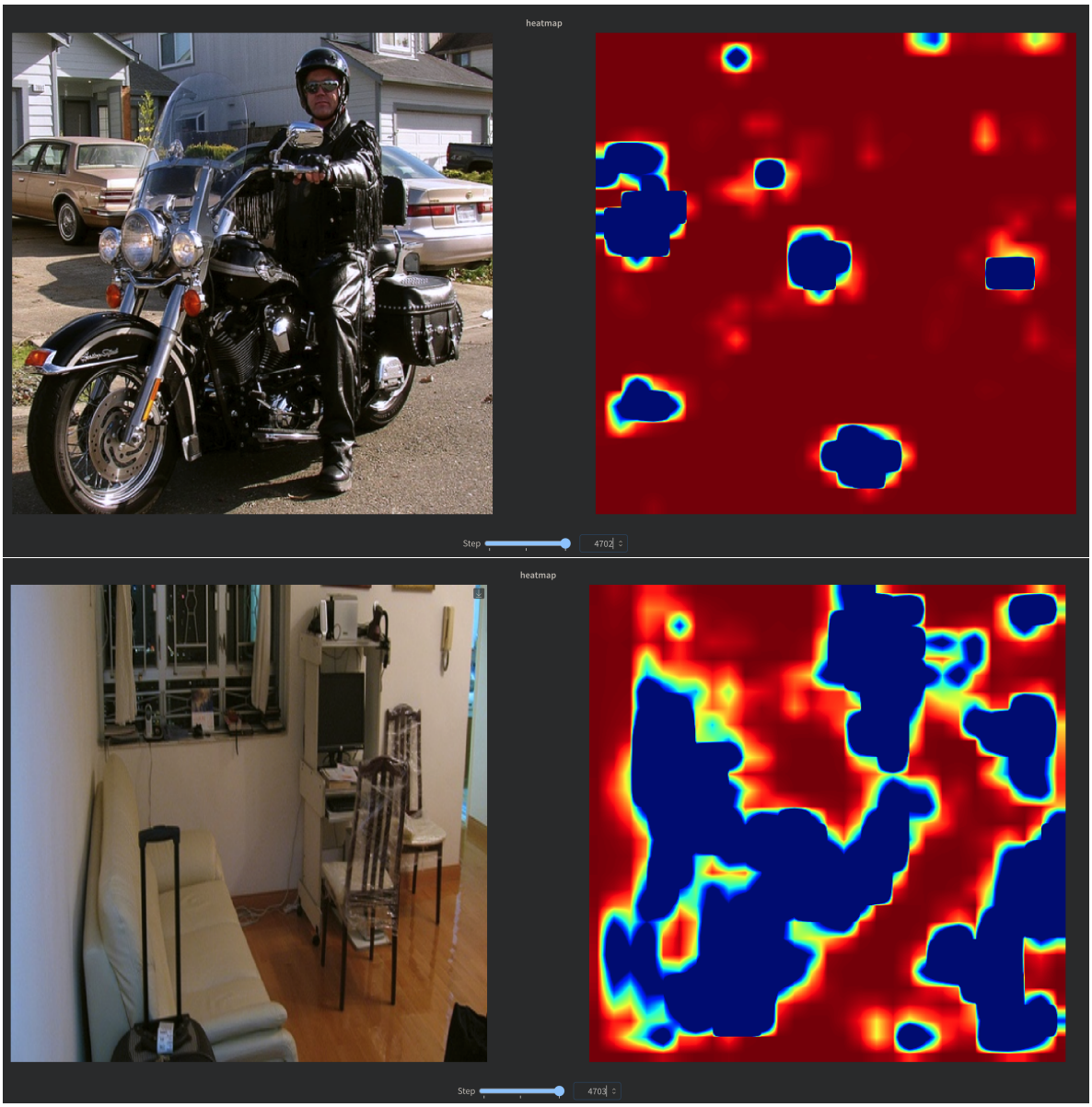
Q 1.6


val 2
Add Panel
val/metric2
val/metric1
train 3
Add Panel
train/metric1
train/metric2
train/loss
heatmap
Step
156


heatmap
Step
157
heatmap
Step
2349


heatmap
Step
2350
heatmap
Step
4699

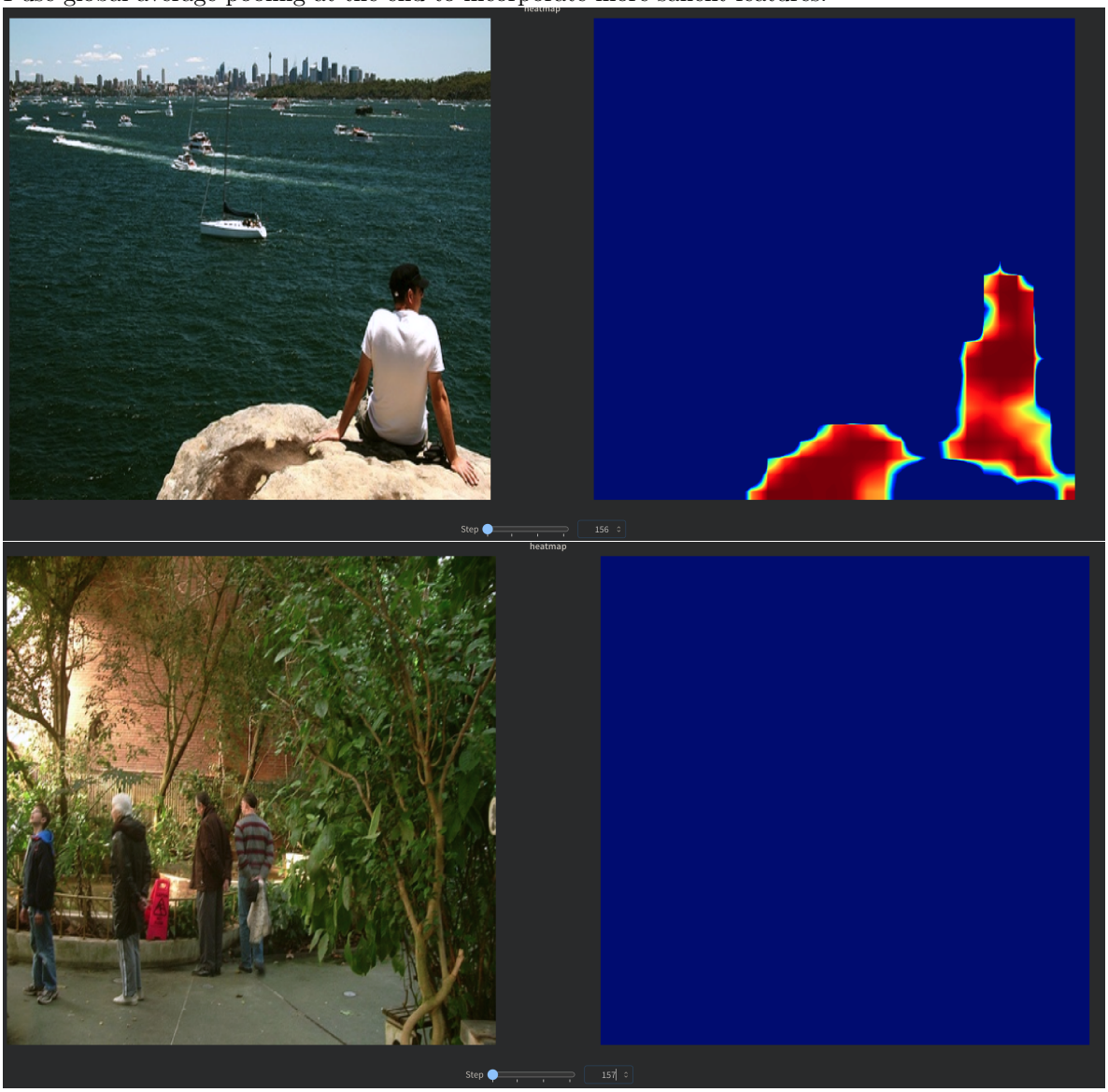

heatmap
Step
4700
heatmap
Step
4701

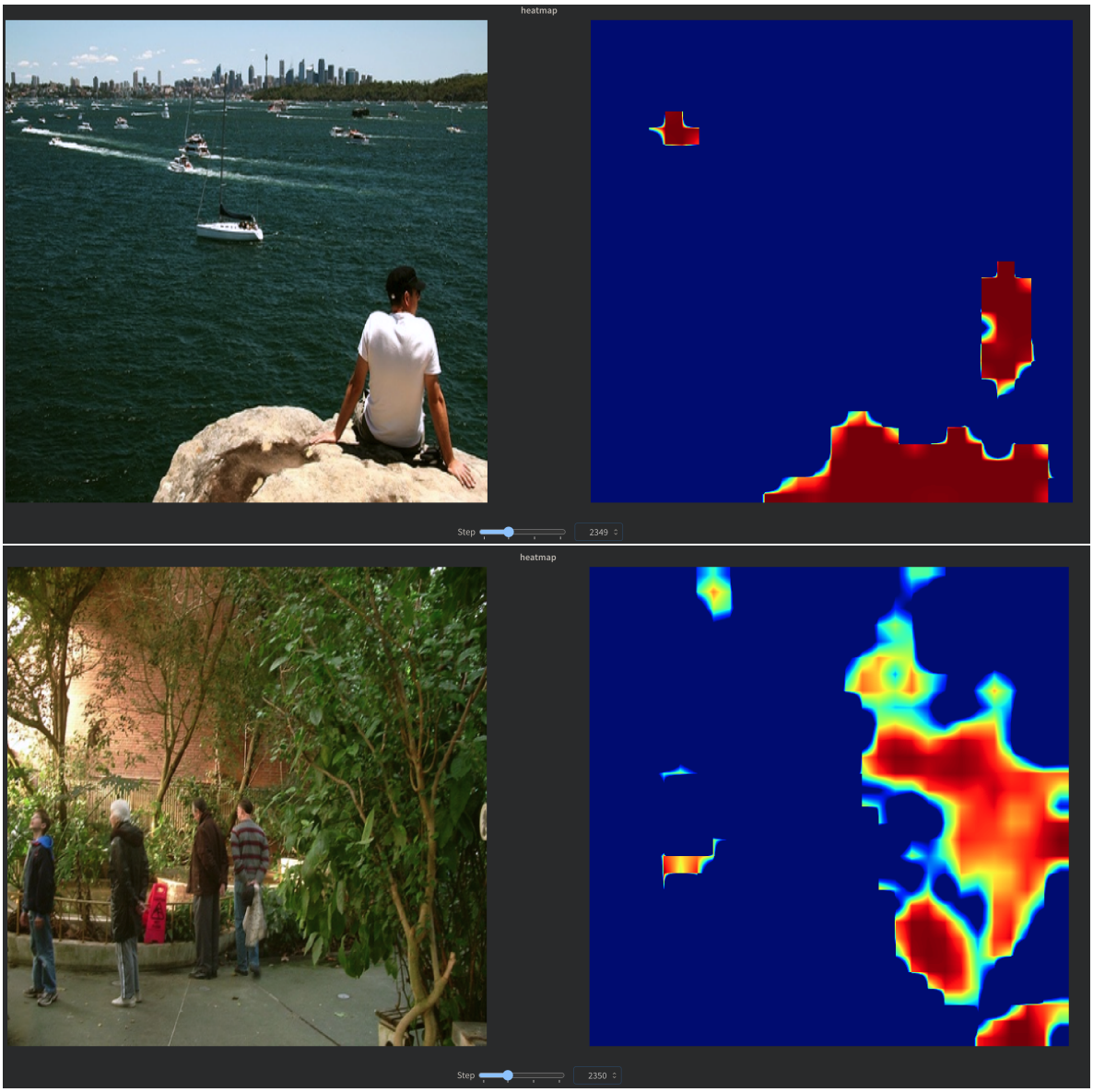


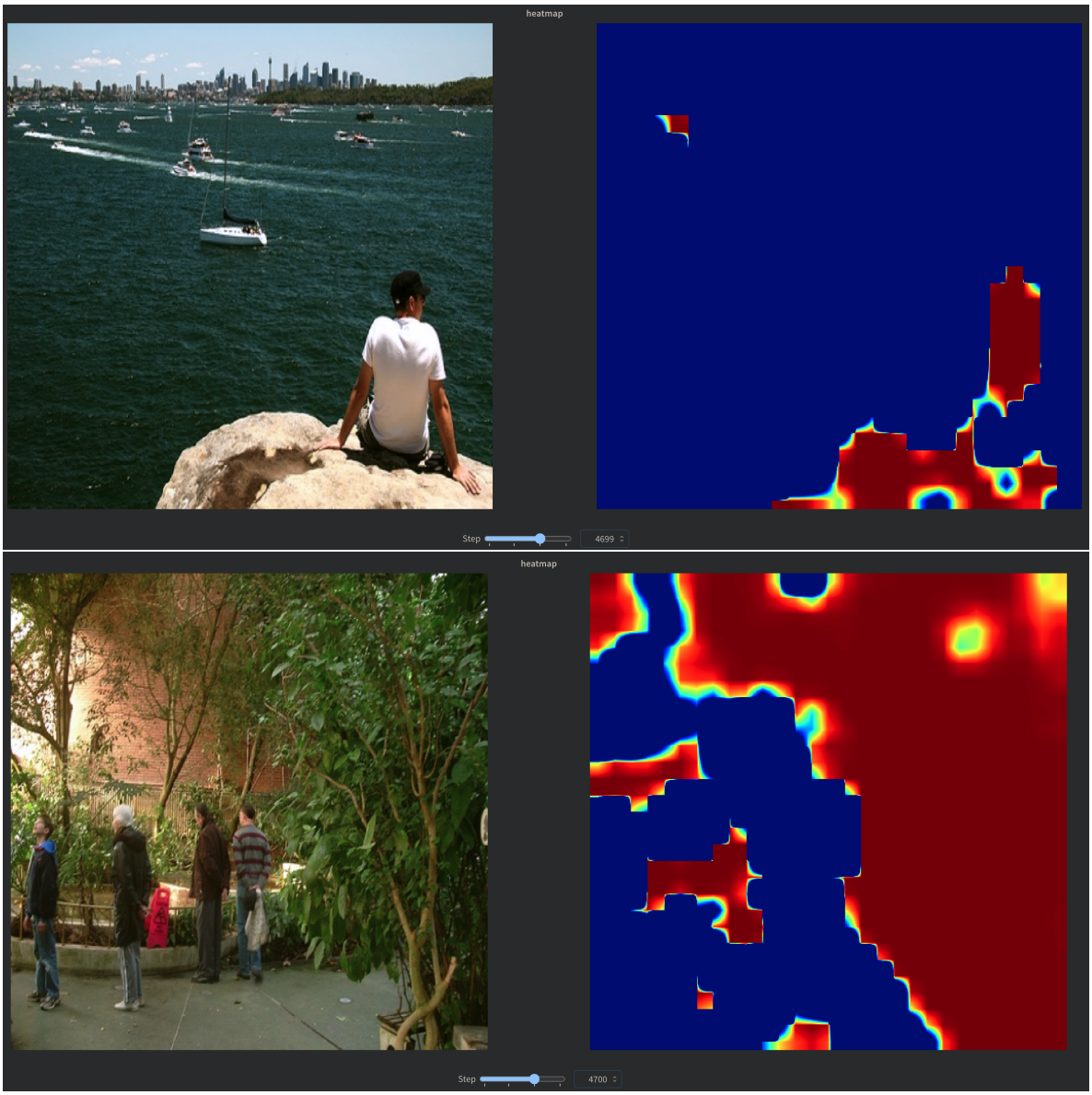
Training loss: 0.09067, metric1: 0.7621, metric2: 0.7839
Validation metric1: 0.4481, metric2: 0.6009

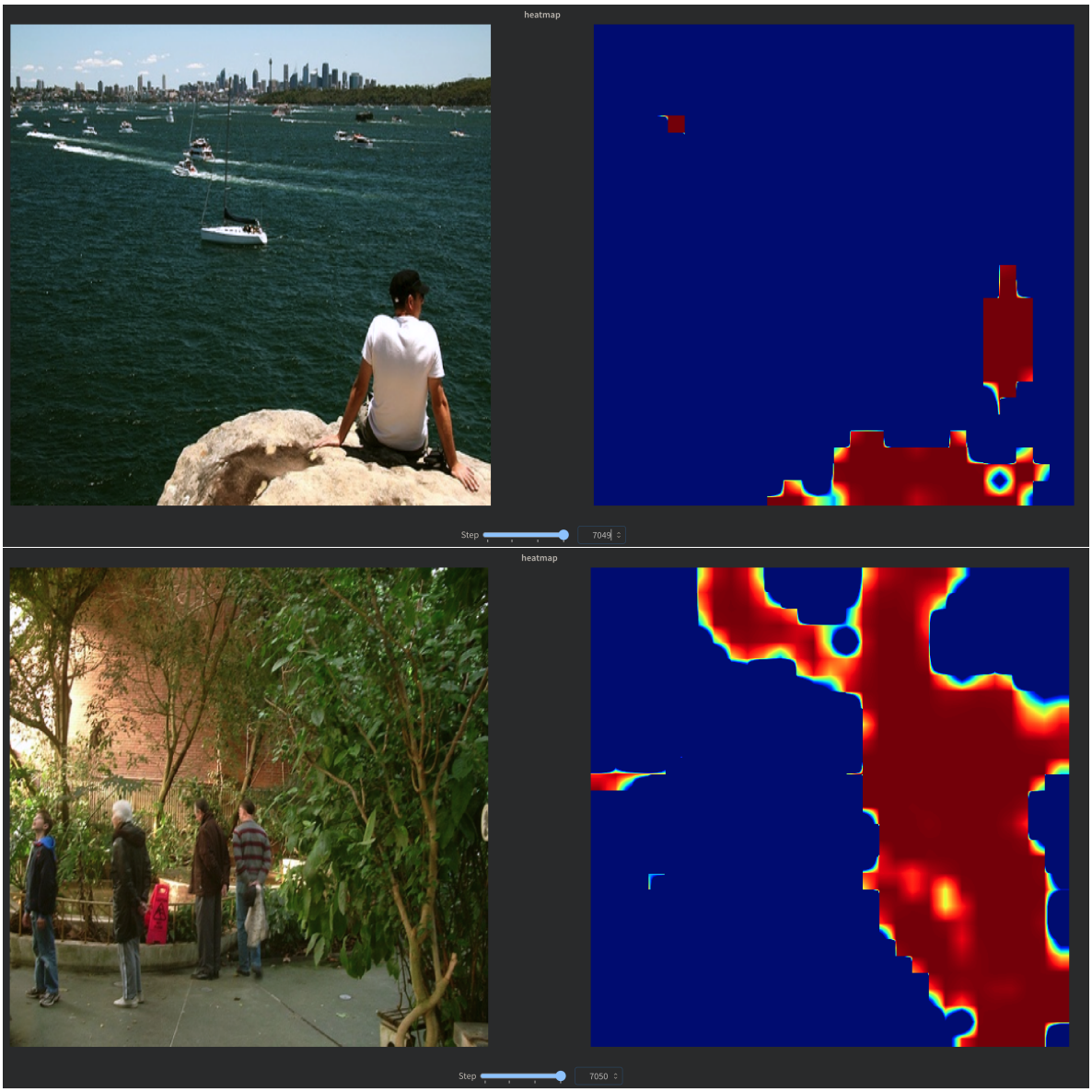
Q 1.7

I use global average pooling at the end to incorporate more salient features.

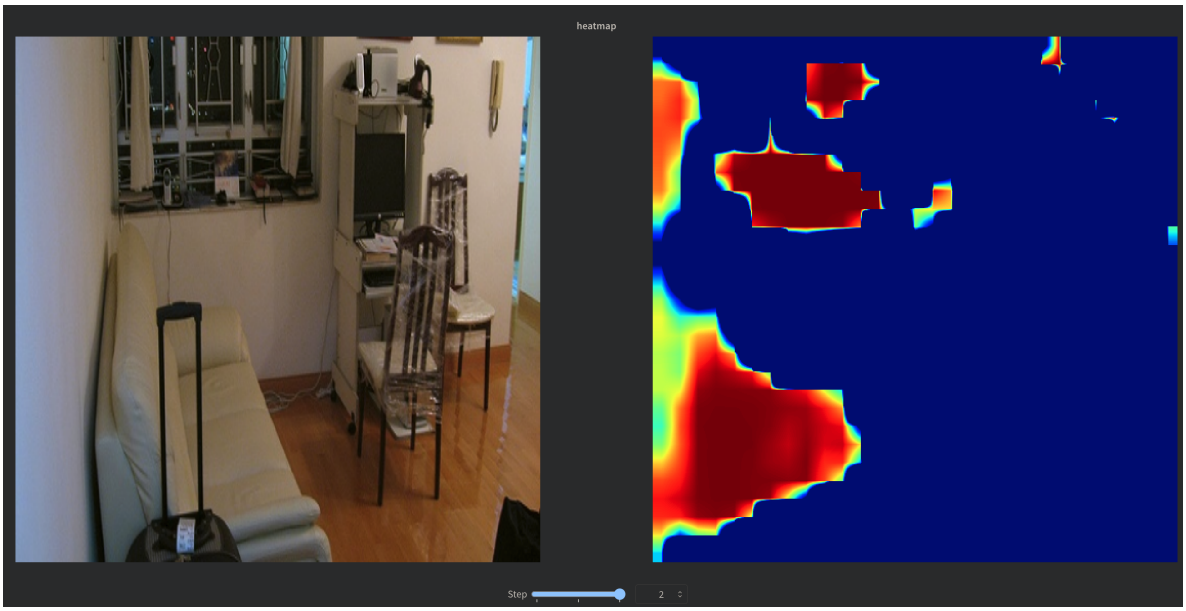



heatmap
Step
2349
heatmap
Step
2350


heatmap
Step
4699
heatmap
Step
4700


heatmap
Step
7049
heatmap
Step
7050


heatmap
Step
0
heatmap
Step
1



Training loss: 0.02455, metric1: 0.9821, metric2: 0.9453
Validation metric1: 0.7845, metric2: 0.7181

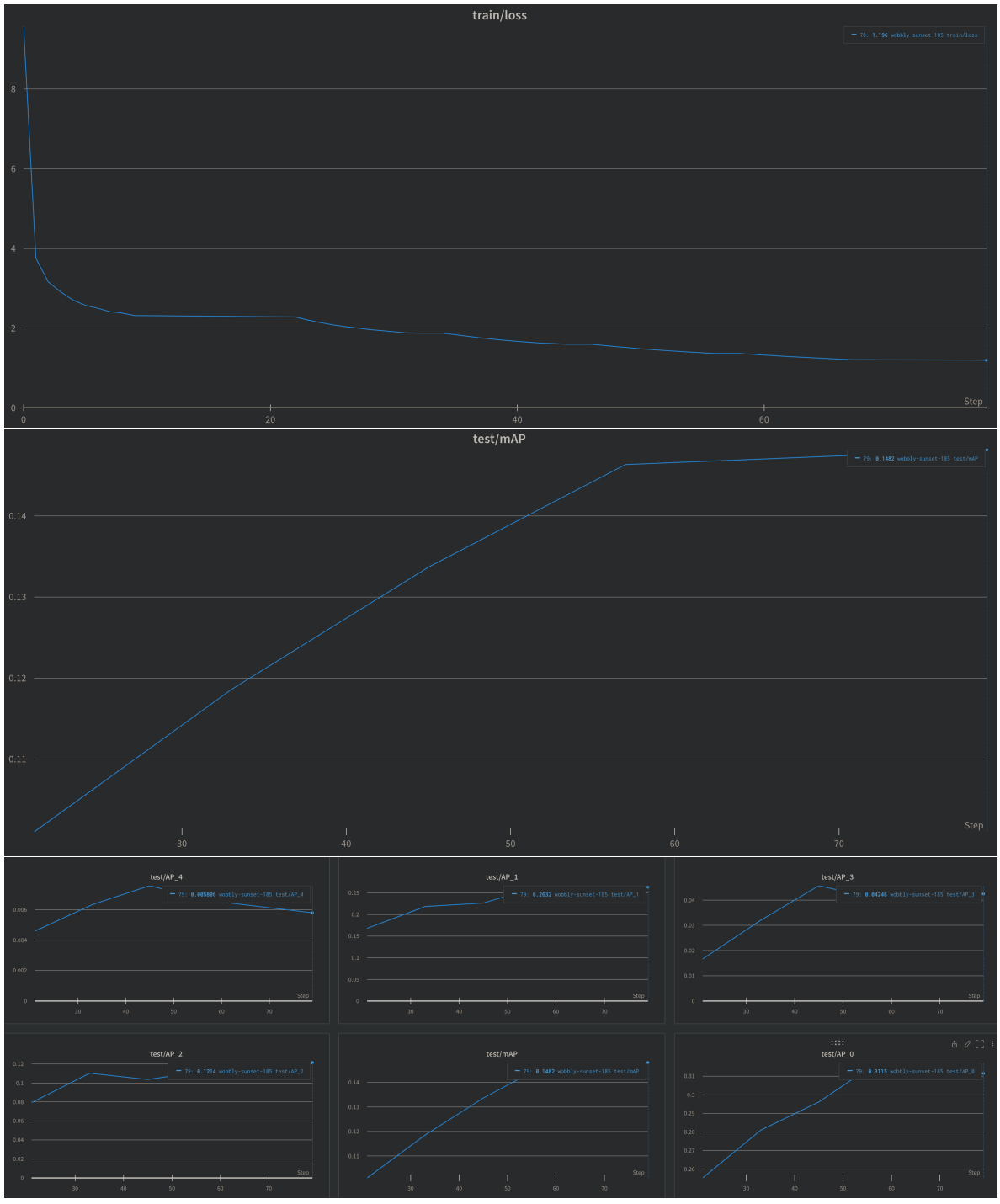
Q2.3

```python
# TODO: given bounding boxes and corresponding scores, perform non max suppression
def nms(bounding_boxes, confidence_score, threshold=0.05):
    """
    bounding boxes of shape     Nx4
    confidence scores of shape  N
    threshold: confidence threshold for boxes to be considered

    return: list of bounding boxes and scores
    """
    # sort bounding boxes based on confidence score
    pairs = [x for x in list(zip(confidence_score, bounding_boxes)) if x[0] > threshold]
    pairs = sorted(pairs, key=lambda x: x[0], reverse=True)
    boxes, scores = [], []
    visited = set()
    # loop to visit all bounding boxes
    while len(visited) < len(pairs):
        idx = 0
        # get the first index that is not visited before
        for i in range(len(pairs)):
            if i in visited:
                continue
            idx = i
            break
        # add this bounding box to output list
        boxes.append(pairs[idx][1])
        scores.append(pairs[idx][0])
        visited.add(idx)
        # loop over other bounding boxes to remove overlapping bboxes
        for i in range(len(pairs)):
            if i in visited:
                continue
            # ignore overlapping bboxes with iou > 0.3
            if iou(pairs[idx][1], pairs[i][1]) > 0.3:
                visited.add(i)

    return boxes, scores

# TODO: calculate the intersection over union of two boxes
def iou(box1, box2):
    """
    Calculates Intersection over Union for two bounding boxes (xmin, ymin, xmax, ymax)
    returns IoU vallue
    """
    # get the boarder coordiantes of union of two bboxes
    x_left = max(box1[0], box2[0])
    y_top = max(box1[1], box2[1])
    x_right = min(box1[2], box2[2])
```

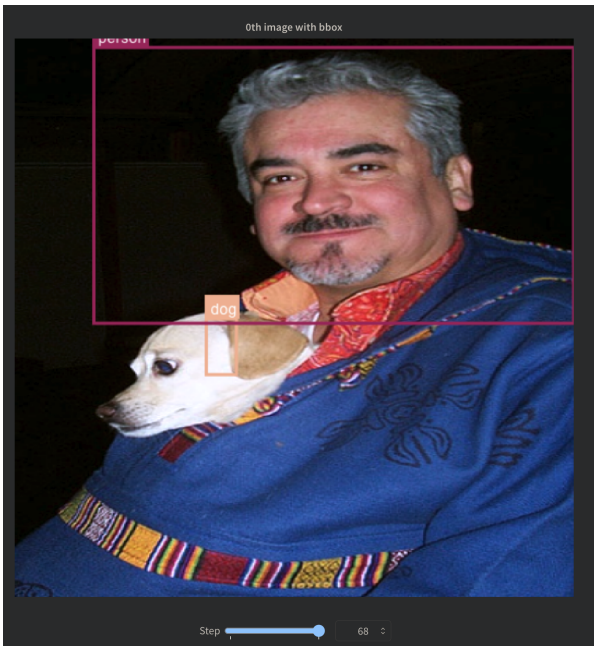
```
    y_bottom = min(box1[3], box2[3])

    # area is zero if two bboxes cannot be unioned
    if x_right < x_left or y_bottom < y_top:
        return 0.0

    # calculate intersection area
    intersection = (x_right - x_left) * (y_bottom - y_top)
    area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
    area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])
    # iou is intersection divided by union (addition of areas of two bboxes minus
    ↪ intersection)
    iou = intersection / (area1 + area2 - intersection)
    return iou

res = [[], []]
# TODO (Q2.3): Iterate over each class (follow comments)
labels = []
bboxes = []
for class_num in range(20):
    # get valid rois and cls_scores based on thresh
    # use NMS to get boxes and scores
    boxes, scores = nms(rois.cpu(), cls_prob[:, class_num].cpu(), thresh)
    if boxes and scores:
        for i in range(len(boxes)):
            res[0].append((scores[i], boxes[i], class_num, iter))
            if args.use_wandb and (epoch == 0 or epoch == 4) and (cnt < 10 or iter
    ↪ in logged):
                labels.append(class_num)
                bboxes.append(boxes[i].detach().numpy().astype(float))
res[1].append([iter, gt_boxes, gt_class_list])

def calculate_map(res):
    """
    Calculate the mAP for classification.
    """
    # TODO (Q2.3): Calculate mAP on test set.
    # Feel free to write necessary function parameters.
    # get predictions and ground truths
    pred = res[0]
    pred = sorted(pred, key=lambda x: x[0], reverse=True)
    gt = res[1]
    # iterate all ground truths to get number of gt_bboxes for each class
    allgtbboxes = [0] * 20
    for i in range(len(gt)):
        for j in range(len(gt[i][2])):
            allgtbboxes[gt[i][2][j]] += 1
    # total for each class
    AP = [0] * 20
    TP = [0] * 20
    FP = [0] * 20
```

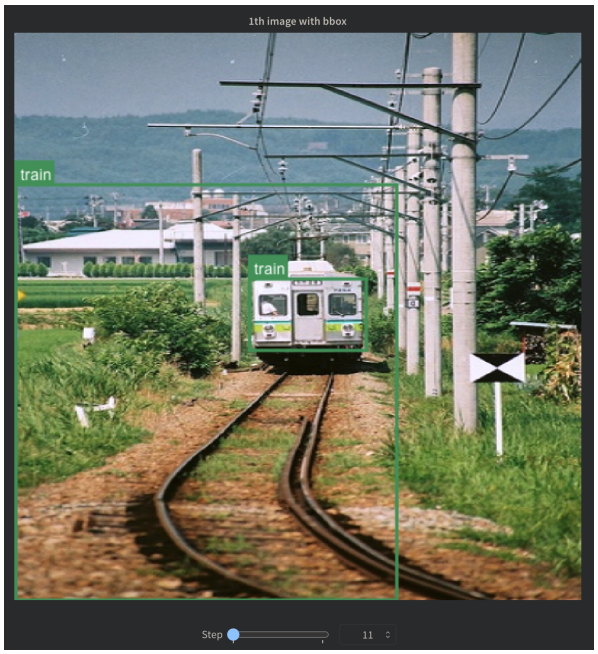
```
    # visited set for each ground truth bboxes list, used for if one gt_bbox is match,
    # it cannot be matched again
    visited = [set() for _ in range(len(gt))]
    # precision and recall list for AP (AUC) calculation for each class
    precision = [[] for _ in range(20)]
    recall = [[] for _ in range(20)]
    # iterate prediction bounding boxes to calculate precision and recall
    for _, box, class_num, iter in pred:
        gt_boxes = gt[iter][1]
        gt_class_list = gt[iter][2]
        assert gt[iter][0] == iter
        assert len(gt_boxes) == len(gt_class_list)
        # if not more gt bboxes to match a prediction, this prediction is a false positive
        if len(gt_boxes) == 0 or len(gt_boxes) == len(visited[iter]):
            FP[class_num] += 1
        else:
            match = False
            # iterate gt bboxes of the image
            for i in range(len(gt_boxes)):
                if i in visited[iter]:
                    continue
                # prediction is true positive if label is matched and iou exceeds threshold
                if class_num == gt_class_list[i] and iou(box, gt_boxes[i]) > 0.45:
                    TP[class_num] += 1
                    visited[iter].add(i)
                    match = True
                    break
            # prediction is not matched to any gt bboxes and is regarded a a false positive
            if not match:
                FP[class_num] += 1

        # add precision and recall to list
        precision[class_num].append(TP[class_num] / (TP[class_num] + FP[class_num]))
        recall[class_num].append(TP[class_num] / allgtbboxes[class_num])
    # using sklearn auc function to calculate AP for each class
    for i in range(20):
        AP[i] = sklearn.metrics.auc(recall[i], precision[i])
    return AP
```

Q2.3


train/loss
test/mAP
test/AP_4
test/AP_1
test/AP_3
test/AP_2
test/mAP
test/AP_0
Step

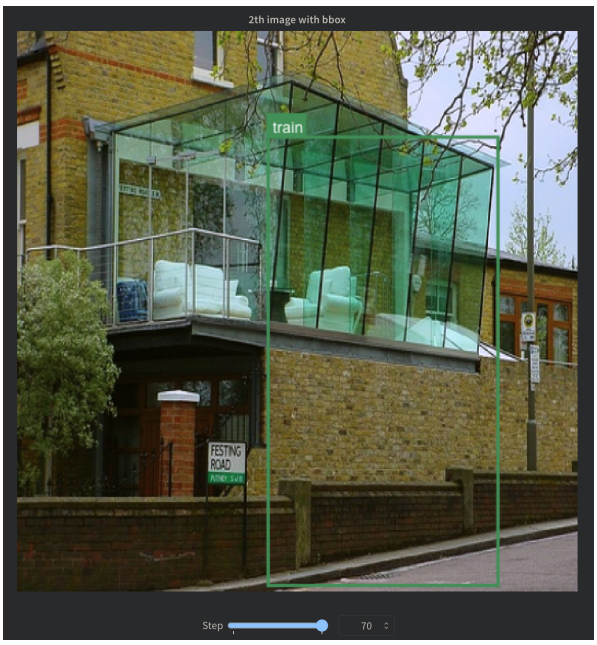

0th image with bbox
person
Step
10


0th image with bbox
person
dog
Step
68

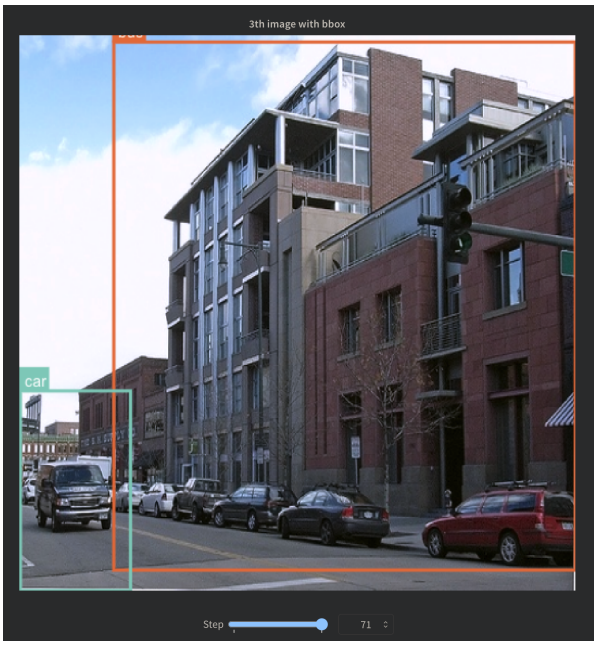

1th image with bbox
train
train
Step
11

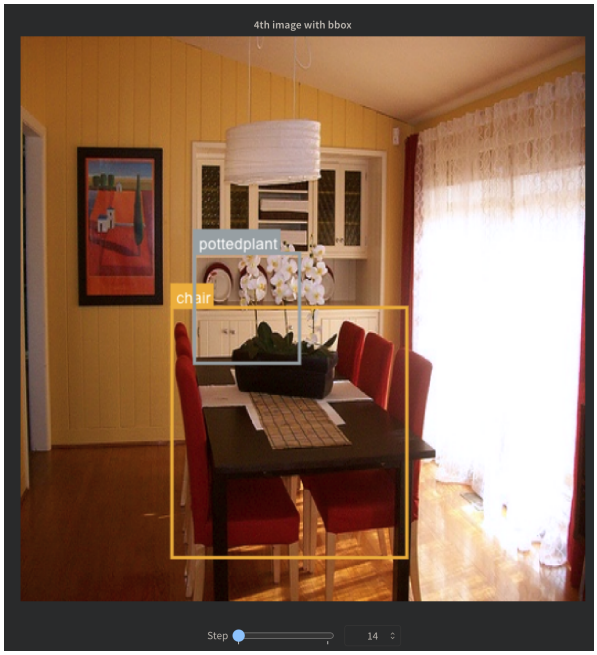

1th image with bbox
train
Step
69

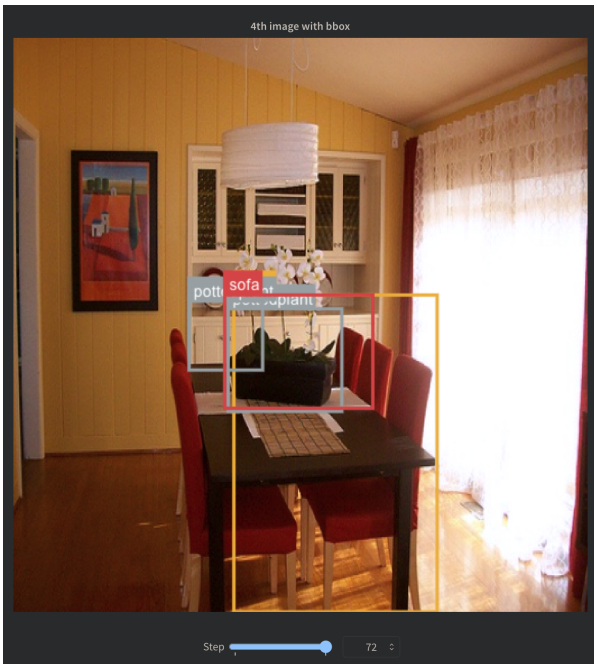

2th image with bbox
train
FESTING ROAD
Step
12

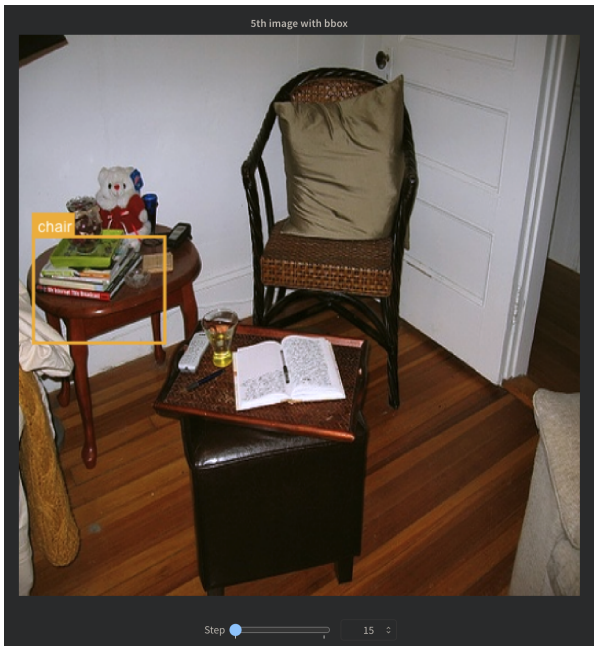

2th image with bbox
train
FESTING ROAD
Step
70


3th image with bbox
person
car
Step
13

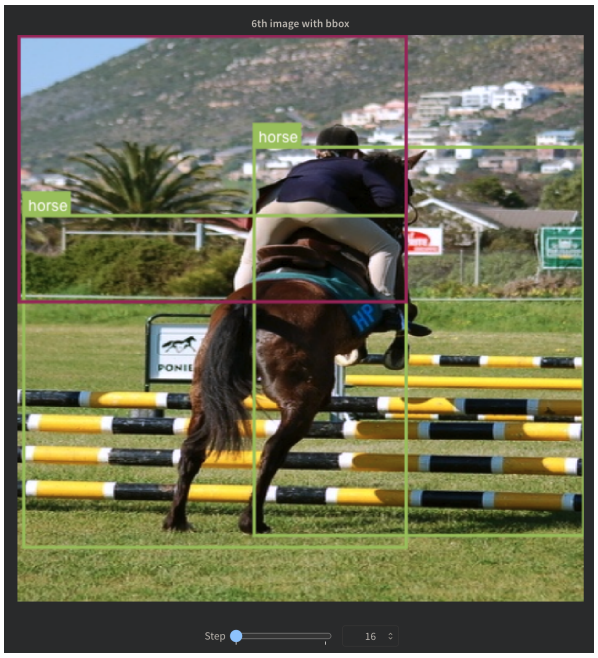

3th image with bbox
car
Step
71

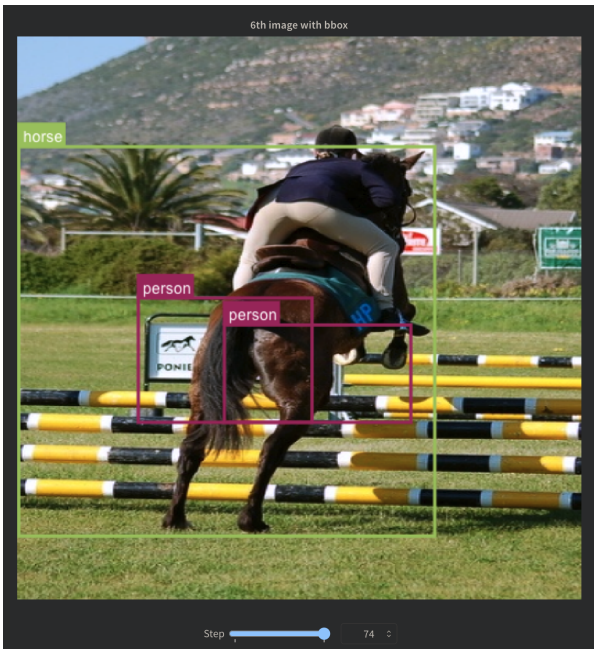

4th image with bbox
pottedplant
chair
Step
14

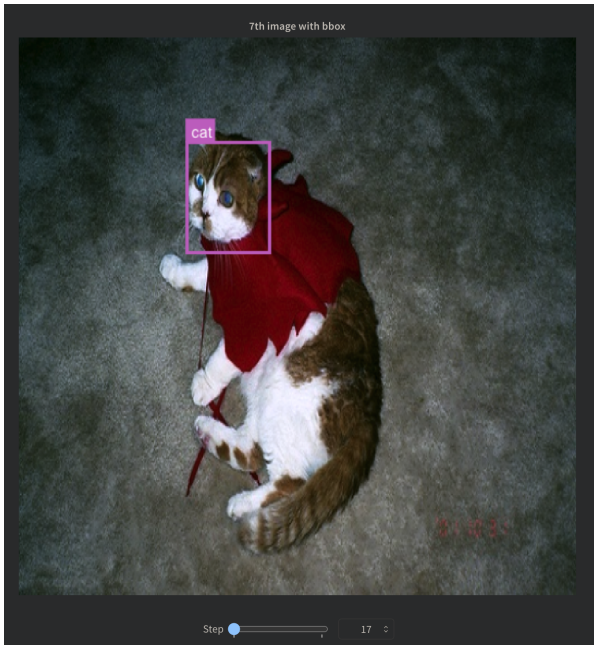

4th image with bbox
sofa
Step
72


5th image with bbox
chair
Step
15

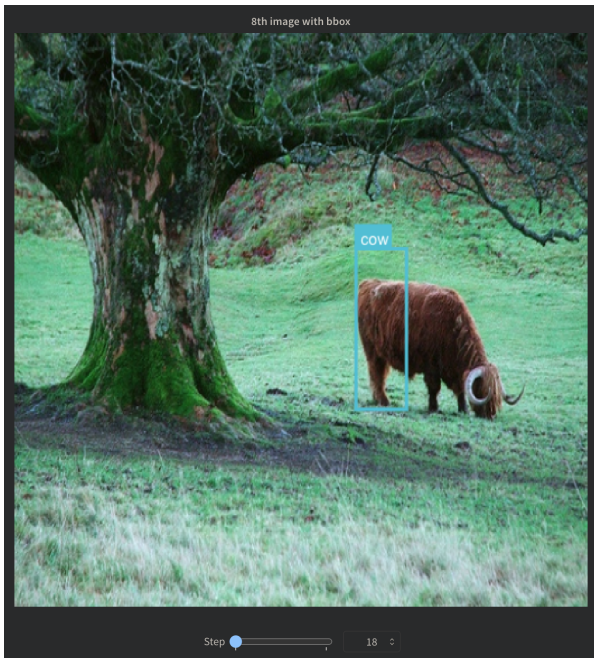

5th image with bbox
person
chair
tvmonitor
Step
73

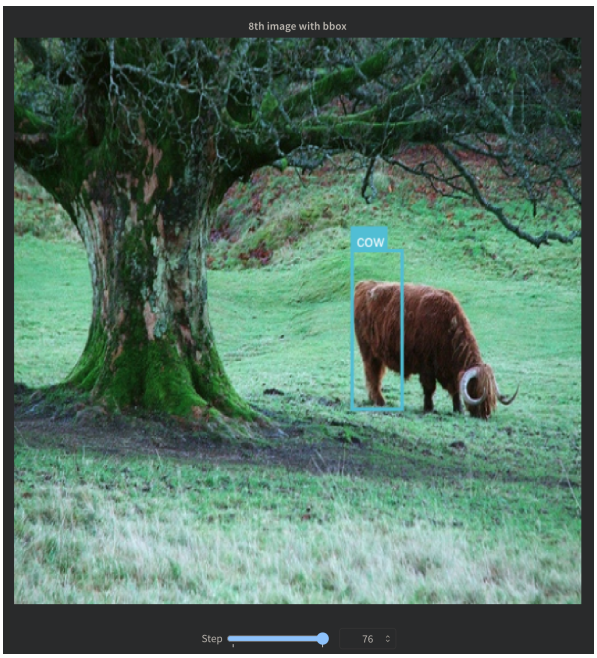

6th image with bbox
horse
horse
Step
16

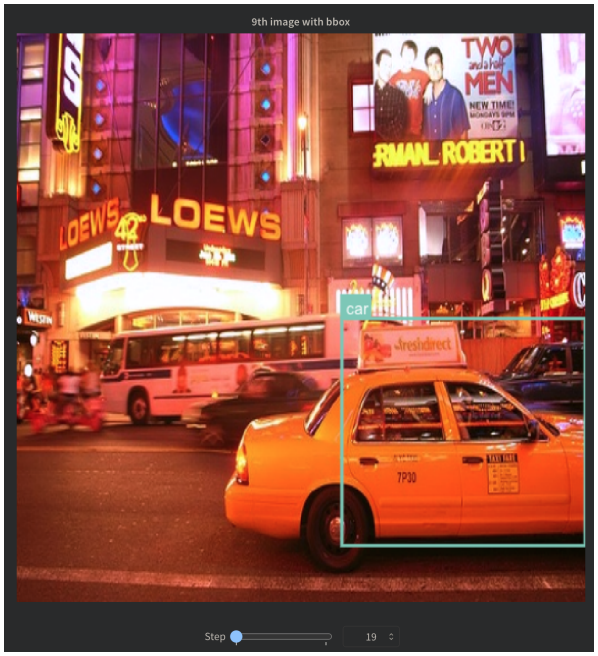

6th image with bbox
horse
person
person
Step
74


7th image with bbox
cat
Step
17


7th image with bbox
cat
Step
75


8th image with bbox
cow
Step
18


8th image with bbox
cow
Step
76


9th image with bbox
TWO and a Half MEN
NEW TIME!
LOEWS
car
freshdirect
Step
19


9th image with bbox
TWO and a Half MEN
NEW TIME!
LOEWS
car
freshdirect
Step
77

final AP: [0.3114676597298536, 0.26322355466182873, 0.12143539348699098, 0.04246064846296949, 0.0058059822798605994, 0.26113716801143544, 0.21391140685913415, 0.18842382856393336, 0.01045631626372508, 0.08239310206964146, 0.05043815832782442, 0.1562528503884718, 0.28339286269794695, 0.2778879876497678, 0.035609007110676896, 0.04869998075540583, 0.06565027043847324, 0.05325066230271974, 0.3421963485568527, 0.14914270503358934]
final mAP: 0.14816179468255505