YOLO

October 27, 2024

[26]:
```
#st124092@ait.asia
```

[1]:
```
import torch
import torch.nn as nn
```

```
/home/st124092/work/cuda116/.venv/lib/python3.8/site-packages/tqdm/auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

[2]:
```
'cuda' if torch.cuda.is_available() else'cpu'
```

[2]:
```
'cuda'
```

1 YOLO ARCHITECTURE

Image Reference: https://www.datacamp.com/blog/yolo-object-detection-explained

The following code is inspired and taken from this repo : https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/object_detection/Y

1.1 Lets build our model

[3]:
```
"""
Information about architecture config:
Tuple is structured by (kernel_size, filters, stride, padding)
"M" is simply maxpooling with stride 2x2 and kernel 2x2
List is structured by tuples and lastly int with number of repeats
"""

architecture_config = [
    (7, 64, 2, 3),
    "M",
    (3, 192, 1, 1),
    "M",
    (1, 128, 1, 0),
    (3, 256, 1, 1),
    (1, 256, 1, 0),
```

```
    (3, 512, 1, 1),
    "M",
    [(1, 256, 1, 0), (3, 512, 1, 1), 4], # list architecture
    (1, 512, 1, 0),
    (3, 1024, 1, 1),
    "M",
    [(1, 512, 1, 0), (3, 1024, 1, 1), 2],
    (3, 1024, 1, 1),
    (3, 1024, 2, 1),
    (3, 1024, 1, 1),
    (3, 1024, 1, 1),
]
```

[4]:

```
class CNNBlock(nn.Module):
    def __init__(self, in_channels, out_channels, **kwargs):
        super(CNNBlock, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, bias=False, **kwargs)
        self.batchnorm = nn.BatchNorm2d(out_channels)           # Not used in␣
 ↪original implementation!
        self.leakyrelu = nn.LeakyReLU(0.1)

    def forward(self,x):
        return self.leakyrelu(self.batchnorm(self.conv(x)))
```

[5]:

```
class Yolov1(nn.Module):
    def __init__(self, in_channels=3, **kwargs):
        super(Yolov1, self).__init__()
        self.architecture = architecture_config
        self.in_channels = in_channels
        self.darknet = self._create_conv_layers(self.architecture)
        self.fcs = self._create_fc(**kwargs)

    def forward(self,x):
        x = self.darknet(x)
        return self.fcs(torch.flatten(x, start_dim=1))

    def _create_conv_layers(self, architecture):
        layers = []
        in_channels = self.in_channels

        for x in architecture:
            if type(x) == tuple:
                layers += [
                    CNNBlock(in_channels, out_channels = x[1],␣
 ↪kernel_size=x[0], stride=x[2], padding=x[3],)]
                in_channels = x[1]
```

```
            if type(x) == str:
                layers += [
                    nn.MaxPool2d(kernel_size=2, stride=2)]

            if type(x) == list:
                conv1 = x[0]
                conv2 = x[1]
                num_repeats = x[2]

                for _ in range(num_repeats):
                    layers += [
                        CNNBlock(in_channels, out_channels = conv1[1],␣
↪kernel_size= conv1[0], stride=conv1[2], padding=conv1[3],)]
                    layers += [
                        CNNBlock(conv1[1], out_channels = conv2[1],␣
↪kernel_size= conv2[0], stride=conv2[2], padding=conv2[3],)]
                    in_channels = conv2[1]
        return nn.Sequential(*layers)

    def _create_fc(self, split_size, num_boxes, num_classes):
        S,B,C = split_size, num_boxes, num_classes
        return nn.Sequential(
            nn.Flatten(),
            nn.Linear(1024*S*S , 4096),
            nn.Dropout(0.5),                          # not implemented in paper
            nn.LeakyReLU(0.1),
            nn.Linear(4096, S*S*(C + B*5)),            # C+5*B = 30
        )
```

[6]:
```
# Test our model

model = Yolov1(split_size=7, num_boxes=2, num_classes=20)
print(model)
```

```
Yolov1(
  (darknet): Sequential(
    (0): CNNBlock(
      (conv): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
      (batchnorm): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (2): CNNBlock(
      (conv): Conv2d(64, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
```

```
bias=False)
      (batchnorm): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (4): CNNBlock(
      (conv): Conv2d(192, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batchnorm): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (5): CNNBlock(
      (conv): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (6): CNNBlock(
      (conv): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batchnorm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (7): CNNBlock(
      (conv): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (9): CNNBlock(
      (conv): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batchnorm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (10): CNNBlock(
      (conv): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
```

```
    (11): CNNBlock(
      (conv): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batchnorm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (12): CNNBlock(
      (conv): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (13): CNNBlock(
      (conv): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batchnorm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (14): CNNBlock(
      (conv): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (15): CNNBlock(
      (conv): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batchnorm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (16): CNNBlock(
      (conv): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (17): CNNBlock(
      (conv): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (18): CNNBlock(
      (conv): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
```

```
      (batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (20): CNNBlock(
      (conv): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (21): CNNBlock(
      (conv): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (22): CNNBlock(
      (conv): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (batchnorm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (23): CNNBlock(
      (conv): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (24): CNNBlock(
      (conv): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (25): CNNBlock(
      (conv): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (26): CNNBlock(
      (conv): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
```

```
1), bias=False)
      (batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
    (27): CNNBlock(
      (conv): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (batchnorm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (leakyrelu): LeakyReLU(negative_slope=0.1)
    )
  )
  (fcs): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=50176, out_features=4096, bias=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): LeakyReLU(negative_slope=0.1)
    (4): Linear(in_features=4096, out_features=1470, bias=True)
  )
)
```

[7]:

```
x = torch.randn(2, 3, 448, 448)
print(model(x).shape)                          # 7*7*30 = 1470
```

```
torch.Size([2, 1470])
```

1.2 Lets build our Loss Function

[8]:

```
from dataset import VOCDataset
from utils import intersection_over_union
```

[9]:

```
class YoloLoss(nn.Module):
    """
    Calculate the loss for yolo (v1) model
    """

    def __init__(self, S=7, B=2, C=20):
        super(YoloLoss, self).__init__()
        self.mse = nn.MSELoss(reduction="sum")

        """
        S is split size of image (in paper 7),
        B is number of boxes (in paper 2),
        C is number of classes (in paper and VOC dataset is 20),
        """
        self.S = S
        self.B = B
```

```
        self.C = C

        # These are from Yolo paper, signifying how much we should
        # pay loss for no object (noobj) and the box coordinates (coord)
        self.lambda_noobj = 0.5
        self.lambda_coord = 5

    def forward(self, predictions, target):
        # predictions are shaped (BATCH_SIZE, S*S(C+B*5) when inputted
        predictions = predictions.reshape(-1, self.S, self.S, self.C + self.B *␣
↪5)

        # Calculate IoU for the two predicted bounding boxes with target bbox
        iou_b1 = intersection_over_union(predictions[..., 21:25], target[...,␣
↪21:25])
        iou_b2 = intersection_over_union(predictions[..., 26:30], target[...,␣
↪21:25])
        ious = torch.cat([iou_b1.unsqueeze(0), iou_b2.unsqueeze(0)], dim=0)

        # Take the box with highest IoU out of the two prediction
        # Note that bestbox will be indices of 0, 1 for which bbox was best
        iou_maxes, bestbox = torch.max(ious, dim=0)
        exists_box = target[..., 20].unsqueeze(3)  # in paper this is Iobj_i

        # ======================== #
        #   FOR BOX COORDINATES    #
        # ======================== #

        # Set boxes with no object in them to 0. We only take out one of the␣
↪two
        # predictions, which is the one with highest Iou calculated previously.
        box_predictions = exists_box * (
            (
                bestbox * predictions[..., 26:30]
                + (1 - bestbox) * predictions[..., 21:25]
            )
        )

        box_targets = exists_box * target[..., 21:25]

        # Take sqrt of width, height of boxes to ensure that
        box_predictions[..., 2:4] = torch.sign(box_predictions[..., 2:4]) *␣
↪torch.sqrt(
            torch.abs(box_predictions[..., 2:4] + 1e-6)
        )
        box_targets[..., 2:4] = torch.sqrt(box_targets[..., 2:4])
```

```python
        box_loss = self.mse(
            torch.flatten(box_predictions, end_dim=-2),
            torch.flatten(box_targets, end_dim=-2),
        )

        # ==================== #
        #   FOR OBJECT LOSS    #
        # ==================== #

        # pred_box is the confidence score for the bbox with highest IoU
        pred_box = (
            bestbox * predictions[..., 25:26] + (1 - bestbox) * predictions[...
↪, 20:21]
        )

        object_loss = self.mse(
            torch.flatten(exists_box * pred_box),
            torch.flatten(exists_box * target[..., 20:21]),
        )

        # ====================== #
        #   FOR NO OBJECT LOSS    #
        # ====================== #

        #max_no_obj = torch.max(predictions[..., 20:21], predictions[..., 25:
↪26])
        #no_object_loss = self.mse(
        #    torch.flatten((1 - exists_box) * max_no_obj, start_dim=1),
        #    torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1),
        #)

        no_object_loss = self.mse(
            torch.flatten((1 - exists_box) * predictions[..., 20:21],␣
↪start_dim=1),
            torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1),
        )

        no_object_loss += self.mse(
            torch.flatten((1 - exists_box) * predictions[..., 25:26],␣
↪start_dim=1),
            torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1)
        )

        # ================== #
        #   FOR CLASS LOSS   #
        # ================== #
```

```
        class_loss = self.mse(
            torch.flatten(exists_box * predictions[..., :20], end_dim=-2,),
            torch.flatten(exists_box * target[..., :20], end_dim=-2,),
        )

        loss = (
            self.lambda_coord * box_loss  # first two rows in paper
            + object_loss  # third row in paper
            + self.lambda_noobj * no_object_loss  # forth row
            + class_loss  # fifth row
        )

        return loss
```

[10]:
```
import torchvision.transforms as transforms
import torch.optim as optim
import torchvision.transforms.functional as FT
from tqdm import tqdm
from torch.utils.data import DataLoader
from utils import (
    non_max_suppression,
    mean_average_precision,
    intersection_over_union,
    cellboxes_to_boxes,
    get_bboxes,
    plot_image,
    save_checkpoint,
    load_checkpoint,
)
```

[11]:
```
seed = 123
torch.manual_seed(seed)
```

[11]: <torch._C.Generator at 0x7fee807d5610>

[12]:
```
# Hyperparameters etc.
LEARNING_RATE = 2e-5
DEVICE = "cuda" if torch.cuda.is_available else "cpu"
BATCH_SIZE = 16 # 64 in original paper but I don't have that much vram, grad␣
↪accum?
WEIGHT_DECAY = 0
EPOCHS = 10
NUM_WORKERS = 2
PIN_MEMORY = True
LOAD_MODEL = False
LOAD_MODEL_FILE = "best_so_far.pth.tar"
IMG_DIR = "dataset/images"
```

```
LABEL_DIR = "dataset/labels"
```

[13]:
```
class Compose(object):
    def __init__(self, transforms):
        self.transforms = transforms

    def __call__(self, img, bboxes):
        for t in self.transforms:
            img, bboxes = t(img), bboxes

        return img, bboxes


transform = Compose([transforms.Resize((448, 448)), transforms.ToTensor(),])
```

[14]:
```
def train_fn(train_loader, model, optimizer, loss_fn):
    loop = tqdm(train_loader, leave=True)
    mean_loss = []

    for batch_idx, (x, y) in enumerate(loop):
        x, y = x.to(DEVICE), y.to(DEVICE)
        out = model(x)
        loss = loss_fn(out, y)
        mean_loss.append(loss.item())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # update progress bar
        loop.set_postfix(loss=loss.item())

    print(f"Mean loss was {sum(mean_loss)/len(mean_loss)}")
```

[15]:
```
def main():
    model = Yolov1(split_size=7, num_boxes=2, num_classes=20).to(DEVICE)
    optimizer = optim.Adam(
        model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY
    )
    loss_fn = YoloLoss()

    if LOAD_MODEL:
        load_checkpoint(torch.load(LOAD_MODEL_FILE), model, optimizer)

    train_dataset = VOCDataset(
        "dataset/100examples.csv",
        transform=transform,
        img_dir=IMG_DIR,
```

```
        label_dir=LABEL_DIR,
    )

    test_dataset = VOCDataset(
        "dataset/test.csv", transform=transform, img_dir=IMG_DIR,␣
↪label_dir=LABEL_DIR,
    )

    train_loader = DataLoader(
        dataset=train_dataset,
        batch_size=BATCH_SIZE,
        num_workers=NUM_WORKERS,
        pin_memory=PIN_MEMORY,
        shuffle=True,
        drop_last=True,
    )

    test_loader = DataLoader(
        dataset=test_dataset,
        batch_size=BATCH_SIZE,
        num_workers=NUM_WORKERS,
        pin_memory=PIN_MEMORY,
        shuffle=True,
        drop_last=True,
    )

    for epoch in range(EPOCHS):
        pred_boxes, target_boxes = get_bboxes(
            train_loader, model, iou_threshold=0.5, threshold=0.4
        )

        mean_avg_prec = mean_average_precision(
            pred_boxes, target_boxes, iou_threshold=0.5, box_format="midpoint"
        )
        print(f"Train mAP: {mean_avg_prec}")

        #if mean_avg_prec > 0.9:
        #    checkpoint = {
        #        "state_dict": model.state_dict(),
        #        "optimizer": optimizer.state_dict(),
        #    }
        #    save_checkpoint(checkpoint, filename=LOAD_MODEL_FILE)
        #    import time
        #    time.sleep(10)

        train_fn(train_loader, model, optimizer, loss_fn)
```

[16]:
```
main()
```

```
Train mAP: 0.0
100%|          | 6/6 [00:02<00:00,  2.02it/s, loss=715]
Mean loss was 889.3602294921875


Train mAP: 0.0
100%|          | 6/6 [00:02<00:00,  2.56it/s, loss=395]
Mean loss was 537.6829884847006


Train mAP: 0.0
100%|          | 6/6 [00:02<00:00,  2.43it/s, loss=503]
Mean loss was 478.6702575683594


Train mAP: 0.0
100%|          | 6/6 [00:02<00:00,  2.62it/s, loss=526]
Mean loss was 435.08201090494794


Train mAP: 0.0
100%|          | 6/6 [00:02<00:00,  2.64it/s, loss=306]
Mean loss was 368.8504587809245


Train mAP: 6.61375597701408e-05
100%|          | 6/6 [00:02<00:00,  2.40it/s, loss=281]
Mean loss was 321.43519083658856


Train mAP: 3.561253106454387e-05
100%|          | 6/6 [00:02<00:00,  2.28it/s, loss=202]
Mean loss was 282.7339324951172


Train mAP: 0.0
100%|          | 6/6 [00:02<00:00,  2.21it/s, loss=231]
```

```
Mean loss was 252.1270980834961


Train mAP: 5.733943544328213e-05
100%|          | 6/6 [00:02<00:00,  2.44it/s, loss=265]
Mean loss was 251.42701212565103


Train mAP: 0.00028062198543921113
100%|          | 6/6 [00:02<00:00,  2.23it/s, loss=214]
Mean loss was 229.84151458740234
```

1.2.1 write a test function for validation data,

[17]:
```python
def valid_fn(valid_loader, model, loss_fn):
    model.eval()  # Set model to evaluation mode
    mean_loss = []
    loop = tqdm(valid_loader, leave=True)

    with torch.no_grad():
        for batch_idx, (x, y) in enumerate(loop):
            x, y = x.to(DEVICE), y.to(DEVICE)
            out = model(x)
            loss = loss_fn(out, y)
            mean_loss.append(loss.item())

            # update progress bar
            loop.set_postfix(loss=loss.item())

    print(f"Mean validation loss was {sum(mean_loss)/len(mean_loss)}")
    model.train()
    return sum(mean_loss)/len(mean_loss)
```

1.2.2 train the yolo model with training and validation set on PASCAL VOC data.

[18]:
```python
from sklearn.model_selection import train_test_split
from torch.utils.data import Subset
import torch

full_dataset = VOCDataset(
    "dataset/train.csv",
    transform=transform,
    img_dir=IMG_DIR,
    label_dir=LABEL_DIR,
```

```
)

dataset_size = len(full_dataset)
indices = list(range(dataset_size))
train_indices, val_indices = train_test_split(
    indices,
    test_size=0.2,
    random_state=seed
)

train_dataset = Subset(full_dataset, train_indices)
val_dataset = Subset(full_dataset, val_indices)


total_size = len(full_dataset)
train_size = len(train_dataset)
val_size = len(val_dataset)

print(f"Total dataset size: {total_size}")
print(f"Training set size: {train_size} ({train_size/total_size*100:.1f}%)")
print(f"Validation set size: {val_size} ({val_size/total_size*100:.1f}%)")
```

```
Total dataset size: 16550
Training set size: 13240 (80.0%)
Validation set size: 3310 (20.0%)
```

[19]:
```
def main():
    model = Yolov1(split_size=7, num_boxes=2, num_classes=20).to(DEVICE)
    optimizer = optim.Adam(
        model.parameters(), lr=LEARNING_RATE, weight_decay=WEIGHT_DECAY
    )
    loss_fn = YoloLoss()

    if LOAD_MODEL:
        load_checkpoint(torch.load(LOAD_MODEL_FILE), model, optimizer)

    train_loader = DataLoader(
        dataset=train_dataset,
        batch_size=BATCH_SIZE,
        num_workers=NUM_WORKERS,
        pin_memory=PIN_MEMORY,
        shuffle=True,
        drop_last=True,
    )

    val_loader = DataLoader(
        dataset=val_dataset,
```

```
        batch_size=BATCH_SIZE,
        num_workers=NUM_WORKERS,
        pin_memory=PIN_MEMORY,
        shuffle=False,
        drop_last=True,
    )

    for epoch in range(EPOCHS):
        pred_boxes, target_boxes = get_bboxes(
            train_loader, model, iou_threshold=0.5, threshold=0.4
        )

        mean_avg_prec = mean_average_precision(
            pred_boxes, target_boxes, iou_threshold=0.5, box_format="midpoint"
        )
        print(f"Train mAP: {mean_avg_prec}")

        train_fn(train_loader, model, optimizer, loss_fn)
        val_loss = valid_fn(val_loader, model, loss_fn)

        checkpoint = {
        "state_dict": model.state_dict(),
        "optimizer": optimizer.state_dict(),
        }
        save_checkpoint(checkpoint, filename=LOAD_MODEL_FILE)
    print("Model saved after 10 epochs.")
```

[20]:
```
!export CUDA_VISIBLE_DEVICES=0,1,2,3
```

[21]:
```
main()
```

```
Train mAP: 0.0

100%|      | 827/827 [03:24<00:00,  4.04it/s, loss=192]

Mean loss was 251.90797998135437

100%|      | 206/206 [00:52<00:00,  3.96it/s, loss=163]

Mean validation loss was 163.26945588195207
=> Saving checkpoint
Train mAP: 0.002744792029261589

100%|      | 827/827 [03:02<00:00,  4.53it/s, loss=182]

Mean loss was 179.07935164427383

100%|      | 206/206 [00:44<00:00,  4.64it/s, loss=153]

Mean validation loss was 162.85283397933813
=> Saving checkpoint
```

```
Train mAP: 0.007169117219746113

100%|      | 827/827 [04:43<00:00,  2.92it/s, loss=135]

Mean loss was 161.70346530605255

100%|      | 206/206 [01:13<00:00,  2.82it/s, loss=148]

Mean validation loss was 148.96882344218133
=> Saving checkpoint
Train mAP: 0.012085916474461555

100%|      | 827/827 [04:05<00:00,  3.37it/s, loss=128]

Mean loss was 152.16088289678314

100%|      | 206/206 [01:09<00:00,  2.95it/s, loss=146]

Mean validation loss was 151.80737538013642
=> Saving checkpoint
Train mAP: 0.015234148129820824

100%|      | 827/827 [03:06<00:00,  4.43it/s, loss=136]

Mean loss was 145.9815778074991

100%|      | 206/206 [02:26<00:00,  1.41it/s, loss=139]

Mean validation loss was 144.69658575706111
=> Saving checkpoint
Train mAP: 0.019742552191019058

100%|      | 827/827 [05:48<00:00,  2.38it/s, loss=128]

Mean loss was 142.86766745072612

100%|      | 206/206 [00:53<00:00,  3.82it/s, loss=150]

Mean validation loss was 147.16873750408877
=> Saving checkpoint
Train mAP: 0.015162885189056396

100%|      | 827/827 [03:35<00:00,  3.83it/s, loss=163]

Mean loss was 136.3023274063055

100%|      | 206/206 [00:54<00:00,  3.80it/s, loss=152]

Mean validation loss was 137.13396129793333
=> Saving checkpoint
Train mAP: 0.02364708110690117

100%|      | 827/827 [03:16<00:00,  4.20it/s, loss=85.3]

Mean loss was 130.03348006187355

100%|      | 206/206 [00:45<00:00,  4.55it/s, loss=130]
```

```
Mean validation loss was 122.83234498107318
=> Saving checkpoint
Train mAP: 0.03647088259458542

100%|      | 827/827 [03:09<00:00,  4.36it/s, loss=121]

Mean loss was 124.56289040912596

100%|      | 206/206 [00:54<00:00,  3.79it/s, loss=138]

Mean validation loss was 121.81533954212966
=> Saving checkpoint
Train mAP: 0.033590167760849

100%|      | 827/827 [02:51<00:00,  4.83it/s, loss=112]

Mean loss was 118.8215225754799

100%|      | 206/206 [00:50<00:00,  4.10it/s, loss=133]

Mean validation loss was 136.5670263420031
=> Saving checkpoint
Model saved after 10 epochs.
```

1.2.3 the performance of trained model on test data of PASCAL VOC in terms of mAP.

[22]:
```python
LOAD_MODEL = True
LOAD_MODEL_FILE = "best_so_far.pth.tar"
```

[23]:
```python
model = Yolov1(split_size=7, num_boxes=2, num_classes=20).to(DEVICE)
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE,␣
 ↪weight_decay=WEIGHT_DECAY)

# if LOAD_MODEL:
#     load_checkpoint(torch.load(LOAD_MODEL_FILE), model, optimizer)
#     print("Model loaded successfully from", LOAD_MODEL_FILE)

if LOAD_MODEL:
    checkpoint = torch.load(LOAD_MODEL_FILE)
    load_checkpoint(checkpoint, model, optimizer)  # Use the imported function
    print("Model loaded successfully from", LOAD_MODEL_FILE)

test_dataset = VOCDataset(
    "dataset/test.csv", transform=transform, img_dir=IMG_DIR,␣
 ↪label_dir=LABEL_DIR,
)

test_loader = DataLoader(
    dataset=test_dataset,
    batch_size=BATCH_SIZE,
```

```
    num_workers=NUM_WORKERS,
    pin_memory=PIN_MEMORY,
    shuffle=False,
    drop_last=True,
)
```

```
=> Loading checkpoint
Model loaded successfully from best_so_far.pth.tar
```

[25]:

```
# Define the evaluation function
def evaluate_test_set(test_loader, model, device, iou_threshold=0.5,␣
 ↪conf_threshold=0.4):
    model.eval()
    pred_boxes, target_boxes = [], []

    model = model.to(device)

    with torch.no_grad():
        pred_boxes, target_boxes = get_bboxes(
            test_loader, model, iou_threshold=iou_threshold,␣
 ↪threshold=conf_threshold, device=device
        )

    mean_avg_prec = mean_average_precision(
        pred_boxes, target_boxes, iou_threshold=iou_threshold,␣
 ↪box_format="midpoint"
    )

    print(f"Test Set mAP: {mean_avg_prec:.4f}")

evaluate_test_set(test_loader, model, DEVICE, iou_threshold=0.5,␣
 ↪conf_threshold=0.4)
```

```
Test Set mAP: 0.0251
```