BBM 418 - Computer Vision Laboratory
Assignment #4 - Object Tracking on Videos

Emirhan Topcu
21827899
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
emirhantopcu50@gmail.com

Part 1 - Object Classification and Localization

```python
class ClassificationHead(nn.Module):
    def __init__(self, resnet_model):
        super(ClassificationHead, self).__init__()
        self.resnet_model = torchvision.models.resnet18(pretrained=True)
        for param in self.resnet_model.parameters():
            param.requires_grad = False
        self.resnet_model.fc = nn.Linear(512, 1)

    def forward(self, x):
        x = F.relu(self.resnet_model(x))
        return x


class RegressionHead(nn.Module):
    def __init__(self, resnet_model):
        super(RegressionHead, self).__init__()
        self.resnet_model = resnet_model
        for param in self.resnet_model.parameters():
            param.requires_grad = False
        self.resnet_model.fc = nn.Linear(512, 256)
        self.linear1 = nn.Linear(256, 128)
        self.linear2 = nn.Linear(128, 4)

    def forward(self, x):
        x = F.relu(self.resnet_model(x))
        x = F.relu(self.linear1(x))
        x = self.linear2(x)
        return x
```

I created Classification and Regression heads for the ResNet18 model. Above, you can see the settings I have applied to them.

```python
annotations = pd.read_csv(r'raccoon/train/_annotations.txt', header=None, sep=" ", on_bad_lines='skip')

annotations = annotations.reindex(index=order_by_index(annotations.index,
                                                       index_natsorted(annotations[0], reverse=False)))
annotations.reset_index(drop=True, inplace=True)


train_dataset = RaccoonDataset(annotations, 'raccoon/train', IMAGE_RESIZE, transform=train_transform)
train_dataloader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True
)

annotations_test = pd.read_csv(r'raccoon/test/_annotations.txt', header=None, sep=" ", on_bad_lines='skip')

annotations_test = annotations_test.reindex(index=order_by_index(annotations_test.index,
                                                       index_natsorted(annotations_test[0], reverse=False)))
annotations_test.reset_index(drop=True, inplace=True)

test_dataset = RaccoonDataset(annotations_test, 'raccoon/test', IMAGE_RESIZE, transform=train_transform)
test_dataloader = DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True
)

annotations_test = pd.read_csv(r'raccoon/valid/_annotations.txt', header=None, sep=" ", on_bad_lines='skip')
```

I prepared the data loaders for train, validation and test sets.

```python
class RaccoonDataset(Dataset):
    def __init__(self, annotations_df, root_dir, image_resize, transform=None):
        self.annotations = annotations_df
        self.root_dir = root_dir
        self.transform = transform
        self.image_names = self.annotations[:][0]
        self.labels = self.annotations[:][1]
        self.image_resize = image_resize

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

    def __getitem__(self, index):
        path = self.root_dir + "/" + self.image_names.iloc[index]
        image = cv2.imread(path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        shape_y = image.shape[0]
        shape_x = image.shape[1]

        image = self.transform(image)
        targets = self.labels[index].split(",")[:-1]
        object_class = int(self.labels[index].split(",")[-1])
        targets = [int(target) for target in targets]
        targets[0] = int(targets[0] * (self.image_resize/shape_x))
        targets[1] = int(targets[1] * (self.image_resize/shape_y))
        targets[2] = int(targets[2] * (self.image_resize/shape_x))
        targets[3] = int(targets[3] * (self.image_resize/shape_y))
        targets = torch.tensor(targets)
        object_class = torch.tensor(object_class)
        sample = (image, targets, object_class)

        return sample
```

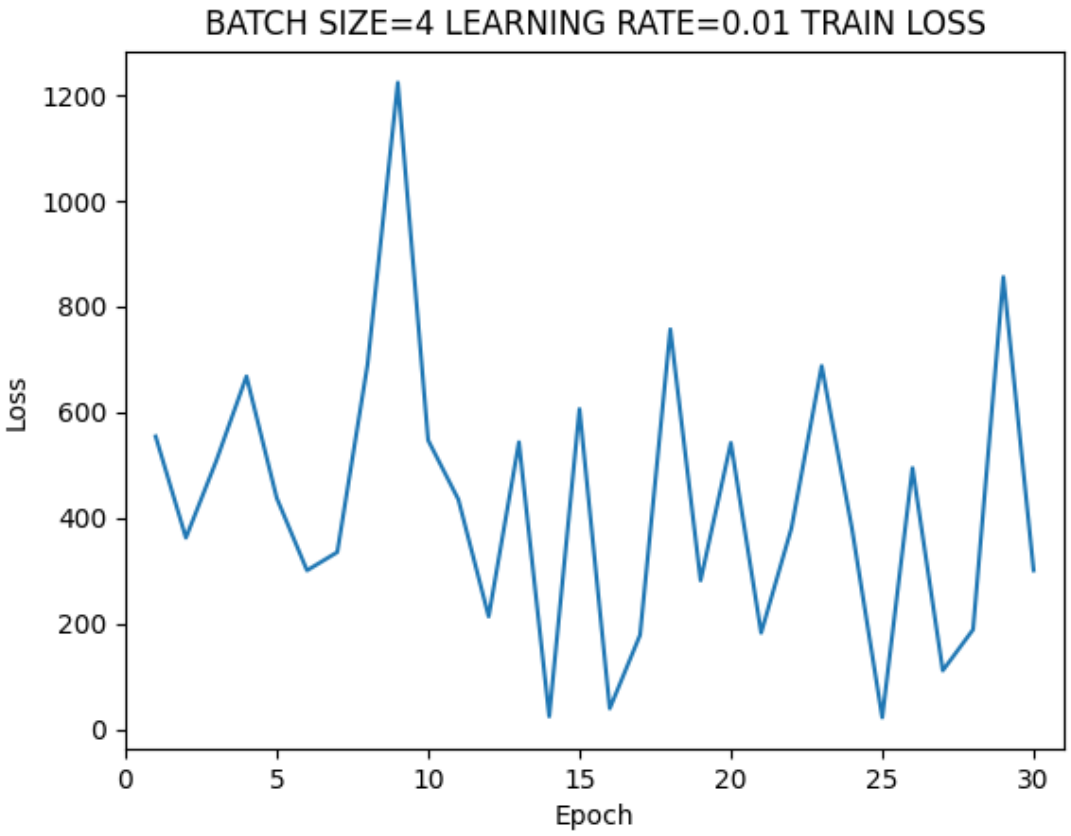
Also here is the custom dataset I created for this Raccoon dataset.

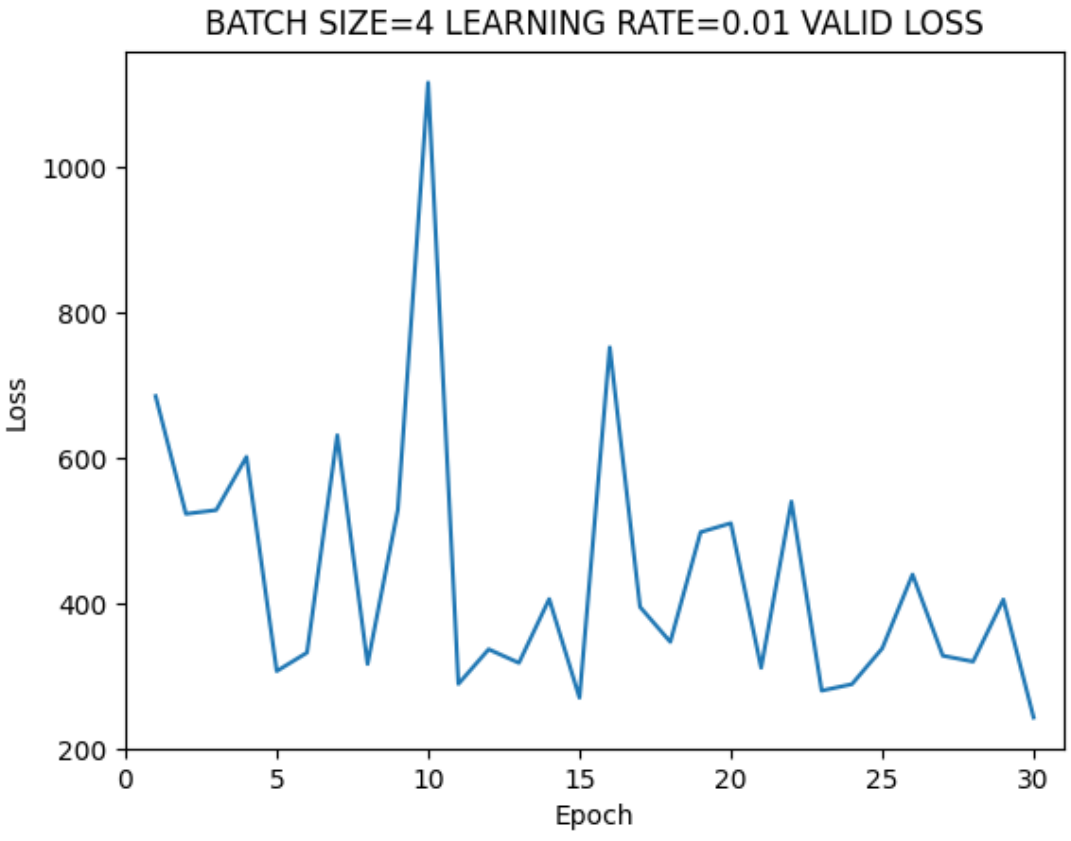
```
init_resnet_model = torchvision.models.resnet18(pretrained=True)
init_resnet_model2 = torchvision.models.resnet18(pretrained=True)
modelC = ClassificationHead(init_resnet_model).to(device)
modelR = RegressionHead(init_resnet_model).to(device)
modelR.to(device)
optimizerSM = torch.optim.SGD(modelC.parameters(), lr=LEARNING_RATE)
optimizerL2 = torch.optim.Adam(modelR.parameters(), lr=LEARNING_RATE)
criterionSM = nn.CrossEntropyLoss()
criterionL2 = nn.MSELoss()
```

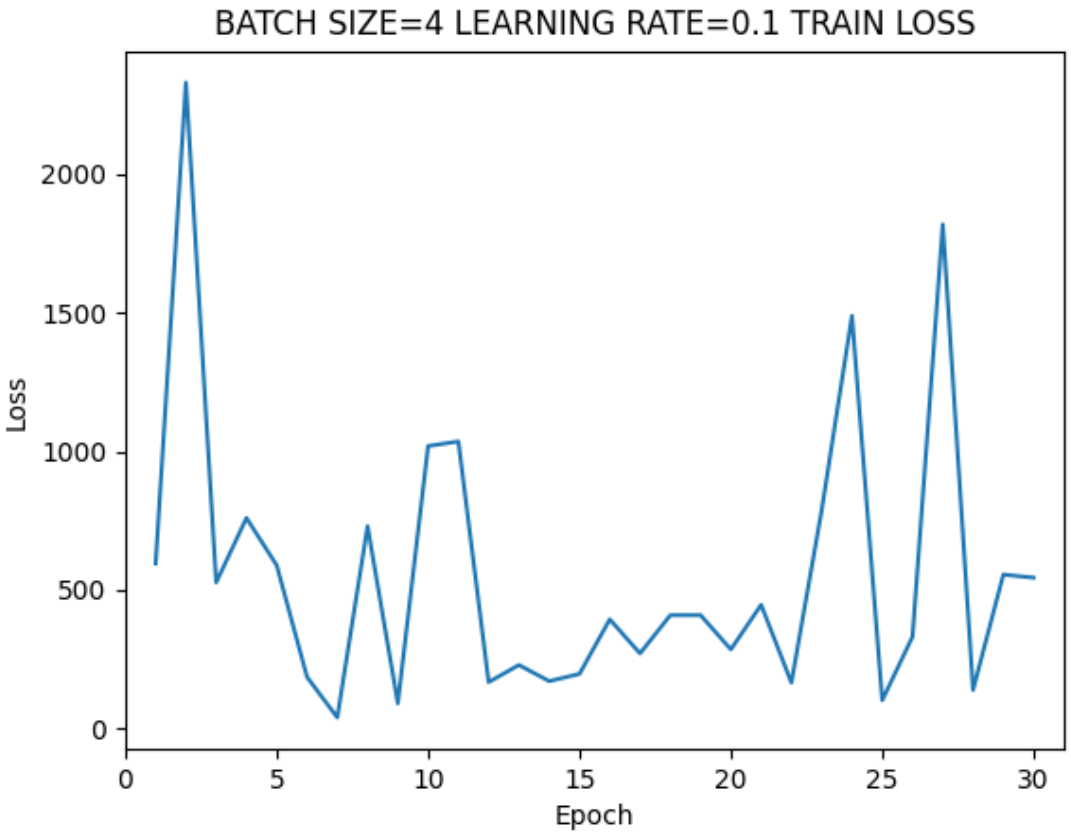
I initialized the model as above. I used CrossEntropyLoss() for softmax and MSELoss() for l2. I chose Adam as optimizer for I observed it gave the best results.

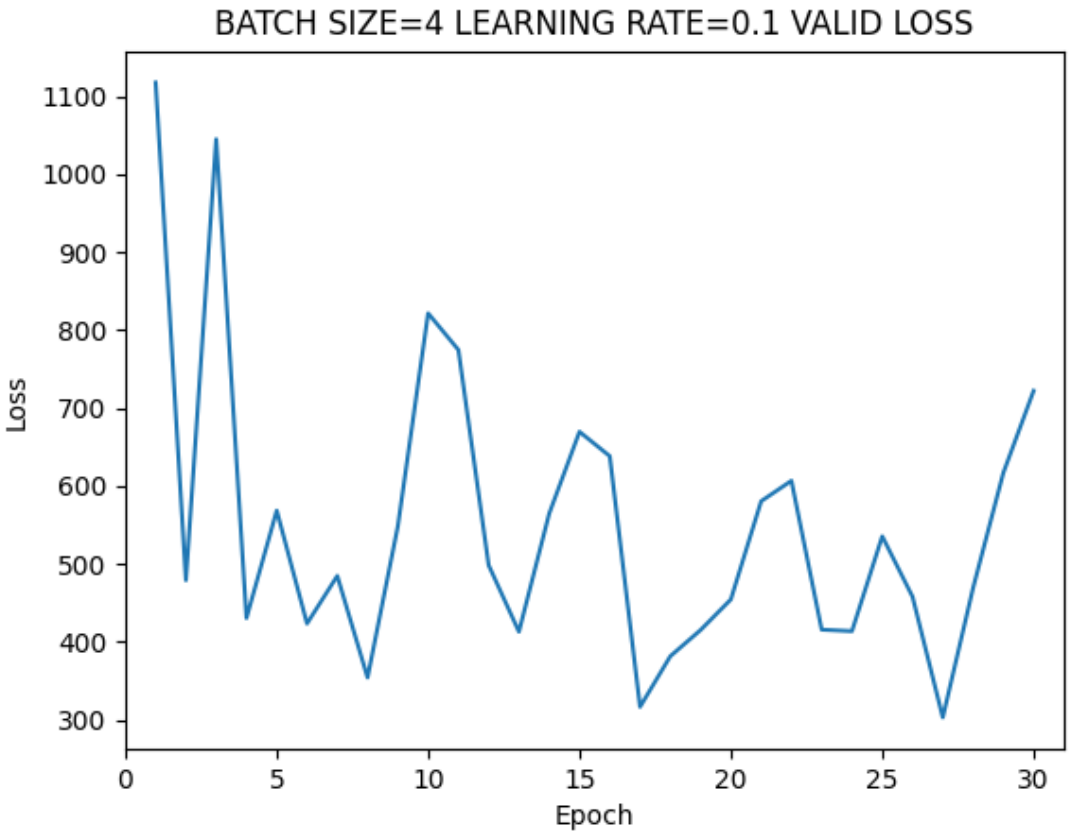
```
def train():

    valid_loss_array = []
    train_loss_array = []
    for epoch in range(EPOCH_NUM):

        for i, (images, labels, object_class) in enumerate(train_dataloader):
            images = images.to(device)
            labels = labels.to(torch.float32)
            labels = labels.to(device)

            object_class = object_class.to(device)

            outputsC = modelC(images)
            outputsR = modelR(images)

            lossSM = criterionSM(outputsC, object_class)
            lossL2 = criterionL2(outputsR, labels)

            optimizerSM.zero_grad()
            optimizerL2.zero_grad()

            lossSM.backward()
            lossL2.backward()

            optimizerSM.step()
            optimizerL2.step()

        print(f'Epoch [{epoch + 1}/{EPOCH_NUM}], Classification Train Loss: {lossSM.item():.4f}')
        print(f'Epoch [{epoch + 1}/{EPOCH_NUM}], Regression Train Loss: {lossL2.item():.4f}')

        train_loss_array.append(lossL2.item())
        valid_loss_array.append(validate(valid_dataloader))
```

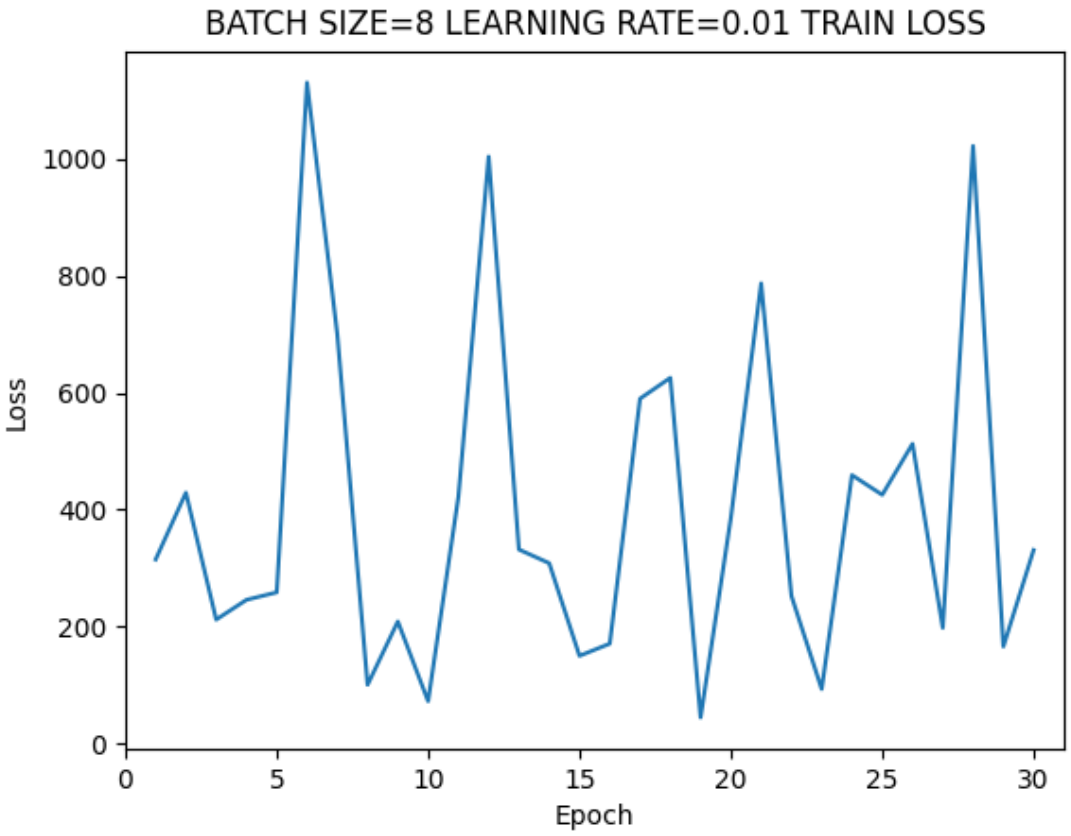
This is the train function I wrote. We will have several different plots from this function.

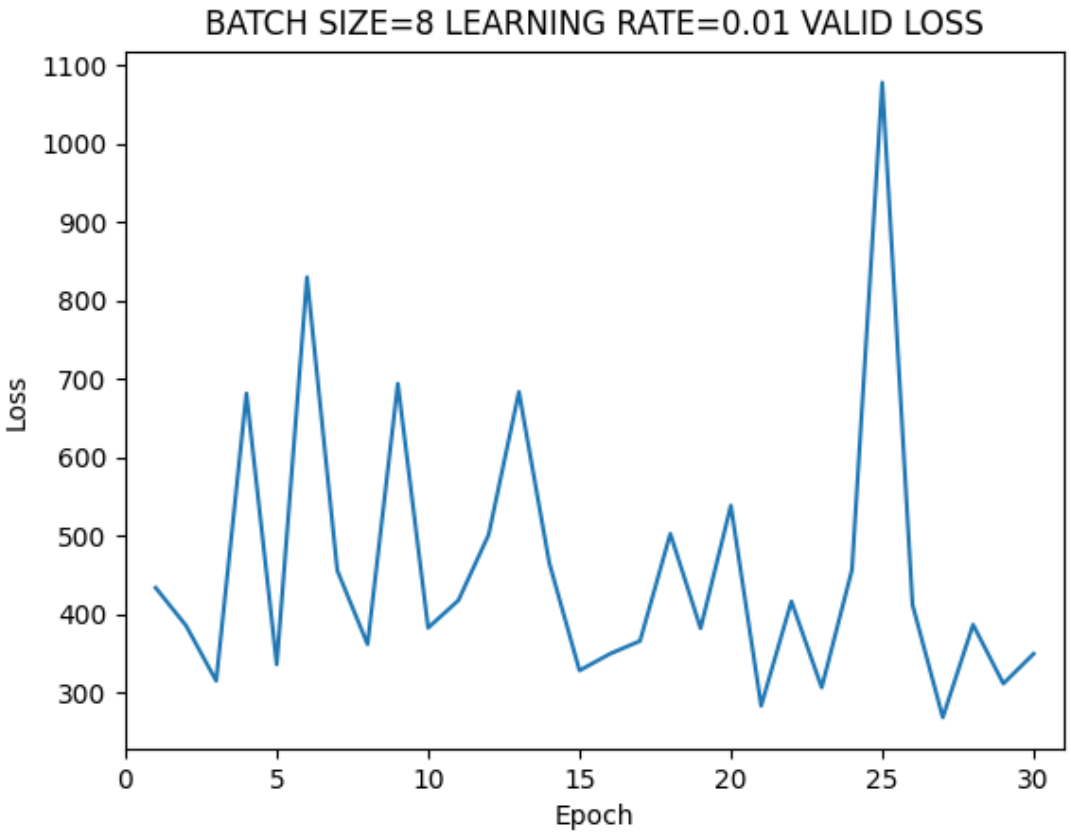

BATCH SIZE=4 LEARNING RATE=0.01 TRAIN LOSS
Loss
Epoch
0
200
400
600
800
1000
1200
0
5
10
15
20
25
30


BATCH SIZE=4 LEARNING RATE=0.01 VALID LOSS
Loss
Epoch
200
400
600
800
1000
0
5
10
15
20
25
30


BATCH SIZE=4 LEARNING RATE=0.1 TRAIN LOSS
Loss
Epoch
0
500
1000
1500
2000
0
5
10
15
20
25
30


BATCH SIZE=4 LEARNING RATE=0.1 VALID LOSS
Loss
Epoch
300
400
500
600
700
800
900
1000
1100
0
5
10
15
20
25
30


BATCH SIZE=8 LEARNING RATE=0.01 TRAIN LOSS
Loss
Epoch
0
200
400
600
800
1000
0
5
10
15
20
25
30

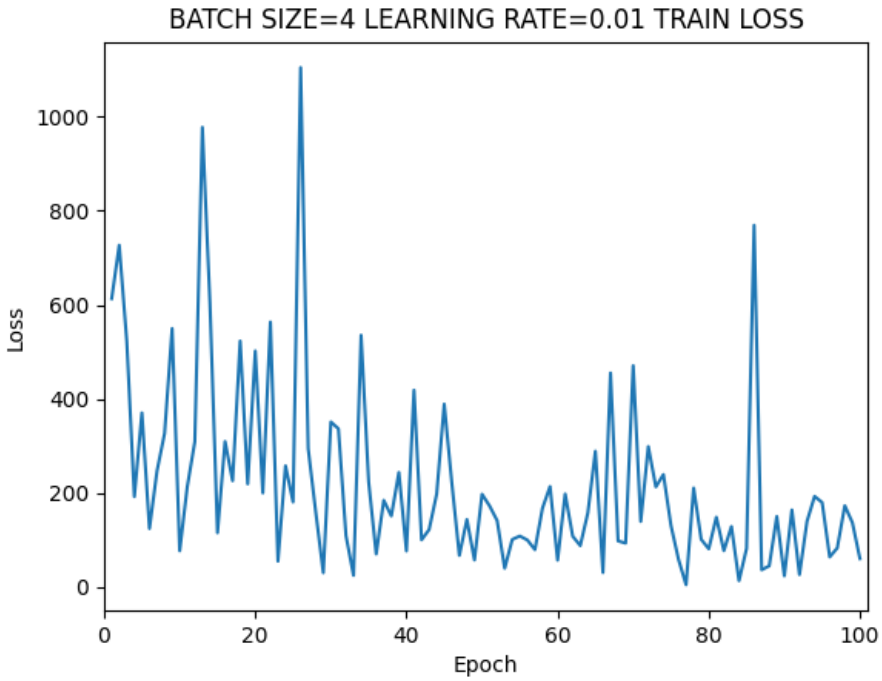

BATCH SIZE=8 LEARNING RATE=0.01 VALID LOSS
Loss
Epoch
300
400
500
600
700
800
900
1000
1100
0
5
10
15
20
25
30


BATCH SIZE=8 LEARNING RATE=0.1 TRAIN LOSS
Loss
Epoch
0
200
400
600
800
1000
1200
0
5
10
15
20
25
30


BATCH SIZE=8 LEARNING RATE=0.1 VALID LOSS
Loss
Epoch
250
500
750
1000
1250
1500
1750
2000
0
5
10
15
20
25
30

I used epoch number 30 for this subject because it seemed sufficient to me. I observed it not going down after that level so many times that it did not seem necessary to use more epochs.

Although, I'm including the plots with 100 epochs, batch size 4 as extra.



I think the spikes on the plots are the results of the low number of images in the dataset. The dataset we use for this assignment is so little that the loss value is not decreasing consistently. But still, we can see a downtrend on the plot generally.

```
Classification loss after testing = 0.0
Regression loss after testing = 309.2611389160156
Max IoU: 0.9139521792239946
Min IoU: 0.19555969891462233
Mean IoU: 0.6295509688502777
Median IoU: 0.7112419860748053
```

These are the test results. Although the max IoU value is 0.91(I think it's a good score given the dataset we worked with) mean of the IoU values of all test samples is 0.6 and I would not say that's successful.

Part 2 - Object Tracking use YOLOv3 and Mean-Shift

```python
class VotDataset(Dataset):
    def __init__(self, images, ground_truth):
        super(VotDataset, self).__init__()
        self.images = images
        self.annotations = ground_truth

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

    def __getitem__(self, item):
        return self.images[item], self.annotations[item]
```

I created a custom dataset for VOT dataset.

```python
if __name__ == '__main__':
    name = "/soccer1"

    path = "vot2017"
    frames = []
    ground_truths = []
    ground_truth = pd.read_csv(path + name + "/groundtruth.txt", header=None, sep=",", on_bad_lines="skip")
    images_file = os.listdir(path + name + "/color")

    for file in range(len(images_file)):
        img = cv2.imread(path + name + "/color/" + images_file[file])
        frames.append(img)
        list_ground_truth = list(ground_truth.iloc[file])
        list_ground_truth = [float(gt) for gt in list_ground_truth]
        list_ground_truth = torch.tensor(list_ground_truth)
        ground_truths.append(list_ground_truth)

    # Load data

    votddataset = VotDataset(frames, ground_truths)
    votd_loader = DataLoader(votddataset, batch_size=1, pin_memory=True, num_workers=1)

    yolov3 = cv2.dnn.readNetFromDarknet("yolov3-tiny.cfg", "yolov3-tiny.weights")
```

Preparing dataset data loaders and pretrained yolov3 net.

```python
for i, (frame, gt) in enumerate(votd_loader):
    blob_frame = torch.Tensor.numpy(frame)
    blob_frame = blob_frame[0]
    gt = gt[0]

    blob = cv2.dnn.blobFromImage(blob_frame, 1 / 255, (416, 416), [0, 0, 0], 1, crop=False)

    yolov3.setInput(blob)
    layers = yolov3.getLayerNames()
    outputs_names = [layers[j - 1] for j in yolov3.getUnconnectedOutLayers()]

    outputs = yolov3.forward(outputs_names)
    coords, bbox = box_finder(outputs, blob_frame)
    if len(coords) != 0:
        loss_list = []
        for coord in coords:
            loss = loss_function(torch.Tensor(coord), gt)
            loss_list.append(loss.item())
        bounding_box = bbox[np.argmin(loss_list)]

        new_frame = cv2.rectangle(blob_frame, (int(bounding_box[0]), int(bounding_box[1])),
                                  (int(bounding_box[2]), int(bounding_box[3])),
                                  color=(0, 0, 255), thickness=2)

        cv2.imwrite(f"gif/image{i}.jpg", new_frame)

        print("Humanoid found in this frame..")

    else:
        cv2.imwrite(f"gif/image{i}.jpg", blob_frame)

        print("Humanoid not found in this frame.")
```

```python
def box_finder(outputs, image):
    height, width, cr = image.shape
    bounding_box_list = []
    confidentiality_list = []
    coordinates = []

    for output in outputs:
        for elem in output:
            scores = elem[5:]
            predicted_class = np.argmax(scores)
            if predicted_class == 0:
                confidence = scores[predicted_class]
                if confidence > 0.4:
                    w, h = int(elem[2] * width), int(elem[3] * height)
                    x0, y0 = int(elem[0] * width), int(elem[1] * height)
                    x1, y1 = x0 - (w / 2), y0 - (h / 2)
                    x2, y2 = x0 + (w / 2), y0 - (h / 2)
                    x3, y3 = x0 + (w / 2), y0 + (h / 2)
                    x4, y4 = x0 - (w / 2), y0 + (h / 2)
                    confidentiality_list.append(confidence)
                    bounding_box_list.append([x1, y1, x3, y3])
                    coordinates.append([x1, y1, x2, y2, x3, y3, x4, y4])
```

These are the codes I used to find bounding boxes on each frame of the videos.

On the datasets I've tried the YoloV3 net with, I've seen that it's not so consistent. In my 3 example gifs, it can not find objects unless they are extremely obvious. Also, sometimes it does not find the correct bounding box.

Here's the drive link of my examples:

https://drive.google.com/drive/folders/1uo2qFO-KHveyIY4xt5ZSWMgG1PHN7Ffx?usp=sharing