BBM418 – Computer Vision Laboratory

Programming Assignment 4

Single Object Tracking with Regression Networks

Burak Özüesen – 21827761

1. Introduction

In this assignment, we implemented a basic single object tracker by training the network on the given videos. We used a two-frame architecture so the network used two features of frames at the same time. We utilized VOT2017 dataset for training and testing the tracker by considering the ground truth bounding box annotations.

2. Setup

In this part of my report, I would like to talk about the things I did before moving on to the Training and Testing part of the assignment.

In this part, I pulled the dataset onto CoLab very easily with the commands given by our friend "Mert Çökelek" in Piazza.

```
!pip install git+https://github.com/votchallenge/vot-toolkit-python
!vot 'initialize' vot2017 --workspace /content/drive/MyDrive/Assignment4/vot-workspace
```

Right after that, I performed the import operations of the extra libraries I used. While doing the assignment, I applied transformations between data types of many libraries. Although I've removed some of these from my final file, I think they should stay here as they are topics I've researched.

```
import os
import numpy as np
import pandas as pd
from PIL import Image
from time import time
import datetime
import random
import sys
from matplotlib import pyplot as plt
from IPython.display import display
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
from torchvision import models
from torchsummary import summary
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets, transforms
from torch.autograd import Variable
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import glob
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2
from google.colab.patches import cv2_imshow
import torchvision.transforms.functional as F
import torch.optim as optim
import torch.nn.functional as F1
```

3. Model Setup

How to create the Resnet-18 model and the custom model requested in the assignment document, and the selection of criterion and optimizer functions are as follows.

```
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()

        self.linear1 = nn.Linear(1024, 1024)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(1024, 4)




    def forward(self, x):
        residual = x
        out = F1.relu(self.linear1(x))
        out = F1.relu(self.linear1(out))
        out = self.linear2(x+residual)
        return out



model = models.resnet18(pretrained=True)
model.fc = nn.Sequential()
myModel = MyModel().to(device)

criterion = nn.MSELoss()
optimizer = optim.Adam(myModel.parameters(),lr= 0.001)
```

4. Training

4.1. Creating Dataset as Frame Pairs

By navigating the entire dataset I downloaded, I first defined it as 2-dimensional elements in an array. I have stored the names of the images and the groundTruth values as elements of this array. Then, if the file directories of the consecutive array elements are the same, I combined them into another array as the elements of the frame pairs. Afterwards, I had the chance to randomly generate a new dataset each time by using the Shuffle function on this array.

```
class_image_paths = []
groundTruths = []
path = "/content/drive/MyDrive/Assignment4/vot-workspace/sequences"
class_image_paths = []
class_count = 0
for dir in os.listdir(path):
    try:
      class_count += 1
      dirPath = path + "/" + dir
      groundtruth = open(dirPath + "/groundtruth.txt", "r")
      count = 0
      for img in sorted(os.listdir(dirPath + "/color/")):
          imgPath = dirPath + "/color/" + img
          count += 1
          class_image_paths.append([imgPath, class_count-1])
          tempImg = Image.open(imgPath)

          groundTruths.append([imgPath, class_count-1, groundtruth.readline().strip(), tempImg])

      groundtruth.close()
    except:
      continue

pairElements = []
for i in range(len(groundTruths) - 1):
  if (groundTruths[i][1] == groundTruths[i + 1][1]):
    pairElements.append([groundTruths[i], groundTruths[i + 1]])


random.shuffle(pairElements)
```

4.2. Cropping Process

At this stage, I drew both the BB and cropped the photo according to the BB values of the first photo. Then I cropped the second photo with the same coordinates I cropped. After scaling the BB values of the second photo from the original shape (224,224) to its size, I returned 3 objects in this function. The first of these objects is the first frame of size (224, 224) with a cropped and bounding box drawn. The second is the second frame with the BB undrawn (224, 224). The third parameter is a (1, 4) vector consisting of the points we want our model to give.

```
# This part opens image and takes groundTruth, draw BB and crop image according to enlarged BB
firstFrame = cv2.imread(image[0][0])
groundTruthString = image[0][2]
coordinates = groundTruthString.split(",")
xCoordinatesOfGroundTruth = []
yCoordinatesOfGroundTruth = []
for i in range(0, len(coordinates), 2):
  xCoordinatesOfGroundTruth.append(round(float((coordinates[i]))))
  yCoordinatesOfGroundTruth.append(round(float((coordinates[i+1]))))

x_of_left_upper = min(xCoordinatesOfGroundTruth)
y_of_left_upper = min(yCoordinatesOfGroundTruth)

x_of_right_bottom = max(xCoordinatesOfGroundTruth)
y_of_right_bottom = max(yCoordinatesOfGroundTruth)

X_of_center_bounding_box = (x_of_left_upper + x_of_right_bottom) // 2
Y_of_center_bounding_box = (y_of_left_upper + y_of_right_bottom) // 2

xDiff = abs(X_of_center_bounding_box - x_of_left_upper)
yDiff = abs(Y_of_center_bounding_box - y_of_left_upper)

leftUpper = (x_of_left_upper, y_of_left_upper)
rightLower = (x_of_right_bottom, y_of_right_bottom)

cv2.rectangle(firstFrame, leftUpper, rightLower, (0, 255, 0), 1)
croppedFirstImage = firstFrame[max(0, y_of_left_upper-yDiff):y_of_right_bottom+yDiff, max(0, x_of_left_upper-xDiff):x_of_right_bottom+xDiff]
```

```
# This part crops second frame according to first frames coordinates
secondFrame = cv2.imread(image[1][0])

croppedSecondFrame = secondFrame[max(0, y_of_left_upper-yDiff):y_of_right_bottom+yDiff, max(0, x_of_left_upper-xDiff):x_of_right_bottom+xDiff]

cropped_X_Coordinates = [max(0, x_of_left_upper-xDiff), x_of_right_bottom+xDiff]
cropped_Y_Coordinates = [max(0, y_of_left_upper-yDiff), y_of_right_bottom+yDiff]

targetVector = []

predictGroundTruthString = image[1][2]
predictCoordinates = predictGroundTruthString.split(",")
x_coord_of_predict = []
y_coord_of_predict = []
for i in range(0, len(predictCoordinates), 2):
  x_coord_of_predict.append((round(float((predictCoordinates[i])))))
  y_coord_of_predict.append((round(float((predictCoordinates[i+1])))))

x_of_left_upper = min(x_coord_of_predict)
y_of_left_upper = min(y_coord_of_predict)

x_of_right_bottom = max(x_coord_of_predict)
y_of_right_bottom = max(y_coord_of_predict)

X_of_center_bounding_box = (x_of_left_upper + x_of_right_bottom) // 2
Y_of_center_bounding_box = (y_of_left_upper + y_of_right_bottom) // 2

leftUpper = (x_of_left_upper, y_of_left_upper)
rightLower = (x_of_right_bottom, y_of_right_bottom)
```

```
# This part calculates target vector values
bigX = cropped_X_Coordinates[1]
bigY = cropped_Y_Coordinates[1]
smallX = cropped_X_Coordinates[0]
smallY = cropped_Y_Coordinates[0]


leftUpperX = leftUpper[0]
leftUpperY = leftUpper[1]
rightLowerX = rightLower[0]
rightLowerY = rightLower[1]


c = 224 * (bigX-rightLowerX) // (bigX - smallX)
d = 224 * (bigY-rightLowerY) // (bigY - smallY)

a = 224 * (bigX-leftUpperX) // (bigX - smallX)
b = 224 * (bigY-leftUpperY) // (bigY - smallY)

resultFrame = cv2.resize(croppedFirstImage, (224, 224))
cv2.rectangle(resultFrame, (a, b), (c, d), (0, 0, 255), 2)

targetVector.append([abs(a), abs(b), abs(c), abs(d)])

cropCord = [smallX, smallY, bigX, bigY]
```

4.3. Training Results

During the training phase, I divided my mixed Pair Elements array into 2 different arrays with the array slicing method and defined one of them as a test set and the other as a training set.

Afterwards, I train my model for as many generations as I want with objects that return from the function I mentioned above. I've also included a print function to average the result each time the epoch is finished.

```
random.shuffle(pairElements)
trainSet = pairElements[0:14197]
testSet = pairElements[14197:]



epoch = 30
for i in range(epoch):
  epoch_loss = []
  start_time = time()
  for j in range(len(trainSet)):

    firstFrame, secondFrame, targetVector, _ = cropImage(trainSet[j])
    targetVector = torch.tensor(targetVector).to(torch.float32)

    output1 = model(firstFrame)
    output2 = model(secondFrame)
    resultVector = torch.cat((output1, output2), 1)
    output3 = myModel(resultVector.to(device))


    loss = criterion(output3.to(device), targetVector.to(device))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    epoch_loss.append(loss.item())



  end_time = time()
  print("Epoch = ", i, "Loss = ", sum(epoch_loss)/len(epoch_loss))
```

I wanted to show 2 different train results in the screenshots below. I couldn't find why the change in my success loss function was small even though I used different LR.

```
Epoch =  0 Loss =  159.89528708457948
Epoch =  1 Loss =  163.01328732252122
Epoch =  2 Loss =  162.62776781082152
Epoch =  3 Loss =  162.74211639642715
Epoch =  4 Loss =  162.86435365200043
Epoch =  5 Loss =  162.97886829137803
Epoch =  6 Loss =  163.085964782238
Epoch =  7 Loss =  163.18555384635926
Epoch =  8 Loss =  163.27767705917358
Epoch =  9 Loss =  163.3626504588127
Epoch =  10 Loss =  163.44043920993806
Epoch =  11 Loss =  163.51128946304323
Epoch =  12 Loss =  163.5753149652481
Epoch =  13 Loss =  163.63267952919006
Epoch =  14 Loss =  163.68361186504364
Epoch =  15 Loss =  163.72830617427826
Epoch =  16 Loss =  163.7669138622284
Epoch =  17 Loss =  163.799632024765
Epoch =  18 Loss =  163.82667558431626
Epoch =  19 Loss =  163.8482250714302
Epoch =  20 Loss =  163.86452107191087
Epoch =  21 Loss =  163.87575994491579
Epoch =  22 Loss =  163.882094604969
Epoch =  23 Loss =  163.88380174636842
Epoch =  24 Loss =  163.8810149717331
Epoch =  25 Loss =  163.87394211769103
Epoch =  26 Loss =  163.8627495241165
Epoch =  27 Loss =  163.84770272731782
Epoch =  28 Loss =  163.82885667800903
Epoch =  29 Loss =  163.8065033721924
```

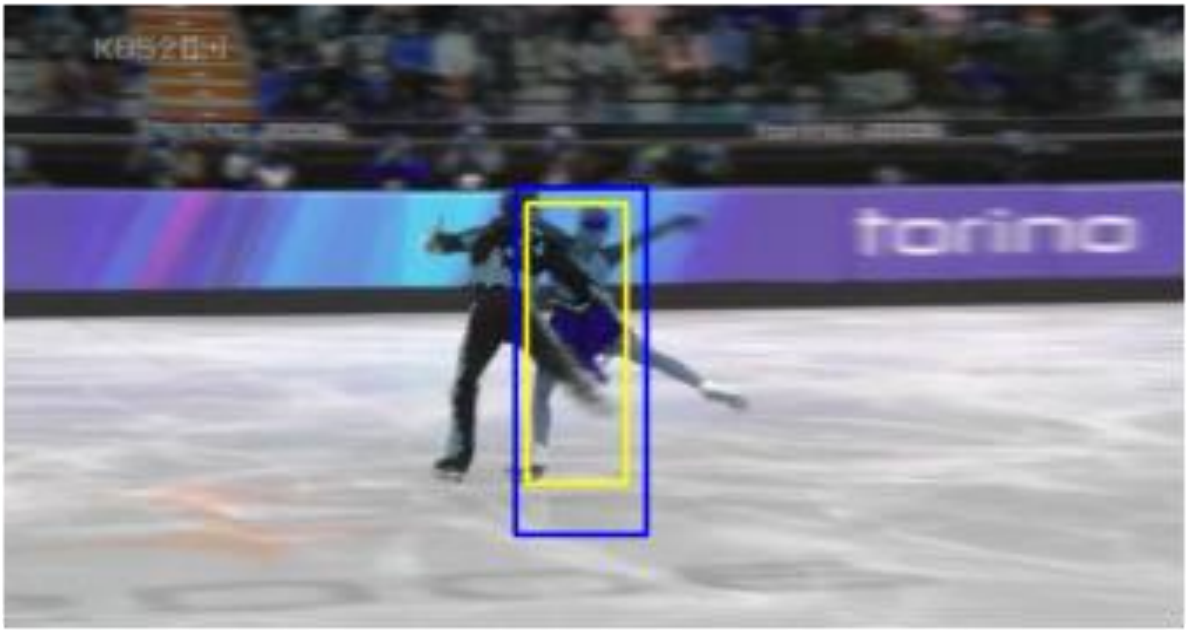
```
Epoch =  0 Loss =  150.34564599990844
Epoch =  1 Loss =  135.15954647064208
Epoch =  2 Loss =  126.52902221679688
Epoch =  3 Loss =  128.0228816986084
Epoch =  4 Loss =  130.36739654541014
Epoch =  5 Loss =  130.60004386901855
Epoch =  6 Loss =  130.03811664581298
Epoch =  7 Loss =  129.54720878601074
Epoch =  8 Loss =  129.30636863708497
Epoch =  9 Loss =  129.25598526000977
Epoch =  10 Loss =  129.26922340393065
Epoch =  11 Loss =  129.2640645980835
Epoch =  12 Loss =  129.22432460784913
Epoch =  13 Loss =  129.16526985168457
Epoch =  14 Loss =  129.10338249206544
Epoch =  15 Loss =  129.04596939086915
Epoch =  16 Loss =  128.99323005676268
Epoch =  17 Loss =  128.94220428466798
Epoch =  18 Loss =  128.89084587097167
Epoch =  19 Loss =  128.83845558166504
Epoch =  20 Loss =  128.78541927337648
Epoch =  21 Loss =  128.73219547271728
Epoch =  22 Loss =  128.6789867401123
Epoch =  23 Loss =  128.62593421936035
Epoch =  24 Loss =  128.5727705001831
Epoch =  25 Loss =  128.51951541900635
Epoch =  26 Loss =  128.4661548614502
Epoch =  27 Loss =  128.4127336502075
Epoch =  28 Loss =  128.3591209411621
Epoch =  29 Loss =  128.3055700302124
```

5. Testing

```
test_loss = 0
start_time = time()

accuracy = []

with torch.no_grad():
    for i in range(len(testSet)):
        firstFrame, secondFrame, targetVector, cropCord = cropImage(testSet[i])

        targetVector = torch.tensor(targetVector).to(torch.float32)

        output1 = model(firstFrame)
        output2 = model(secondFrame)
        resultVector = torch.cat((output1, output2), 1)

        output3 = myModel(resultVector.to(device))

        output3 = output3.tolist()[0]

        originalSecondFrame = cv2.imread(testSet[i][1][0])

        croppedBoxedSecondFrame = originalSecondFrame[cropCord[1]:cropCord[3], cropCord[0]:cropCord[2]]

        myShape = croppedBoxedSecondFrame.shape
        croppedBoxedSecondFrame = cv2.resize(croppedBoxedSecondFrame, (224,224))


        a = int(round(output3[0]))
        b = int(round(output3[1]))
        c = int(round(output3[2]))
        d = int(round(output3[3]))


        drawedRectangle = cv2.rectangle(croppedBoxedSecondFrame, (a,b), (c,d), (255,255,0), 3)
```

```
drawedRectangle = cv2.rectangle(croppedBoxedSecondFrame, (a,b), (c,d), (255,255,0), 3)

finalCordX1 = (a * myShape[1] // 224) + cropCord[0]
finalCordY1 = (b * myShape[0] // 224) + cropCord[1]
finalCordX2 = (c * myShape[1] // 224) + cropCord[0]
finalCordY2 = (d * myShape[0] // 224) + cropCord[1]

drawedRectangle = cv2.rectangle(originalSecondFrame, (finalCordX1,finalCordY1), (finalCordX2,finalCordY2), (0, 0, 255), 2)

coordinates = testSet[i][1][2].split(",")

xCoordinatesOfGroundTruth = []
yCoordinatesOfGroundTruth = []
for j in range(0, len(coordinates), 2):
  xCoordinatesOfGroundTruth.append(round(float((coordinates[j]))))
  yCoordinatesOfGroundTruth.append(round(float((coordinates[j+1]))))

x_of_left_upper = min(xCoordinatesOfGroundTruth)
y_of_left_upper = min(yCoordinatesOfGroundTruth)

x_of_right_bottom = max(xCoordinatesOfGroundTruth)
y_of_right_bottom = max(yCoordinatesOfGroundTruth)

X_of_center_bounding_box = (x_of_left_upper + x_of_right_bottom) // 2
Y_of_center_bounding_box = (y_of_left_upper + y_of_right_bottom) // 2

xDiff = abs(X_of_center_bounding_box - x_of_left_upper)
yDiff = abs(Y_of_center_bounding_box - y_of_left_upper)

leftUpper = (x_of_left_upper, y_of_left_upper)
rightLower = (x_of_right_bottom, y_of_right_bottom)

cv2.rectangle(originalSecondFrame, leftUpper, rightLower, (255, 255, 0), 2)



# there is break for seeing single image if you want see test loss please comment imshow and break
plt.axis("off")
plt.imshow(originalSecondFrame)
#break
```

I have included 3 examples below. The blue boxes in these pictures are the correct boxes, and the yellows are the points my model predicted.


torino