BBM 418 - Computer Vision Laboratory

HACETTEPE UNIVERSITY

Department of Computer Engineering

Programming Assignment-4
Spring 2020-2021
Dr. Nazli Ikizler Cinbis

Due Date: 30.05.2021, 23:59pm

Single Object Tracking with Regression Networks

1) Introduction

In this assignment we are learning single tracker implementation with given challenge VOT2017 dataset, using CNN implementation with pre-trained resnet18. Resnet18 Google Colab was used.

2) Implementation Details

About Dataset: The dataset has 8 ground truth points and video frames. Data is read in the ***read_data()*** function. First we take ground truth points with splitting the newline and commas. After that frames are read pairwise as well as the ground truths. X contains (frame[index], frame[index+1]). Y contains[ground_truth[index], ground_truth[index+1]] and so on. Before passing them to the Dataloader(from torch.utils.data.DataLoader), ***class MyDataset*** (CustomDataset) is initialized. It takes the read_data() outputs. In the __getitem__ we resize the pairwise images and returns the resized version of the ground truth as well. A the end of this function MyDataset:

```
returns (resized images), (resized ground truths), (img paths)
```

Dataset is splitted 50%-25%-25% (train-val-test). Then passes through Dataloader. Shuffling is handled before passing dataloader with `SubsetRandomSampler`..

About Helper Functions: There are some helper functions such as ***enlarge_bb()*** for enlarging the bounding box for the search region. It also check the image bounds. Reverse operation is handled in the ***shrink_bb(). CropImg()*** is he main operation function. It first enlarge the bounding box then crop the image tensor with enlarged bounding box coordinates. In the get_bb() function we create the bounding box with the given coordinates in the ground truths. The bounding box returns the left,upper bounds and the right bottom bounds. Therefore our created bounding box is a rectangle.

```
model = models.resnet18(pretrained=True)
self.convnet = nn.Sequential(*list(model.children())[:-1])
```

```
self.classifier = nn.Sequential(
        nn.Linear(1024, 4096),
        nn.ReLU(inplace=True),
        nn.Dropout(),
        nn.Linear(4096, 4096),
        nn.ReLU(inplace=True),
        nn.Dropout(),
        nn.Linear(4096, 4096),
        nn.ReLU(inplace=True),
        nn.Dropout(),
        nn.Linear(4096, 4),#four bbox outputs are set
        )
```

```
for param in self.convnet.parameters():
    param.requires_grad = False
```

About MyNet*()* module: It is taken from pretrained resnet18 model. It is not classification model so that I have to remove last layer. After that as in the paper FC-ReLU-FC-ReLU- FC-ReLU-FC. And residual connection is done.

```
def forward(self, x, y):
    x1 = self.convnet(x)
    x1 = x1.view(x.size(0), -1)
    x2 = self.convnet(y)
    x2 = x2.view(x.size(0), -1)
    x = torch.cat((x1, x2), 1)#two images are concatenated.
    x = self.classifier(x)
    return x
```

Concatenation is handled in the forward part.

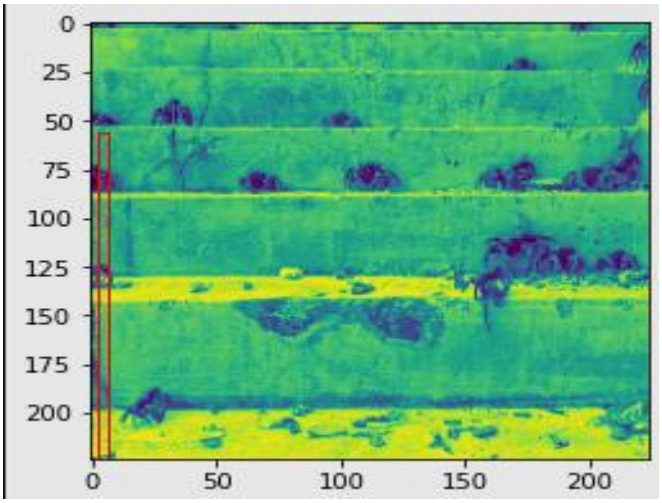
Train model() function: is for both train and evaluation. Parameters are our Cnn model, loss function is MSE. Creates a criterion that measures the mean squared error (squared L2 norm) between each element in the input *x* and target *y*. Optimizer is SGD (stochastic gradient descent). Here, we need to freeze all the network except the final layer. We need to set `requires_grad == False` to freeze the parameters so that the gradients are not computed in `backward()`. I use schedular but it is not needed because I use small epoch numbers. When we set the is_train to True is means ı am doing training else testing. First of all i pass the dataset elements to the GPU. bb_list() is created to store the ground_truth of the second one of the pair. ***zero_grad()*** restarts looping without losses from the last step if you use the gradient method for decreasing the error (or losses). Gradient tracking IS enabled inside the with torch.set_grad_enabled. At the end the best model is saved for further testing when validation is handled in this function.

```
bb_list=list()
for i in range(len(local_labels[0])): #image cropping was handled this loop
  prev_gt=local_labels[0][i]#prev ground truth
  curr_gt=local_labels[1][i]#curr ground truth
  bb=get_bb(prev_gt)
  if epoch==0:#crop the img only at the first epoch
    crop_img,bb_new=cropImg(local_batch[0][i],bb)
    crop_img=F.resize(crop_img,(224,224))
    local_batch[0][i]=crop_img#change

  bb2=get_bb(curr_gt)
  bb_list.append(bb2)
  if epoch==0:#crop the img only at the first epoch
    crop_img2,bb_new2=cropImg(local_batch[1][i],bb)
    crop_img2=F.resize(crop_img2,(224,224))
    local_batch[1][i]=crop_img2

bb_list=torch.Tensor(bb_list)/224.
bb_list=bb_list.to(device)
```

Note: In here cropping an image is done at the first epoch. Then cropped image set as an local batch image.

3) Experimental Results:

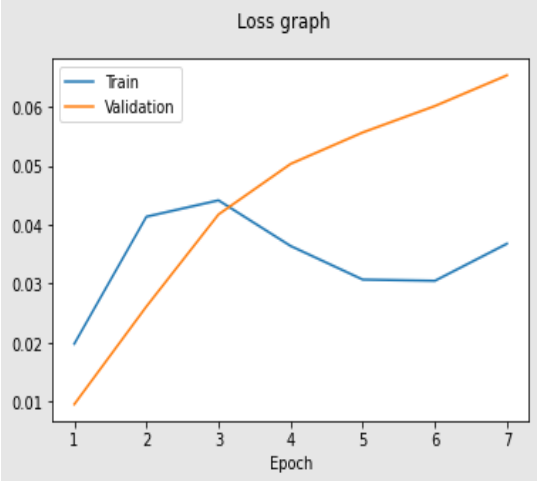
Two different learning rates and two different batch_size is implemented. It does not converge at all to a good loss. So that i early stop the epoch(at 7). Adding residual connection does not decrease the loss value. But adding dropout is very critical because there are so many parameters and avoid overfitting. Decreasing step is good to avoid the oscilattion. Dropping the last batch for balancing the dataloaders.



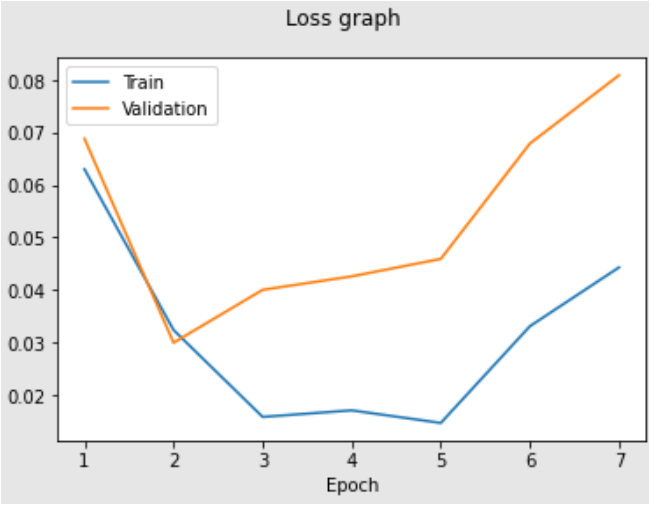
Worst case: The bounding box shown at left with red rectangle. The results are almost always worst case. Because of my bad resize implementations.

According to loss graphs, function does not converge well.

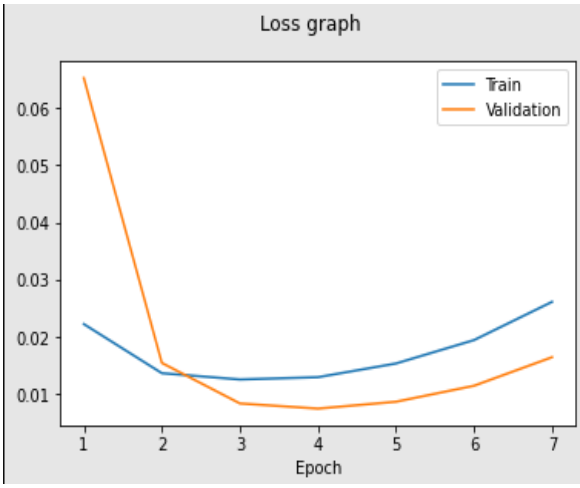
Batch size:256, Learning Rate:0,001



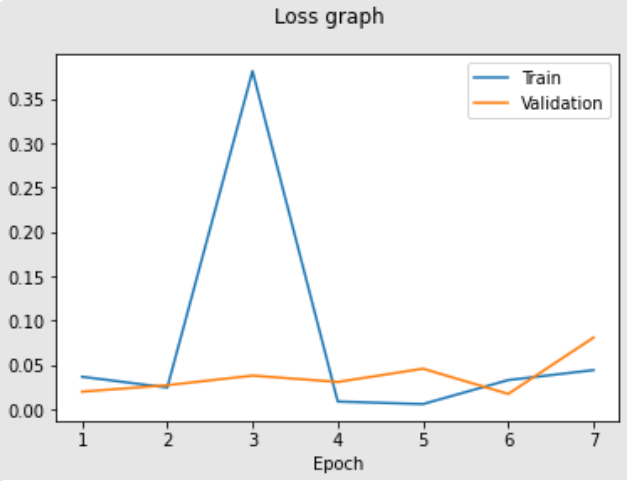
Batch size:256, Learning Rate:0,01



Batch size:128, Learning Rate:0,001



Batch size: 128, Learning Rate:0,01



4) Conclusion:

As a result my work failed. Main reason is resizing too much and therefore loosing information. Resizing the image is losing information. In my code I resize an image 1024,1024 to 224,224, After cropping the image it needs to be resized again to 224,224. This time we loose quality again. Therefore in my code losing information is crucial. Prediction of ground_truths are obtaining as normalized version. As you can see above part results.

Some weaknesses of that network are variations due to geometric changes Eg:- Pose, articulation, the scale of objects. Low quality image. Crowded scenes such as concerts or markets. Similar objects in the scene.

The strengths of the network are without any prior knowledge about the appearance and number of targets deep learning can learn the features. Features can be extracted fastly but according to my implementation it takes much because the code read images in each epoch.

Pytorch is a very difficult framework because of tensor. Converting operations are cumbersome. Dataloader gives me so much error according to my Custom Dataset. Using tuple in tensor or processing them almost impossible. Also splitting dataset takes much time in my implementation. I wish i splitted dataset before passing all of them to the myDataset(custom dataset).

About future works multibox detection can be handled. Calculating Intersection Over Union is a good choice for better accuracy calculation.