



HACETTEPE UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING
SPRING 2019

BBM 418 Computer Vision Laboratory Problem Set-3

Supervisors:

Dr. Nazlı İkizler Cinbis

TA Özge Yalcinkaya

Author:

Ege Berke Balseven

Number:

21590776

Subject:

Single Object Tracking with

Regression Networks

Due Date:

30.05.2019

Contents

1	Introduction	2
2	Implementation Details	2
3	Experimental Results	3
3.1	PART 1: Training	3
3.2	PART 2: Testing	7
3.3	Bonus:	7
4	Conclusion	8

1 Introduction

In this experiment, i got familiar the object tracking in Pytorch. Object tracking is intended to follow an object in a video sequence. A tracking algorithm is started with a frame of a video and a bounding box to show the location of the interested object in tracking. The tracking algorithm outputs a bounding box for all subsequent frames.

I used pre-trained VGG16 model from torch library. Then i extract features from the last layer before FC layers and applied average pooling. A Convolutional Neural Network (CNN) is trained to predict the position of the bounding box in the current frame. Network has 3 fully connected layers with Relu and dropout. Loss function is the mean square error loss and as an optimizer Adam optimizer used.

2 Implementation Details

We have two frames. First frame is previous frame, the location of the object is known and the frame is cropped to times the size of the bounding box around the object. The object in the first cropped image is always centered. Then the position of the object in the current frame must be predicted. The bounding box used to crop the previous frame is also used to crop the current frame. Because the object may have moved, it is not centered in the current frame. Convolutional Neural Network is trained to predict the location of the bounding box in the current frame. Both frames pass through a convolutional layers. Then test starts with known initial bounding box. After that, the network predicts the bounding boxes of the rest.

In this experiment I've had too many problems. Some of these are: frames and annotations reading, processing, TypeError: forward() takes 2 positional arguments but 3 were given, ValueError: not enough values to unpack (expected 2, got 0), RuntimeError: shape '[8, 200704]' is invalid for input of size 200704, return self._apply(lambda t: t.cuda(device)) TypeError: _apply() missing 1 required positional argument: 'fn', RuntimeError: CUDA error: unknown error, PermissionError: [Errno 13] Permission denied: '../dataset/videos/test/video0002' , RuntimeError: CUDA out of memory. KeyError, RuntimeError: Expected object of backend CPU but got backend CUDA for argument 2 'weight', PIL.Image.DecompressionBombError: Image size (409630806 pixels) exceeds limit of 178956970 pixels, could be decompression bomb DOS attack. RuntimeError: Expected 4-dimensional input for 4-dimensional weight 64 3 3 3 0, but got 5-dimensional input of size [1, 64, 227, 227, 3]

instead. RuntimeError: Given groups=1, weight of size 64 3 3 3, expected input[4, 227, 227, 3] to have 3 channels, but got 227 channels instead. Also test and validation part was too difficult etc. I spent a lot of hours solving all of these problems.

3 Experimental Results

3.1 PART 1: Training

Hyper-Parameters:

Learning Rate:

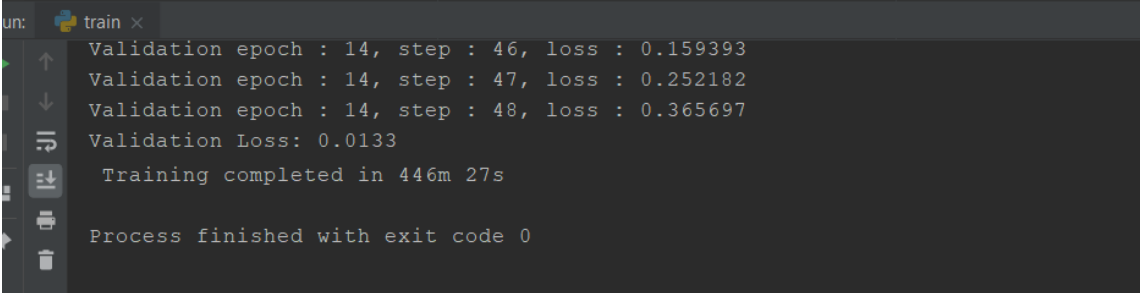
The Learning Rate is one of the parameters that determines the weight change while the weights are updated while the forward and backpropagation operations are performed on the Neural Network. If this parameter is too less, the loss value will be higher because each epoch will learn less. However, if the Learning Rate is high, also loss value is higher. With big learning rate we can not reach minimum loss because steps are too big.

Batch Size:

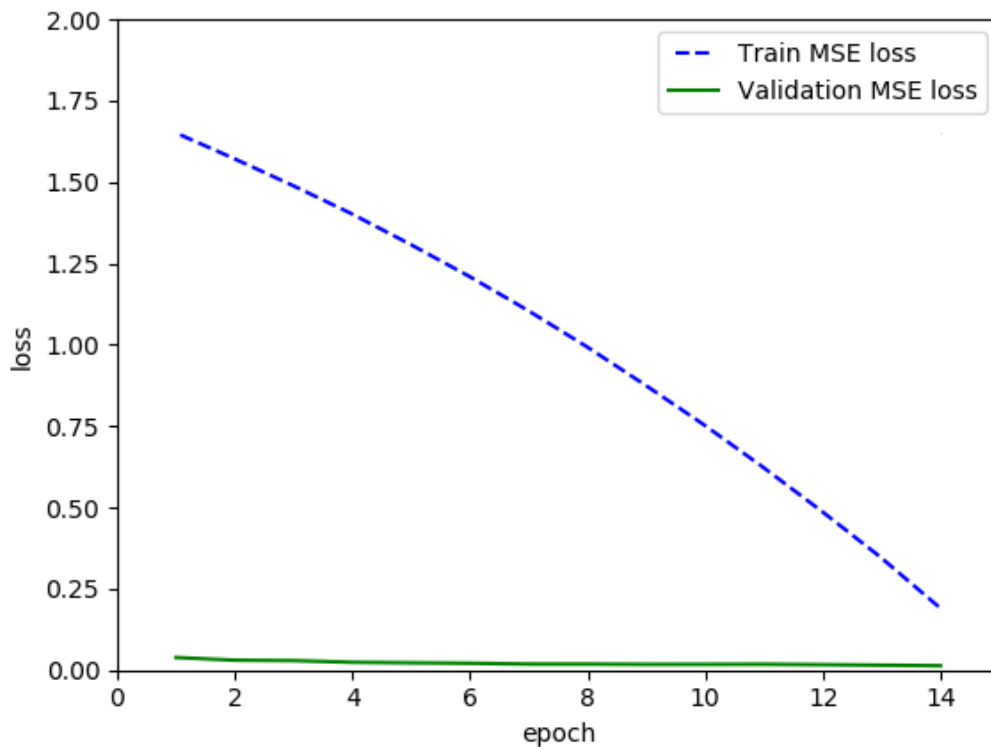
Batch determines how much data is to be taken. If there is more, it will load more data, which will use more memory. When the Batch Size increase, more RAM is being used. But as the batch size increases, the loss often increases. Because it learns with more data and the learning time is getting shorter. If the batch is reduced to you all the time, we can not say that the loss is get worse. There must be an optimum range for this.

Train and Validation Losses:

```
19 # parameters
20 epochs = 14
21 batch_size = 32
22 learning_rate = 0.00001
23 save_directory = '../save/pa3/'
24
```



```
Validation epoch : 14, step : 46, loss : 0.159393
Validation epoch : 14, step : 47, loss : 0.252182
Validation epoch : 14, step : 48, loss : 0.365697
Validation Loss: 0.0133
Training completed in 446m 27s
Process finished with exit code 0
```



```
use_gpu = torch.cuda.is_available()

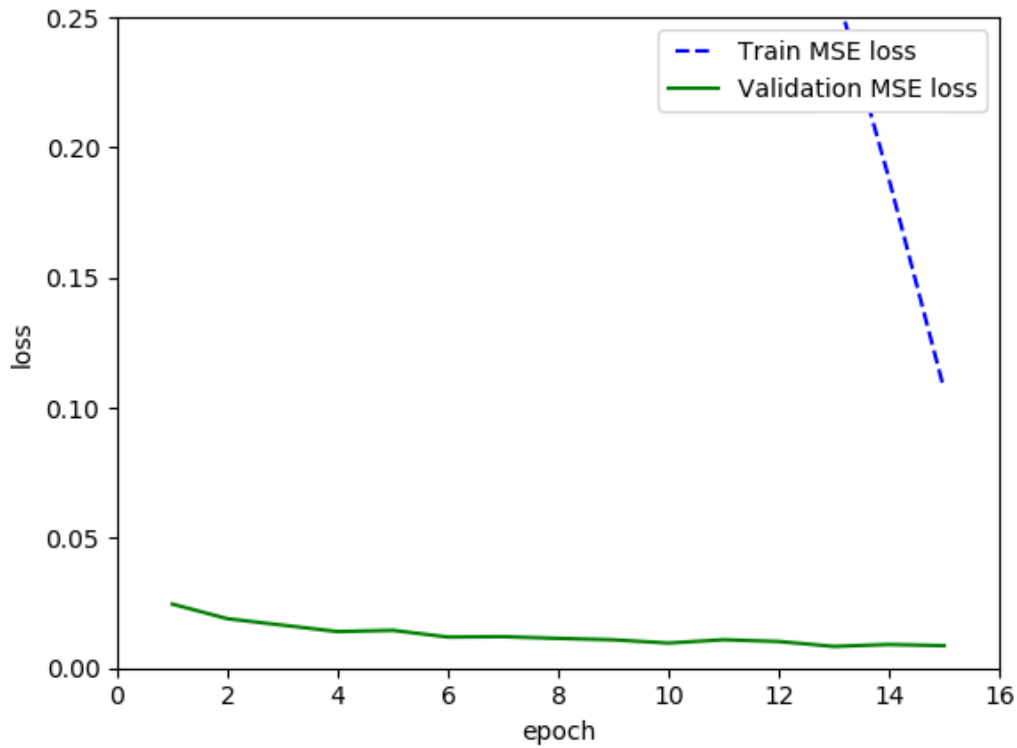
# parameters
epochs = 15
batch_size = 64
learning_rate = 0.0001
save_directory = '../save/pa3/'

main()

train × test ×

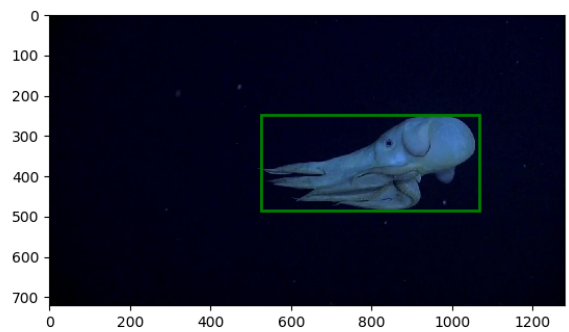
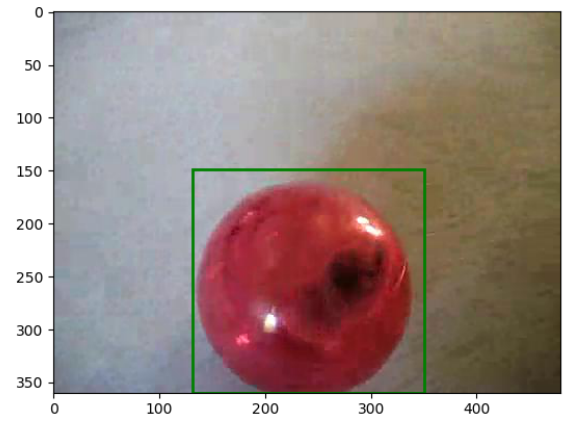
Validation epoch : 15, step : 22, loss : 0.346377
Validation epoch : 15, step : 23, loss : 0.357678
Validation epoch : 15, step : 24, loss : 0.208816
Validation Loss: 0.0086
Training completed in 484m 12s

Process finished with exit code 0
```



Training and validation losses are too good. I guess, it is low because of the i normalized the bounding boxes. That's why mean square error got low value probably. The training took too long, it took 484 minutes(8 hours) for 15 epoch. So i could not a lot try with big epochs. But i have tried different batch sizes and learning rates, with low batch size and low learning rate, learning becomes a little slower.

Some best results:



3.2 PART 2: Testing

Some initial and predicted bounding box results:

```

Run: test x
C:\Users\Berke\Anaconda3\python.exe C:/Users/Berke/Desktop/418-3/418-3/src/test.py
initial box ground truth :
[690.32  80.307 887.85 327.46 ]
test results:
[707.4300575142075, 111.99991678575692, 929.1127319604816, 386.53497288708115]
[669.2008070866154, 81.69996777609686, 863.2927306569497, 353.01652440425886]
[659.3952239988489, 68.83355827416095, 886.0740173925764, 392.9928009666809]
[640.8374522886537, 80.33601766762455, 896.922719357022, 427.05757446927873]
[657.9691029526989, 110.07513764888166, 899.5818164686017, 424.5152665620373]
[633.7145219198196, 89.65527054267586, 864.8836013693863, 394.7146385185663]
[641.2459643189353, 67.25563316426785, 835.7505894372638, 316.4781554054723]
[630.3284929197166, 64.62321772670414, 808.664468175389, 294.1691570267974]
[618.7788171653766, 53.852165420605616, 771.303426411143, 260.464274841313]
[611.535486559431, 60.06500506367449, 777.5827970476885, 270.6624189893582]
[620.0906721382981, 60.23013027008734, 780.8336770471648, 244.8132779281728]
[707.4013968432171, 84.93101920941669, 902.6626836906178, 314.36132929353]
[697.1416646171416, 107.63298346156783, 907.4932265647917, 351.9283057407483]
[700.1442211703683, 108.50750347437565, 943.7135750336963, 367.99815903928663]
[706.3403629873579, 118.09383373156979, 924.0806696651066, 349.32097839848683]
[676.3722284108427, 124.33290769338078, 896.9687180215764, 366.54739968142434]
[648.0047731271269, 117.99975537465176, 899.9400523214237, 408.6943343278243]
[633.8230969179958, 110.84490001890319, 883.0175606042326, 394.8254136204374]
[629.4168456759035, 133.8614773983115, 847.1070376110738, 371.15438960616916]
[633.0457320702311, 107.33646727326911, 772.4567222933194, 294.6677921032148]
[625.3649092767278, 121.98091939518737, 769.3521846568759, 304.37813281929806]
[609.5280407030881, 100.67025251065989, 726.3401456675932, 267.9305481279412]

initial box ground truth :
[762.7  433.91 803.07 503.23]
test results:
[758.6925236386876, 433.27556612930636, 800.0549672238524, 504.8719188646493]
[758.5476462407646, 437.85956628903165, 798.7401758177358, 504.2879122930364]
[757.1551430965366, 445.9284417794382, 804.8191699082598, 521.0365121070662]
[757.3501207221534, 446.13530528490287, 801.0412647489051, 514.6615305416815]
[759.7253889645744, 442.0650240028393, 798.6278521837066, 507.2803881390787]
[759.716343687494, 450.6649664585184, 795.2125814939308, 512.3313871510131]
[757.5180170540614, 441.8279387205964, 789.5651217440771, 495.2232518006544]
[774.8172131710289, 457.7605728015694, 804.1837318209493, 508.4672683269829]
[777.9496358787069, 467.9429383009533, 804.2478077913933, 518.4474331393681]
[776.1222526985562, 466.1495275007392, 799.5684861157854, 513.5213625003579]
[776.2371271181383, 457.35950197815794, 791.7965652812433, 492.84998030713115]
[776.1861512374688, 454.995958636794, 789.2208206231527, 483.80018354140645]
[778.1837476297526, 456.1888393924005, 790.3368536977098, 481.93125018568173]
[776.3375788805291, 459.4860534259659, 785.7674064529158, 480.7020727176173]
[776.4053547646963, 462.025301981368, 783.6146251299416, 478.1924348310901]
[773.9152549756684, 457.99948358049807, 777.2499023126168, 465.36708843964806]

```

According to initial box, predicted bounding box positions in frames seems good.

3.3 Bonus:

Dropout:

Dropout is the cancellation of some neurons in order to prevent overfitting. This is done according to a certain threshold. The neurons below the thresh- old are not included in the process, thus overfitting is prevented. In the end, i used dropout in

my network.

4 Conclusion

Results are interesting. Loss is too good. I didn't expect that. I didn't have enough time to convert the results into video. The results I have shown above are the best result in what i tried. The experiment was difficult, i failed too much nearly everywhere but i learned a lot from the experiment.