7_DeepLearning_students

July 13, 2019

1 Hands-on Machine Learning

1.1 Session 7: Deep Learning

by Daniel Bug

1.1.1 Goal of this session

In this session you will: * implement a deep neural network * learn about different layer types and activation functions * experiment with visualization techniques

Mind that there are still a few things we hide behind the scenes: * Data Loading / Handling is done using framework utilities (see additional python scripts and pytorch doc) * Data Augmentation will be covered in a later session

1.1.2 Dataset

This session uses the PascalVOC dataset, which is accessible on this server. The dataset comprises input images that can be classified with different strategies. We consider an image classification problem, i.e. decide which objects from a finite set of classes appear in the input image. Since multiple objects may appear in each image, this is a multi-label classification task.

Let's dive right into the task: Make sure to run the imports and continue loading the dataset to RAM.

```
In [1]: import os
        os.environ['CUDA_VISIBLE_DEVICES'] = '7'  # Set this parameter to YOUR GROUP NUMBER

        import torch
        import torch.nn as nn
        import torch.nn.functional as F
        import numpy as np
        import matplotlib
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline

        import voc as dsetVOC
        import torchvision.transforms as standard_transforms
        import transforms as extended_transforms
```

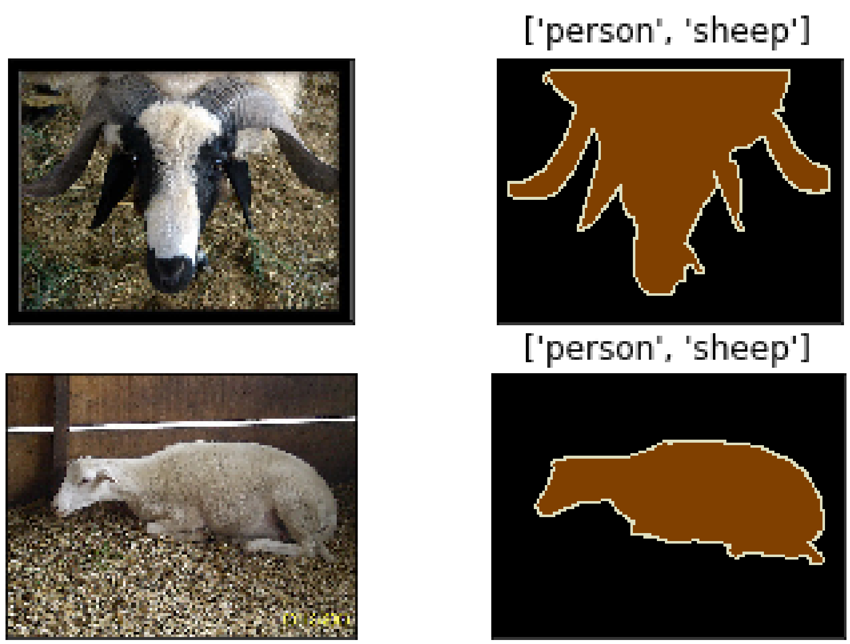
```python
from torch.autograd import Variable
from torch.utils.data import DataLoader
from torch  import optim
```

Loading the dataset We prepared a loader for you that can automatically grab a training- and testset. Run the cell below a few times to get an overview of the data available.

You can print the cls2lbl dictionary to see the list of all classes.

In [2]:
```python
train_set = dsetVOC.VOC('train')
valid_set = dsetVOC.VOC('valid')
cls2lbl = dsetVOC.class_to_label

for i in range(2):
    img, mask, classes = valid_set.__getitem__(np.random.randint(0, 100 + 1), colorize
    plt.subplot(2,2,2*i+1)
    plt.imshow(img)
    plt.xticks(()); plt.yticks(())
    plt.subplot(2,2,2*i+2)
    plt.imshow(mask)
    plt.xticks(()); plt.yticks(())
    plt.title(str([cls2lbl[x+1] for x in np.where(classes>0.)[0]]))
plt.show()
```



1.2 Building a network

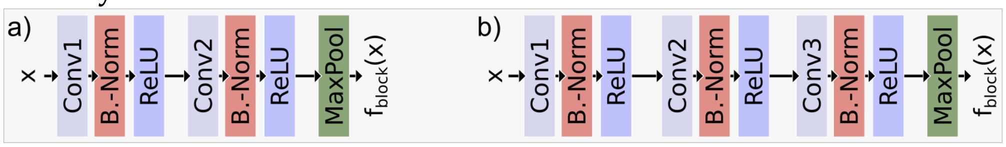
1.2.1 Defining functional units

As introduced in the preparation, we are going to implement a VGGNet model as a classifier. From the previous session you know the setup of classical Multilayer Perceptrons.

Q1a: Which other layer types appear in the VGG architecture?

Convolutional , Relu, Maxpool, FC

Q1b: How can a VGG net be structured in small functional units (3-4 layers)?

... **write your answers/ideas in this box** The blocks of the VGGNet should look like this: 

Note that this structure is a little bit different from the original architecture. and reflects some recent advances in Deep Learning

TASK Implement the function below:

Hint: implement a functional block by defining a new python class which inherits from nn.Module (see pytorch doc). You have to write an *init* and *forward* method.

```
In [3]: class VGGBlock(nn.Module):
            def __init__(self, ifeat, ofeat, N=2):
                super(VGGBlock, self).__init__()
                assert(N in (2, 3))

                '''
                self.N=N
                # setup all layers inside a VGG block here
                self.conv1 = nn.Conv2d(ifeat, ofeat,kernel_size=3, padding=0)

                self.conv2 = nn.Conv2d(ofeat, ofeat,kernel_size=3, padding=0)

                self.conv3 = nn.Conv2d(ofeat, ofeat,kernel_size=1, padding=0)
                self.bnl1 = nn.BatchNorm2d(ofeat)
                self.bnl2 = nn.BatchNorm2d(ofeat)
                self.bnl3 = nn.BatchNorm2d(ofeat)
                self.maxpool = nn.MaxPool2d((3,3), stride=2)
                '''

                self.N = N
                self.conv1 = nn.Conv2d(ifeat, ofeat, 3, padding=0)
                self.conv2 = nn.Conv2d(ofeat, ofeat, 3, padding=0)
                self.bn1 = nn.BatchNorm2d(ofeat)
                self.bn2 = nn.BatchNorm2d(ofeat)
                self.maxpool = nn.MaxPool2d((3, 3), stride=2)
                if 3==N:
                    self.conv3 = nn.Conv2d(ofeat, ofeat, 1, padding=0)
                    self.bn3 = nn.BatchNorm2d(ofeat)
```

```
        def forward(self, x):
            # define the forward method,  note that there are 2 or 3 convolution layers
            '''
            if self.N==2:
                x = F.relu(self.bnl1(self.conv1(x)))
                x = F.relu(self.bnl2(self.conv2(x)))
                x = self.maxpool(x)
                #print(x.shape)
                return x
            elif self.N==3:
                x = F.relu(self.bnl1(self.conv1(x)))
                x = F.relu(self.bnl2(self.conv2(x)))
                x = F.relu(self.bnl3(self.conv3(x)))
                x = self.maxpool(x)
                return x
            return x
            '''
            x = self.conv1(x)
            x = self.bn1(x)
            x = F.relu(x)
            x = self.conv2(x)
            x = self.bn2(x)
            x = F.relu(x)
            if self.N==3:
                x = self.conv3(x)
                x = self.bn3(x)
                x = F.relu(x)
            x = self.maxpool(x)
            return x
```

1.2.2 Main Architecture

The standard VGG Net is used to predict a single class. Recall that in PascalVOC multiple objects may be present in an image.

Q2: What changes between a single and multi-label scenario?

... write your answers/ideas in this box Since the number of classes in PascalVOC is much smaller than in the ILSCVR Challenge (where VGG16 was benchmarked) the number of parameters for the Linear Layers can drastically be reduced in this session. Use *1024* instead of *4096* parameters.

TASK Using the block diagram in the preparation and your pytorch module above, implement a VGG-16 network as nn.Module.

```
In [4]: class VGG16(nn.Module):
            def __init__(self):
                super(VGG16, self).__init__()
```

```
        # set up the blocks for the feature extractor part
        self.block1 = VGGBlock(3, 64, N=2)
        self.block2 = VGGBlock(64, 128,N=2)
        self.block3 = VGGBlock(128, 256, N=3)
        self.block4 = VGGBlock(256,512, N=3)
        self.block5 = VGGBlock(512,512, N=3)

        k = 5

        self.FC1=nn.Linear(512*k*k,1024)
        self.DO1=nn.Dropout(p=0.5, inplace = True)
        self.FC2=nn.Linear(1024,1024)
        self.DO2=nn.Dropout(p=0.1, inplace = True)
        self.FC3=nn.Linear(1024,20)

        # set up the dense layers here, this is the classifier part of the network
        # don't forget the Dropout for a better learning behaviour
        # ... your part

    def forward(self, x):
        # implement the forward function
        x=self.block1(x)
        x=self.block2(x)
        x=self.block3(x)
        x=self.block4(x)
        x=self.block5(x)



        #print(x.shape)  # useful for finding the 'k' above
        x = x.view(x.size(0), -1)  # this call of view transforms the 2D feature field
        # implement the classifier function
        x=F.relu(self.DO1(self.FC1(x)))
        x=F.relu(self.DO2(self.FC2(x)))
        x=F.relu(self.FC3(x))

        return x
```

In [5]: 512*25

Out[5]: 12800

1.2.3 Data Preparation

This part covers necessary preparations to use the images from PascalVOC in the training process. Read through the implementations below and using the pytorch doc explore what is done here.

```
In [6]: def get_data_loaders():
            mean_std = ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

            inp_transform = standard_transforms.Compose([
                standard_transforms.Pad(300),
                standard_transforms.CenterCrop(500),
                standard_transforms.Resize(320),
                standard_transforms.RandomHorizontalFlip(),  # We can include this here, since
                standard_transforms.ToTensor(),
                standard_transforms.Normalize(*mean_std)
            ])

            tgt_transform = standard_transforms.Compose([
                standard_transforms.Pad(300),
                standard_transforms.CenterCrop(500),
                standard_transforms.Resize(320),
                standard_transforms.ToTensor()])

            train_set = dsetVOC.VOC('train', transform=inp_transform, target_transform=tgt_tran
            valid_set = dsetVOC.VOC('valid', transform=inp_transform, target_transform=tgt_tran

            train_loader = DataLoader(train_set, batch_size=16, num_workers=4, shuffle=True)
            valid_loader = DataLoader(valid_set, batch_size=1, num_workers=4, shuffle=False)

            return train_loader, valid_loader
```

1.3 Training Loop

Q3: How is the training process structured? Which steps form an epoch?

. . . write your answers/ideas in this box **TASK** Implement a training loop for the VGG16 model. In PyTorch you need to set up model, loss function and optimizer. This is done as initialization before entering the loop. During trainig we iterate multiple times through the dataset until the loss is not reduced any further. Iteration in mini-batches is necessary since using the entire dataset at once would largely exceed the GPUs memory capacity.

For an improved estimation of the actual performance we iterate over the validation data as well (*Recap* session on unbiased evaluation).

```
In [7]: network = VGG16()

        network.cuda()

        # initialize your loss criterion
        #ost_func = nn.BCEwithLogitsloss()
        criterion = torch.nn.BCEWithLogitsLoss()


        # initialize an optimizer object
```

```python
optimizer= optim.SGD(network.parameters(),lr=0.001,  weight_decay=1e-7)

train_loader, valid_loader = get_data_loaders()

for epoch in range(11):

    network.train(True)
    ep_train_losses = []
    for batch_nr, data in enumerate(train_loader):
        # get the required data and labels and wrap them in variables for the GPU proc
        '''
        img, mask, classes =data
        input_images =Variable(img).cuda()
        labels = Variable(classes).cuda()

        output = network(input_images)

        loss =cost_func(output,labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        ep_train_losses.append(loss)
        '''


        imgs, _, lbls = data
        inputs = Variable(imgs).cuda()
        labels = Variable(lbls).cuda()

        outputs = network(inputs)

        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
# zero old gradient values
# compute new gradients
# update weights
        ep_train_losses.append(loss.cpu().data.numpy())
        if batch_nr % 10 == 0:
            print('\r{} mean {:.5f}'.format(batch_nr, 100*np.mean(ep_train_losses)))
        #print(loss.cpu().data.numpy())

        # ...

        # (optional) write some visualization to check if it works
        # ...
```

```
            # compute a loss from the network output
            # ...

            # ...  # zero old gradient values

            # ...  # compute new gradients
            # ...  # update weights

            # write some updating output for the current loss
            # print("batch_number",batch_nr,"loss",ep_train_losses(batch_nr))
            # ...


        network.train(False)
        ep_valid_losses = []
        for batch_nr, data in enumerate(valid_loader):
            # repeat the steps above for the validation set
            # which steps have to be skipped?
            '''

            img, mask, classes =data
            input_images =Variable(img).cuda()
            labels = Variable(classes).cuda()

            output = network(input_images)

            loss =cost_func(output,labels)

            ep_train_losses.append(loss)
            '''
            imgs, _, lbls = data
            inputs = Variable(imgs).cuda()
            labels = Variable(lbls).cuda()
            outputs = network(inputs)

            loss = criterion(outputs, labels)
            ep_valid_losses.append(loss.cpu().data.numpy())
        print('\rValidation {} mean {:.5f} std {:.5f}'.format(epoch, 100*np.mean(ep_valid_l

        #torch.save(network, './model_ep{:02d}_adam.pth'.format(epoch))
    torch.save(network, './model_ep{:02d}_adam.pth'.format(epoch))

    # save your trained model
    # ...
```

```
0 mean 71.29960
10 mean 71.03553
```

```
20 mean 70.93263
30 mean 70.83833
40 mean 70.71649
50 mean 70.63509
60 mean 70.54645
70 mean 70.47180
80 mean 70.40493
90 mean 70.34268
100 mean 70.28902
110 mean 70.24473
120 mean 70.20467
130 mean 70.16270
140 mean 70.12650
150 mean 70.09365
160 mean 70.06394
170 mean 70.03664
180 mean 70.00948
190 mean 69.98473
200 mean 69.96211
210 mean 69.94058
220 mean 69.92265
230 mean 69.90364
240 mean 69.88649
250 mean 69.87066
260 mean 69.85623
270 mean 69.84188
280 mean 69.82876
290 mean 69.81640
300 mean 69.80423
310 mean 69.79216
320 mean 69.78123
330 mean 69.77056
340 mean 69.76030
350 mean 69.75130
360 mean 69.74192
370 mean 69.73390
380 mean 69.72556
390 mean 69.71756
400 mean 69.71006
410 mean 69.70325
420 mean 69.69664
430 mean 69.68972
440 mean 69.68328
450 mean 69.67689
460 mean 69.67053
470 mean 69.66494
480 mean 69.65932
490 mean 69.65412
```

```
500 mean 69.64886
510 mean 69.64349
520 mean 69.63822
530 mean 69.63317
Validation 0 mean 69.31025 std 0.01611
0 mean 69.34968
10 mean 69.37881
20 mean 69.38261
30 mean 69.38185
40 mean 69.38134
50 mean 69.37999
60 mean 69.38053
70 mean 69.38277
80 mean 69.38300
90 mean 69.38224
100 mean 69.37999
110 mean 69.37870
120 mean 69.37807
130 mean 69.37700
140 mean 69.37532
150 mean 69.37466
160 mean 69.37361
170 mean 69.37303
180 mean 69.37215
190 mean 69.37139
200 mean 69.37092
210 mean 69.37026
220 mean 69.36992
230 mean 69.36898
240 mean 69.36909
250 mean 69.36859
260 mean 69.36821
270 mean 69.36799
280 mean 69.36789
290 mean 69.36711
300 mean 69.36662
310 mean 69.36609
320 mean 69.36572
330 mean 69.36546
340 mean 69.36510
350 mean 69.36515
360 mean 69.36476
370 mean 69.36425
380 mean 69.36371
390 mean 69.36318
400 mean 69.36283
410 mean 69.36244
420 mean 69.36222
```

```
430 mean 69.36188
440 mean 69.36169
450 mean 69.36114
460 mean 69.36085
470 mean 69.36054
480 mean 69.36025
490 mean 69.36004
500 mean 69.35977
510 mean 69.35953
520 mean 69.35912
530 mean 69.35896
Validation 1 mean 69.31450 std 0.00317
0 mean 69.34544
10 mean 69.34661
20 mean 69.34589
30 mean 69.34577
40 mean 69.34620
50 mean 69.34557
60 mean 69.34602
70 mean 69.34583
80 mean 69.34553
90 mean 69.34617
100 mean 69.34550
110 mean 69.34590
120 mean 69.34606
130 mean 69.34513
140 mean 69.34465
150 mean 69.34428
160 mean 69.34369
170 mean 69.34393
180 mean 69.34401
190 mean 69.34372
200 mean 69.34382
210 mean 69.34366
220 mean 69.34316
230 mean 69.34258
240 mean 69.34205
250 mean 69.34205
260 mean 69.34161
270 mean 69.34139
280 mean 69.34134
290 mean 69.34152
300 mean 69.34147
310 mean 69.34119
320 mean 69.34121
330 mean 69.34115
340 mean 69.34105
350 mean 69.34065
```

```
360 mean 69.34039
370 mean 69.34028
380 mean 69.34025
390 mean 69.34028
400 mean 69.34004
410 mean 69.33980
420 mean 69.33980
430 mean 69.33967
440 mean 69.33941
450 mean 69.33922
460 mean 69.33919
470 mean 69.33911
480 mean 69.33898
490 mean 69.33884
500 mean 69.33885
510 mean 69.33873
520 mean 69.33869
530 mean 69.33860
Validation 2 mean 69.31472 std 0.00001
0 mean 69.33831
10 mean 69.33736
20 mean 69.33734
30 mean 69.33297
40 mean 69.33490
50 mean 69.33402
60 mean 69.33339
70 mean 69.33268
80 mean 69.33249
90 mean 69.33192
100 mean 69.33272
110 mean 69.33316
120 mean 69.33293
130 mean 69.33293
140 mean 69.33327
150 mean 69.33348
160 mean 69.33309
170 mean 69.33321
180 mean 69.33342
190 mean 69.33354
200 mean 69.33333
210 mean 69.33351
220 mean 69.33380
230 mean 69.33358
240 mean 69.33354
250 mean 69.33343
260 mean 69.33344
270 mean 69.33361
280 mean 69.33337
```

```
290 mean 69.33343
300 mean 69.33308
310 mean 69.33291
320 mean 69.33280
330 mean 69.33253
340 mean 69.33242
350 mean 69.33247
360 mean 69.33244
370 mean 69.33239
380 mean 69.33240
390 mean 69.33246
400 mean 69.33233
410 mean 69.33215
420 mean 69.33213
430 mean 69.33202
440 mean 69.33200
450 mean 69.33184
460 mean 69.33190
470 mean 69.33182
480 mean 69.33188
490 mean 69.33166
500 mean 69.33157
510 mean 69.33143
520 mean 69.33131
530 mean 69.33121
Validation 3 mean 69.31472 std 0.00001
0 mean 69.32166
10 mean 69.32834
20 mean 69.33078
30 mean 69.33022
40 mean 69.32924
50 mean 69.32890
60 mean 69.32915
70 mean 69.32843
80 mean 69.32775
90 mean 69.32727
100 mean 69.32727
110 mean 69.32673
120 mean 69.32667
130 mean 69.32659
140 mean 69.32671
150 mean 69.32706
160 mean 69.32690
170 mean 69.32674
180 mean 69.32674
190 mean 69.32620
200 mean 69.32636
210 mean 69.32635
```

```
220 mean 69.32629
230 mean 69.32652
240 mean 69.32626
250 mean 69.32606
260 mean 69.32603
270 mean 69.32572
280 mean 69.32583
290 mean 69.32592
300 mean 69.32588
310 mean 69.32598
320 mean 69.32592
330 mean 69.32594
340 mean 69.32600
350 mean 69.32602
360 mean 69.32580
370 mean 69.32576
380 mean 69.32575
390 mean 69.32575
400 mean 69.32563
410 mean 69.32566
420 mean 69.32580
430 mean 69.32579
440 mean 69.32583
450 mean 69.32577
460 mean 69.32569
470 mean 69.32564
480 mean 69.32560
490 mean 69.32575
500 mean 69.32580
510 mean 69.32580
520 mean 69.32573
530 mean 69.32570
Validation 4 mean 69.31472 std 0.00001
0 mean 69.31488
10 mean 69.32226
20 mean 69.32090
30 mean 69.32164
40 mean 69.32388
50 mean 69.32428
60 mean 69.32437
70 mean 69.32441
80 mean 69.32440
90 mean 69.32441
100 mean 69.32421
110 mean 69.32468
120 mean 69.32472
130 mean 69.32408
140 mean 69.32398
```

```
150 mean 69.32405
160 mean 69.32386
170 mean 69.32406
180 mean 69.32397
190 mean 69.32368
200 mean 69.32364
210 mean 69.32374
220 mean 69.32384
230 mean 69.32385
240 mean 69.32401
250 mean 69.32386
260 mean 69.32390
270 mean 69.32386
280 mean 69.32406
290 mean 69.32397
300 mean 69.32395
310 mean 69.32390
320 mean 69.32384
330 mean 69.32384
340 mean 69.32380
350 mean 69.32389
360 mean 69.32392
370 mean 69.32390
380 mean 69.32396
390 mean 69.32392
400 mean 69.32389
410 mean 69.32378
420 mean 69.32383
430 mean 69.32374
440 mean 69.32382
450 mean 69.32381
460 mean 69.32377
470 mean 69.32370
480 mean 69.32354
490 mean 69.32362
500 mean 69.32362
510 mean 69.32366
520 mean 69.32362
530 mean 69.32361
Validation 5 mean 69.31472 std 0.00001
0 mean 69.31445
10 mean 69.31944
20 mean 69.31939
30 mean 69.31924
40 mean 69.31980
50 mean 69.31991
60 mean 69.32144
70 mean 69.32180
```

```
80 mean 69.32212
90 mean 69.32266
100 mean 69.32245
110 mean 69.32264
120 mean 69.32318
130 mean 69.32281
140 mean 69.32250
150 mean 69.32233
160 mean 69.32230
170 mean 69.32220
180 mean 69.32232
190 mean 69.32220
200 mean 69.32210
210 mean 69.32207
220 mean 69.32219
230 mean 69.32204
240 mean 69.32206
250 mean 69.32226
260 mean 69.32229
270 mean 69.32223
280 mean 69.32212
290 mean 69.32214
300 mean 69.32212
310 mean 69.32209
320 mean 69.32219
330 mean 69.32213
340 mean 69.32201
350 mean 69.32203
360 mean 69.32221
370 mean 69.32225
380 mean 69.32217
390 mean 69.32222
400 mean 69.32217
410 mean 69.32221
420 mean 69.32217
430 mean 69.32217
440 mean 69.32231
450 mean 69.32219
460 mean 69.32233
470 mean 69.32240
480 mean 69.32231
490 mean 69.32226
500 mean 69.32226
510 mean 69.32217
520 mean 69.32209
530 mean 69.32204
Validation 6 mean 69.31472 std 0.00001
0 mean 69.31737
```

```
10 mean 69.31586
20 mean 69.31790
30 mean 69.31797
40 mean 69.31815
50 mean 69.31837
60 mean 69.31896
70 mean 69.31905
80 mean 69.31906
90 mean 69.31969
100 mean 69.31972
110 mean 69.31974
120 mean 69.32014
130 mean 69.32022
140 mean 69.32024
150 mean 69.32023
160 mean 69.32004
170 mean 69.31981
180 mean 69.31995
190 mean 69.32015
200 mean 69.32009
210 mean 69.32006
220 mean 69.32000
230 mean 69.31999
240 mean 69.32009
250 mean 69.31996
260 mean 69.32016
270 mean 69.32033
280 mean 69.32027
290 mean 69.32034
300 mean 69.32047
310 mean 69.32030
320 mean 69.32020
330 mean 69.32027
340 mean 69.32027
350 mean 69.32033
360 mean 69.32030
370 mean 69.32042
380 mean 69.32038
390 mean 69.32036
400 mean 69.32027
410 mean 69.32032
420 mean 69.32024
430 mean 69.32020
440 mean 69.32026
450 mean 69.32017
460 mean 69.32021
470 mean 69.32016
480 mean 69.32017
```

```
490 mean 69.32016
500 mean 69.32026
510 mean 69.32030
520 mean 69.32031
530 mean 69.32033
Validation 7 mean 69.31472 std 0.00001
0 mean 69.31190
10 mean 69.32433
20 mean 69.32389
30 mean 69.32362
40 mean 69.32305
50 mean 69.32212
60 mean 69.32183
70 mean 69.32181
80 mean 69.32138
90 mean 69.32114
100 mean 69.32122
110 mean 69.32117
120 mean 69.32105
130 mean 69.32104
140 mean 69.32096
150 mean 69.32089
160 mean 69.32086
170 mean 69.32065
180 mean 69.32065
190 mean 69.32065
200 mean 69.32076
210 mean 69.32057
220 mean 69.32059
230 mean 69.32060
240 mean 69.32053
250 mean 69.32065
260 mean 69.32055
270 mean 69.32060
280 mean 69.32065
290 mean 69.32056
300 mean 69.32042
310 mean 69.32043
320 mean 69.32039
330 mean 69.32024
340 mean 69.32031
350 mean 69.32029
360 mean 69.32041
370 mean 69.32044
380 mean 69.32049
390 mean 69.32042
400 mean 69.32044
410 mean 69.32049
```

```
420 mean 69.32047
430 mean 69.32045
440 mean 69.32039
450 mean 69.32033
460 mean 69.32037
470 mean 69.32032
480 mean 69.32034
490 mean 69.32031
500 mean 69.32029
510 mean 69.32024
520 mean 69.32029
530 mean 69.32026
Validation 8 mean 69.31472 std 0.00001
0 mean 69.32043
10 mean 69.32103
20 mean 69.32047
30 mean 69.32054
40 mean 69.32029
50 mean 69.31957
60 mean 69.31964
70 mean 69.31944
80 mean 69.31940
90 mean 69.31922
100 mean 69.31913
110 mean 69.31869
120 mean 69.31869
130 mean 69.31878
140 mean 69.31886
150 mean 69.31892
160 mean 69.31872
170 mean 69.31883
180 mean 69.31902
190 mean 69.31912
200 mean 69.31930
210 mean 69.31933
220 mean 69.31959
230 mean 69.31961
240 mean 69.31960
250 mean 69.31955
260 mean 69.31959
270 mean 69.31961
280 mean 69.31968
290 mean 69.31971
300 mean 69.31964
310 mean 69.31958
320 mean 69.31959
330 mean 69.31961
340 mean 69.31957
```

```
350 mean 69.31952
360 mean 69.31948
370 mean 69.31948
380 mean 69.31951
390 mean 69.31944
400 mean 69.31945
410 mean 69.31941
420 mean 69.31936
430 mean 69.31933
440 mean 69.31931
450 mean 69.31927
460 mean 69.31927
470 mean 69.31932
480 mean 69.31928
490 mean 69.31923
500 mean 69.31929
510 mean 69.31933
520 mean 69.31934
530 mean 69.31933
Validation 9 mean 69.31472 std 0.00001
0 mean 69.31335
10 mean 69.31621
20 mean 69.31853
30 mean 69.31867
40 mean 69.31892
50 mean 69.31879
60 mean 69.31829
70 mean 69.31818
80 mean 69.31820
90 mean 69.31855
100 mean 69.31863
110 mean 69.31852
120 mean 69.31850
130 mean 69.31852
140 mean 69.31834
150 mean 69.31825
160 mean 69.31838
170 mean 69.31833
180 mean 69.31852
190 mean 69.31852
200 mean 69.31841
210 mean 69.31840
220 mean 69.31847
230 mean 69.31846
240 mean 69.31875
250 mean 69.31882
260 mean 69.31889
270 mean 69.31890
```

```
280 mean 69.31885
290 mean 69.31897
300 mean 69.31901
310 mean 69.31905
320 mean 69.31902
330 mean 69.31901
340 mean 69.31911
350 mean 69.31925
360 mean 69.31925
370 mean 69.31916
380 mean 69.31907
390 mean 69.31904
400 mean 69.31900
410 mean 69.31900
420 mean 69.31903
430 mean 69.31896
440 mean 69.31901
450 mean 69.31911
460 mean 69.31906
470 mean 69.31903
480 mean 69.31897
490 mean 69.31906
500 mean 69.31914
510 mean 69.31908
520 mean 69.31903
530 mean 69.31904
Validation 10 mean 69.31472 std 0.00001
```

```
/usr/local/anaconda3/lib/python3.6/site-packages/torch/serialization.py:241: UserWarning: Coul
  "type " + obj.__name__ + ". It won't be checked "
/usr/local/anaconda3/lib/python3.6/site-packages/torch/serialization.py:241: UserWarning: Coul
  "type " + obj.__name__ + ". It won't be checked "
```

1.3.1 Visualize some examples

Verification is important! To assess the quality of the model we have to find suitable measures to quantify the results for our test and validation data (ideally in a way we can easily understand), but more often than not, it will help tremendously to simply browse through some examples and verify the model by manual inspection.

```
In [8]: network = torch.load('./model_ep10_adam.pth')
        network.train(False)

        _, browse_loader = get_data_loaders()
        cls2lbl = dsetVOC.class_to_label
```

```python
num_labels = 1e-12
top_5_scores = []
top_1_scores = []
myscore = [ [] for i in range(5) ]
for img_nr, data in enumerate(browse_loader):
    imgs, _, lbls = data

    inputs = Variable(imgs).cuda()
    output = torch.sigmoid(network(inputs))
    prediction = np.squeeze(output.cpu().data.numpy())
    args = np.argsort(prediction)
    top5 = np.argsort(prediction)[-5:]
    top1 = np.argsort(prediction)[-1]

    labels = lbls.numpy()[0]

    # implementation top5 error
    top_5_score = 0
    for ll in np.where(labels>0)[0]:
        if ll in top5:
            top_5_score += 1
    top_5_score = top_5_score/np.sum(labels)
    top_5_scores.append(top_5_score)

    # Implement your on error measure
    # ...
    num_labels = np.sum(labels)
    top_predicted = np.argsort(prediction)[-num_labels:]
    if np.in1d(top_predicted, np.where(labels==1)).all():
        top_1_scores.append(1)
    else:
        top_1_scores.append(0)
    print('#########')


print('Top 5 score: {:.2f}%'.format(100*np.mean(top_5_scores)))
# Print your on error measure
# ...
print('All correct classified: {:.2f}%'.format(100*np.mean(top_1_scores)))
```

```
/usr/local/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:34: VisibleDeprecationWa

#########
#########
#########
#########
#########
```

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########

```
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
#########
Top 5 score: 12.69%
```

1.3.2 Top-N Accuracy

Extend the above computation by implementing a Top-5 error measure, i.e. we score whenever a label we expect from the ground-truth appears in our Top-5 predictions.

Statement: The Top-5 error is a fair measure for this evaluation.

Q4: Think, discuss, reason.

Q5: Suggest and implement your own idea to compute a human interpretable score for the network performance.

... write your answers/ideas in this box We hope you enjoyed the session! - Feel free to give your feedback and help us improve this lab...

... your ideas for a improvements, peace and a better world in general, here pls :D