PE_projekt

February 23, 2021

[1]:
```
import os
import numpy as np
import torch

import torchvision

from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt

%matplotlib inline
```

[2]:
```
# check if CUDA is available
train_on_gpu = False # torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available.  Training on CPU ...')
else:
    print('CUDA is available!  Training on GPU ...')
```

```
CUDA is not available.  Training on CPU …
```

[3]:
```
# define training and test data directories
data_dir = 'dataset/'
train_dir = os.path.join(data_dir, 'train/')
test_dir = os.path.join(data_dir, 'test/')
valid_dir = os.path.join(data_dir, 'valid/')

# classes are folders in each directory with these names
classes = ['melanoma', 'nevus', 'seborrheic_keratosis']
```

[4]:
```
# load and transform data using ImageFolder

# VGG-16 Takes 224x224 images as input, so we resize all of them
data_transform = transforms.Compose([transforms.Resize(250),
                                     transforms.CenterCrop(224),
                                      transforms.ToTensor()])
train_data = datasets.ImageFolder(train_dir, transform=data_transform)
test_data = datasets.ImageFolder(test_dir, transform=data_transform)
```

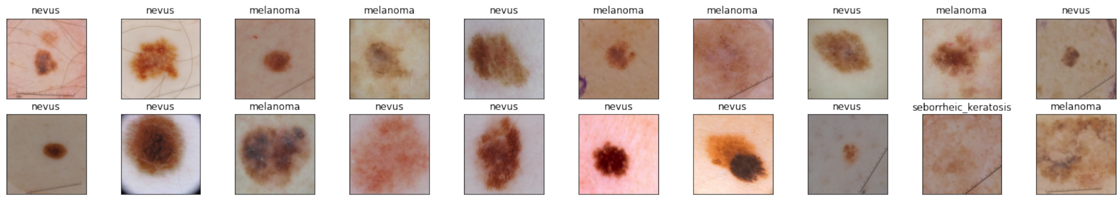
```python
valid_data = datasets.ImageFolder(valid_dir, transform=data_transform)
# print out some data stats
print('Num training images: ', len(train_data))
print('Num test images: ', len(test_data))
print('Num validation images: ', len(valid_data))
```

```
Num training images:  2000
Num test images:  600
Num validation images:  150
```

[5]:
```python
# define dataloader parameters
batch_size = 20
num_workers=0

# prepare data loaders
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
                                           num_workers=num_workers,␣
 ↪shuffle=True)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size,
                                          num_workers=num_workers, shuffle=True)
valid_loader = torch.utils.data.DataLoader(valid_data, batch_size=batch_size,
                                          num_workers=num_workers, shuffle=True)
```

[6]:
```python
# Visualize some sample data

# obtain one batch of training images
dataiter = iter(train_loader)
images, labels = dataiter.next()
images = images.numpy() # convert images to numpy for display

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(20):
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    plt.imshow(np.transpose(images[idx], (1, 2, 0)))
    ax.set_title(classes[labels[idx]])
```

```
/home/lozinske/miniconda3/envs/PE_projekt/lib/python3.7/site-
packages/ipykernel_launcher.py:11: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
  # This is added back by InteractiveShellApp.init_path()
```



[7]:
```python
# Load the pretrained model from pytorch
vgg16 = models.vgg16(pretrained=True)

# print out the model structure
print(vgg16)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

[8]:
```python
print(vgg16.classifier[6].in_features)
print(vgg16.classifier[6].out_features)
```

```
4096
1000
```

[9]:
```python
# Freeze training for all "features" layers
for param in vgg16.features.parameters():
    param.require_grad = False
```

[10]:
```python
import torch.nn as nn

n_inputs = vgg16.classifier[6].in_features

# add last linear layer (n_inputs -> 3 classes)
# new layers automatically have requires_grad = True
last_layer = nn.Linear(n_inputs, len(classes))

vgg16.classifier[6] = last_layer

# if GPU is available, move the model to GPU
if train_on_gpu:
    vgg16.cuda()

# check to see that your last layer produces the expected number of outputs
print(vgg16.classifier[6].out_features)
```

```
3
```

[11]:
```
import torch.optim as optim
# CrossEntropyLoss function is good with dataset with multiple unbalanced␣
↪classes (different number of samples)
# specify loss function (categorical cross-entropy)
criterion = nn.CrossEntropyLoss()

# specify optimizer (stochastic gradient descent) and learning rate = 0.001,␣
↪optimize parameters in loss function
optimizer = optim.SGD(vgg16.classifier.parameters(), lr=0.001) # lr depends on␣
↪learning results(may differ)
```

[12]:
```
# number of epochs to train the model
n_epochs = 20
valid_loss_min = np.Inf

for epoch in range(1, n_epochs+1):

    # keep track of training and validation loss
    train_loss = 0.0
    valid_loss = 0.0

    ###################
    # train the model #
    ###################
    # model by default is set to train
    vgg16.train()
    for batch_i, (data, target) in enumerate(train_loader):
        # move tensors to GPU if CUDA is available
        if train_on_gpu:
            data, target = data.cuda(), target.cuda()
        # clear the gradients of all optimized variables, to prevent error's␣
↪accumulation
        optimizer.zero_grad()
        # forward pass: compute predicted outputs by passing inputs to the model
        output = vgg16(data)
        # calculate the batch loss
        loss = criterion(output, target)
        # backward pass: compute gradient of the loss with respect to model␣
↪parameters
        loss.backward()
        # perform a single optimization step (parameter update)
        optimizer.step()
        # update training loss
        train_loss += loss.item()

        if batch_i % 20 == 19:    # print training loss every specified number␣
↪of mini-batches
```

```
            print('Training: Epoch %d, Batch %d loss: %.16f' %
                  (epoch, batch_i + 1, train_loss / 20))

    vgg16.eval() # prep model for evaluation
    for batch_i, (data, target) in enumerate(valid_loader):
        # forward pass: compute predicted outputs by passing inputs to the model
        output = vgg16(data)
        # calculate the loss
        loss = criterion(output, target)
        # update running validation loss
        valid_loss += loss.item()

        if batch_i % 20 == 19:    # print training loss every specified number of mini-batches
            print('Validation: Epoch %d, Batch %d loss: %.16f' %
                  (epoch, batch_i + 1, valid_loss / 20))

    # print training/validation statistics
    # calculate average loss over an epoch
    train_loss = train_loss/len(train_loader.dataset)
    valid_loss = valid_loss/len(valid_loader.dataset)

    print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
        epoch+1,
        train_loss,
        valid_loss
        ))

    # save model if validation loss has decreased
    if valid_loss <= valid_loss_min:
        print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
        valid_loss_min,
        valid_loss))
        torch.save(vgg16.state_dict(), 'model.pt')
        valid_loss_min = valid_loss
```

```
Training: Epoch 1, Batch 20 loss: 0.8937232404947281
Training: Epoch 1, Batch 40 loss: 1.7520716816186905
Training: Epoch 1, Batch 60 loss: 2.5959528654813768
Training: Epoch 1, Batch 80 loss: 3.3879931241273882
Training: Epoch 1, Batch 100 loss: 4.1711155086755749
Epoch: 2        Training Loss: 0.041711         Validation Loss: 0.054611
Validation loss decreased (inf --> 0.054611).  Saving model …
Training: Epoch 2, Batch 20 loss: 0.7967709451913834
Training: Epoch 2, Batch 40 loss: 1.5836860701441764
Training: Epoch 2, Batch 60 loss: 2.3688531503081323
```

```
Training: Epoch 2, Batch 80 loss: 3.1566525831818582
Training: Epoch 2, Batch 100 loss: 3.9125271186232569
Epoch: 3        Training Loss: 0.039125         Validation Loss: 0.053532
Validation loss decreased (0.054611 --> 0.053532).  Saving model …
Training: Epoch 3, Batch 20 loss: 0.7570281594991684
Training: Epoch 3, Batch 40 loss: 1.4544058024883271
Training: Epoch 3, Batch 60 loss: 2.2318968027830124
Training: Epoch 3, Batch 80 loss: 2.9838651895523070
Training: Epoch 3, Batch 100 loss: 3.7736583530902861
Epoch: 4        Training Loss: 0.037737         Validation Loss: 0.050985
Validation loss decreased (0.053532 --> 0.050985).  Saving model …
Training: Epoch 4, Batch 20 loss: 0.7237058714032173
Training: Epoch 4, Batch 40 loss: 1.4230968475341796
Training: Epoch 4, Batch 60 loss: 2.1800740018486975
Training: Epoch 4, Batch 80 loss: 2.9340826943516731
Training: Epoch 4, Batch 100 loss: 3.6843535974621773
Epoch: 5        Training Loss: 0.036844         Validation Loss: 0.049986
Validation loss decreased (0.050985 --> 0.049986).  Saving model …
Training: Epoch 5, Batch 20 loss: 0.7418365031480789
Training: Epoch 5, Batch 40 loss: 1.4786423802375794
Training: Epoch 5, Batch 60 loss: 2.2104373246431352
Training: Epoch 5, Batch 80 loss: 2.9078957885503769
Training: Epoch 5, Batch 100 loss: 3.6336459845304487
Epoch: 6        Training Loss: 0.036336         Validation Loss: 0.051263
Training: Epoch 6, Batch 20 loss: 0.6575327008962631
Training: Epoch 6, Batch 40 loss: 1.3752990230917930
Training: Epoch 6, Batch 60 loss: 2.0779342621564867
Training: Epoch 6, Batch 80 loss: 2.7813130050897596
Training: Epoch 6, Batch 100 loss: 3.5476864770054819
Epoch: 7        Training Loss: 0.035477         Validation Loss: 0.049243
Validation loss decreased (0.049986 --> 0.049243).  Saving model …
Training: Epoch 7, Batch 20 loss: 0.6547839269042015
Training: Epoch 7, Batch 40 loss: 1.3855033650994302
Training: Epoch 7, Batch 60 loss: 2.0509373128414152
Training: Epoch 7, Batch 80 loss: 2.7901690840721129
Training: Epoch 7, Batch 100 loss: 3.5324532687664032
Epoch: 8        Training Loss: 0.035325         Validation Loss: 0.049117
Validation loss decreased (0.049243 --> 0.049117).  Saving model …
Training: Epoch 8, Batch 20 loss: 0.6893760800361634
Training: Epoch 8, Batch 40 loss: 1.4061234891414642
Training: Epoch 8, Batch 60 loss: 2.0938444033265116
Training: Epoch 8, Batch 80 loss: 2.7429019048810006
Training: Epoch 8, Batch 100 loss: 3.4175614401698113
Epoch: 9        Training Loss: 0.034176         Validation Loss: 0.049955
Training: Epoch 9, Batch 20 loss: 0.6904447585344314
Training: Epoch 9, Batch 40 loss: 1.4238117396831513
Training: Epoch 9, Batch 60 loss: 2.1013706773519516
Training: Epoch 9, Batch 80 loss: 2.7333745867013932
```

```
Training: Epoch 9, Batch 100 loss: 3.3881410002708434
Epoch: 10       Training Loss: 0.033881         Validation Loss: 0.046973
Validation loss decreased (0.049117 --> 0.046973).  Saving model …
Training: Epoch 10, Batch 20 loss: 0.6508076488971710
Training: Epoch 10, Batch 40 loss: 1.3149775698781014
Training: Epoch 10, Batch 60 loss: 1.9995783671736718
Training: Epoch 10, Batch 80 loss: 2.6770193666219710
Training: Epoch 10, Batch 100 loss: 3.3526929676532746
Epoch: 11       Training Loss: 0.033527         Validation Loss: 0.047722
Training: Epoch 11, Batch 20 loss: 0.6703696221113205
Training: Epoch 11, Batch 40 loss: 1.3369824320077897
Training: Epoch 11, Batch 60 loss: 2.0257978051900865
Training: Epoch 11, Batch 80 loss: 2.7131043881177903
Training: Epoch 11, Batch 100 loss: 3.3446144044399260
Epoch: 12       Training Loss: 0.033446         Validation Loss: 0.049403
Training: Epoch 12, Batch 20 loss: 0.6251097574830056
Training: Epoch 12, Batch 40 loss: 1.3113566055893897
Training: Epoch 12, Batch 60 loss: 1.9923828795552254
Training: Epoch 12, Batch 80 loss: 2.6851920038461685
Training: Epoch 12, Batch 100 loss: 3.2976576119661329
Epoch: 13       Training Loss: 0.032977         Validation Loss: 0.049921
Training: Epoch 13, Batch 20 loss: 0.6624703913927078
Training: Epoch 13, Batch 40 loss: 1.3157240837812423
Training: Epoch 13, Batch 60 loss: 1.9309425324201583
Training: Epoch 13, Batch 80 loss: 2.5767317861318588
Training: Epoch 13, Batch 100 loss: 3.2658273309469221
Epoch: 14       Training Loss: 0.032658         Validation Loss: 0.047134
Training: Epoch 14, Batch 20 loss: 0.6946130171418190
Training: Epoch 14, Batch 40 loss: 1.3513974413275718
Training: Epoch 14, Batch 60 loss: 1.9639334157109261
Training: Epoch 14, Batch 80 loss: 2.6439152702689173
Training: Epoch 14, Batch 100 loss: 3.2515433162450789
Epoch: 15       Training Loss: 0.032515         Validation Loss: 0.048121
Training: Epoch 15, Batch 20 loss: 0.6201885998249054
Training: Epoch 15, Batch 40 loss: 1.3013881355524064
Training: Epoch 15, Batch 60 loss: 1.9442445278167724
Training: Epoch 15, Batch 80 loss: 2.6474666625261305
Training: Epoch 15, Batch 100 loss: 3.2632124155759810
Epoch: 16       Training Loss: 0.032632         Validation Loss: 0.047128
Training: Epoch 16, Batch 20 loss: 0.6384350180625915
Training: Epoch 16, Batch 40 loss: 1.2155491665005684
Training: Epoch 16, Batch 60 loss: 1.8687177345156669
Training: Epoch 16, Batch 80 loss: 2.4969396248459814
Training: Epoch 16, Batch 100 loss: 3.1371933057904244
Epoch: 17       Training Loss: 0.031372         Validation Loss: 0.047257
Training: Epoch 17, Batch 20 loss: 0.6355775818228722
Training: Epoch 17, Batch 40 loss: 1.2600370973348618
Training: Epoch 17, Batch 60 loss: 1.8376768201589584
```

```
Training: Epoch 17, Batch 80 loss: 2.5044110119342804
Training: Epoch 17, Batch 100 loss: 3.1364293128252028
Epoch: 18       Training Loss: 0.031364         Validation Loss: 0.047042
Training: Epoch 18, Batch 20 loss: 0.5969465628266335
Training: Epoch 18, Batch 40 loss: 1.2547573953866960
Training: Epoch 18, Batch 60 loss: 1.8532465308904649
Training: Epoch 18, Batch 80 loss: 2.4716737613081934
Training: Epoch 18, Batch 100 loss: 3.0856217473745344
Epoch: 19       Training Loss: 0.030856         Validation Loss: 0.049140
Training: Epoch 19, Batch 20 loss: 0.6411809772253036
Training: Epoch 19, Batch 40 loss: 1.2447782739996911
Training: Epoch 19, Batch 60 loss: 1.8757026448845864
Training: Epoch 19, Batch 80 loss: 2.4807352736592292
Training: Epoch 19, Batch 100 loss: 3.1147315084934233
Epoch: 20       Training Loss: 0.031147         Validation Loss: 0.046418
Validation loss decreased (0.046973 --> 0.046418).  Saving model …
Training: Epoch 20, Batch 20 loss: 0.5723712176084519
Training: Epoch 20, Batch 40 loss: 1.1699392110109330
Training: Epoch 20, Batch 60 loss: 1.7476233392953873
Training: Epoch 20, Batch 80 loss: 2.3714562132954597
Training: Epoch 20, Batch 100 loss: 3.0541798934340476
Epoch: 21       Training Loss: 0.030542         Validation Loss: 0.047077
```

[13]:
```python
# track test loss
# over 3 classes
test_loss = 0.0
class_correct = list(0. for i in range(5))
class_total = list(0. for i in range(5))

vgg16.eval() # eval mode

# iterate over test data
for data, target in test_loader:
    # move tensors to GPU if CUDA is available
    if train_on_gpu:
        data, target = data.cuda(), target.cuda()
    # forward pass: compute predicted outputs by passing inputs to the model
    output = vgg16(data)
    # calculate the batch loss
    loss = criterion(output, target)
    # update  test loss
    test_loss += loss.item()*data.size(0)
    # convert output probabilities to predicted class
    _, pred = torch.max(output, 1)
    # compare predictions to true label
    correct_tensor = pred.eq(target.data.view_as(pred))
```

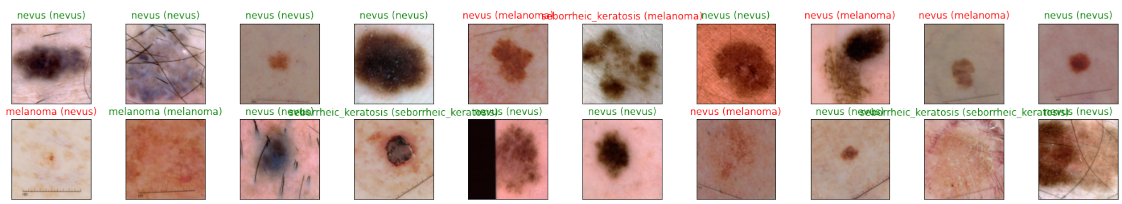
```
    correct = np.squeeze(correct_tensor.numpy()) if 0 else np.
↪squeeze(correct_tensor.cpu().numpy()) # not train_on_gpu
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

# calculate avg test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(3):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            classes[i], 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
    100. * np.sum(class_correct) / np.sum(class_total),
    np.sum(class_correct), np.sum(class_total)))
```

```
Test Loss: 0.820256

Test Accuracy of melanoma: 25% (30/117)
Test Accuracy of nevus: 80% (317/393)
Test Accuracy of seborrheic_keratosis: 36% (33/90)

Test Accuracy (Overall): 63% (380/600)
```

[14]:
```
# obtain one batch of test images
dataiter = iter(test_loader)
images, labels = dataiter.next()
images.numpy()

# move model inputs to cuda, if GPU available
if train_on_gpu:
    images = images.cuda()

# get sample outputs
output = vgg16(images)
# convert output probabilities to predicted class
_, preds_tensor = torch.max(output, 1)
preds = np.squeeze(preds_tensor.numpy()) if 0 else np.squeeze(preds_tensor.
↪cpu().numpy()) # not train_on_gpu
```

```python
# plot the images in the batch, along with predicted and true labels
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(20):
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
    plt.imshow(np.transpose(images[idx], (1, 2, 0)))
    ax.set_title("{} ({})".format(classes[preds[idx]], classes[labels[idx]]),
                 color=("green" if preds[idx]==labels[idx].item() else "red"))
```

```
/home/lozinske/miniconda3/envs/PE_projekt/lib/python3.7/site-
packages/ipykernel_launcher.py:19: MatplotlibDeprecationWarning: Passing non-
integers as three-element position specification is deprecated since 3.3 and
will be removed two minor releases later.
```



[15]:
```python
torch.save(vgg16.state_dict(), 'model_backup.pt')
```