4.1_resnet18_PyTorch

March 30, 2020

Pre-trained-Models with PyTorch

In this lab, you will use pre-trained models to classify between the negative and positive samples; you will be provided with the dataset object. The particular pre-trained model will be resnet18; you will have three questions:

change the output layer

train the model

identify several misclassified samples

You will take several screenshots of your work and share your notebook.

Table of Contents

Download Data

Imports and Auxiliary Functions

Dataset Class

Question 1

Question 2

Question 3

Estimated Time Needed: 120 min

Download Data

Download the dataset and unzip the files in your data directory, unlike the other labs, all the data will be deleted after you close the lab, this may take some time:

[1]:
```
# !wget https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/
↪CognitiveClass/DL0321EN/data/images/Positive_tensors.zip
```

```
--2020-03-30 08:57:31--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-
courses-data/CognitiveClass/DL0321EN/data/images/Positive_tensors.zip
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)… 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)|67.228.254.196|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 2598656062 (2.4G) [application/zip]
```

```
Saving to: ‘Positive_tensors.zip’

Positive_tensors.zi 100%[===================>]   2.42G  10.8MB/s    in 5m 53s

2020-03-30 09:03:24 (7.03 MB/s) - ‘Positive_tensors.zip’ saved
[2598656062/2598656062]
```

[2]:
```
# !unzip -q Positive_tensors.zip
```

[5]:
```
# !wget https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/
 ↪CognitiveClass/DL0321EN/data/images/Negative_tensors.zip

# !unzip -q Negative_tensors.zip
```

```
--2020-03-30 09:11:55--  https://s3-api.us-geo.objectstorage.softlayer.net/cf-
courses-data/CognitiveClass/DL0321EN/data/images/Negative_tensors.zip
Resolving s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)… 67.228.254.196
Connecting to s3-api.us-geo.objectstorage.softlayer.net (s3-api.us-
geo.objectstorage.softlayer.net)|67.228.254.196|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 2111408108 (2.0G) [application/zip]
Saving to: ‘Negative_tensors.zip’

Negative_tensors.zi 100%[===================>]   1.97G  8.20MB/s    in 4m 52s

2020-03-30 09:16:48 (6.91 MB/s) - ‘Negative_tensors.zip’ saved
[2111408108/2111408108]
```

We will install torchvision:

[ ]:
```
# !pip install torchvision
```

Imports and Auxiliary Functions

The following are the libraries we are going to use for this lab. The torch.manual_seed() is for forcing the random function to give the same number every time we try to recompile it.

[1]:
```
# These are the libraries will be used for this lab.
import torchvision.models as models
from PIL import Image
import pandas

from torchvision import transforms
import torch.nn as nn
import time
import torch
```

```
import matplotlib.pylab as plt
import numpy as np
from torch.utils.data import Dataset, DataLoader
import h5py
import os
import glob

torch.manual_seed(0)
```

[1]: <torch._C.Generator at 0x7f87f0013390>

[2]:
```
from matplotlib.pyplot import imshow
# import matplotlib.pylab as plt
# from PIL import Image
import pandas as pd
import os
```

Dataset Class

This dataset class is essentially the same dataset you build in the previous section, but to speed things up, we are going to use tensors instead of jpeg images. **Therefore for each iteration, you will skip the reshape step, conversion step to tensors and normalization step.**

[3]:
```
## Create your own dataset object

class Dataset(Dataset):
    ## Constructor
    def __init__(self, transform=None, train=True):
        directory = "./" # "/home/dsxuser/work"
        positive  = 'Positive_tensors'
        negative  = 'Negative_tensors'

        positive_file_path = os.path.join(directory,positive)
        negative_file_path = os.path.join(directory,negative)
        positive_files = [os.path.join(positive_file_path,file) \
                          for file in os.listdir(positive_file_path) if file.
 ↪endswith(".pt")]
        negative_files = [os.path.join(negative_file_path,file) \
                          for file in os.listdir(negative_file_path) if file.
 ↪endswith(".pt")]
        number_of_samples = len(positive_files) + len(negative_files)
        self.all_files = [None] * number_of_samples
        self.all_files[::2]  = positive_files
        self.all_files[1::2] = negative_files

        self.transform = transform # The transform is goint to be used on image
        self.Y=torch.zeros([number_of_samples]).type(torch.LongTensor) # torch.
 ↪LongTensor
```

```
        self.Y[::2]  = 1
        self.Y[1::2] = 0

        if train:
            self.all_files = self.all_files[0:30000]
            self.Y = self.Y[0:30000]
        else:
            self.all_files = self.all_files[30000:]
            self.Y = self.Y[30000:]
        self.len = len(self.all_files)

    ## Get the length
    def __len__(self):
        return self.len

    ## Getter
    def __getitem__(self, ix):
        image = torch.load(self.all_files[ix])
        y = self.Y[ix]
        # # If there is any transform method, apply it onto the image
        if self.transform: image = self.transform(image)
        return image, y

print("done")
```

```
done
```

We create two dataset objects, one for the training data and one for the validation data.

[4]:
```
train_dataset = Dataset(train=True)
validation_dataset = Dataset(train=False)
print("done")
```

```
done
```

Question 1

Prepare a pre-trained resnet18 model :

Step 1: Load the pre-trained model resnet18 Set the parameter pretrained to true:

[5]:
```
## Type your code here

model = models.resnet18(pretrained=True)
```

Step 2: Set the attribute requires_grad to False. As a result, the parameters will not be affected by training.

[6]:
```
## Type your code here
```

```
for parm in model.parameters(): parm.requires_grad = False
```

Step 3: Replace the output layer model.fc of the neural network with a nn.Linear object, to classify 2 different classes. For the parameters in_features remember the last hidden layer has 512 neurons.

[7]:
```
## Type your code here

model.fc = nn.Linear(in_features=512, out_features=2) # nn.Linear(512, 2)
```

Print out the model in order to show whether you get the correct answer. (Your peer reviewer is going to mark based on what you print here.)

[8]:
```
print(model)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
```

```
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=2, bias=True)
)
```

Question 2: Train the Model

In this question you will train your, model:

Step 1: Create a cross entropy criterion function

[9]:
```
## Type your code here
```

```
criterion = nn.CrossEntropyLoss()
```

Step 2: Create a training loader and validation loader object, the batch size should have 100 samples each.

[10]:
```
## Type your code here

training_loader = torch.utils.data.DataLoader(dataset=train_dataset,␣
↪batch_size=100)
                                                # shuffle=True)

validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset,␣
↪batch_size=100)
                                                  # shuffle=False)
```

Step 3: Use the following optimizer to minimize the loss

[11]:
```
optimizer = torch.optim.Adam([parameters for parameters in model.parameters() \
                              if parameters.requires_grad], lr=0.001)
```

Complete the following code to calculate the accuracy on the validation data for one epoch; this should take about 45 minutes.
Make sure you calculate the accuracy on the validation data.

[12]:
```
n_epochs = 1
loss_list, accuracy_list = [], []
n_val = len(validation_dataset)
# n_train = len(train_dataset)
start_time = time.time()
loss = 0

for epoch in range(n_epochs):
    for x, y in training_loader:
        model.train()
        optimizer.zero_grad()   # clear gradient
        z = model(x)            # make a prediction

        loss = criterion(z, y)  # calculate loss
        loss.backward()         # calculate gradients of parameters
        optimizer.step()        # update parameters

        loss_list.append(loss.data)

    correct = 0
    for x_val, y_val in validation_loader:
        model.eval()                                # set model to eval
        z = model(x_val)                            # make a prediction
```

```
        _, y_hat = torch.max(z, 1)                # find max
        correct += (y_hat == y_val).sum().item() # Calculate misclassified␣
↪samples in mini-batch

    accuracy = correct / n_val
    print("Epoch {:2d} - Time: {:3.5f}s / Accuracy: {:2.5f}".format(epoch + 1,
                                                                    time.time()␣
↪- start_time,
                                                                    accuracy))
```

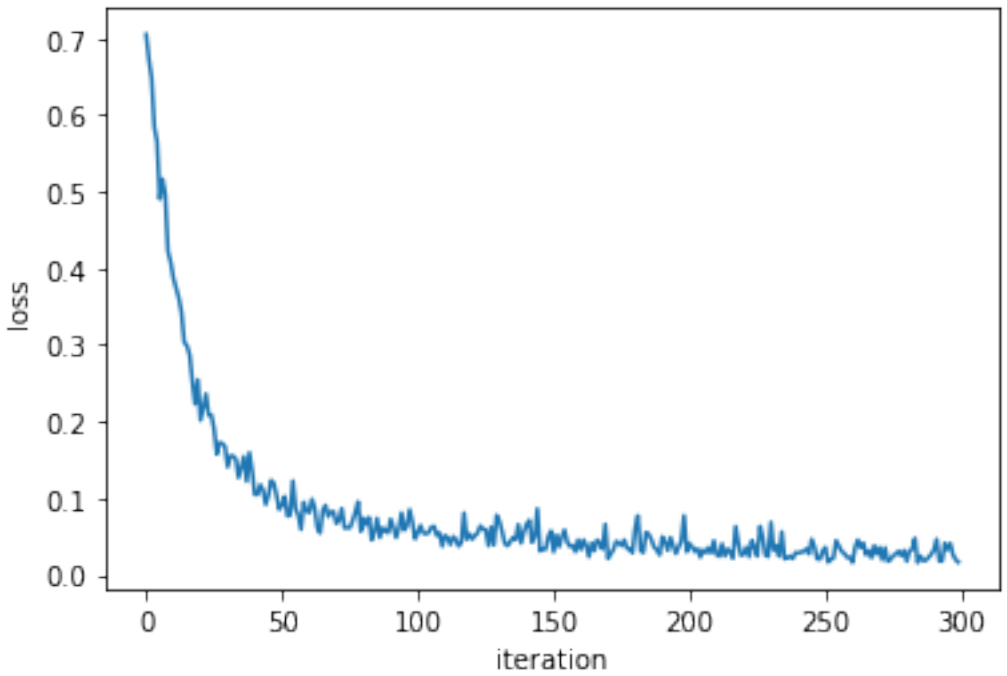
```
Epoch  1 - Time: 720.67477s / Accuracy: 0.99330
```

Print out the Accuracy and plot the loss stored in the list loss_list for every iteration and take a screen shot.

[13]:
```
accuracy
```

[13]:
```
0.9933
```

[14]:
```
plt.plot(loss_list)
plt.xlabel("iteration")
plt.ylabel("loss")
plt.show()
```



Question 3: Find the misclassified samples

Identify the first four misclassified samples using the validation data:

[44]:
```
def find_misclassified(validation_loader):
    samples = []
    n_batch = 0
    for x_val, y_val in validation_loader:
        model.eval()                              # set model to eval
        z = model(x_val)                          # make a prediction
        _, y_hat = torch.max(z, 1)

        for ix in range(0, len(y_hat)):
            if (y_hat[ix] != y_val[ix]):
                sample_num = n_batch + ix
                samples.append({'snum': sample_num, 'x': x_val[ix], 'y_val':␣
↪y_val[ix], 'y_hat': y_hat[ix]})
            if len(samples) >= 4: return samples
        n_batch += len(y_val)                     # batch of 100

    return samples
```

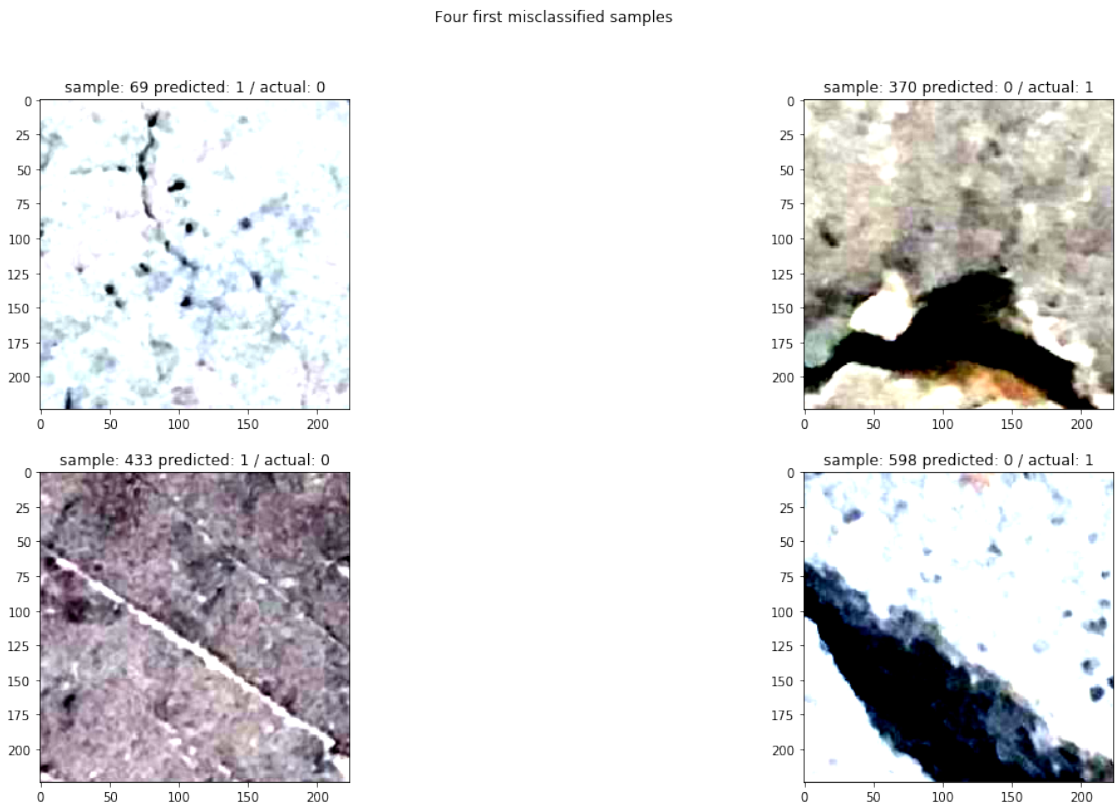
[45]:
```
samples = find_misclassified(validation_loader)
```

[46]:
```
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))

ix = 0
for ax1 in axs:
    for ax2 in ax1:
        sample = samples[ix]
        ax2.imshow(sample['x'].permute(1, 2, 0).numpy())
        ax2.title.set_text("sample: " + str(sample['snum']) + " predicted: " +␣
↪str(sample['y_hat'].item()) + \
                           " / actual: " + str(sample['y_val'].item()))
        ix += 1

fig.suptitle('Four first misclassified samples')
plt.show()
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```



CLICK HERE Click here to see how to share your notebook.

About the Authors:

Joseph Santarcangelo has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.