Chuyển giao học tập trong Pytorch

1. Phân loại

Chuyển giao học tập có 2 loại:

- Tinh chỉnh mô hình: sử dụng một mô hình được huấn luyện trước và cập nhật ***tất cả*** các tham số của mô hình trên bài toán mới, nếu cần thiết thì huấn luyện lại toàn bộ mô hình.
- Bộ trích xuất đặc trưng: sử dụng một mô hình được huấn luyện trước và chỉ cập nhật lại trọng số của tầng cuối cùng để dự đoán phân loại

Tham khảo cụ thể trong link sau: <https://cs231n.github.io/transfer-learning/>__ and here <https://ruder.io/transfer-learning/>

2. Các thao tác chung khi áp dụng tinh chỉnh mô hình và trích xuất đặc trưng

- Khởi tạo mô hình huấn luyện trước
- Điều chỉnh lại đầu ra tầng cuối cùng phù hợp với số class của bài toán phân loại
- Định nghĩa thuật toán tối ưu với các tham số cần cập nhật trong quá trình huấn luyện
- Thực hiện các bước huấn luyện mô hình

3. Thực hành trên bài toán phân loại covid (âm tính, dương tính) dựa trên ảnh x-quang ngực với VGG19_bn

Nguồn dataset: https://drive.google.com/file/d/18aMf57_1u2AWInnMB67s3Xku0sPzm28u/view

B1. Import các thư viện cần thiết

```
#importing the libraries
import pandas as pd
import numpy as np
import time
import copy
import warnings
from tqdm import tqdm

#for Data preprocessing and Augmentation
import os
from imutils import paths
import cv2
# import Augmentor
import torchvision.transforms as transforms

# from sklearn.preprocessing import OneHotEncoder

#for reading and displaying images
import matplotlib.pyplot as plt
from PIL import Image

#Pytorch libraries and modules
import torch
import torch.nn as nn
```

```
import torch.nn.functional as F
from torchsummary import summary

from torch.nn import Linear, CrossEntropyLoss
from torch.optim import Adam, lr_scheduler
from torch.utils.data import DataLoader,Dataset
from sklearn.model_selection import train_test_split

#torchvision for pre-trained models
import torchvision
from torchvision import datasets, models

#for evaluating model
from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report, ConfusionMatrixDisplay
```

B2. Dữ liệu đầu vào

A. Khởi tạo giá trị các tham số

```
BATCH_SIZE = 32
classes = ['Normal', 'Covid']
num_classes = 2
num_epochs = 15
criterion = CrossEntropyLoss()
CHECKPOINT_PATH = '../FEVGG19bn/FEVGG19bn1.pt'
path_dataset =
'/media/trucloan/Data/Research/TransferLearningVGG19bnCovid1910k_images/dataset10K_
images/'

std = np.array([0.229, 0.224, 0.225])
mean = np.array([0.485, 0.456, 0.406])
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# Flag for feature extracting. When False, we finetune the whole model,
#   when True we only update the reshaped layer params
feature_extract = False
```

B. Đường dẫn đến tập dữ liệu

```
normal = path_dataset + 'NORMAL/'
covid = path_dataset + 'COVID/'

dir_normal = os.listdir(normal)
dir_covid = os.listdir(covid)
```

```
dict_data0 = {'path': normal, 'image_name': dir_normal, 'labels': 0}
dict_data1 = {'path': covid, 'image_name': dir_covid, 'labels': 1}
```

```
df0 = pd.DataFrame(dict_data0)
df1 = pd.DataFrame(dict_data1)
df = pd.concat([df0, df1])
df.to_csv('./data.csv', index = False)
```

```
train_data, test_data=train_test_split(df, test_size=0.2, random_state = 42, shuffle=True)
train_data, val_data =train_test_split(train_data, test_size=0.2, random_state = 42, shuffle=True)
```

C. Định nghĩa lớp dataset

```
class ImageDataset(Dataset):
    def __init__(self, csv, transform):
        self.transform = transform
        self.csv = csv

        self.image_name = self.csv[:]['image_name']
        self.label = np.array(self.csv[:]['labels'])

    def __len__(self):
        return len(self.label)

    def __getitem__(self, index):
                            images = Image.open(self.csv['path'].iloc[index] + self.image_name.iloc[index]).convert('RGB')

        images = self.transform(images)
        targets = self.label[index]
        targets = torch.tensor(targets, dtype = torch.long)
#         sample = {'image': image, 'labels': targets}

        return images, targets
```

D. Biến đổi và đưa dữ liệu vào mô hình huấn luyện theo từng batch

```
transform = transforms.Compose([
```

```
    transforms.Grayscale(3),
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])

train_dataset = ImageDataset(train_data, transform)
val_dataset = ImageDataset(val_data, transform)
test_dataset = ImageDataset(test_data, transform)
```

```
train_dataloader = DataLoader(train_dataset, batch_size = BATCH_SIZE, shuffle = True)
val_dataloader = DataLoader(val_dataset, batch_size = BATCH_SIZE, shuffle = True)
test_dataloader = DataLoader(test_dataset, batch_size = BATCH_SIZE, shuffle = True)
```

B3. Định nghĩa hàm train model

```
def training_loop(model, optimizer, loss_list, acc_list):
    model.to(device)
    #List to store loss to visualize
    lossli = loss_list
    accli = acc_list

    y_true = []
    y_pred = []

    valid_loss_min = np.Inf # track change in validation loss
    count = 0
    patience = 8 # nếu val_loss tăng 15 lần thì ngừng
    for epoch in range(num_epochs):

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

        train_acc = 0.0
        valid_acc = 0.0

        ###################
        # train the model #
        ###################

        model.train()
```

```
        for data, label in tqdm(train_dataloader):
            data = data.to(device)
            label = label.to(device)
            optimizer.zero_grad()
            output = model(data)

            loss = criterion(output, label)
            loss.backward()
            optimizer.step()


            train_loss += loss.item()*data.size(0)

            _, pred = torch.max(output, 1)

            train_acc += pred.eq(label).sum().item()

#         scheduler.step() ############

        ######################
        # validate the model #
        ######################

        model.eval()
        with torch.no_grad():
            for data, label in tqdm(val_dataloader):
                data = data.to(device)
                label = label.to(device)
                output = model(data)
                loss = criterion(output, label)
                valid_loss += loss.item()*data.size(0)

                # Calculate accuracy
                _, pred = torch.max(output, 1)
#                 y_true += target.tolist()
#                 y_pred += pred.tolist()

                valid_acc +=  pred.eq(label).sum().item()

        # calculate average losses
        train_loss = train_loss/len(train_dataloader.dataset)
        valid_loss = valid_loss/len(val_dataloader.dataset)
```

```
        lossli.append({'epoch':epoch,'train_loss':
train_loss,'valid_loss':valid_loss})

        train_acc = train_acc*100/len(train_dataloader.dataset)
        valid_acc = valid_acc*100/len(val_dataloader.dataset)
        accli.append({'epoch':epoch,'train_acc': train_acc,'valid_acc':valid_acc})

        ####################
        # Early stopping #
        ##################

        # print training/validation statistics
            print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f} \n
\tTraining Acc: {:.6f} \tValidation Acc: {:.6f}'.format(
            epoch, train_loss, valid_loss, train_acc, valid_acc))
        # save model if validation loss has decreased

        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'train_acc': accli,
            'loss_list': lossli,
            'loss': loss
            }, CHECKPOINT_PATH)

        if valid_loss <= valid_loss_min:
                print('Validation loss decreased ({:.6f} --> {:.6f}). Saving model
...'.format(
                valid_loss_min,
                valid_loss))

            count = 0
            print('count = ',count)
            torch.save(model, '../FTVGG19bn/FTVGG19bn_model.pt') #save model

            valid_loss_min = valid_loss
        else:
            count += 1
            print('count = ',count)
            if count >= patience:
                print('Early stopping!')
```

```
            return lossli, accli

    return lossli, accli
```

B4. Đặt thuộc tính *.requires_grad* của các tham số trong model

- Mặc định khi load 1 mô hình được huấn luyện trước thì tất cả các tham số có thuộc tính .requires_grad = True, phù hợp với trường hợp huấn luyện từ đầu hoặc tinh chỉnh mô hình.
- Nếu trích xuất đặt trưng (feature extracting) thì chỉ tính lại gradient cho tầng được khởi tạo mới (tầng cuối) do đó không cần cập nhật lại gradient. Vì vậy cần xây dựng hàm để đặt tất cả các thuộc tính .requires_grad = False như sau:

```
def set_parameter_requires_grad (model, feature_extracting):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False
```

B5. Khởi tạo và reshape model

- Tất cả các model được huấn luyện trước trên Imagenet (1000 classes)

```
(classifier): Sequential(
    ...
    (6): Linear(in_features=4096, out_features=1000, bias=True)
 )
```

do đó cần reshape layer cuối theo số classes của bài toán phân loại cụ thể

```
model.classifier[6] = nn.Linear(4096,num_classes)
```

- Nếu trích xuất đặc trưng, chỉ cập nhật lại layer cuối được reshape (mặc định), không cần tính lại gradients của nhưng tham số không thay đổi ở các layers trước. (dùng hàm set_parameter_requires_grad được định nghĩa ở bước trên để đặt .requires_grad = False).

Chú ý đây là bước khác nhau cơ bản giữa tinh chỉnh và trích xuất đặc trưng

```
def initialize_model(num_classes, feature_extract, use_pretrained = True):
    model_ft = models.vgg19_bn(pretrained = use_pretrained)
    set_parameter_requires_grad(model_ft, feature_extract)
    num_ftrs = model_ft.classifier[6].in_features
    model_ft.classifier[6] = nn.Linear(num_ftrs, num_classes)
    input_size = 224

    return model_ft, input_size

# Initialize the model for this run
model_ft, input_size = initialize_model(num_classes, feature_extract,
use_pretrained = True)

print (model_ft)
```

B6. Sử dụng hàm tối ưu

Hàm tối ưu chỉ cập nhật các tham số mong muốn.
Nếu trích xuất đặc trưng thì chỉ cập nhật ở layer cuối (layer được khởi tạo lại với đầu ra phù hợp bài toán phân loại), các layers trước đó đã đặt .requires_grad = False ở hàm set_parameter_requires_grad.
Nếu tinh chỉnh hoặc huấn luyện từ đầu thì tất cả các tham số có .requires_grad = True (mặc định) sẽ được tối ưu.

```
# Send the model to GPU
model_ft = model_ft.to(device)

# Gather the parameters to be optimized/updated in this run. If we are
#  finetuning we will be updating all parameters. However, if we are
#  doing feature extract method, we will only update the parameters
#  that we have just initialized, i.e. the parameters with requires_grad
#  is True.
params_to_update = model_ft.parameters()
print("Params to learn:")
if feature_extract:
   params_to_update = []
   for name,param in model_ft.named_parameters():
       if param.requires_grad == True:
           params_to_update.append(param)
           print("\t",name)
else:
   for name,param in model_ft.named_parameters():
       if param.requires_grad == True:
           print("\t",name)

# Observe that all parameters are being optimized
optimizer = Adam(params_to_update ,lr = 0.001, weight_decay=1e-5)
```

B7. Chạy các bước huấn luyện và kiểm định mô hình

```
loss_list, acc_list = [],[]

since = time.time()

loss, acc = training_loop(
   model = model_ft,
   optimizer = optimizer,
   loss_list = loss_list,
   acc_list = acc_list
)
time_elapsed = time.time() - since
```

```
print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
```

B8. So sánh với mô hình huấn luyện từ đầu

Sử dụng kiến trúc của mô hình mà không dùng chuyển giao học tập. So sánh với thời gian và độ chính xác tổng quát

```
# Initialize the non-pretrained version of the model used for this run
scratch_model,_ = initialize_model(num_classes, feature_extract=False, use_pretrained=False)
scratch_model = scratch_model.to(device)
scratch_optimizer = optim.SGD(scratch_model.parameters(), lr=0.001, momentum=0.9)
scratch_criterion = nn.CrossEntropyLoss()
_,scratch_hist = train_model(scratch_model, dataloaders_dict, scratch_criterion, scratch_optimizer, num_epochs=num_epochs, is_inception=(model_name=="inception"))

# Plot the training curves of validation accuracy vs. number
#  of training epochs for the transfer learning method and
#  the model trained from scratch
ohist = []
shist = []

ohist = [h.cpu().numpy() for h in hist]
shist = [h.cpu().numpy() for h in scratch_hist]

plt.title("Validation Accuracy vs. Number of Training Epochs")
plt.xlabel("Training Epochs")
plt.ylabel("Validation Accuracy")
plt.plot(range(1,num_epochs+1),ohist,label="Pretrained")
plt.plot(range(1,num_epochs+1),shist,label="Scratch")
plt.ylim((0,1.))
plt.xticks(np.arange(1, num_epochs+1, 1.0))
plt.legend()
plt.show()
```

TỔNG KẾT

Huấn luyện từ đầu:

`initialize_model(num_classes, feature_extract=False, use_pretrained=False)`

Chuyển giao học tập:

- Trích xuất đặc trưng:

`initialize_model(num_classes, feature_extract=True, use_pretrained=True)`

- Tinh chỉnh:

`initialize_model(num_classes, feature_extract=False, use_pretrained=True)`

link tham khảo:

https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html#create-the-optimizer

KẾT QUẢ THỰC NGHIỆM

NETWORK	Precision	Recall	F1-Score	Accuracy	Specificity
VGG19bn	0.91	0.90	0.90	0.90	
FT_VGG19bn	0.90	0.90	0.90	0.90	
FT_VGG19bn (classifier)	0.99	0.99	0.99	0.9925	
FE_VGG19bn	0.96	0.95	09.5	0.953	
VGG16	0.97	0.97	0.96	0.965	
FT_VGG16 (Lớp cuối)	0.96	0.96	0.96	0.961	
FT_VGG16 (classifier)	0.94	0.94	0.94	0.943	
FT_VGG16 (paper)	0.97	0.97	0.97	0.966	
FE_VGG16	0.95	0.94	0.94	0.944	