Лабораторная работа №5

Transfer Learning

Прудникова Анастасия Алексеевна М8О-408Б-20

Для решения задачи был выбран датасет с рентгеновскими снимками.

Ввод [1]:

```
# License: BSD
# Author: Sasank Chilamkurthy

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torch.backends.cudnn as cudnn
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
from PIL import Image
from tempfile import TemporaryDirectory

cudnn.benchmark = True
plt.ion()
```

```
WARNING:tensorflow:From C:\Users\nprud\anaconda3\Lib\site-packages\keras
\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is d
eprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy in
stead.

WARNING:tensorflow:From C:\Users\nprud\anaconda3\Lib\site-packages\keras
\src\backend.py:873: The name tf.get_default_graph is deprecated. Please
use tf.compat.v1.get_default_graph instead.
```

Out[1]:

```
<contextlib.ExitStack at 0x1ac1eb6db50>
```

Ввод [2]:

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

data_dir = 'dataset/archive/images'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size
                                             shuffle=True, num_workers=4)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Ввод [3]:

```
dataset_sizes
```

Out[3]:

```
{'train': 1022, 'val': 1139}
```

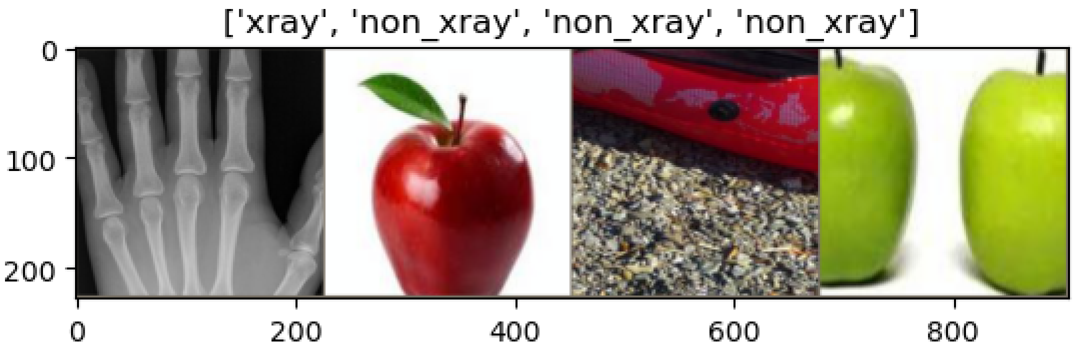
Ввод [4]:

```
class_names
```

Out[4]:

```
['non_xray', 'xray']
```

Ввод [5]:
```
def imshow(inp, title=None):
    """Display image for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)



inputs, classes = next(iter(dataloaders['train']))

out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in classes])
```



Ввод [6]:
```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    with TemporaryDirectory() as tempdir:
        best_model_params_path = os.path.join(tempdir, 'best_model_params.p

        torch.save(model.state_dict(), best_model_params_path)
        best_acc = 0.0

        for epoch in range(num_epochs):
            print(f'Epoch {epoch}/{num_epochs - 1}')
            print('-' * 10)

            for phase in ['train', 'val']:
                if phase == 'train':
                    model.train()
                else:
                    model.eval()

                running_loss = 0.0
                running_corrects = 0

                for inputs, labels in dataloaders[phase]:
                    inputs = inputs.to(device)
                    labels = labels.to(device)

                    optimizer.zero_grad()

                    with torch.set_grad_enabled(phase == 'train'):
                        outputs = model(inputs)
                        _, preds = torch.max(outputs, 1)
                        loss = criterion(outputs, labels)

                        if phase == 'train':
                            loss.backward()
                            optimizer.step()

                    running_loss += loss.item() * inputs.size(0)
                    running_corrects += torch.sum(preds == labels.data)
                if phase == 'train':
                    scheduler.step()

                epoch_loss = running_loss / dataset_sizes[phase]
                epoch_acc = running_corrects.double() / dataset_sizes[phase

                print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}

                if phase == 'val' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    torch.save(model.state_dict(), best_model_params_path)

            print()

        time_elapsed = time.time() - since
        print(f'Training complete in {time_elapsed // 60:.0f}m {time_elapse
        print(f'Best val Acc: {best_acc:4f}')

        model.load_state_dict(torch.load(best_model_params_path))
```

```
    return model
```

Ввод [7]:

```
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloaders['val']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title(f'predicted: {class_names[preds[j]]}')
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return
        model.train(mode=was_training)
```

Ввод [8]:

```
model_ft = models.resnet18(weights='IMAGENET1K_V1')
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1
```

Ввод [9]:
```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=4)
```

```
Epoch 0/3
----------
train Loss: 0.2315 Acc: 0.9119
val Loss: 0.0041 Acc: 0.9991

Epoch 1/3
----------
train Loss: 0.2475 Acc: 0.9100
val Loss: 0.0018 Acc: 1.0000

Epoch 2/3
----------
train Loss: 0.2435 Acc: 0.9149
val Loss: 0.0012 Acc: 0.9991

Epoch 3/3
----------
train Loss: 0.2333 Acc: 0.9315
val Loss: 0.0303 Acc: 0.9939

Training complete in 24m 0s
Best val Acc: 1.000000
```

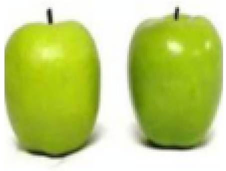
Ввод [10]:
```
model_conv = torchvision.models.resnet18(weights='IMAGENET1K_V1')
for param in model_conv.parameters():
    param.requires_grad = False

num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=
```

Ввод [11]:
```
model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=4)
```

```
Epoch 0/3
----------
train Loss: 0.2885 Acc: 0.8669
val Loss: 0.0228 Acc: 0.9956

Epoch 1/3
----------
train Loss: 0.2719 Acc: 0.8914
val Loss: 0.0182 Acc: 0.9947

Epoch 2/3
----------
train Loss: 0.2981 Acc: 0.8992
val Loss: 0.0008 Acc: 1.0000

Epoch 3/3
----------
train Loss: 0.3531 Acc: 0.8816
val Loss: 0.0032 Acc: 1.0000

Training complete in 14m 40s
Best val Acc: 1.000000
```

Ввод [12]:
```
visualize_model(model_conv)

plt.ioff()
plt.show()
```

predicted: non_xray



predicted: non_xray

predicted: non_xray

predicted: non_xray

predicted: xray

predicted: xray

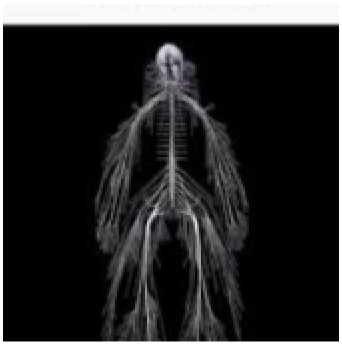
Ввод [13]:

```
def visualize_model_predictions(model,img_path):
    was_training = model.training
    model.eval()

    img = Image.open(img_path)
    img = data_transforms['val'](img)
    img = img.unsqueeze(0)
    img = img.to(device)

    with torch.no_grad():
        outputs = model(img)
        _, preds = torch.max(outputs, 1)

        ax = plt.subplot(2,2,1)
        ax.axis('off')
        ax.set_title(f'Predicted: {class_names[preds[0]]}')
        imshow(img.cpu().data[0])

        model.train(mode=was_training)
```

Ввод [14]:

```
visualize_model_predictions(
    model_conv,
    img_path='example.jpg'
)

plt.ioff()
plt.show()
```

Predicted: non_xray



Ввод [17]:

```
visualize_model_predictions(
    model_conv,
    img_path='example2.jpg'
)

plt.ioff()
plt.show()
```

Predicted: xray

Как видно по картинкам, несмотря на попытки запутать нейросеть, изображения были распознаны с поражающей точностью.

Вывод

В ходе выполнения лабораторной работы я произвела тонкую настройку (fine-tuning) предварительно обученной нейронной сети с целью развить способность распознавать, представляет ли анализируемое изображение рентгеновский снимок или нет. Данная задача относится к области передачи обучения (transfer learning) в машинном обучении.

Ввод [ ]:

```
```