Решение задачи "Классификация кошек" в рамках хакатона 24-25 апреля 2021 Американского центра

В этом ноутбуке будем решать задачу классификации фотографий кошек с помощью предобученной нейронной сети ResNet-18 для последующего использования модели в Телеграмм боте.

```
# подключаем гугл диск на котором данные
from google.colab import drive
drive.mount ('/content/gdrive', force_remount = True)
```

```
     Mounted at /content/gdrive
```

```
!ls /content/gdrive/'My Drive'/dataset_cats_small2/test
```

```
     00000011_007_ginger.jpg          00000013_025_ginder.jpg
     00000011_014_grey_and_green.jpg  00000014_027_grey.jpg
     00000011_017_black.jpg           00000015_016_grey.jpg
     00000013_005_grey_and_green.jpg  00000015_019_grey.jpg
     00000013_021_grey_and_green.jpg  00000015_020_ginger.jpg
```

Строим нейросеть

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F
import matplotlib.pyplot as plt
import numpy as np
import torchvision.models as models
```

Создаем даталоадеры для обучения нейросети:

```
images_dataset = torchvision.datasets.ImageFolder("../content/gdrive/My Drive/dataset_cats
  transforms.ToTensor(),
  # нормализуем как в ImageNet
  torchvision.transforms.Normalize([0.485, 0.456, 0.406],
                                   [0.229, 0.224, 0.225]),

]))

images_dataloader = torch.utils.data.DataLoader(images_dataset, batch_size=1,
                                          shuffle=True)
```

Будем использовать предобученную на ImageNet сеть ResNet18:

```
net = models.resnet18(True, True).cuda()
net
```

```
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
      (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_
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (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_
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
      (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_
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
```

```
    (fc): Linear(in_features=512, out_features=1000, bias=True)
  )
```

Заменим последний полносвязный слой сети на слой, который будет выдавать 5 значения на выходе (т.к. у нас 5 классов):

```
net.fc = nn.Linear(512, 5)
```

Заморозим все слом нейросети, кроме самого последнего, только что добавленного fc-слоя. Будем обучать только последний слой сети.

```
for i, child in enumerate(net.children()):
    if i == 9:
        break
    for param in child.parameters():
        param.requires_grad = False
```

Объявляем лосс-функцию и оптимизатор:

```
# стандартная лосс-функция для задачи классификации
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.00001, momentum=0.95)
```

```
# для обучения на GPU
device = 'cuda:0'
net.to(device)
```

```
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
          (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_
          )
        )
        (1): BasicBlock(
          (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
        )
      )
      (layer3): Sequential(
        (0): BasicBlock(
          (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
          (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_
          )
        )
        (1): BasicBlock(
          (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
        )
      )
      (layer4): Sequential(
        (0): BasicBlock(
          (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
          (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
          (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_
          )
        )
        (1): BasicBlock(
          (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
          (relu): ReLU(inplace=True)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
          (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_
        )
      )
      (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
      (fc): Linear(in_features=512, out_features=5, bias=True)
    )
```

Обучаем сеть 5 эпох:

```
n_epochs = 5
print_every = 5

total_step = len(images_dataloader)

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

    print(f'Epoch {epoch}\n')
    for batch_idx, (data, target) in enumerate(images_dataloader):
        # кладем данные на GPU
        data, target = data.to(device), target.to(device)

        # делаем шаг обучения сети
        optimizer.zero_grad()
        outputs = net(data)
        loss = criterion(outputs, target)
        loss.backward()
        optimizer.step()
        if (batch_idx) % 20 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch, n_epochs, batch_idx, total_step, loss.item()))
```

```
Epoch 1

Epoch [1/5], Step [0/90], Loss: 0.9546
Epoch [1/5], Step [20/90], Loss: 1.8599
Epoch [1/5], Step [40/90], Loss: 1.7213
Epoch [1/5], Step [60/90], Loss: 1.7897
Epoch [1/5], Step [80/90], Loss: 1.4428
Epoch 2

Epoch [2/5], Step [0/90], Loss: 1.2752
Epoch [2/5], Step [20/90], Loss: 1.1970
Epoch [2/5], Step [40/90], Loss: 1.9494
Epoch [2/5], Step [60/90], Loss: 1.3462
Epoch [2/5], Step [80/90], Loss: 1.8911
Epoch 3

Epoch [3/5], Step [0/90], Loss: 2.1333
Epoch [3/5], Step [20/90], Loss: 1.9808
Epoch [3/5], Step [40/90], Loss: 1.3560
Epoch [3/5], Step [60/90], Loss: 2.2480
Epoch [3/5], Step [80/90], Loss: 1.9629
Epoch 4

Epoch [4/5], Step [0/90], Loss: 1.2609
Epoch [4/5], Step [20/90], Loss: 1.7421
Epoch [4/5], Step [40/90], Loss: 2.0927
Epoch [4/5], Step [60/90], Loss: 1.9611
Epoch [4/5], Step [80/90], Loss: 1.3790
Epoch 5

Epoch [5/5], Step [0/90], Loss: 1.1557
Epoch [5/5], Step [20/90], Loss: 0.9951
Epoch [5/5], Step [40/90], Loss: 1.0392
Epoch [5/5], Step [60/90], Loss: 2.3141
Epoch [5/5], Step [80/90], Loss: 1.8869
```

Тестируем обученную нейросеть на тестовом наборе картинок:

```
images_testset = torchvision.datasets.ImageFolder("../content/gdrive/My Drive/dataset_cats
  transforms.ToTensor(),
  torchvision.transforms.Normalize([0.485, 0.456, 0.406],
                                   [0.229, 0.224, 0.225]),

]))

images_testloader = torch.utils.data.DataLoader(images_testset, batch_size=1,
                                          shuffle=False)


batch_loss = 0
total=0
correct=0
```

```
with torch.no_grad():
        net.eval()

        for data, target in (images_testloader):
            # кладем данные на GPU
            data, target = data.to(device), target.to(device)
            outputs = net(data)
            # считаем loss
            loss = criterion(outputs, target)
            batch_loss += loss.item()

            # считаем accuracy
            _, pred = torch.max(outputs, dim=1)
            correct += torch.sum(pred==target).item()
            total += target.size(0)

        print("Acc", 100 * correct/total)
        print("Loss", batch_loss/len(images_testloader))
```

```
Acc 23.333333333333332
Loss 1.5339624755912358
```

сохраняем модель, чтобы использовать ее в Телеграмм боте.

```
torch.save (net, 'cats_clf02.h5')
```

```
from google.colab import files

files.download('cats_clf02.h5')
```

✓ 0 сек. выполнено в 18:39 ● ×