Домашнее задание: дообучение нейросети

Привет!

В этом домашнем задании вам предстоит самим дообучить сеть на новом датасете для классификации.

В последнем практическом занятии этого модуля мы решали задачу классификации кошек и собак с помощью предобученной на ImageNet сети. На самом деле, это не совсем честно, потому что среди классов ImageNet наверняка есть классы "кошка" и "собака", и сеть, обученная на ImageNet и без дообучения может классифицировать кошек и собак. Поэтому в домашнем задании мы возьмем датасет фотографий Intel Image Classification (https://www.kaggle.com/puneet6060/intel-image-classification) природы.

1. Загрузка датасета

Для загрузки датасета из интернета воспользуемся командами bash. Bash -- язык командной строки linux. В ячейках jupyter notebook можно запускать команды bash, предварительно написав ! в начале ячейки.

```
# команда wget скачивает файлы из интернета по ссылке
! wget -c
"https://storage.googleapis.com/kaggle-data-sets/111880%2F269359%2Fbun
dle%2Farchive.zip?GoogleAccessId=gcp-kaggle-com@kaggle-
161607.iam.gserviceaccount.com&Expires=1599859461&Signature=M3%2F
%2FZh4ZO0UGqPgY88LNgkNJJ%2BpTtdVIHN9sB%2FuvNJBHQMFz5OhWJUwtuSDc%2Fr1Q
%2FD%2F3gD2wUFBA8vgxpumu75Rh7EdViQzWJU3Iybet5%2FfFVhzg1onuCONRljsrXAZG
%2BGxb
%2B4rDdOP2eZ1OFCVNqVRNO0ASQhwjsQFLTW8sp9gvd8dcVCO0iNFOnZNJuiGRp0IU
%2FI3BVf%2FA3zLe1ECHkwjG5%2Ba6Zo4lBRbhYsr5LxtlSH
%2BDJcPTZkfWa3vqUCvjM99zK4ESPdNuC4X72nbONb1crKcWS
```

```
%2BlJRfBNKUXECt1dFDU9SFTS0e1iI%2BPAFCzoG4WH4rwvsA0IsN%2BIGg5R9Q%3D%3D"
-O dataset.zip
# команда unzip разархивирует zip-архив
! unzip dataset.zip

# команда ls выводит список файлов в текущей директории
! ls

# команда ls ИМЯ_ПАПКИ выводит список файлов в указанной папке
! ls seg_train

! ls seg_test/seg_test

import numpy as np
from tqdm import tqdm_notebook
import matplotlib.pyplot as plt
%matplotlib inline

# модули библиотеки PyTorch
import torch
from torchvision import datasets, transforms
# метрика качества
from sklearn.metrics import accuracy_score
```

Задание 1 (0 баллов)

Как обычно, начнем с загрузки даатсета в пайторч с помощью ImageFolder.

Для начала объявим трансформации.

Объявите трансформации для тренировочного и тестового датасета:

- перевод картинки в тензор
- нормализация с mean=[0.485, 0.456, 0.406] и std=[0.229, 0.224, 0.225]

Resize делать не надо! Все картинки этого датасета уже приведены к одному форматц 150*150

```
from torchvision import transforms

# Трансформации для тренировочного датасета
transform_train = transforms.Compose([
    transforms.ToTensor(),  # Перевод картинки в тензор
    transforms.Normalize(mean=[0.485, 0.456, 0.406],  # Нормализация с
mean и std
                         std=[0.229, 0.224, 0.225])
])

# Трансформации для валидационного датасета
transform_val = transforms.Compose([
    transforms.ToTensor(),  # Перевод картинки в тензор
```

```
    transforms.Normalize(mean=[0.485, 0.456, 0.406],  # Нормализация с
mean и std
                         std=[0.229, 0.224, 0.225])
])
```

И теперь объявим датасеты:

```
train_data = datasets.ImageFolder("seg_train",
transform=transform_train)
test_data = datasets.ImageFolder("seg_test", transform=transform_val)

train_data
```

Объявите даталоадеры. Помните, что для тренировочного даталоадера важно перемешивать данные, для тестового -- неважно. Батч сайз возьмите проивзольный. Но не берите слишком маленький, будете долго ждать обучения сети.

```
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder

# Определите произвольный размер батча
batch_size = 32

# Путь к директориям с данными
train_data_path = 'path_to_train_data'
test_data_path = 'path_to_test_data'

# Загрузка тренировочного и тестового датасетов с использованием
ImageFolder
train_dataset = ImageFolder(root=train_data_path,
transform=transform_train)
test_dataset = ImageFolder(root=test_data_path,
transform=transform_val)

# Даталоадеры
train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)
```

Отлично! Теперь давайте посмотрим на примеры картинок из датасета, и заодно проверим, что вы объявили даталоадеры верно.

```
dataiter = iter(train_loader)
# батч картинок и батч ответов к картинкам
images, labels = dataiter.next()

images.shape, labels.shape
```

```
def show_imgs(imgs, labels):
    f, axes= plt.subplots(1, 10, figsize=(30,5))
    for i, axis in enumerate(axes):
      axes[i].imshow(np.squeeze(np.transpose(imgs[i].numpy(), (1, 2,
0))), cmap='gray')
      axes[i].set_title(labels[i].numpy())
    plt.show()

show_imgs(images, labels)
```

Импортируем нужные модули для обучения сети:

```
# модуль, где определены слои для нейронных сетей
import torch.nn as nn
# модуль, где определены активайии для слоев нейронных сетей
import torch.nn.functional as F
```

Задание 2. Обучение сети с нуля. (4 балла код + 1 балл вывод)

Построим сверточную нейронную сеть, обучим ее и посчитаем метрику accuracy на тестовой выборке.

Так как в этом датасете картинки разрешения 150*150 -- больше, чем разрешение картинок из датасета с практического занятия -- то давайте построим сеть с тремя сверточными слоями и двумя макспуллингами.

```
# класс для удобного перевода картинки из двумерного объекта в вектор
class Flatten(nn.Module):
    def forward(self, input):
        return input.view(input.size(0), -1)

class ConvNet(nn.Module):
    def __init__(self):
        super().__init__()
        # объявите слои: Conv-MaxPool-Conv-MaxPool-Conv.
        # Первый conv слой с 5 ядрами, второй -- с 4, третий -- с 3.
        # MaxPool оба с параметрами (2, 2)

        # YOUR CODE

        # после этого объявите два полносвязных слоя: первый с 256
нейронами,
        # второй -- с 6 (выходной слой, 6 -- количество классов в
датасете)
        # ВАЖНО! вам предстоит узнать, какое количество нейронов будет
в первом
        # полносвязном слое после растягивания карт активации в вектор
```

```
(Linear(?, 256))
        # проще всего это узнать, запустив обучение сети и посмотрев
на текст ошибки:
        # в нем будет указано, сколько нейронов ожидается в линейном
слое.
        # P.S. не забудтье Flatten!

        # YOUR CODE

    def forward(self, x):
        # forward pass сети
        # напишите forward pass сети, используйте relu
        # в качестве промежудточных активаций и softmax
        # в качестве активации последнего слоя

        return x
```

Тут, как обычно, функция обучения сети:

```
def train(net, n_epoch=5):
  # выбираем функцию потерь
  loss_fn = torch.nn.CrossEntropyLoss()

  # выбираем алгоритм оптимизации и learning_rate
  learning_rate = 1e-3
  optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)

  # обучаем сеть 5 эпохи
  for epoch in tqdm_notebook(range(n_epoch)):

      running_loss = 0.0
      train_dataiter = iter(train_loader)
      for i, batch in enumerate(tqdm_notebook(train_dataiter)):
          # так получаем текущий батч
          X_batch, y_batch = batch

          # обнуляем веса
          optimizer.zero_grad()

          # forward pass (получение ответов на батч картинок)
          y_pred = net(X_batch)
          # вычисление лосса от выданных сетью ответов и правильных
ответов на батч
          loss = loss_fn(y_pred, y_batch)
          # bsckpropagation (вычисление градиентов)
          loss.backward()
          # обновление весов сети
          optimizer.step()

          # выведем текущий loss
```

```
            running_loss += loss.item()
            # выведем качество каждые 500 батчей
            if i % 10 == 9:
                print('[%d, %5d] loss: %.3f, acc: %3f' %
                      (epoch + 1, i + 1, running_loss / 500,
accuracy_score(y_batch.numpy(), np.argmax(y_pred.detach().numpy(),
axis=1))))
                running_loss = 0.0

    print('Обучение закончено')
    return net

# объявляем сеть
net = ConvNet()
# теперь обучим сеть. Выберите нужное уоличество эпох для обучения.
# Постарайтесь подобрать его так, чтобы сеть не переобучилась, но и не
недообучилась.
# можно поставить побольше эпох и остановить обучение, когда покажется
нужным
net = train(net, epoch=#YOUR CODE)
```

Посчитаем accuracy на test:

```
test_dataiter = iter(test_loader)
images, labels = test_dataiter.next()

accuracy_score(labels.numpy(),
np.argmax(net.forward(images).detach().numpy(), axis=1))
```

Задание: Вывод:

Проанализирейте: переобучилась ли ваша сеть? Достаточно ли хорошо она предсказывает картинки на тесте?

Тут напишите ваш вывод

Задание 3. Дообучение AlexNet (2+2 балла код + 1 балл вывод)

Теперь давайте попробуем дообучить сеть, предобученную на ImageNet, на нашем датасете.

На практическом занятии мы дообучали Vgg16, теперь даватйе возьмем другую сеть -- AlexNet.

Список сетей, предобученных на ImageNet, которые есть в библиотеке PyTorch: https://pytorch.org/docs/stable/torchvision/models.html

```
from torchvision import models
AlexNet = models.alexnet(pretrained=True)

AlexNet.parameters
```

Задание 3.1:

Напишите класс New_AlexNet на основе AlexNet, у которой заменили последний слой. Заморозьте все слои, кроме слоев классификатора (всех линейных слоев)

```
import torch
import torch.nn as nn
from torchvision import models

class New_AlexNet(nn.Module):
    def __init__(self, num_classes):
        super(New_AlexNet, self).__init__()

        # Загрузка предобученной модели AlexNet
        self.alexnet = models.alexnet(pretrained=True)

        # Замораживаем все параметры
        for param in self.alexnet.parameters():
            param.requires_grad = False

        # Заменяем последний слой классификатора
        self.alexnet.classifier[6] = nn.Linear(in_features=4096,
out_features=num_classes)

    def forward(self, x):
        # Пропускаем данные через модель
        x = self.alexnet(x)
        return x

net = New_AlexNet()
train(net, epoch=# YOUR CODE)
```

И посмотрим на скор на тесте:

```
test_dataiter = iter(test_loader)
images, labels = test_dataiter.next()

accuracy_score(labels.numpy(),
np.argmax(net.forward(images).detach().numpy(), axis=1))
```

Задание 3.2:

Напишите класс New_AlexNet на основе AlexNet, у которой заменили последний слой(как в задании выше) Заморозьте все слои, кроме **двух последних** слоев классификатора

```
import torch
import torch.nn as nn
from torchvision import models

class New_AlexNet_v2(nn.Module):
    def __init__(self, num_classes):
        super(New_AlexNet_v2, self).__init__()

        # Загрузка предобученной модели AlexNet
        self.alexnet = models.alexnet(pretrained=True)

        # Замораживаем все параметры
        for param in self.alexnet.parameters():
            param.requires_grad = False

        # Размерозим два последних слоя классификатора
        for param in self.alexnet.classifier[5:].parameters():
            param.requires_grad = True

        # Заменяем последний слой классификатора
        self.alexnet.classifier[6] = nn.Linear(in_features=4096,
out_features=num_classes)

    def forward(self, x):
        # Пропускаем данные через модель
        x = self.alexnet(x)
        return x

net = New_VGG16_1()
train(net)
```

И посмотрим на скор на тесте:

```
test_dataiter = iter(test_loader)
images, labels = test_dataiter.next()

accuracy_score(labels.numpy(),
np.argmax(net.forward(images).detach().numpy(), axis=1))
```

Задание 3.3: Вывод:

Какие результаты у вас получились? Классифицирует ли картинки природы лучше сеть, предобученная на ImageNet, или ваша сеть из задания два получилась лучше? Есть ли разница между дообучением двух последних слоев сети или всех линейных слоев сети?

Тут напишите ваш вывод