

▼ Практическое задание №1

Установка необходимых пакетов:

```
!pip install -q tqdm
!pip install --upgrade --no-cache-dir gdown

Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.6.6)
Collecting gdown
  Downloading gdown-4.7.1-py3-none-any.whl (15 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from gdown) (1.16.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.7.22)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.6.6
    Uninstalling gdown-4.6.6:
      Successfully uninstalled gdown-4.6.6
  Successfully installed gdown-4.7.1
```

Монтирование Вашего Google Drive к текущему окружению:

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

Mounted at /content/drive
```

Константы, которые пригодятся в коде далее, и ссылки (gdrive идентификаторы) на предоставляемые наборы данных:

```
EVALUATE_ONLY = False
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '1EkQL97xSiNgjYOE3rHuTfOX9Vd2841P1',
    'train_small': '1yFYg7RuOGofOJs2exrOx1syZSteDPb8F',
    'train_tiny': '1cXyZyfhIn8Xq5jMR_FW9dmkuqoteJnzX',
    'test': '1pCiol3n3Tj4jkPW-Sf6muqhGxkCE2L1K',
    'test_small': '1mmfoAtfIhcjhGESUGhBHpCnL6HSVRSSP',
    'test_tiny': '18e8gzRxqBLGHoyTwRwO3SCf_kAITLE07'
}
```

Импорт необходимых зависимостей:

```
from pathlib import Path
import numpy as np
from typing import List
from tqdm.notebook import tqdm
from time import sleep
from PIL import Image
import IPython.display
from sklearn.metrics import balanced_accuracy_score
import gdown
from sklearn.preprocessing import LabelEncoder
from torch.utils.data import Dataset, DataLoader
```

▼ Класс Dataset

Предназначен для работы с наборами данных, обеспечивает чтение изображений и соответствующих меток, а также формирование пакетов (батчей).

```

class ImgDataset(Dataset):

    def __init__(self, name):
        self.name = name
        self.is_loaded = False
        url = f"https://drive.google.com/uc?id={DATASETS_LINKS[name]}"
        # url = f"{DATASETS_LINKS[name]}"
        output = f'{name}.npz'
        gdown.download(url, output, quiet=False)
        print(f'Loading dataset {self.name} from npz.')
        np_obj = np.load(f'{name}.npz')
        self.images = np_obj['data']
        self.labels = np_obj['labels']
        self.n_files = self.images.shape[0]
        self.is_loaded = True
        self.len_ = len(self.labels)
        print(f'Done. Dataset {name} consists of {self.n_files} images.')

    def image(self, i):
        # read i-th image in dataset and return it as numpy array
        if self.is_loaded:
            return self.images[i, :, :, :]

    def images_seq(self, n=None):
        # sequential access to images inside dataset (is needed for testing)
        for i in range(self.n_files if not n else n):
            yield self.image(i)

    def random_image_with_label(self):
        # get random image with label from dataset
        i = np.random.randint(self.n_files)
        return self.image(i), self.labels[i]

    def random_batch_with_labels(self, n):
        # create random batch of images with labels (is needed for training)
        indices = np.random.choice(self.n_files, n)
        imgs = []
        for i in indices:
            img = self.image(i)
            imgs.append(self.image(i))
        logits = np.array([self.labels[i] for i in indices])
        return np.stack(imgs), logits

    def image_with_label(self, i: int):
        # return i-th image with label from dataset
        return self.image(i), self.labels[i]

    def __len__(self):
        return self.len_

    def __getitem__(self, index):
        return self.images[index], self.labels[index]

```

▼ Пример использования класса Dataset

Загрузим обучающий набор данных, получим произвольное изображение с меткой. После чего визуализируем изображение, выведем метку. В будущем, этот кусок кода можно закомментировать или убрать.

```

d_train_tiny = ImgDataset('train_tiny')

img, lbl = d_train_tiny.random_image_with_label()
print()
print(f'Got numpy array of shape {img.shape}, and label with code {lbl}.')
print(f'Label code corresponds to {TISSUE_CLASSES[lbl]} class.')

pil_img = Image.fromarray(img)
IPython.display.display(pil_img)

```

Downloading...

From (uriginal): https://drive.google.com/uc?id=1cXyZyfHin8Xg5jMR_FW9dmkuqoteJnzX

From (redirected): https://drive.google.com/uc?id=1cXyZyfHin8Xg5jMR_FW9dmkuqoteJnzX&confirm=t&uuiid=5d4bab40-

To: /content/train_tiny.npz

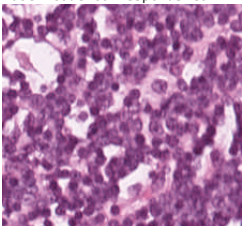
100%[██████████] 105M/105M [00:01<00:00, 94.9MB/s]

Loading dataset train_tiny from npz.

Done. Dataset train_tiny consists of 900 images.

Got numpy array of shape (224, 224, 3), and label with code 3.

Label code corresponds to LYM class.



len(d_train_tiny)

900

▼ Класс Metrics

Реализует метрики точности, используемые для оценивания модели:

1. точность,

2. сбалансированную точность.

```
class Metrics:

    @staticmethod
    def accuracy(gt: List[int], pred: List[int]):
        assert len(gt) == len(pred), 'gt and prediction should be of equal length'
        return sum(int(i[0] == i[1]) for i in zip(gt, pred)) / len(gt)

    @staticmethod
    def accuracy_balanced(gt: List[int], pred: List[int]):
        return balanced_accuracy_score(gt, pred)

    @staticmethod
    def print_all(gt: List[int], pred: List[int], info: str):
        print(f'metrics for {info}:')
        print('\t accuracy {:.4f}:'.format(Metrics.accuracy(gt, pred)))
        print('\t balanced accuracy {:.4f}:'.format(Metrics.accuracy_balanced(gt, pred)))
```

▼ Класс Model

Класс, хранящий в себе всю информацию о модели.

Вам необходимо реализовать методы `save`, `load` для сохранения и загрузки модели. Особенно актуально это будет во время тестирования на дополнительных наборах данных.

Пожалуйста, убедитесь, что сохранение и загрузка модели работает корректно. Для этого обучите модель, протестируйте, сохраните ее в файл, перезапустите среду выполнения, загрузите обученную модель из файла, вновь протестируйте ее на тестовой выборке и убедитесь в том, что получаемые метрики совпадают с полученными для тестовой выборки ранее.

Также, Вы можете реализовать дополнительные функции, такие как:

1. валидацию модели на части обучающей выборки;
2. использование кроссвалидации;
3. автоматическое сохранение модели при обучении;
4. загрузку модели с какой-то конкретной итерации обучения (если используется итеративное обучение);
5. вывод различных показателей в процессе обучения (например, значение функции потерь на каждой эпохе);
6. построение графиков, визуализирующих процесс обучения (например, график зависимости функции потерь от номера эпохи обучения);
7. автоматическое тестирование на тестовом наборе/наборах данных после каждой эпохи обучения (при использовании итеративного обучения);
8. автоматический выбор гиперпараметров модели во время обучения;
9. сохранение и визуализацию результатов тестирования;
10. Использование аугментации и других способов синтетического расширения набора данных (дополнительным плюсом будет обоснование необходимости и обоснование выбора конкретных типов аугментации)
11. и т.д.

Полный список опций и дополнений приведен в презентации с описанием задания.

При реализации дополнительных функций допускается добавление параметров в существующие методы и добавление новых методов в класс модели.

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

DEVICE = torch.device("cuda")

class Model:

    def __init__(self):
        # todo
        self.batch_size = 16
        self.epochs = 15

        # LBL6
        self.model = models.alexnet(pretrained=True)
        '''for param in self.model.features[:-3].parameters():
            param.requires_grad = False'''

        # num_features -- это размерность вектора фич, поступающего на вход FC-слою
        self.num_features = 9216

        self.model.classifier = nn.Sequential(nn.Linear(self.num_features, 100), nn.Linear(100, 9))

        self.model.to(DEVICE)

    def save(self, name: str):
        # todo
        # example demonstrating saving the model to PROJECT_DIR folder on gdrive with name 'name'
        torch.save(self.model.state_dict(), f'/content/drive/MyDrive/{name}.pth')

    def load(self, name: str):
        # todo
        # example demonstrating loading the model with name 'name' from gdrive using link
        name_to_id_dict = {
            'best': '1-2h3QHip_3sqJZmCJ-_wauwURL38FWHc',
            'best_during_training': '1ZrJaKVxuGRZQ3-3IGSXPPileDNca20ID'
        }
        output = f'{name}.pth'
        gdown.download(f'https://drive.google.com/uc?id={name_to_id_dict[name]}', output, quiet=False)
        np_obj = torch.load(f'./{name}.pth')
        self.model.load_state_dict(np_obj)
```

```

self.model.to_device(device=self.device)

def train(self, train_dataset: ImgDataset, val_dataset: ImgDataset):
    # you can add some plots for better visualization,
    # you can add model autosaving during training,
    # etc.
    print(f'training started')
    train_loader = DataLoader(train_dataset, batch_size=self.batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=self.batch_size, shuffle=True)

    history = []
    max_val_acc = 0
    log_template = "\nEpoch {ep:03d} train_loss: {t_loss:0.4f} val_loss {v_loss:0.4f} train_acc {t_acc:0.4f} val_acc {v_acc:0.4f}"

    with tqdm(desc="epoch", total=self.epochs) as pbar_outer:
        l_rate = 1e-5
        opt = torch.optim.Adam(self.model.parameters(), lr=l_rate)
        criterion = nn.CrossEntropyLoss()

        for epoch in range(self.epochs):

            train_loss, train_acc = self.fit_epoch(train_loader, criterion, opt)
            print("loss", train_loss)

            #LBL2
            val_loss, val_acc = self.eval_epoch(val_loader, criterion)
            history.append((train_loss, train_acc, val_loss, val_acc))

            #LBL1
            if (val_acc > max_val_acc):
                max_val_acc = val_acc
                self.save('best_during_training')

            #LBL5
            pbar_outer.update(1)
            tqdm.write(log_template.format(ep=epoch+1, t_loss=train_loss, \
                                          v_loss=val_loss, t_acc=train_acc, v_acc=val_acc))

    return history
    print(f'training done')

def fit_epoch(self, train_loader, criterion, optimizer):
    running_loss = 0.0
    running_corrects = 0
    processed_data = 0

    for inputs, labels in train_loader:
        inputs = inputs.to(DEVICE).float().movedim(-1, 1)
        # print(inputs.size())
        labels = labels.to(DEVICE).long()
        optimizer.zero_grad()

        outputs = self.model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        preds = torch.argmax(outputs, 1)
        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)
        processed_data += inputs.size(0)

    train_loss = running_loss / processed_data
    train_acc = running_corrects.cpu().numpy() / processed_data
    return train_loss, train_acc

def eval_epoch(self, val_loader, criterion):
    self.model.eval()
    running_loss = 0.0
    running_corrects = 0
    processed_size = 0

    for inputs, labels in val_loader:
        inputs = inputs.to(DEVICE).float().movedim(-1, 1)
        labels = labels.to(DEVICE).long()

        with torch.set_grad_enabled(False):
            outputs = self.model(inputs)
            loss = criterion(outputs, labels)
            preds = torch.argmax(outputs, 1)

        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)
        processed_size += inputs.size(0)
    val_loss = running_loss / processed_size
    val_acc = running_corrects.double() / processed_size
    return val_loss, val_acc

def test_on_dataset(self, dataset: ImgDataset, limit=None):
    # you can upgrade this code if you want to speed up testing using batches
    self.model.eval()

    inputs = torch.from_numpy(dataset.images)

    inputs = inputs.to(DEVICE).float().movedim(-1, 1)

    with torch.set_grad_enabled(False):
        outputs = self.model(inputs)
        print("outp: ")
        print(outputs.size())
        print("\n")
        print("inp: ")
        print(inputs.size())
        print("\n")
        preds = torch.argmax(outputs, 1)

```

```

predictions = preds.detach().clone()

n = dataset.n_files if not limit else int(dataset.n_files * limit)
return np.array(torch.Tensor(predictions).flatten().cpu())

def test_on_image(self, img: np.ndarray):
    # todo: replace this code
    self.model.eval()
    prediction = self.model(torch.from_numpy(img).to(DEVICE).float().unsqueeze(1))
    return prediction

```

▼ Классификация изображений

Используя введенные выше классы можем перейти уже непосредственно к обучению модели классификации изображений. Пример общего пайплайна решения задачи приведен ниже. Вы можете его расширять и улучшать. В данном примере используются наборы данных 'train_small' и 'test_small'.

```

d_train = ImgDataset('train_small')
d_val = ImgDataset('test_tiny')
d_test = ImgDataset('test_small')

Downloading...
From (uriginal): https://drive.google.com/uc?id=iyFYg7RuOGofOJs2exr0XisyZSteDPb8F
From (redirected): https://drive.google.com/uc?id=iyFYg7RuOGofOJs2exr0XisyZSteDPb8F&confirm=t&uuiid=b21709fe-c303-44c7-a72d-ced82e6d4045
To: /content/train_small.npz
100%|██████████| 841M/841M [00:05<00:00, 142MB/s]
Loading dataset train_small from npz.
Done. Dataset train_small consists of 7200 images.
Downloading...
From: https://drive.google.com/uc?id=18e8gzRxqBLGHoyTwRwO3SCf\_kAITLE07
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 115MB/s]
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
Downloading...
From (uriginal): https://drive.google.com/uc?id=1mmfoAtfIhcjhGE5UGhBHpCnL6HSVRSSP
From (redirected): https://drive.google.com/uc?id=1mmfoAtfIhcjhGE5UGhBHpCnL6HSVRSSP&confirm=t&uuiid=0ff890d8-6d70-446a-b418-edfc66fd981c
To: /content/test_small.npz
100%|██████████| 211M/211M [00:01<00:00, 114MB/s]
Loading dataset test_small from npz.
Done. Dataset test_small consists of 1800 images.

d_train_full = ImgDataset('train')

Downloading...
From (uriginal): https://drive.google.com/uc?id=1EkOL97xSiNgjYOE3rHuTFox9Vd2841P1
From (redirected): https://drive.google.com/uc?id=1EkOL97xSiNgjYOE3rHuTFox9Vd2841P1&confirm=t&uuiid=41cc9913-48f2-4cbe-ac73-105c6a4c54d6
To: /content/train.npz
100%|██████████| 2.10G/2.10G [00:10<00:00, 208MB/s]
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.

model = Model()
if not EVALUATE_ONLY:
    history = model.train(d_train_full, d_val)
    model.save('best')
else:
    #todo: your link goes here
    model.load('best')

```

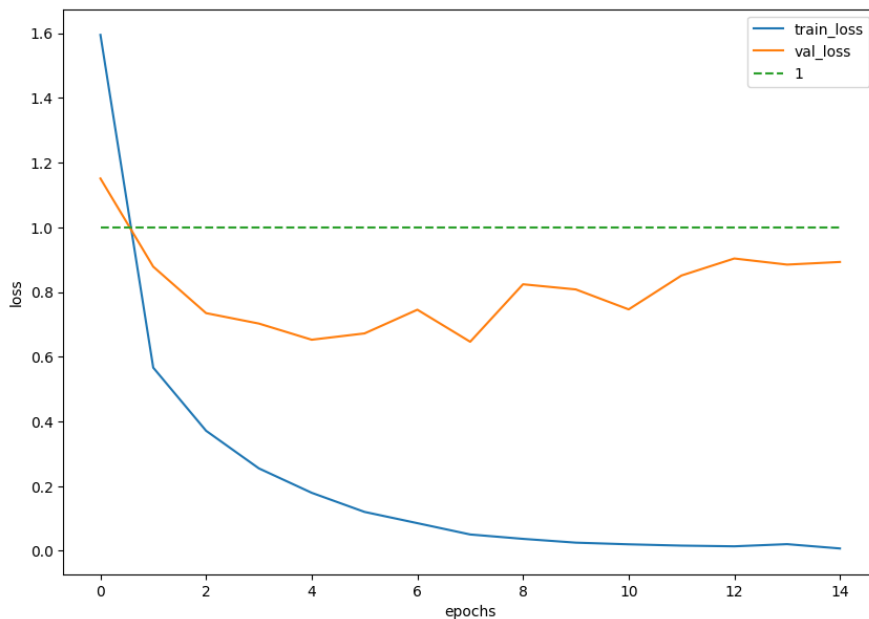
```

/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretra
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than
warnings.warn(msg)
training started
epoch: 100%
15/15 [04:46<00:00, 19.04s/it]
loss 1.595020732793543

Epoch 001 train_loss: 1.5950 val_loss 1.1510 train_acc 0.6224 val_acc 0.7222
loss 0.5662980242570241
loss, acc, val_loss, val_acc = zip(*history)
loss 0.37114934640874464
from matplotlib import colors, pyplot as plt
%matplotlib inline

#LBL3
plt.figure(figsize=(10, 7))
plt.plot(loss, label="train_loss")
plt.plot(val_loss, label="val_loss")
plt.plot(np.ones(np.shape(val_loss)), ls='--', label="1")
plt.legend(loc='best')
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()

```



Пример тестирования модели на части набора данных:

```

# evaluating model on 10% of test dataset
pred_1 = model.test_on_dataset(d_test, limit=0.1)
Metrics.print_all(d_test.labels[:len(pred_1)], pred_1, '10% of test')

outp:
torch.Size([1800, 9])

inp:
torch.Size([1800, 3, 224, 224])

metrics for 10% of test:
accuracy 0.9122:
balanced accuracy 0.9122:

```

Пример тестирования модели на полном наборе данных:

```

d_test_full = ImgDataset('test')

Downloading...
From (original): https://drive.google.com/uc?id=1pCi0l3n3Tj4jkPW-Sf6muqbGxkCE2L1K
From (redirected): https://drive.google.com/uc?id=1pCi0l3n3Tj4jkPW-Sf6muqbGxkCE2L1K&confirm=t&uuiid=3bad33dd-1035-498e-9f36-bbca8773f1e8
To: /content/test.npz
100%|██████████| 525M/525M [00:02<00:00, 188MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.

# evaluating model on full test dataset (may take time)
if TEST_ON_LARGE_DATASET:
    pred_2 = model.test_on_dataset(d_test_full)
    Metrics.print_all(d_test_full.labels, pred_2, 'test')

outp:
torch.Size([4500, 9])

inp:
torch.Size([4500, 3, 224, 224])

```