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 (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.12.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.16.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.6)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdow
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdow
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->
```

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 = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '1kcU115ZWapHdTGtVRP6FcNdbd_aDPo-0',
    'train_small': '1wChkmW5nmdDioqu2nDstt6fXeUov9fhJ',
    'train_tiny': '1_e1XSwTWDbUBoecZFO6lHkpgdbcH4kA0',
    'test': '1MJsag3OJ1xLQMr10O5yUY7o3nJUToEdW',
    'test_small': '1yJr057ly_9QKmbp5AehTUoRmg6JqlT9j',
    'test_tiny': '1qa0dK1R4a9qoO_GYHLyhQVZ_YQ-d_IOf'
}
```

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, accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
import gdown

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms

import os
import json
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
```

Класс Dataset

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

```
class Dataset:

    def __init__(self, name):
        self.name = name
        self.is_loaded = False
        url = f"https://drive.google.com/uc?export=download&confirm=pbef&id={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
        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]



class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, images, labels, transform=None):
        self.images = images
        self.labels = labels
        self.transform = transform

    def __iter__(self):
        for i in range(self.n_files):
            yield self.image(i), self.labels[i]

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

    def __getitem__(self, idx):
        image = self.images[idx]
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        return image, label
```

Пример испольвания 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 = Dataset('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: https://drive.google.com/uc?export=download&confirm=pbef&id=1_e1XSwTWDbUBoecZFO6lHkpgdbcH4kA0
To: /content/train_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 35.4MB/s]Loading dataset train_tiny from npz.
Done. Dataset train_tiny consists of 90 images.

Got numpy array of shape (224, 224, 3), and label with code 1.
Label code corresponds to BACK class.
```

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 заргрузки 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 выбрки 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.

```
#LBL1 -- Аугментация данных
data_augumentation = transforms.Compose([
            transforms.ToPILImage(),
            transforms.Resize((224, 224)),
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.RandomRotation(degrees=20),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406],
                                 [0.229, 0.224, 0.225])
        ])


class Model:

    def __init__(self):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = models.resnet50(pretrained=True)
        num_ftrs = self.model.fc.in_features
        self.model.fc = nn.Linear(num_ftrs, len(TISSUE_CLASSES))
        self.model = self.model.to(self.device)
        # для сохранения графиков обучения
        self.history = None
        # аугментация данных
        self.data_transforms = data_augumentation
        self.best_val_loss = float('inf')

    def save(self, name: str):
        torch.save(self.model.state_dict(), f'{name}.pth')
        print(f'Model saved to {name}.pth')

    def save_model(self, epoch, val_loss, path="/content/drive/MyDrive/Dataset for nn in image processing/checkpoints"):
        os.makedirs(path, exist_ok=True)
        save_path = os.path.join(path, f"model_epoch_{epoch}_val_loss_{val_loss:.4f}.pth")
        torch.save(self.model.state_dict(), save_path)
        print(f"Model saved to {save_path}")

    def load(self, name: str):
        name_to_id_dict = {
            'best': '1-jf-NX6_AMGvjS_48c-rFiWOpVTzywjb'
          }

        if name not in name_to_id_dict:
                raise ValueError(f"Модель '{name}' не найдена в словаре name_to_id_dict.")

        file_id = name_to_id_dict[name]
        url = f'https://drive.google.com/uc?id={file_id}'

        output = f'{name}.pth'

        gdown.download(url, output, quiet=False)
        self.model = self.model.to(self.device)
        state_dict = torch.load(output, map_location=self.device)
        self.model.load_state_dict(state_dict)
        print(f'Модель загружена из {output}')

    def train(self, dataset: Dataset):
        print(f'Training started')

        images = dataset.images
        labels = dataset.labels
        #LBL2 -- Валидация на части обучающей выборки
        x_train, x_val, y_train, y_val = train_test_split(
        images, labels, test_size=0.25, random_state=42)

        train_dataset = CustomDataset(images, labels, transform=self.data_transforms)
        val_dataset = CustomDataset(x_val, y_val, transform=self.data_transforms)

        train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=True)
        val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=32, shuffle=False)
```

```
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.SGD(self.model.parameters(), lr=0.001, momentum=0.9)

        history = {"loss": [], "accuracy": [], "val_loss": [], "val_accuracy": []}
        num_epochs = 10
        for epoch in range(num_epochs):
            self.model.train()
            running_loss = 0.0
            correct_predictions = 0
            total_samples = 0

            for inputs, labels in tqdm(train_loader):
                inputs = inputs.to(self.device)
                labels = labels.to(self.device)

                optimizer.zero_grad()

                outputs = self.model(inputs)
                loss = criterion(outputs, labels)

                loss.backward()
                optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                _, preds = torch.max(outputs, 1)
                correct_predictions += torch.sum(preds == labels).item()
                total_samples += labels.size(0)

            epoch_loss = running_loss / len(train_dataset)
            epoch_accuracy = correct_predictions / total_samples

            history["loss"].append(epoch_loss)
            history["accuracy"].append(epoch_accuracy)

            self.model.eval()
            val_running_loss = 0.0
            val_correct_predictions = 0
            val_total_samples = 0

            with torch.no_grad():
                for val_inputs, val_labels in val_loader:
                    val_inputs = val_inputs.to(self.device)
                    val_labels = val_labels.to(self.device)

                    val_outputs = self.model(val_inputs)
                    val_loss = criterion(val_outputs, val_labels)

                    val_running_loss += val_loss.item() * val_inputs.size(0)
                    _, val_preds = torch.max(val_outputs, 1)
                    val_correct_predictions += torch.sum(val_preds == val_labels).item()
                    val_total_samples += val_labels.size(0)

            val_epoch_loss = val_running_loss / len(val_dataset)
            val_epoch_accuracy = val_correct_predictions / val_total_samples

            #LBL3 -- Вывод различных показателей в процессе обучения (например, значение функции потерь на каждой эпохе)
            history["val_loss"].append(val_epoch_loss)
            history["val_accuracy"].append(val_epoch_accuracy)

            print(f'Epoch {epoch+1}/{num_epochs}, '
                  f'Loss: {epoch_loss:.4f}, Accuracy: {epoch_accuracy:.4f}, '
                  f'Val Loss: {val_epoch_loss:.4f}, Val Accuracy: {val_epoch_accuracy:.4f}')
            self.save_model(epoch, val_loss)

        #LBL4 -- Автоматическое сохранения модели при обучении
        if val_loss < self.best_val_loss:
            self.best_val_loss = val_loss
            self.save_model(epoch, val_loss, path="/content/drive/MyDrive/2Dataset for nn in image processing/best_checkpoir
        print(f'Training done')
        self.history = history
        self.training_plots(history)
        return history

    def test_on_dataset(self, dataset: Dataset, limit=None):
        self.model.eval()
        n = dataset.n_files if not limit else int(dataset.n_files * limit)
        predictions = []
        with torch.no_grad():
            for img in tqdm(dataset.images_seq(n), total=n):
                img = self.data_transforms(img)
                img = img.unsqueeze(0)
```

```
                img = img.to(self.device)
                outputs = self.model(img)
                _, preds = torch.max(outputs, 1)
                predictions.append(preds.item())
        return predictions

    def test_on_image(self, img: np.ndarray):
        self.model.eval()

        img = self.data_transforms(img)
        img = img.unsqueeze(0)
        img = img.to(self.device)

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

    #LBL5 -- Построение графиков, визуализирующих процесс обучения
    def training_plots(self, history):
        acc = history['accuracy']
        val_acc = history['val_accuracy']
        loss = history['loss']
        val_loss = history['val_loss']
        plt.figure(figsize=(10, 5))
        plt.plot(loss, label='Потери на обучающем наборе')
        plt.plot(val_loss, label='Потери на валидационном наборе')
        plt.title('График функции потерь')
        plt.xlabel('Эпоха')
        plt.ylabel('Потери')
        plt.legend()
        plt.show()
        plt.figure(figsize=(10, 5))
        plt.plot(acc, label='Точность на обучающем наборе')
        plt.plot(val_acc, label='Точность на валидационном наборе')
        plt.title('График точности модели')
        plt.xlabel('Эпоха')
        plt.ylabel('Точность')
        plt.legend()
        plt.show()

    #LBL6 -- Построение матрицы ошибок, оценивание чувствительности и специфичности модели
    def evaluate_model(self, data_loader, class_names):
        self.model.eval()
        all_preds = []
        all_labels = []

        with torch.no_grad():
            for inputs, labels in data_loader:
                inputs, labels = inputs.to(self.device), labels.to(self.device)
                outputs = self.model(inputs)
                _, preds = torch.max(outputs, 1)
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

        # Построение матрицы ошибок
        cm = confusion_matrix(all_labels, all_preds)
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
        plt.xlabel("Predicted Labels")
        plt.ylabel("True Labels")
        plt.title("Confusion Matrix")
        plt.show()

        # Чувствительность и специфичность
        print(classification_report(all_labels, all_preds, target_names=class_names))
        return cm
```

Классификация изображений

В следующем разделе представлены результаты обучения построенной модели на датасетах разных размеров, а также построены графики обучения.

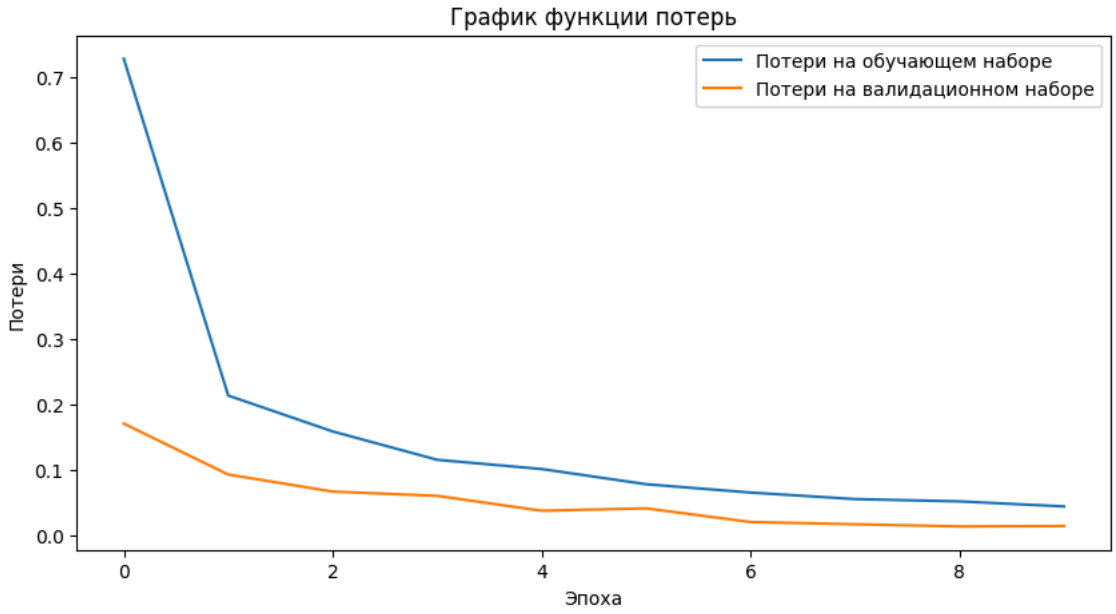
```
d_train = Dataset('train_small')
d_test = Dataset('test_small')
```

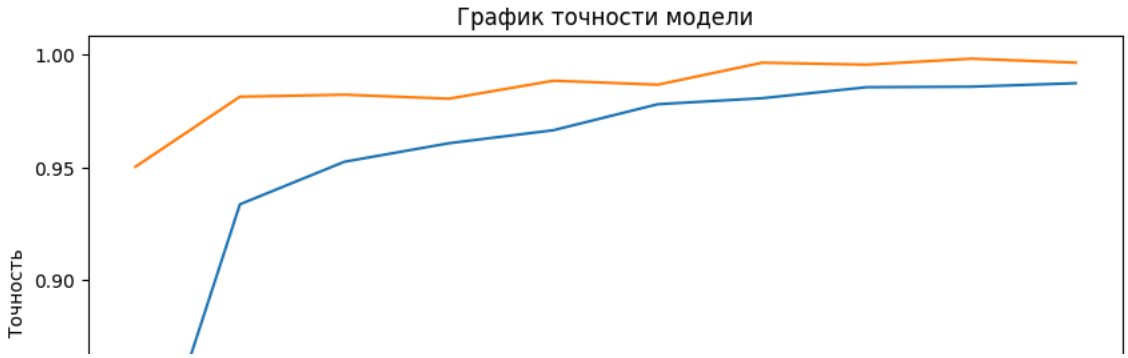
```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1wChkmW5nmdDioqu2nDstt6fXeUov9fhJ
To: /content/train_small.npz
100%|██████████| 525M/525M [00:05<00:00, 94.0MB/s]
Loading dataset train_small from npz.
Done. Dataset train_small consists of 4500 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1yJr057ly_9QKmbp5AehTUoRmg6JqlT9j
To: /content/test_small.npz
100%|██████████| 2.10G/2.10G [00:24<00:00, 85.1MB/s]
Loading dataset test_small from npz.
Done. Dataset test_small consists of 18000 images.
```

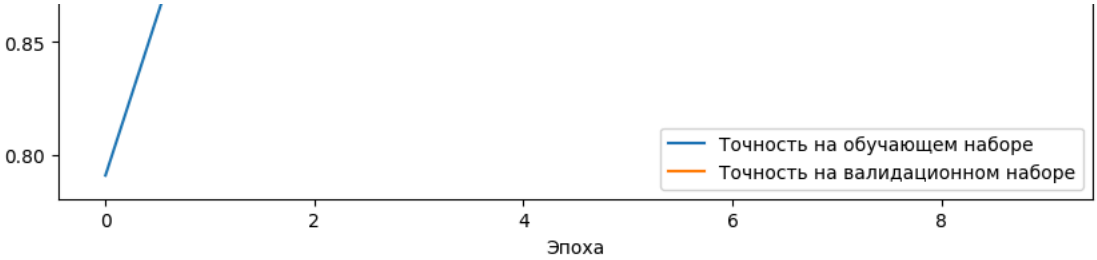
```
EVALUATE_ONLY = False
```

```
model = Model()
history = model.train(d_train)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is d
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight e
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet5
100%|██████████| 97.8M/97.8M [00:00<00:00, 188MB/s]
Training started
100%                                  141/141 [00:46<00:00,  3.30it/s]
Epoch 1/10, Loss: 0.7282, Accuracy: 0.7909, Val Loss: 0.1703, Val Accuracy: 0.9502
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_0_val_loss_0.1048.pth
100%                                  141/141 [00:47<00:00,  3.26it/s]
Epoch 2/10, Loss: 0.2133, Accuracy: 0.9336, Val Loss: 0.0925, Val Accuracy: 0.9813
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_1_val_loss_0.0552.pth
100%                                  141/141 [00:47<00:00,  3.31it/s]
Epoch 3/10, Loss: 0.1583, Accuracy: 0.9524, Val Loss: 0.0664, Val Accuracy: 0.9822
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_2_val_loss_0.0212.pth
100%                                  141/141 [00:48<00:00,  3.28it/s]
Epoch 4/10, Loss: 0.1150, Accuracy: 0.9607, Val Loss: 0.0599, Val Accuracy: 0.9804
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_3_val_loss_0.0109.pth
100%                                  141/141 [00:47<00:00,  3.28it/s]
Epoch 5/10, Loss: 0.1009, Accuracy: 0.9664, Val Loss: 0.0371, Val Accuracy: 0.9884
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_4_val_loss_0.0060.pth
100%                                  141/141 [00:48<00:00,  3.27it/s]
Epoch 6/10, Loss: 0.0776, Accuracy: 0.9780, Val Loss: 0.0407, Val Accuracy: 0.9867
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_5_val_loss_0.0066.pth
100%                                  141/141 [00:47<00:00,  3.26it/s]
Epoch 7/10, Loss: 0.0649, Accuracy: 0.9807, Val Loss: 0.0199, Val Accuracy: 0.9964
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_6_val_loss_0.0093.pth
100%                                  141/141 [00:48<00:00,  3.27it/s]
Epoch 8/10, Loss: 0.0549, Accuracy: 0.9856, Val Loss: 0.0164, Val Accuracy: 0.9956
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_7_val_loss_0.0040.pth
100%                                  141/141 [00:47<00:00,  3.27it/s]
Epoch 9/10, Loss: 0.0514, Accuracy: 0.9858, Val Loss: 0.0131, Val Accuracy: 0.9982
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_8_val_loss_0.0035.pth
100%                                  141/141 [00:48<00:00,  3.26it/s]
Epoch 10/10, Loss: 0.0437, Accuracy: 0.9873, Val Loss: 0.0137, Val Accuracy: 0.9964
Model saved to /content/drive/MyDrive/Dataset for nn in image processing/checkpoints/model_epoch_9_val_loss_0.0044.pth
Model saved to /content/drive/MyDrive/2Dataset for nn in image processing/best_checkpoint/model_epoch_9_val_loss_0.004
Training done
```



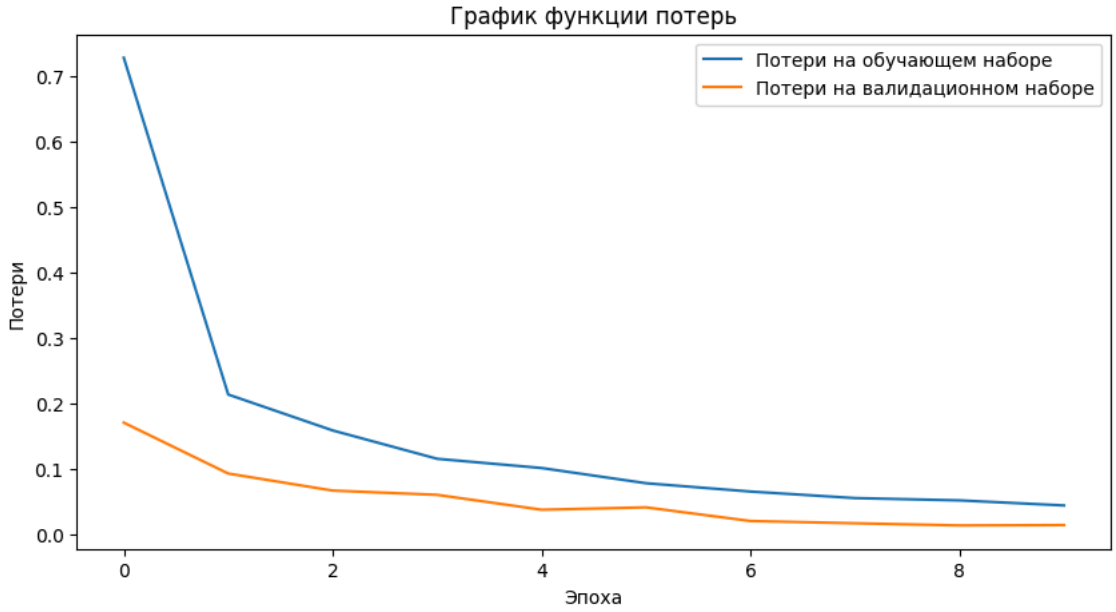


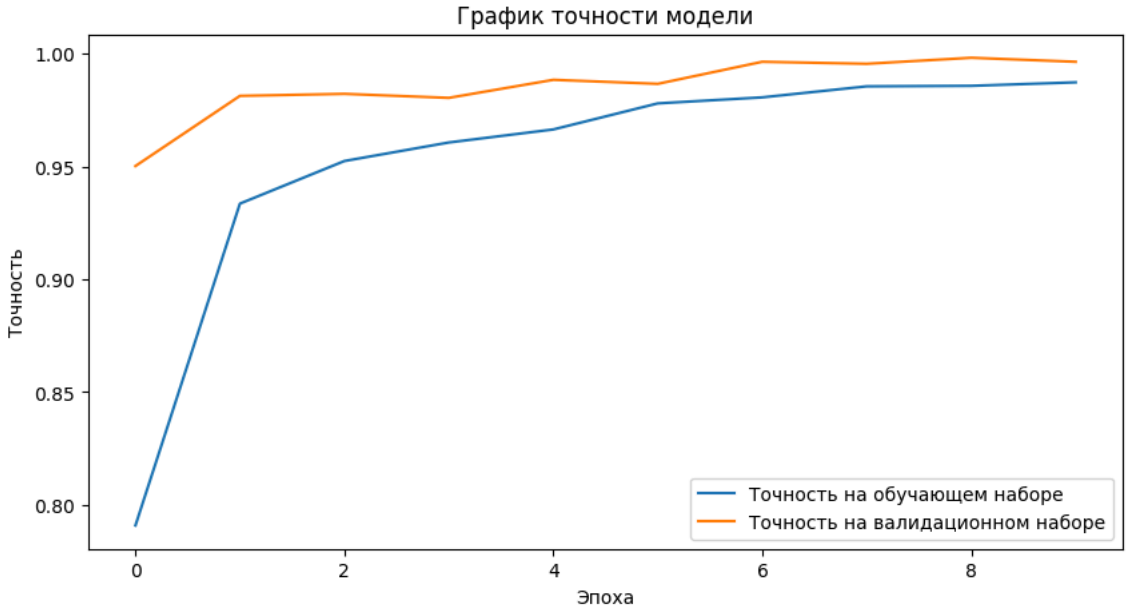



```
model.save('best')
```

```
Model saved to best.pth
```

```
model.training_plots(history)
```






```
# 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')
```

```
100%                                                  1800/1800 [00:15<00:00, 118.12it/s]
metrics for 10% of test:
         accuracy 0.9956:
         balanced accuracy 0.9956:
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:2480: UserWarning: y_pred contains classes no
  warnings.warn("y_pred contains classes not in y_true")
```

Пример тестирования модели на полном наборе данных:

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

```
100%                                                  18000/18000 [02:29<00:00, 87.10it/s]
metrics for test:
         accuracy 0.9824:
         balanced accuracy 0.9824:
```

```
model2 = Model()
model2.load('best')
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is dep
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enu
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-
100%|██████████| 97.8M/97.8M [00:00<00:00, 184MB/s]
Downloading...
From (original): https://drive.google.com/uc?id=1-jf-NX6_AMGvjS_48c-rFiWOpVTzywjb
From (redirected): https://drive.google.com/uc?id=1-jf-NX6_AMGvjS_48c-rFiWOpVTzywjb&confirm=t&uuid=1100b42a-c089-4929-b0
To: /content/best.pth
100%|██████████| 94.4M/94.4M [00:02<00:00, 45.7MB/s]Модель загружена из best.pth

<ipython-input-10-2d6f2e12acc0>:40: FutureWarning: You are using `torch.load` with `weights_only=False` (the current def
  state_dict = torch.load(output, map_location=self.device)
```

```
# evaluating model on 10% of test dataset
pred_1 = model2.test_on_dataset(d_test, limit=0.1)
Metrics.print_all(d_test.labels[:len(pred_1)], pred_1, '10% of test')
```

```
100%                                                  1800/1800 [00:16<00:00, 111.87it/s]
metrics for 10% of test:
         accuracy 0.9961:
         balanced accuracy 0.9961:
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:2480: UserWarning: y_pred contains classes no
  warnings.warn("y_pred contains classes not in y_true")
```

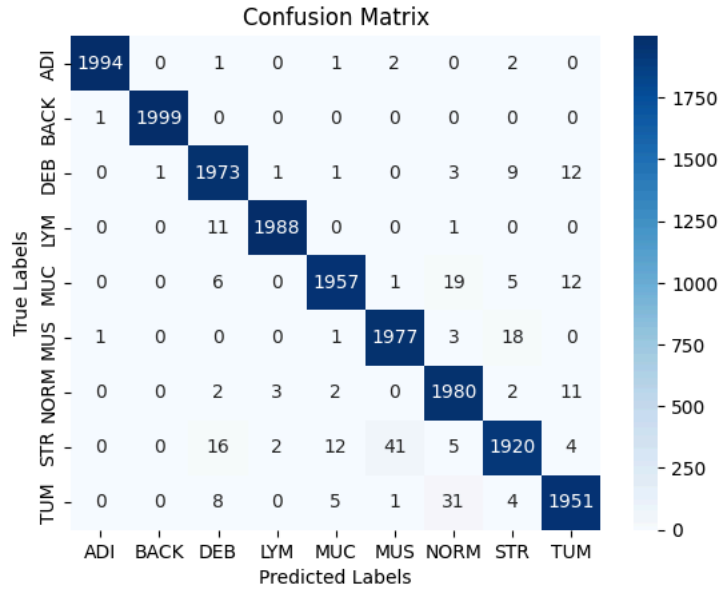
```
# evaluating model on full test dataset (may take time)
if TEST_ON_LARGE_DATASET:
    pred_2 = model2.test_on_dataset(d_test)
    Metrics.print_all(d_test.labels, pred_2, 'test')
```

```
100%                                                  18000/18000 [02:34<00:00, 116.72it/s]
metrics for test:
         accuracy 0.9825:
         balanced accuracy 0.9825:
```

```
data_transforms = transforms.Compose([
        transforms.ToPILImage(),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])


test_dataset = CustomDataset(d_test.images, d_test.labels, transform=data_transforms)
test_dataset_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32, shuffle=False)


cm = model.evaluate_model(test_dataset_loader, TISSUE_CLASSES)
```



	precision	recall	f1-score	support
ADI	1.00	1.00	1.00	2000
BACK	1.00	1.00	1.00	2000
DEB	0.98	0.99	0.98	2000
LYM	1.00	0.99	1.00	2000
MUC	0.99	0.98	0.98	2000
MUS	0.98	0.99	0.98	2000
NORM	0.97	0.99	0.98	2000
STR	0.98	0.96	0.97	2000
TUM	0.98	0.98	0.98	2000
accuracy			0.99	18000
macro avg	0.99	0.99	0.99	18000
weighted avg	0.99	0.99	0.99	18000

Результат работы пайплайна обучения и тестирования выше тоже будет оцениваться. Поэтому не забудьте присылать на проверку ноутбук с выполненными ячейками кода с демонстрациями метрик обучения, графиками и т.п. В этом пайплайне Вам необходимо продемонстрировать работу всех реализованных дополнений, улучшений и т.п.

Настоятельно рекомендуется после получения пайплайна с полными результатами обучения экспортировать ноутбук в pdf (файл -> печать) и прислать этот pdf вместе с самим ноутбуком.

Тестирование модели на других наборах данных

Ваша модель должна поддерживать тестирование на других наборах данных. Для удобства, Вам предоставляется набор данных test_tiny, который представляет собой малую часть (2% изображений) набора test. Ниже приведен фрагмент кода, который будет осуществлять тестирование для оценивания Вашей модели на дополнительных тестовых наборах данных.

Прежде чем отсылать задание на проверку, убедитесь в работоспособности фрагмента кода ниже.

```
final_model = Model()
final_model.load('best')
d_test_tiny = Dataset('test_tiny')
pred = final_model.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, 'test-tiny')
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is dep
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enu
  warnings.warn(msg)
Downloading...
From (original): https://drive.google.com/uc?id=1-jf-NX6_AMGvjS_48c-rFiWOpVTzywjb
From (redirected): https://drive.google.com/uc?id=1-jf-NX6_AMGvjS_48c-rFiWOpVTzywjb&confirm=t&uuid=78bfe64f-131b-4a54-88
To: /content/best.pth
100%|██████████| 94.4M/94.4M [00:03<00:00, 27.9MB/s]
<ipython-input-10-2d6f2e12acc0>:40: FutureWarning: You are using `torch.load` with `weights_only=False` (the current def
  state_dict = torch.load(output, map_location=self.device)
Модель загружена из best.pth
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1qa0dK1R4a9qoO_GYHLyhQVZ_YQ-d_IOf
To: /content/test_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 113MB/s]
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 900 images.
```

100% 900/900 [00:07<00:00, 107.39it/s]

```
metrics for test-tiny:
         accuracy 0.9822:
         balanced accuracy 0.9822:
```

Отмонтировать Google Drive.

```
drive.flush_and_unmount()
```

Дополнительные "полезности"

Ниже приведены примеры использования различных функций и библиотек, которые могут быть полезны при выполнении данного практического задания.

Измерение времени работы кода

Измерять время работы какой-либо функции можно легко и непринужденно при помощи функции timeit из соответствующего модуля:

```
import timeit

def factorial(n):
    res = 1
    for i in range(1, n + 1):
        res *= i
    return res


def f():
    return factorial(n=1000)

n_runs = 128
print(f'Function f is caluclated {n_runs} times in {timeit.timeit(f, number=n_runs)}s.')
```

Scikit-learn

Для использования "классических" алгоритмов машинного обучения рекомендуется использовать библиотеку scikit-learn (https://scikit-learn.org/stable/). Пример классификации изображений цифр из набора данных MNIST при помощи классификатора SVM:

```
# Standard scientific Python imports
import matplotlib.pyplot as plt

# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split

# The digits dataset
digits = datasets.load_digits()

# The data that we are interested in is made of 8x8 images of digits, let's
# have a look at the first 4 images, stored in the `images` attribute of the
```

```
# dataset.  If we were working from image files, we could load them using
# matplotlib.pyplot.imread.  Note that each image must have the same size. For these
# images, we know which digit they represent: it is given in the 'target' of
# the dataset.
_, axes = plt.subplots(2, 4)
images_and_labels = list(zip(digits.images, digits.target))
for ax, (image, label) in zip(axes[0, :], images_and_labels[:4]):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Training: %i' % label)

# To apply a classifier on this data, we need to flatten the image, to
# turn the data in a (samples, feature) matrix:
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Create a classifier: a support vector classifier
classifier = svm.SVC(gamma=0.001)

# Split data into train and test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False)

# We learn the digits on the first half of the digits
classifier.fit(X_train, y_train)

# Now predict the value of the digit on the second half:
predicted = classifier.predict(X_test)

images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
for ax, (image, prediction) in zip(axes[1, :], images_and_predictions[:4]):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Prediction: %i' % prediction)

print("Classification report for classifier %s:\n%s\n"
      % (classifier, metrics.classification_report(y_test, predicted)))
disp = metrics.plot_confusion_matrix(classifier, X_test, y_test)
disp.figure_.suptitle("Confusion Matrix")
print("Confusion matrix:\n%s" % disp.confusion_matrix)

plt.show()
```

Scikit-image

Реализовывать различные операции для работы с изображениями можно как самостоятельно, работая с массивами numpy, так и используя специализированные библиотеки, например, scikit-image (https://scikit-image.org/). Ниже приведен пример использования Canny edge detector.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi

from skimage import feature


# Generate noisy image of a square
im = np.zeros((128, 128))
im[32:-32, 32:-32] = 1

im = ndi.rotate(im, 15, mode='constant')
im = ndi.gaussian_filter(im, 4)
im += 0.2 * np.random.random(im.shape)

# Compute the Canny filter for two values of sigma
edges1 = feature.canny(im)
edges2 = feature.canny(im, sigma=3)

# display results
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(8, 3)
                                    sharex=True, sharey=True)

ax1.imshow(im, cmap=plt.cm.gray)
ax1.axis('off')
ax1.set_title('noisy image', fontsize=20)

ax2.imshow(edges1, cmap=plt.cm.gray)
ax2.axis('off')
ax2.set_title(r'Canny filter, $\sigma=1$', fontsize=20)
```

```
ax3.imshow(edges2, cmap=plt.cm.gray)
ax3.axis('off')
ax3.set_title(r'Canny filter, $\sigma=3$', fontsize=20)

fig.tight_layout()

plt.show()
```

Tensorflow 2

Для создания и обучения нейросетевых моделей можно использовать фреймворк глубокого обучения Tensorflow 2. Ниже приведен пример простейшей нейроной сети, использующейся для классификации изображений из набора данных MNIST.

```
# Install TensorFlow

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```