

▼ Практическое задание №1

Установка необходимых пакетов:

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

Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch->timm) (75.2.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (3.6)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (12.6.7)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (12.6)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (12.6.8)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (10.3.7.7)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (12.5.4)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (1.11.1.6)
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch->timm) (3.5.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from torchvision->timm) (1.11.1.6)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from torchvision->timm) (11.3.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch->timm) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch->timm) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface_hub->timm) (3.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface_hub->timm) (3.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface_hub->timm) (3.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->huggingface_hub->timm) (3.1)
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.9.0+cu126)
Requirement already satisfied: torchvision in /usr/local/lib/python3.12/dist-packages (0.24.0+cu126)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch) (3.20.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1.14.0)
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch) (3.6)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch) (2.27.5)
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch) (3.3.20)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch) (3.5.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from torchvision) (2.0.2)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from torchvision) (11.3.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch) (3.0.3)
```

Монтируем 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': '1j0G54j1f2-f-fcbuv89Wzn_wX-wTrR9I',
    'train_small': '1fYTCskVYCXUnQntQg1xpqEfYIcSuoEBA',
    'train_tiny': '17V3ly7zCNSK819Glhqv1ENDgbFowJXgQ',
    'test': '1YwpLnEdftoAOH_C9ti6IPxEzYK8tMyx',
    'test_small': '1ov0fxHc4lxRs3MSSXV2tu6LnYOBQE1Ay',
```

```

'test_tiny': '1eAyZ9oUQeg_E887gNQNaK2x81EGIVTXK'
} ..
'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-ClI',
'train_small': '1qd45xFDwdZjktLFwQb-et-mAaFeCzOR',
'train_tiny': '1I-2Z0uXLd4QwhZQQltp817Kn3J0Xgbui',
'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr',
'test_small': '1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ21I',
'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}'''
```

'\n 'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-ClI',\n 'train_small': '1qd45xFDwdZjktLFwQb-et-mAaFeCzOR',\n 'train_tiny': '1I-2Z0uXLd4QwhZQQltp817Kn3J0Xgbui',\n 'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr',\n 'test_small': '1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ21I',\n 'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'\n}'

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
```

▼ 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
        file_id = DATASETS_LINKS[name]
        url = f"https://drive.google.com/uc?export=download&id={file_id}"
        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', allow_pickle=True)

        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]
```

▼ Пример использования класса Dataset

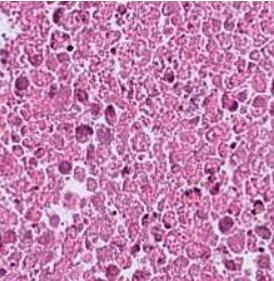
Загрузим обучающий набор данных, получим произвольное изображение с меткой. После чего визуализируем изображение, выведем метку. В будущем, этот кусок кода можно закомментировать или убрать.

```
d_train_tiny = 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 (original): https://drive.google.com/uc?export=download&id=17V3ly7zCNSK819GLhgylENDgbFowJXgQ
From (redirected): https://drive.google.com/uc?export=download&id=17V3ly7zCNSK819GLhgvlENDgbFowJXgQ&confirm=t&uuid=84aeb34d-bc2a-46de-
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 131MB/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 2.
Label code corresponds to DEB class.

```

▼ Обёртка над Dataset для использования с PyTorch

```
# =====
# (Опционально) Обёртка над Dataset для использования с PyTorch
# =====

# ВНИМАНИЕ:
# Этот код нужен только тем, кто хочет решать задание с помощью PyTorch.
# Он показывает, как "подключить" наш Dataset к torch.utils.data.DataLoader.

try:
    import torch
    from torch.utils.data import Dataset as TorchDataset, DataLoader
    import torchvision.transforms as T
    from PIL import Image

    class HistologyTorchDataset(TorchDataset):
        """
        Обёртка над Dataset для использования с PyTorch.

        base_dataset: экземпляр Dataset('train'), Dataset('train_small'), etc.
        transform: функция/объект, преобразующий изображение (PIL.Image -> torch.Tensor).

        """
        def __init__(self, base_dataset, transform=None):
            self.base = base_dataset
            # Минимальный transform по умолчанию:
            # np.uint8 [0, 255] -> float32 [0.0, 1.0]
            self.transform = transform or T.ToTensor()

        def __len__(self):
            # Размер датасета
            return len(self.base.images)

        def __getitem__(self, idx):
            """
            Возвращает (image_tensor, label) для PyTorch.
            image_tensor: torch.Tensor формы [3, H, W]
            label: int
            """
            img, label = self.base.image_with_label(idx) # img: np.ndarray (H, W, 3)
            img = Image.fromarray(img) # в PIL.Image
            img = self.transform(img) # в torch.Tensor
            return img, label

except ImportError:
    HistologyTorchDataset = None
    print("PyTorch / torchvision не найдены. Обёртка HistologyTorchDataset недоступна.")
```

▼ Пример использования класса HistologyTorchDataset

```
if "HistologyTorchDataset" not in globals() or HistologyTorchDataset is None:
    print("PyTorch не установлен или обёртка недоступна – пример пропущен.")
else:
    print("Пример использования PyTorch-обёртки над Dataset")

base_train = Dataset('train_tiny')

# Создаём PyTorch-совместимый датасет
train_ds = HistologyTorchDataset(base_train)

# DataLoader автоматически создаёт батчи и перемешивает данные
from torch.utils.data import DataLoader
train_loader = DataLoader(train_ds, batch_size=8, shuffle=True)

# Берём один батч и выводим информацию
images_batch, labels_batch = next(iter(train_loader))

print("Форма батча изображений:", tuple(images_batch.shape)) # [batch, 3, 224, 224]
print("Форма батча меток:", tuple(labels_batch.shape)) # [batch]
print("Пример меток:", labels_batch[:10].tolist())

print("Тип images_batch:", type(images_batch))
print("Тип labels_batch:", type(labels_batch))
```

```
Пример использования PyTorch-обёртки над Dataset
Downloading...
From (original): https://drive.google.com/uc?export=download&id=17V3ly7zCNSK819GLhgylENDgbFowJXgQ
From (redirected): https://drive.google.com/uc?export=download&id=17V3ly7zCNSK819GLhgylENDgbFowJXgQ&confirm=t&uuid=484dc22a-1e18-48cd-
To: /content/train_tiny.npz
100%[██████████] 105M/105M [00:00<00:00, 193MB/s]
Loading dataset train_tiny from npz.
Done. Dataset train_tiny consists of 900 images.
Форма батча изображений: (8, 3, 224, 224)
Форма батча меток: (8,)
Пример меток: [1, 0, 6, 3, 2, 0, 3, 0]
Тип images_batch: <class 'torch.Tensor'>
Тип labels_batch: <class 'torch.Tensor'>
```

▼ 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('\taccuracy {:.4f}'.format(Metrics.accuracy(gt, pred)))
        print('\tbalanced 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.

```

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
import timm
import matplotlib.pyplot as plt
import json
from datetime import datetime
from tqdm.notebook import tqdm
import numpy as np
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

class Model:

    def __init__(self):
        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.num_classes = len(TISSUE_CLASSES)
        self.model = None
        self._setup_model()
        self.history = {
            'train_loss': [], 'val_loss': [],
            'train_acc': [], 'val_acc': []
        }

    def _setup_model(self):
        self.model = timm.create_model(
            'efficientnet_b0',
            pretrained=True,
            num_classes=self.num_classes
        )
        self.model = self.model.to(self.device)

#6
    self.train_transform = transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.RandomRotation(15),
        transforms.ColorJitter(brightness=0.2, contrast=0.2),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                            std=[0.229, 0.224, 0.225])
    ])

    self.test_transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                            std=[0.229, 0.224, 0.225])
    ])

    def save(self, name: str):
        torch.save({
            'model_state_dict': self.model.state_dict(),
            'history': self.history,
            'num_classes': self.num_classes
        }, f'/content/drive/MyDrive/{name}.pth')
        print(f"Модель сохранена как {name}.pth")

    def load(self, name: str):
        file_id = "1DplzCeQ0-PKW8xS-L1aHr9LCJyM8CZcv"
        url = f"https://drive.google.com/uc?id={file_id}"

```

```

output = f'{name}.pth'

gdown.download(url, output, quiet=False)
checkpoint = torch.load(output,
                        map_location=self.device)
self.model.load_state_dict(checkpoint['model_state_dict'])
self.history = checkpoint.get('history', self.history)
print(f"Модель загружена из {url}.pth")

def train(self, dataset: Dataset, epochs=12, validation_split=0.2):

    print(f"Начало обучения")

    # Создаем PyTorch датасеты
    train_dataset = HistologyTorchDataset(dataset, transform=self.train_transform)

#1
    # Разделение на train/val
    dataset_size = len(train_dataset)
    val_size = int(validation_split * dataset_size)
    train_size = dataset_size - val_size

    train_subset, val_subset = torch.utils.data.random_split(
        train_dataset, [train_size, val_size]
    )

    # DataLoader'ы
    train_loader = DataLoader(train_subset, batch_size=32, shuffle=True, num_workers=2)
    val_loader = DataLoader(val_subset, batch_size=32, shuffle=False, num_workers=2)

    # Оптимизатор и функция потерь
    optimizer = optim.AdamW(self.model.parameters(), lr=3e-4, weight_decay=1e-4)
    criterion = nn.CrossEntropyLoss()
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=epochs)

    best_val_acc = 0
    patience = 5
    patience_counter = 0

    for epoch in range(epochs):
        # Обучение
        self.model.train()
        train_loss = 0
        train_correct = 0
        train_total = 0

        train_progress = tqdm(train_loader, desc=f'Обучение',
                             bar_format='{l_bar}{bar:30}{r_bar}{bar:-10b}')

        for batch_idx, (data, target) in enumerate(train_progress):
            data, target = data.to(self.device), target.to(self.device)

            optimizer.zero_grad()
            output = self.model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            _, predicted = output.max(1)
            train_total += target.size(0)
            train_correct += predicted.eq(target).sum().item()

            current_train_loss = train_loss / (batch_idx + 1)
            current_train_acc = 100. * train_correct / train_total

            # Обновляем описание прогресс-бара
            train_progress.set_postfix({
                'Loss': f'{current_train_loss:.4f}',
                'Acc': f'{current_train_acc:.2f}%',
                'LR': f'{optimizer.param_groups[0]["lr"]:.6f}'
            })

        train_acc = 100. * train_correct / train_total
        train_loss = train_loss / len(train_loader)
        print(f"Loss: {train_loss:.4f}")
        print(f"Acc: {train_acc:.2f}%")

        # Валидация
        val_loss, val_acc = self._validate(val_loader, criterion)

        # Обновление scheduler
        scheduler.step()

        # Сохранение истории
        self.history['train_loss'].append(train_loss)
        self.history['train_acc'].append(train_acc)
        self.history['val_loss'].append(val_loss)
        self.history['val_acc'].append(val_acc)

```

```

#3
    print(f'Epoch {epoch+1}/{epochs}:')
    print(f' Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%')
    print(f' Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%')
    print(f' LR: {optimizer.param_groups[0]["lr"]:.6f}')

```

```

predictions = []
n = dataset.n_files if not limit else int(dataset.n_files * limit)

# Используем последовательный доступ к изображениям
for i, img in tqdm(enumerate(dataset.images_seq(n)), total=n, desc="Тестирование"):
    if i >= n:
        break
    prediction = self.test_on_image(img)
    predictions.append(prediction)

return predictions

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

    # Преобразуем numpy array в PIL Image и применяем трансформации
    pil_img = Image.fromarray(img)
    input_tensor = self.test_transform(pil_img).unsqueeze(0).to(self.device)

    with torch.no_grad():
        output = self.model(input_tensor)
    _, prediction = torch.max(output, 1)

    return prediction.item()

#5
def plot_confusion_matrix(self, true_labels, predictions, dataset_name=""):
    cm = confusion_matrix(true_labels, predictions)

    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=TISSUE_CLASSES,
                yticklabels=TISSUE_CLASSES)
    plt.title(f'Матрица ошибок')
    plt.xlabel('Предсказанные классы')
    plt.ylabel('Истинные классы')
    plt.xticks(rotation=45)
    plt.yticks(rotation=0)
    plt.tight_layout()
    plt.show()

    return cm

```

▼ 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 = Dataset('train')
d_test = Dataset('test')

Downloading...
From (original): https://drive.google.com/uc?export=download&id=1j0G54j1f2-f-fcbuv89Wzn\_wX-wTrR9I
From (redirected): https://drive.google.com/uc?export=download&id=1j0G54j1f2-f-fcbuv89Wzn\_wX-wTrR9I&confirm=t&uuid=5229f3a6-fbd9-4afe-to:/content/train.npz
100%|██████████| 2.10G/2.10G [00:25<00:00, 81.4MB/s]
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
Downloading...
From (original): https://drive.google.com/uc?export=download&id=1YwpLnEdftoAOH\_C9ti61IPxEzYK8tMyx
From (redirected): https://drive.google.com/uc?export=download&id=1YwpLnEdftoAOH\_C9ti61IPxEzYK8tMyx&confirm=t&uuid=3b358263-c11e-4d10-to:/content/test.npz
100%|██████████| 525M/525M [00:05<00:00, 99.8MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.

```

```

model = Model()
if not EVALUATE_ONLY:
    model.train(d_train)
    model.save('best')
else:
    #todo: your link goes here
    #https://drive.google.com/file/d/1DplzCeQ0-PKW8xS-L1aHr9LCJyM8CZcv/view?usp=sharing
    #https://drive.google.com/file/d/1DplzCeQ0-PKW8xS-L1aHr9LCJyM8CZcv/view?usp=drive\_link
    model.load('best')

```

```

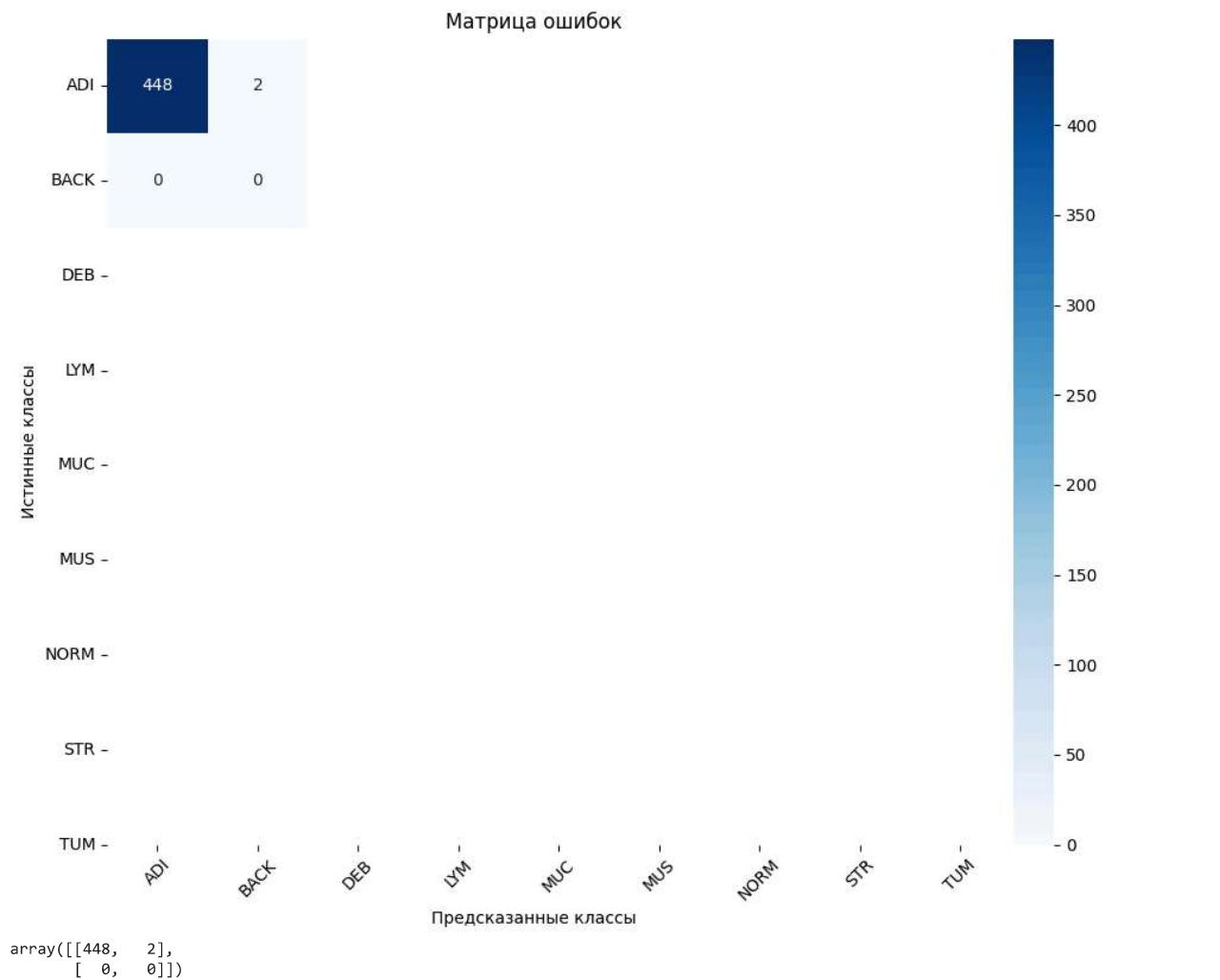
Downloading...
From: https://drive.google.com/uc?id=1DplzCeQ0-PKW8xS-L1aHr9LCJyM8CZcv
To: /content/best.pth
100%|██████████| 16.4M/16.4M [00:00<00:00, 110MB/s]
Модель загружена из https://drive.google.com/uc?id=1DplzCeQ0-PKW8xS-L1aHr9LCJyM8CZcv.pth

```

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')
true_labels_10percent = d_test.labels[:len(pred_1)]
model.plot_confusion_matrix(true_labels_10percent, pred_1)
```

Тестирование: 100% 450/450 [00:05<00:00, 88.38it/s]
metrics for 10% of test:
accuracy 0.9956:
balanced accuracy 0.9956:
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524: UserWarning: y_pred contains classes not in y_true
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')
    true_labels_10percent = d_test.labels[:len(pred_2)]
    model.plot_confusion_matrix(true_labels_10percent, pred_2)
```

Тестирование: 100%

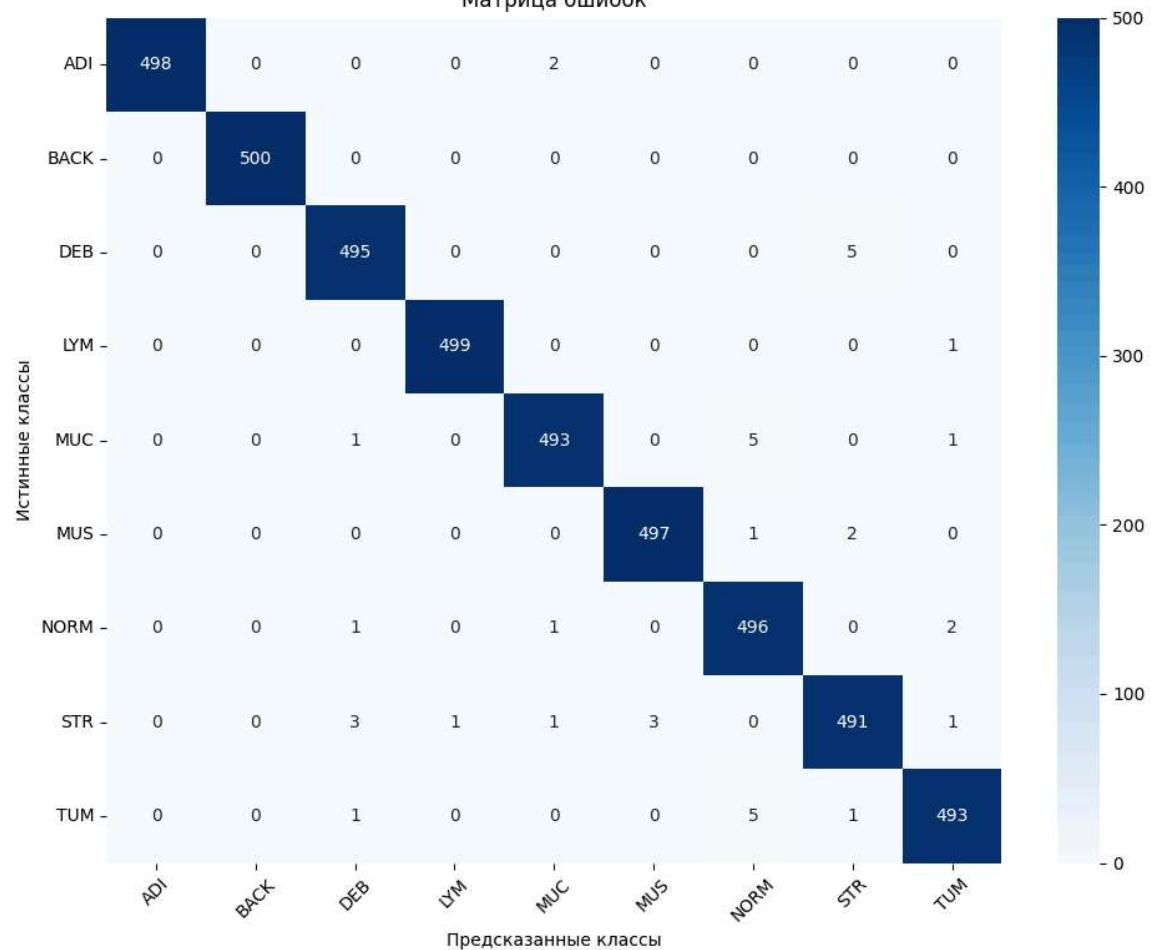
4500/4500 [00:59<00:00, 87.38it/s]

metrics for test:

accuracy 0.9916:

balanced accuracy 0.9916:

Матрица ошибок



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 экспортовать 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')

Downloading...
From: https://drive.google.com/uc?id=1DplzCeQ0-PKw8xS-L1aHr9LCJyM8CZcv
To: /content/best.pth
100%|██████████| 16.4M/16.4M [00:00<00:00, 206MB/s]
Модель загружена из https://drive.google.com/uc?id=1DplzCeQ0-PKw8xS-L1aHr9LCJyM8CZcv.pth
Downloading...
From: https://drive.google.com/uc?export=download&id=1eAyZ9oUOeg\_E887gNONaK2x81EGIVTXK
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 149MB/s]Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
```

Тестирование: 100%

90/90 [00:01<00:00, 61.57it/s]

metrics for test-tiny:

accuracy 0.9889:

balanced accuracy 0.9889:

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 calculated {n_runs} times in {timeit.timeit(f, number=n_runs)}s.')
Function f is calculated 128 times in 0.051963259000046946s.
```

▼ 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.metrics import ConfusionMatrixDisplay
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(f'Training: {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)))

# Матрица ошибок + визуализация в новом API
cm = metrics.confusion_matrix(y_test, predicted)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                               display_labels=classifier.classes_)

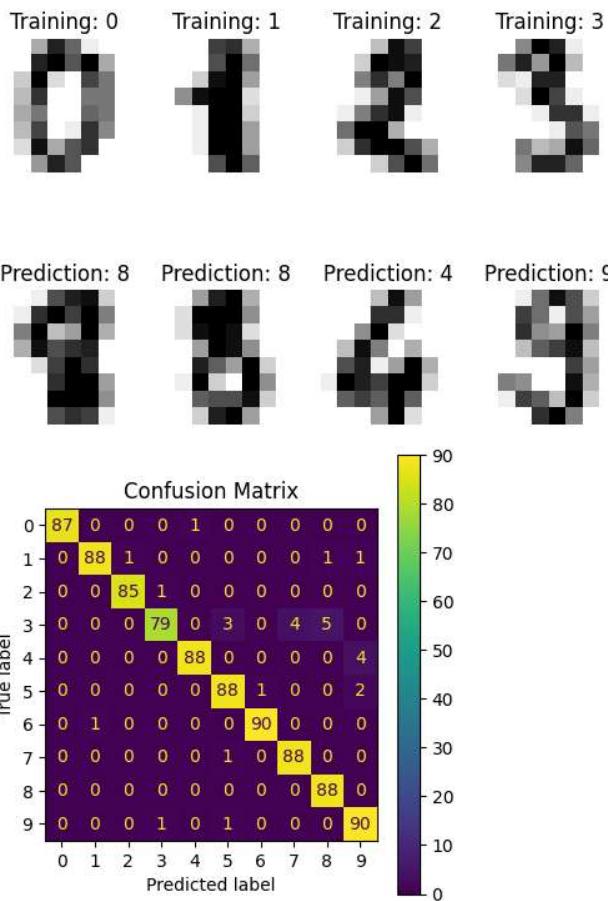
fig, ax = plt.subplots(figsize=(4, 4))
disp.plot(ax=ax)
ax.set_title("Confusion Matrix")
plt.tight_layout()
plt.show()

Classification report for classifier SVC(gamma=0.001):
          precision    recall  f1-score   support

           0       1.00    0.99    0.99     88
           1       0.99    0.97    0.98     91
           2       0.99    0.99    0.99     86
           3       0.98    0.87    0.92     91
           4       0.99    0.96    0.97     92
           5       0.95    0.97    0.96     91
           6       0.99    0.99    0.99     91
           7       0.96    0.99    0.97     89
           8       0.94    1.00    0.97     88
           9       0.93    0.98    0.95     92

    accuracy                           0.97      899
   macro avg       0.97    0.97    0.97      899
weighted avg       0.97    0.97    0.97      899

```



▼ 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)

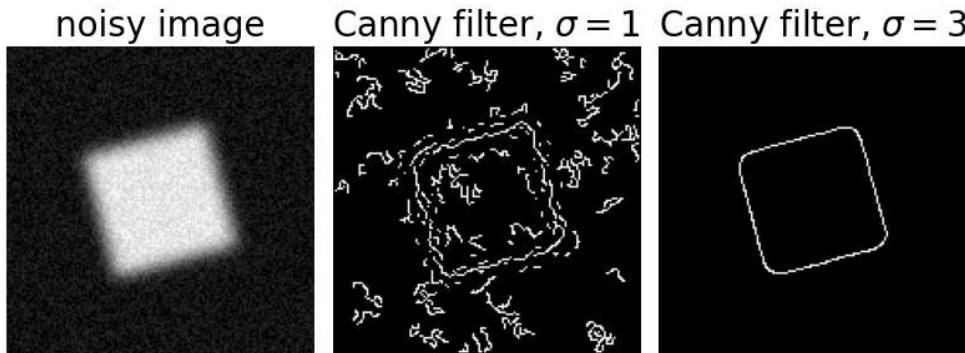
ax2.imshow(edges1, cmap=plt.cm.gray)
ax2.axis('off')
ax2.set_title(r'Canny filter, $\sigma=1$', fontsize=20)

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 нейронной 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')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test, verbose=2)

```

```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ====== 0s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to the constructor of `Flatten`. This argument is only used by the internal `Flatten` layer which is added to the model. If you want to flatten the input, use the `Flatten` layer directly. See: https://keras.io/api/layers/reshaping\_layers/flatten/
super().__init__(**kwargs)
Epoch 1/5
1875/1875 7s 2ms/step - accuracy: 0.8566 - loss: 0.4947
Epoch 2/5
1875/1875 5s 3ms/step - accuracy: 0.9551 - loss: 0.1529
Epoch 3/5
1875/1875 4s 2ms/step - accuracy: 0.9679 - loss: 0.1064
Epoch 4/5
1875/1875 4s 2ms/step - accuracy: 0.9739 - loss: 0.0850
Epoch 5/5
1875/1875 5s 3ms/step - accuracy: 0.9784 - loss: 0.0702
313/313 - 2s - 5ms/step - accuracy: 0.9777 - loss: 0.0748
[0.07479573041200638, 0.9776999950408936]

```

Для эффективной работы с моделями глубокого обучения убедитесь в том, что в текущей среде Google Colab используется аппаратный ускоритель GPU или TPU. Для смены среды выберите "среда выполнения" -> "сменить среду выполнения".

Большое количество туториалов и примеров с кодом на Tensorflow 2 можно найти на официальном сайте <https://www.tensorflow.org/tutorials?hl=ru>.

Также, Вам может понадобиться написать собственный генератор данных для Tensorflow 2. Скорее всего он будет достаточно простым, и его легко можно будет реализовать, используя официальную документацию TensorFlow 2. Но, на всякий случай (если не удалось сразу разобраться или хочется вникнуть в тему более глубоко), можете посмотреть следующий отличный туториал: <https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly>.

▼ PyTorch

```

# =====
# Дополнительно: мини-демо PyTorch
# =====
# Ранний выход, если PyTorch/обёртка недоступны
try:
    import torch
    import torch.nn as nn
    import torch.nn.functional as F
    from torch.utils.data import DataLoader
except ImportError:
    print("PyTorch не установлен – демо пропущено.")
    import sys
    raise SystemExit

if "HistologyTorchDataset" not in globals() or HistologyTorchDataset is None:
    print("HistologyTorchDataset недоступна – демо пропущено.")
    import sys
    raise SystemExit

# --- Данные: tiny-наборы, чтобы выполнялось быстро ---
base_train = Dataset('train_tiny')
base_test = Dataset('test_tiny')

train_ds = HistologyTorchDataset(base_train) # ToTensor по умолчанию
test_ds = HistologyTorchDataset(base_test)

train_loader = DataLoader(train_ds, batch_size=16, shuffle=True)
test_loader = DataLoader(test_ds, batch_size=32, shuffle=False)

# --- Мини-модель: двухслойный CNN + один FC (демонстрация, не решение) ---
class TinyCNN(nn.Module):
    def __init__(self, num_classes=len(TISSUE_CLASSES)):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 8, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(8, 16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2) # 224->112->56
        self.fc = nn.Linear(16 * 56 * 56, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x))) # [B, 3, 224, 224] -> [B, 8, 112, 112]
        x = self.pool(F.relu(self.conv2(x))) # [B, 8, 112, 112] -> [B, 16, 56, 56]
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)

model = TinyCNN(num_classes=len(TISSUE_CLASSES)).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# --- Один учебный шаг "обучения" на одном батче ---

```

```

model.train()
xb, yb = next(iter(train_loader))
xb = xb.to(device)
yb = yb.to(device, dtype=torch.long)

optimizer.zero_grad()
logits = model(xb)
loss = criterion(logits, yb)
float_loss = float(loss.detach().cpu())
loss.backward()
optimizer.step()

print(f"Loss на одном батче train_tiny: {float_loss:.4f}")

# --- Быстрая проверка на одном батче теста (для формы вывода/метрик) ---
model.eval()
with torch.no_grad():
    xt, yt = next(iter(test_loader))
    xt = xt.to(device)
    logits_t = model(xt).cpu()
    y_pred = logits_t.argmax(dim=1).numpy()
    y_true = yt.numpy()

print("Размерности:", {"y_true": y_true.shape, "y_pred": y_pred.shape})
Metrics.print_all(y_true, y_pred, "_") # balanced accuracy/accuracy на одном батче для демонстрации

# для полноценного решения требуется собственный тренировочный цикл по эпохам,
# # аугментации/нормализация, сохранение/загрузка весов, и тестирование на всём наборе.

```

```

Downloading...
From (original): https://drive.google.com/uc?export=download&id=17V3ly7zCNSK819GLhqv1ENDgbFowJXg0
From (redirected): https://drive.google.com/uc?export=download&id=17V3ly7zCNSK819GLhqv1ENDgbFowJXg0&confirm=t&uuid=9ef59ab0-3b65-47cb-
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 254MB/s]
Loading dataset train_tiny from npz.
Done. Dataset train_tiny consists of 900 images.
Downloading...
From: https://drive.google.com/uc?export=download&id=1eAyZ9oUOeg\_E887gNONaK2x81EGIVTXK
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 368MB/s]
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
Device: cuda
Loss на одном батче train_tiny: 2.1851
Размерности: {'y_true': (32,), 'y_pred': (32,)}
metrics for _:
    accuracy 0.2812:
    balanced accuracy 0.2250:
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524: UserWarning: y_pred contains classes not in y_true
  warnings.warn("y_pred contains classes not in y_true")

```

Дополнительные ресурсы по PyTorch

- Официальные туториалы PyTorch — <https://pytorch.org/tutorials/>
- “Deep Learning with PyTorch: 60-Minute Blitz” — https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- Transfer Learning for Computer Vision — https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
- PyTorch Get Started (установка) — <https://pytorch.org/get-started/locally/>
- Dive into Deep Learning (D2L, глава PyTorch) — https://d2l.ai/chapter_preliminaries/index.html
- Fast.ai — Practical Deep Learning for Coders — <https://course.fast.ai/>
- Learn PyTorch.io (Zero to Mastery) — <https://www.learnpytorch.io/>

Numba

В некоторых ситуациях, при ручных реализациях графовых алгоритмов, выполнение многократных вложенных циклов for в python можно существенно ускорить, используя JIT-компилятор Numba (<https://numba.pydata.org/>). Примеры использования Numba в Google Colab можно найти тут:

- [1. https://colab.research.google.com/github/cbernet/maldives/blob/master/numba/numba_cuda.ipynb](https://colab.research.google.com/github/cbernet/maldives/blob/master/numba/numba_cuda.ipynb)
- [2. https://colab.research.google.com/github/evaneschneider/parallel-programming/blob/master/COMPASS_gpu_intro.ipynb](https://colab.research.google.com/github/evaneschneider/parallel-programming/blob/master/COMPASS_gpu_intro.ipynb)

Пожалуйста, если Вы решили использовать Numba для решения этого практического задания, еще раз подумайте, нужно ли это Вам, и есть ли возможность реализовать требуемую функциональность иным способом. Используйте Numba только при реальной необходимости.

▼ Работа с zip архивами в Google Drive

Запаковка и распаковка zip архивов может пригодиться при сохранении и загрузки Вашей модели. Ниже приведен фрагмент кода, иллюстрирующий помещение нескольких файлов в zip архив с последующим чтением файлов из него. Все действия с директориями, файлами и архивами должны осуществляться с примонтированным Google Drive.

Создадим 2 изображения, поместим их в директорию tmp внутри PROJECT_DIR, запакуем директорию tmp в архив tmp.zip.

```
PROJECT_DIR = "/dev/prak_nn_1/"
arr1 = np.random.rand(100, 100, 3) * 255
arr2 = np.random.rand(100, 100, 3) * 255

img1 = Image.fromarray(arr1.astype('uint8'))
img2 = Image.fromarray(arr2.astype('uint8'))

p = "/content/drive/MyDrive/" + PROJECT_DIR

if not (Path(p) / 'tmp').exists():
    (Path(p) / 'tmp').mkdir()

img1.save(str(Path(p) / 'tmp' / 'img1.png'))
img2.save(str(Path(p) / 'tmp' / 'img2.png'))

%cd $p
!zip -r "tmp.zip" "tmp"

-----
FileNotFoundError                         Traceback (most recent call last)
/tmp/ipython-input-1091392201.py in <cell line: 0>()
      9
     10 if not (Path(p) / 'tmp').exists():
--> 11     (Path(p) / 'tmp').mkdir()
     12
     13 img1.save(str(Path(p) / 'tmp' / 'img1.png'))

/usr/lib/python3.12/pathlib.py in mkdir(self, mode, parents, exist_ok)
1309
1310     """
--> 1311     try:
1312         os.mkdir(self, mode)
1313     except FileNotFoundError:
1314         if not parents or self.parent == self:

FileNotFoundError: [Errno 2] No such file or directory: '/content/drive/MyDrive/dev/prak_nn_1/tmp'
```