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/python3.12/di
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/pyth
Requirement already satisfied: filelock in /usr/local/lib/python3.12
Requirement already satisfied: requests[socks] in /usr/local/lib/pyt
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dis
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/pytho
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/loca
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/loca
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/
```

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'
DATASETS_LINKS = {
    'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-cLi',
    'train_small': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui',
    'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr',
    'test_small': '1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ2lI',
    'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}
```

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
        url = f"https://drive.google.com/uc?export=download&confirm=
        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} imag

    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
        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 t
        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 использвания 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.

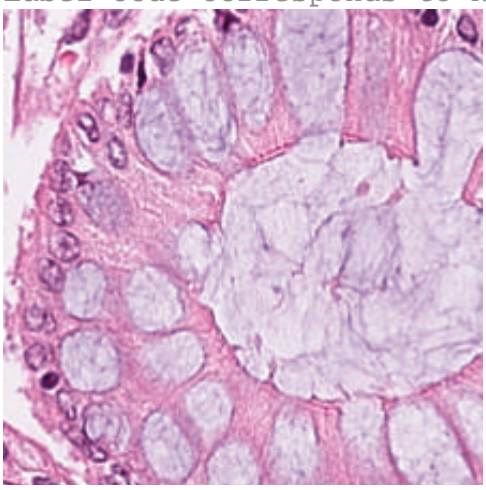
```
d_train_tiny = Dataset('train_tiny')

img, lbl = d_train_tiny.random_image_with_label()
print()
print(f'Got numpy array of shape {img.shape}, and label with code {l
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=1I
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 130MB/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 6.
Label code corresponds to NORM 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 PyT
# Он показывает, как "подключить" наш Dataset к torch.utils.data.Da

try:
    import torch
    from torch.utils.data import Dataset as TorchDataset, DataLoade
    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_sm
        transform:    функция/объект, преобразующий изображение (PI

        """
        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
            img = Image.fromarray(img)                    # в PIL.I
            img = self.transform(img)                     # в torch
            return img, label

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

Пример использвания класса HistologyTorchDataset

```
if "HistologyTorchDataset" not in globals() or HistologyTorchDataset
    print("PyTorch не установлен или обёртка недоступна — пример про
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))  #
    print("Форма батча меток:", tuple(labels_batch.shape))        #
    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: https://drive.google.com/uc?export=download&confirm=pbef&id=1I
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 130MB/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: [7, 3, 4, 0, 1, 5, 6, 7]
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
        return sum(int(i[0] == i[1]) for i in zip(gt, pred)) / len(g

    @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
```

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.

```
class Model:

    def __init__(self):
        import torch
        from torchvision import models
        import torchvision.transforms as T

        self.tissue_classes = TISSUE_CLASSES
        self.num_classes = len(self.tissue_classes)

        self.device = torch.device("cuda" if torch.cuda.is_availabl
        print(f"Using device: {self.device}")
```

```
            try:
                weights = models.EfficientNet_B0_Weights.IMAGENET1K_V1
                backbone = models.efficientnet_b0(weights=weights)
            except AttributeError:
                backbone = models.efficientnet_b0(pretrained=True)

            in_features = backbone.classifier[1].in_features
            backbone.classifier[1] = torch.nn.Linear(in_features, self.

            for param in backbone.parameters():
                param.requires_grad = False
            for param in backbone.classifier.parameters():
                param.requires_grad = True

            self.model = backbone.to(self.device)

            self.criterion = torch.nn.CrossEntropyLoss()
            self.optimizer = torch.optim.Adam(
                filter(lambda p: p.requires_grad, self.model.parameters
                lr=1e-3
            )

            imagenet_mean = [0.485, 0.456, 0.406]
            imagenet_std = [0.229, 0.224, 0.225]

            import torchvision.transforms as T
            self.train_transform = T.Compose([
                T.RandomHorizontalFlip(),
                T.RandomVerticalFlip(),
                T.RandomRotation(20),
                T.ColorJitter(brightness=0.2, contrast=0.2, saturation=
                T.ToTensor(),
                T.Normalize(mean=imagenet_mean, std=imagenet_std),
            ])

            self.eval_transform = T.Compose([
                T.ToTensor(),
                T.Normalize(mean=imagenet_mean, std=imagenet_std),
            ])

            self.best_state_dict = None
            self.history = []


        def save(self, name: str):
            import os
            import torch
```

```
        save_path = f'/content/drive/MyDrive/{name}.pth'
        os.makedirs(os.path.dirname(save_path), exist_ok=True)

        state = {
            'state_dict': self.model.state_dict(),
            'num_classes': self.num_classes,
        }
        torch.save(state, save_path)
        print(f'Model saved to {save_path}')


    def load(self, name: str):
        import os
        import torch

        local_path = f'/content/drive/MyDrive/{name}.pth'

        if not os.path.exists(local_path):
            name_to_id_dict = {
                'best': '1I-hzoTR5HrtQV3-MmWtrPUNo8eRkI8tv'
            }
            if name in name_to_id_dict:
                file_id = name_to_id_dict[name]
                url = f'https://drive.google.com/uc?id={file_id}'
                print(f'Local file not found, downloading from {url
                gdown.download(url, local_path, quiet=False)
            else:
                raise FileNotFoundError(
                    f'Файл {local_path} не найден и ID для имени "{
                    f'в словаре name_to_id_dict не задан.'
                )

        checkpoint = torch.load(local_path, map_location=self.devic
        state_dict = checkpoint.get('state_dict', checkpoint)
        self.model.load_state_dict(state_dict)
        self.model.to(self.device)
        self.model.eval()
        print(f'Model loaded from {local_path}')


    def train(self, dataset: Dataset, epochs: int = 5, batch_size:
        import numpy as np
        import torch
        from torch.utils.data import Dataset as TorchDataset, DataL
        from PIL import Image

        class NumpyHistologyTorchDataset(TorchDataset):
            def __init__(self, base_dataset: Dataset, indices, tran
```

```
                self.base = base_dataset
                self.indices = indices
                self.transform = transform

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

            def __getitem__(self, idx):
                i = self.indices[idx]
                img = self.base.image(i)
                label = int(self.base.labels[i])
                img = Image.fromarray(img)
                if self.transform is not None:
                    img = self.transform(img)
                return img, label

        n_samples = dataset.n_files
        indices = np.arange(n_samples)
        np.random.shuffle(indices)

        n_val = int(n_samples * val_split)
        val_indices = indices[:n_val]
        train_indices = indices[n_val:]

        train_ds = NumpyHistologyTorchDataset(dataset, train_indice
        val_ds = NumpyHistologyTorchDataset(dataset, val_indices, s

        from torch.utils.data import DataLoader
        train_loader = DataLoader(train_ds, batch_size=batch_size,
                                  shuffle=True, num_workers=2, pin_
        val_loader = DataLoader(val_ds, batch_size=batch_size,
                                shuffle=False, num_workers=2, pin_m

        best_val_acc = 0.0
        self.history = []

        for epoch in range(1, epochs + 1):
            print(f'\nEpoch {epoch}/{epochs}')
            #train
            self.model.train()
            running_loss = 0.0
            running_correct = 0
            running_total = 0

            for x, y in tqdm(train_loader, desc='Train'):
                x = x.to(self.device, non_blocking=True)
                y = y.to(self.device, non_blocking=True)
```

```
                self.optimizer.zero_grad()
                logits = self.model(x)
                loss = self.criterion(logits, y)
                loss.backward()
                self.optimizer.step()

                running_loss += loss.item() * x.size(0)
                preds = torch.argmax(logits, dim=1)
                running_correct += (preds == y).sum().item()
                running_total += x.size(0)

            train_loss = running_loss / running_total
            train_acc = running_correct / running_total

            #val
            self.model.eval()
            val_correct = 0
            val_total = 0

            with torch.no_grad():
                for x, y in tqdm(val_loader, desc='Val'):
                    x = x.to(self.device, non_blocking=True)
                    y = y.to(self.device, non_blocking=True)

                    logits = self.model(x)
                    preds = torch.argmax(logits, dim=1)
                    val_correct += (preds == y).sum().item()
                    val_total += x.size(0)

            val_acc = val_correct / val_total if val_total > 0 else

            print(f'Epoch {epoch}: train_loss={train_loss:.4f}, '
                  f'train_acc={train_acc:.4f}, val_acc={val_acc:.4f

            if val_acc > best_val_acc:
                best_val_acc = val_acc
                self.best_state_dict = {k: v.cpu().clone()
                                        for k, v in self.model.stat
                print(f'New best val_acc={best_val_acc:.4f}')

        if self.best_state_dict is not None:
            self.model.load_state_dict(self.best_state_dict)
            self.model.to(self.device)
            self.model.eval()
            print(f'Loaded best model with val_acc={best_val_acc:.4

        print('Training finished.')
```

```
    def finetune(self, dataset: Dataset, epochs: int = 3, batch_siz
                 val_split: float = 0.1, lr: float = 1e-4):
        import torch
        print(f"\nStarting fine-tuning for {epochs} epochs with lr=

        for param in self.model.parameters():
            param.requires_grad = True

        self.optimizer = torch.optim.AdamW(
            self.model.parameters(),
            lr=lr,
            weight_decay=1e-4
        )

        try:
            self.criterion = torch.nn.CrossEntropyLoss(label_smooth
        except TypeError:
            self.criterion = torch.nn.CrossEntropyLoss()

        self.train(dataset, epochs=epochs, batch_size=batch_size, v

        print("Fine-tuning finished.")


    def test_on_dataset(self, dataset: Dataset, limit=None):
        predictions = []
        n = dataset.n_files if not limit else int(dataset.n_files *
        for img in tqdm(dataset.images_seq(n), total=n):
            predictions.append(self.test_on_image(img))
        return predictions


    def test_on_image(self, img: np.ndarray):
        import torch
        from PIL import Image

        self.model.eval()
        img_pil = Image.fromarray(img.astype('uint8'))
        x = self.eval_transform(img_pil).unsqueeze(0).to(self.devic

        with torch.no_grad():
            logits = self.model(x)
            prediction = int(torch.argmax(logits, dim=1).item())
        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 = Dataset('train')
d_test = Dataset('test')
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1X
To: /content/train.npz
100%|██████████| 2.10G/2.10G [00:28<00:00, 72.7MB/s]
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1F
To: /content/test.npz
100%|██████████| 525M/525M [00:06<00:00, 79.3MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
```

```
model = Model()
if not EVALUATE_ONLY:
    # Stage 1
    model.train(d_train)
    model.save('best_stage1')
    # Stage 2 (fine-tuning всей сети)
    model.finetune(d_train, epochs=3, lr=1e-4)
    model.save('best')

else:
    #todo: your link goes here
    model.load('best')
```

```
Using device: cuda

Epoch 1/5
Train: 100%                                   507/507 [01:42<00:00,  6.16it/s]
Val: 100%                                     57/57 [00:02<00:00, 21.87it/s]
Epoch 1: train_loss=0.5639, train_acc=0.8412, val_acc=0.9400
New best val_acc=0.9400
```

```
Epoch 2/5
Train: 100% 507/507 [01:41<00:00, 6.63it/s]
Val: 100% 57/57 [00:02<00:00, 23.03it/s]
Epoch 2: train_loss=0.3273, train_acc=0.8975, val_acc=0.9439
New best val_acc=0.9439

Epoch 3/5
Train: 100% 507/507 [01:40<00:00, 5.42it/s]
Val: 100% 57/57 [00:04<00:00, 14.06it/s]
Epoch 3: train_loss=0.3045, train_acc=0.9003, val_acc=0.9483
New best val_acc=0.9483

Epoch 4/5
Train: 100% 507/507 [01:40<00:00, 5.76it/s]
Val: 100% 57/57 [00:02<00:00, 20.75it/s]
Epoch 4: train_loss=0.2742, train_acc=0.9099, val_acc=0.9500
New best val_acc=0.9500

Epoch 5/5
Train: 100% 507/507 [01:37<00:00, 6.18it/s]
Val: 100% 57/57 [00:02<00:00, 20.31it/s]
Epoch 5: train_loss=0.2815, train_acc=0.9077, val_acc=0.9489
Loaded best model with val_acc=0.9500
Training finished.
Model saved to /content/drive/MyDrive/best_stage1.pth

Starting fine-tuning for 3 epochs with lr=0.0001...

Epoch 1/3
Train: 100% 507/507 [01:56<00:00, 5.51it/s]
Val: 100% 57/57 [00:02<00:00, 21.43it/s]
Epoch 1: train_loss=0.7424, train_acc=0.9307, val_acc=0.9750
New best val_acc=0.9750

Epoch 2/3
Train: 100% 507/507 [01:58<00:00, 5.55it/s]
Val: 100% 57/57 [00:04<00:00, 15.55it/s]
Epoch 2: train_loss=0.6261, train_acc=0.9657, val_acc=0.9861
New best val_acc=0.9861

Epoch 3/3
Train: 100% 507/507 [01:55<00:00, 5.25it/s]
```

```
Val: 100%                                                 57/57 [00:02<00:00, 21.58it/s]
Epoch 3: train_loss=0.5957, train_acc=0.9744, val_acc=0.9889
New best val_acc=0.9889
Loaded best model with val_acc=0.9889
Training finished.
Fine-tuning finished.
Model saved to /content/drive/MyDrive/best.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
```

```
100%                                                 450/450 [00:05<00:00, 98.98it/s]
metrics for 10% of test:
         accuracy 0.9933:
         balanced accuracy 0.9933:
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classificat
  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%                                                 4500/4500 [00:50<00:00, 97.43it/s]
metrics for test:
         accuracy 0.9847:
         balanced accuracy 0.9847:
```

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')
```

```
Using device: cuda
Model loaded from /content/drive/MyDrive/best.pth
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1v
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 43.7MB/s]
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
100%                                                 90/90 [00:01<00:00, 87.89it/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 caluclated {n_runs} times in {timeit.timeit(f
```

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 digit
# have a look at the first 4 images, stored in the `images` attribu
# dataset.  If we were working from image files, we could load them
# matplotlib.pyplot.imread.  Note that each image must have the sam
# images, we know which digit they represent: it is given in the 't
# 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,
# 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:], p
for ax, (image, prediction) in zip(axes[1, :], images_and_predictio
    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, predicte

# 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()
```

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')
])

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)
```

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 удлось 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
    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 п
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 реше
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] ->
        x = self.pool(F.relu(self.conv2(x)))  # [B, 8, 112, 112] ->
        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/метри
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.sha
Metrics.print_all(y_true, y_pred, "_")  # balanced accuracy/accurac

# для полноценного решения требуется собственный тренировочный цикл
# аугментации/нормализация, сохранение/загрузка весов, и тестирован
```

Downloading

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1I
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 239MB/s]
Loading dataset train_tiny from npz.
Done. Dataset train_tiny consists of 900 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1v
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 230MB/s]
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
Device: cuda
Loss на одном батче train_tiny: 2.2130
Размерности: {'y_true': (32,), 'y_pred': (32,)}
metrics for _:
        accuracy 0.3125:
        balanced accuracy 0.2500:
```

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
2. 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 осущетвляться 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"
```

Распакуем архив tmp.zip в директорию tmp2 в PROJECT_DIR. Теперь внутри директории tmp2 содержится директория tmp, внутри которой находятся 2 изображения.

```
p = "/content/drive/MyDrive/" + PROJECT_DIR
%cd $p
!unzip -uq "tmp.zip" -d "tmp2"
```