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/dist-packages (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dist-packages (from gdown) (4.13.5)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from gdown) (3.20.0)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.12/dist-packages (from gdown) (2.32.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from gdown) (4.67.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4->gdown) (2.8)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4->gdo
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gd
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown) (
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdow
```

Монтирование Вашего Google Drive к текущему окружению:

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

```
Mounted at /content/drive
```

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

```
EVALUATE_ONLY = False
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '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
import copy

# PyTorch related imports
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.models as models
import torchvision.transforms as T
```

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]
```

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.

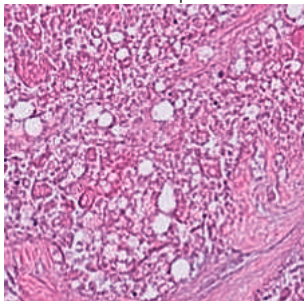
```
d_train_tiny = Dataset('train_tiny')

img, lbl = d_train_tiny.random_image_with_label()
print()
print(f'Got numpy array of shape {img.shape}, and label with code {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=1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:02<00:00, 35.8MB/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 использвания 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: https://drive.google.com/uc?export=download&confirm=pbef&id=1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui
```

```
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 142MB/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, 8, 2, 3, 8, 5, 7, 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 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.

```
class Model:
    def __init__(self, num_classes=None):
        if num_classes is None:
```

```
            num_classes = len(TISSUE_CLASSES)
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        print(f"Using device: {self.device}")

        self.model = models.alexnet(weights=models.AlexNet_Weights.DEFAULT)
        # Modify the classifier to fit the number of classes in our dataset
        # The last layer (index 6) is a Linear layer (fc) that outputs 1000 classes by default
        num_ftrs = self.model.classifier[6].in_features
        self.model.classifier[6] = nn.Linear(num_ftrs, num_classes)
        self.model = self.model.to(self.device)
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.model.parameters(), lr=0.0001)

        self.transform = T.Compose([
            T.Resize(256),
            T.CenterCrop(224),
            T.ToTensor(),
            T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # ImageNet normalization
        ])

    def save(self, name: str):
        save_path = Path(f'/content/drive/MyDrive/{name}.pth')
        torch.save({
            'model_state_dict': self.model.state_dict(),
            'optimizer_state_dict': self.optimizer.state_dict(),
        }, save_path)
        print(f'Model saved to {save_path}')

    def load(self, name: str):
        # Load model state_dict and optimizer state_dict
        # If EVALUATE_ONLY is True, try to download from a specific GDrive ID
        if EVALUATE_ONLY:
            gdrive_id = '1j73eUxnl8dlwenWNGYLtPardVrrV-JD7'
            output_file = f'/content/{name}.pth'
            print(f"Downloading model '{name}' from Google Drive ID: {gdrive_id}")
            try:
                gdown.download(f'https://drive.google.com/uc?id={gdrive_id}', output_file, quiet=False)
                load_path = Path(output_file)
            except Exception as e:
                print(f"Error downloading model: {e}")
                print("Please ensure the Google Drive ID is correct and the file is publicly accessible.")
                return
        else:
            load_path = Path(f'/content/drive/MyDrive/{name}.pth')

        if not load_path.exists():
            print(f"Error: Model file not found at {load_path}")
            return

        checkpoint = torch.load(load_path, map_location=self.device)
        self.model.load_state_dict(checkpoint['model_state_dict'])
        self.optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        self.model.eval()
        print(f'Model loaded from {load_path}')

    def train(self, train_dataset: Dataset, val_dataset: Dataset = None, epochs=10, batch_size=128):
        print(f'training started')
        train_ds = HistologyTorchDataset(train_dataset, transform=self.transform)
        train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True)
        # If validation dataset is provided, prepare its DataLoader
        val_loader = None
        if val_dataset:
            val_ds = HistologyTorchDataset(val_dataset, transform=self.transform)
            val_loader = DataLoader(val_ds, batch_size=batch_size, shuffle=False)

        best_val_loss = float('inf')
        best_model_state = None

        for epoch in range(epochs):
            # Training phase
            self.model.train()
            running_loss = 0.0
            for images, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs} [Train]"):
                images = images.to(self.device)
                labels = labels.to(self.device, dtype=torch.long)

                self.optimizer.zero_grad()
                outputs = self.model(images)
                loss = self.criterion(outputs, labels)
                loss.backward()
                self.optimizer.step()
```

```
                running_loss += loss.item() * images.size(0)

            epoch_train_loss = running_loss / len(train_loader.dataset)
            print(f'Epoch {epoch+1} Train Loss: {epoch_train_loss:.4f}')
            # Validation phase
            if val_loader:
                self.model.eval()
                val_running_loss = 0.0
                with torch.no_grad():
                    for images, labels in tqdm(val_loader, desc=f"Epoch {epoch+1}/{epochs} [Val]"):
                        images = images.to(self.device)
                        labels = labels.to(self.device, dtype=torch.long)

                        outputs = self.model(images)
                        loss = self.criterion(outputs, labels)
                        val_running_loss += loss.item() * images.size(0)

                epoch_val_loss = val_running_loss / len(val_loader.dataset)
                print(f'Epoch {epoch+1} Val Loss: {epoch_val_loss:.4f}')
                # Save best model
                if epoch_val_loss < best_val_loss:
                    best_val_loss = epoch_val_loss
                    best_model_state = copy.deepcopy(self.model.state_dict())
                    print(f'New best validation loss: {best_val_loss:.4f}. Saving model state.')

        print(f'training done')
        # Restore best model state after training
        if best_model_state:
            self.model.load_state_dict(best_model_state)
            print(f'Model restored to best state with validation loss: {best_val_loss:.4f}')

    def test_on_dataset(self, dataset: Dataset, limit=None):
        self.model.eval() # Set model to evaluation mode
        predictions = []
        true_labels = []

        num_samples_to_test = dataset.n_files if not limit else int(dataset.n_files * limit)

        if limit is None:
            test_ds = HistologyTorchDataset(dataset, transform=self.transform)
            test_loader = DataLoader(test_ds, batch_size=64, shuffle=False)

            with torch.no_grad():
                for images, labels in tqdm(test_loader, desc="Testing"):
                    images = images.to(self.device)
                    outputs = self.model(images)
                    _, predicted = torch.max(outputs.data, 1)
                    predictions.extend(predicted.cpu().numpy())
                    true_labels.extend(labels.cpu().numpy())
        else:
            # Iterate through a subset of the images using the existing image_with_label method
            with torch.no_grad():
                for i in tqdm(range(num_samples_to_test), desc="Testing partial dataset"):
                    img_np, label_np = dataset.image_with_label(i)
                    img_pil = Image.fromarray(img_np)
                    img_tensor = self.transform(img_pil).unsqueeze(0).to(self.device) # Add batch dimension

                    output = self.model(img_tensor)
                    _, predicted = torch.max(output.data, 1)
                    predictions.append(predicted.item())
                    true_labels.append(label_np)
        return predictions

    def test_on_image(self, img_np: np.ndarray):
        self.model.eval()
        with torch.no_grad():
            img_pil = Image.fromarray(img_np)
            # Apply the same transformations used during training
            img_tensor = self.transform(img_pil).unsqueeze(0).to(self.device) # Add batch dimension
            output = self.model(img_tensor)
            _, predicted = torch.max(output.data, 1)
            return predicted.item()
```

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_small')
d_test = Dataset('test_small')
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR
To: /content/train_small.npz

  0%|          | 0.00/841M [00:00<?, ?B/s]
  1%|          | 4.72M/841M [00:00<00:22, 37.4MB/s]
  4%|▌         | 29.9M/841M [00:00<00:06, 129MB/s]
  5%|▌         | 43.0M/841M [00:00<00:07, 111MB/s]
  8%|▊         | 65.5M/841M [00:00<00:05, 146MB/s]
 11%|█         | 94.9M/841M [00:00<00:03, 192MB/s]
 14%|█▍        | 115M/841M [00:00<00:05, 137MB/s]
 16%|█▌        | 135M/841M [00:00<00:05, 138MB/s]
 18%|█▊        | 150M/841M [00:01<00:04, 140MB/s]
 20%|██        | 169M/841M [00:01<00:04, 151MB/s]
 23%|██▎       | 190M/841M [00:01<00:03, 163MB/s]
 25%|██▍       | 210M/841M [00:01<00:03, 174MB/s]
 27%|██▋       | 229M/841M [00:04<00:33, 18.3MB/s]
 30%|██▉       | 249M/841M [00:04<00:23, 25.4MB/s]
 31%|███       | 265M/841M [00:04<00:17, 32.6MB/s]
 33%|███▎      | 280M/841M [00:05<00:15, 36.7MB/s]
 35%|███▍      | 293M/841M [00:05<00:13, 39.7MB/s]
 36%|███▌      | 305M/841M [00:05<00:11, 46.6MB/s]
 38%|███▊      | 316M/841M [00:05<00:09, 54.1MB/s]
 39%|███▉      | 327M/841M [00:05<00:10, 50.8MB/s]
 41%|████      | 341M/841M [00:05<00:07, 63.7MB/s]
 43%|████▎     | 360M/841M [00:06<00:05, 84.9MB/s]
 45%|████▍     | 377M/841M [00:06<00:04, 98.6MB/s]
 47%|████▋     | 398M/841M [00:06<00:03, 116MB/s]
 49%|████▉     | 413M/841M [00:06<00:03, 117MB/s]
 51%|█████     | 426M/841M [00:06<00:03, 104MB/s]
 54%|█████▍    | 452M/841M [00:06<00:02, 137MB/s]
 57%|█████▋    | 476M/841M [00:06<00:02, 161MB/s]
 59%|█████▉    | 494M/841M [00:06<00:02, 153MB/s]
 61%|██████    | 511M/841M [00:07<00:03, 86.1MB/s]
 64%|██████▍   | 534M/841M [00:08<00:06, 49.1MB/s]
 65%|██████▌   | 544M/841M [00:09<00:13, 22.2MB/s]
 67%|██████▋   | 565M/841M [00:09<00:08, 31.0MB/s]
 68%|██████▊   | 575M/841M [00:10<00:07, 36.0MB/s]
 70%|██████▉   | 585M/841M [00:10<00:06, 39.5MB/s]
 71%|███████   | 596M/841M [00:10<00:05, 46.5MB/s]
 72%|███████▏  | 605M/841M [00:10<00:04, 51.0MB/s]
 73%|███████▎  | 614M/841M [00:10<00:04, 54.8MB/s]
 75%|███████▍  | 627M/841M [00:10<00:03, 67.2MB/s]
 76%|███████▌  | 636M/841M [00:10<00:03, 65.1MB/s]
 79%|███████▉  | 664M/841M [00:10<00:01, 107MB/s]
 81%|████████  | 678M/841M [00:11<00:02, 73.9MB/s]
 83%|████████▎ | 698M/841M [00:11<00:01, 95.5MB/s]
 86%|████████▌ | 721M/841M [00:11<00:00, 121MB/s]
 88%|████████▊ | 738M/841M [00:11<00:00, 103MB/s]
 89%|████████▉ | 752M/841M [00:11<00:00, 94.0MB/s]
 91%|█████████ | 768M/841M [00:12<00:00, 100MB/s]
 93%|█████████▎| 781M/841M [00:12<00:00, 97.9MB/s]
 94%|█████████▍| 792M/841M [00:15<00:03, 14.0MB/s]
 96%|█████████▌| 803M/841M [00:15<00:02, 17.8MB/s]
 97%|█████████▋| 814M/841M [00:15<00:01, 22.8MB/s]
 98%|█████████▊| 824M/841M [00:15<00:00, 27.0MB/s]
100%|██████████| 841M/841M [00:15<00:00, 53.5MB/s]
Loading dataset train_small from npz.
```

```
model = Model()
if not EVALUATE_ONLY:
    model.train(d_train)
    model.save('best')
else:
    #todo: your link goes here
    model.load('best')
```

```
Using device: cuda
training started
Epoch 1/10 [Train]: 100%                                    57/57 [00:25<00:00,  2.68it/s]
Epoch 1 Train Loss: 0.6013
Epoch 2/10 [Train]: 100%                                    57/57 [00:25<00:00,  2.41it/s]
Epoch 2 Train Loss: 0.2120
Epoch 3/10 [Train]: 100%                                    57/57 [00:24<00:00,  2.52it/s]
Epoch 3 Train Loss: 0.1405
Epoch 4/10 [Train]: 100%                                    57/57 [00:25<00:00,  2.66it/s]
Epoch 4 Train Loss: 0.1164
Epoch 5/10 [Train]: 100%                                    57/57 [00:25<00:00,  3.03it/s]
Epoch 5 Train Loss: 0.0534
Epoch 6/10 [Train]: 100%                                    57/57 [00:25<00:00,  3.02it/s]
Epoch 6 Train Loss: 0.0367
Epoch 7/10 [Train]: 100%                                    57/57 [00:26<00:00,  2.56it/s]
Epoch 7 Train Loss: 0.0419
Epoch 8/10 [Train]: 100%                                    57/57 [00:25<00:00,  3.04it/s]
Epoch 8 Train Loss: 0.0821
Epoch 9/10 [Train]: 100%                                    57/57 [00:25<00:00,  3.06it/s]
Epoch 9 Train Loss: 0.0131
Epoch 10/10 [Train]: 100%                                   57/57 [00:25<00:00,  3.01it/s]
Epoch 10 Train Loss: 0.0099
training done
Model saved to /content/drive/MyDrive/best.pth
```

Пример тестирования модели на части набора данных:

```
pred_1 = model.test_on_dataset(d_test, limit=1)
Metrics.print_all(d_test.labels[:len(pred_1)], pred_1, 'partial test dataset')
```

```
Testing partial dataset: 100%                               1800/1800 [00:06<00:00, 274.17it/s]
metrics for partial test dataset:
     accuracy 0.9533:
     balanced accuracy 0.9533:
```

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')
```

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= Dataset('test')
pred = final_model.test_on_dataset(d_test)
Metrics.print_all(d_test.labels, pred, 'test')
```

```
Using device: cuda
Model loaded from /content/drive/MyDrive/best.pth
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr
To: /content/test.npz

  0%|          | 0.00/525M [00:00<?, ?B/s]
  4%|▍         | 23.1M/525M [00:00<00:02, 229MB/s]
  9%|▉         | 47.2M/525M [00:00<00:02, 235MB/s]
 14%|█▎        | 71.3M/525M [00:00<00:01, 236MB/s]
 18%|█▊        | 94.9M/525M [00:00<00:01, 218MB/s]
 23%|██▎       | 122M/525M [00:00<00:01, 233MB/s]
 28%|██▊       | 145M/525M [00:00<00:01, 229MB/s]
 32%|███▏      | 168M/525M [00:00<00:01, 222MB/s]
 36%|███▋      | 191M/525M [00:00<00:01, 182MB/s]
 40%|████      | 210M/525M [00:02<00:08, 39.1MB/s]
 45%|████▍     | 239M/525M [00:02<00:05, 57.2MB/s]
 50%|████▉     | 265M/525M [00:02<00:03, 76.4MB/s]
 55%|█████▍    | 290M/525M [00:02<00:02, 96.3MB/s]
 60%|█████▉    | 316M/525M [00:02<00:01, 120MB/s]
 65%|██████▍   | 341M/525M [00:02<00:01, 141MB/s]
 69%|██████▉   | 364M/525M [00:03<00:01, 157MB/s]
 74%|███████▎  | 391M/525M [00:03<00:00, 180MB/s]
 79%|███████▊  | 415M/525M [00:03<00:00, 193MB/s]
 84%|████████▎ | 441M/525M [00:03<00:00, 208MB/s]
 89%|████████▊ | 466M/525M [00:03<00:00, 217MB/s]
 94%|█████████▎| 492M/525M [00:03<00:00, 230MB/s]
100%|██████████| 525M/525M [00:03<00:00, 141MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
Testing: 100%                                                    71/71 [00:21<00:00,  3.52it/s]
metrics for test:
     accuracy 0.9524:
     balanced accuracy 0.9524:
```

Отмонтировать Google Drive.

```
drive.flush_and_unmount()
```

Напишите программный код или сгенерируйте его с помощью искусственного интеллекта.

Напишите программный код или сгенерируйте его с помощью искусственного интеллекта.

Напишите программный код или сгенерируйте его с помощью искусственного интеллекта.

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.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('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)))

# 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 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.
```

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 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"
```

Распакуем архив tmp.zip в директорию tmp2 в PROJECT_DIR. Теперь внутри директории tmp2 содержится директория tmp,

Не удается связаться с сервисом reCAPTCHA. Проверьте подключение к Интернету и перезагрузите страницу.