

▼ Практическое задание №1

Установка необходимых пакетов:

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

Requirement already satisfied: tqdm 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->gdown)
Requirement already satisfied: charset_normalizer<4,>=2.0 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown)
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) (2.1.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown) (2023.11.14)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown) (1.7.1)
```

Монтирование Вашего Google Drive к текущему окружению:

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

Mounted at /content/drive
```

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

```
#ссылки были изменены
EVALUATE_ONLY = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
NUM_CLASSES = len(TISSUE_CLASSES)
DATASETS_LINKS = {
    'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-cLi',
    'train_small': '1qd45XXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1S7goa2NrOzC2SkES-8U-q-dMDmCvrnTb',
    'test': '1RFPou3pFKpuHDJZ-D9XDFzgvwpUBF1Dr',
    'test_small': '1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ2lI',
    'test_tiny': '1wvh1bDYv4yRubmoYYT0xN487PS08hy16'
}
```

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

import urllib.request
```

▼ 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 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')

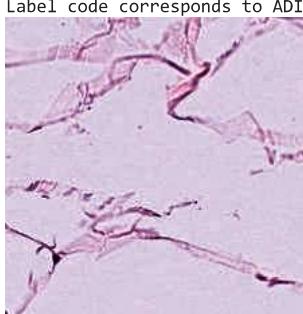
img, lbl = d_train_tiny.random_image_with_label()
print()
print(f'Got numpy array of shape {img.shape}, and label with code {lbl}.')
print(f'Label code corresponds to {TISSUE_CLASSES[lbl]} class.')

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

Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1I-2Z0uXLd40whZ0Qltp817Kn3J0Xgbui
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:01<00:00, 62.2MB/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 0.
Label code corresponds to ADI 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))
```

▼ 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(f'\t accuracy {:.4f}'.format(Metrics.accuracy(gt, pred)))
        print(f'\t balanced accuracy {:.4f}'.format(Metrics.accuracy_balanced(gt, pred)))
```

▼ Класс Model

Класс, хранящий в себе всю информацию о модели.

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

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

Также, Вы можете реализовать дополнительные функции, такие как:

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

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

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

```
class TissueTorchDS(torch.utils.data.Dataset):
    def __init__(self, imgs, lbls, transform):
        self.imgs = imgs
        self.lbls = lbls
        self.tr = transform

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

    def __getitem__(self, i):
        x = self.tr(self.imgs[i])
        y = self.lbls[i]
        return x, y
```

```

class Model:

    def __init__(self, device='cuda' if torch.cuda.is_available() else 'cpu'): #LBL1
        self.device = device
        self.model = models.googlenet(
            weights=models.GoogLeNet_Weights.DEFAULT, #LBL2
            aux_logits=True
        )
        self.model.fc = nn.Linear(self.model.fc.in_features, NUM_CLASSES)
        self.model = self.model.to(device)
        self.best_val_acc = 0.0
        self.transform = transforms.Compose([ #LBL3
            transforms.ToTensor(),
            transforms.Resize((224, 224)),
            transforms.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225]
            )
        ])
    )

    def save(self, name: str):
        save_path = f'/content/drive/MyDrive/{name}.pth'
        torch.save(self.model.state_dict(), save_path)

    def load(self, name: str = 'best'):
        name_to_id_dict = {
            'best': "1nATCEJq6KQ6S8v1mw0J0nWMyfiQ2-lW1"
        }
        if name not in name_to_id_dict:
            raise ValueError(f"Unknown model name '{name}'")
        file_id = name_to_id_dict[name]
        output = f'{name}.pth'
        gdown.download(f'https://drive.google.com/uc?id={file_id}', output, quiet = False)
        state_dict = torch.load(output, map_location=self.device)
        self.model.load_state_dict(state_dict)

    def train(self, dataset, epochs = 5, batch_size = 64, lr = 0.0001, val_split = 0.1):
        images = dataset.images
        labels = dataset.labels
        N = len(images)
        idx = int(N * (1 - val_split))
        train_images, val_images = images[:idx], images[idx:]
        train_labels, val_labels = labels[:idx], labels[idx:]

        train_loader = torch.utils.data.DataLoader(
            TissueTorchDS(train_images, train_labels, self.transform),
            batch_size=batch_size, shuffle = True
        )
        val_loader = torch.utils.data.DataLoader(
            TissueTorchDS(val_images, val_labels, self.transform),
            batch_size=batch_size, shuffle = False
        )

        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(self.model.parameters(), lr = lr)
        scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau( #LBL4
            optimizer,
            mode = 'max',
            factor = 0.5,
            patience = 3,
            min_lr = 1e-6
        )

        for epoch in range(epochs):
            self.model.train()
            epoch_loss = 0
            for x, y in tqdm(train_loader, desc = f"Epoch {epoch + 1}/{epochs}"):
                x, y = x.to(self.device), y.to(self.device)
                optimizer.zero_grad()
                pred = self.model(x)
                loss = criterion(pred.logits, y)
                loss.backward()
                optimizer.step()
                epoch_loss += loss.item()
            val_acc = self.evaluate(val_loader)
            scheduler.step(val_acc)

            print(f"Epoch {epoch + 1}: loss = {epoch_loss:.3f}, val_acc = {val_acc:.3f}") #LBL5

            if val_acc > self.best_val_acc:
                self.best_val_acc = val_acc
                self.save("best_model") #LBL6

```

```

def evaluate(self, loader):
    self.model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for x, y in loader:
            x, y = x.to(self.device), y.to(self.device)
            outputs = self.model(x)
            preds = outputs.argmax(dim = 1)
            correct += (preds == y).sum().item()
            total += len(y)
    return correct / total

def test_on_dataset(self, dataset, limit = None):
    predictions = []
    n = dataset.n_files if not limit else int(dataset.n_files * limit)
    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):
    self.model.eval()
    x = self.transform(img).unsqueeze(0).to(self.device)
    with torch.no_grad():
        outputs = self.model(x)
        pred_class = outputs.argmax(1).item()
    return pred_class

```

▼ 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=1qd45xFDwdZjktLFwQb-et-mAaFeCzOR
To: /content/train_small.npz
100%|██████████| 841M/841M [00:08<00:00, 93.4MB/s]
Loading dataset train_small from npz.
Done. Dataset train_small consists of 7200 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1wbRsog0n7uG1HIPGLhyN-PMeT2kdQ2lI
To: /content/test_small.npz
100%|██████████| 211M/211M [00:01<00:00, 165MB/s]
Loading dataset test_small from npz.
Done. Dataset test_small consists of 1800 images.

```

```

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

Downloading: "https://download.pytorch.org/models/googlenet-1378be20.pth" to /root/.cache/torch/hub/checkpoints/googlenet-1378be20.pth
100%|██████████| 49.7M/49.7M [00:00<00:00, 208MB/s]
/usr/local/lib/python3.12/dist-packages/torchvision/models/googlenet.py:341: UserWarning: auxiliary heads in the pretrained
    warnings.warn(
💡 Скачивание весов модели 'best' с Google Drive...
Downloading...
From (original): https://drive.google.com/uc?id=1nATCEJq6K06S8v1mwOJ0nWMyfiQ2-1W1
From (redirected): https://drive.google.com/uc?id=1nATCEJq6K06S8v1mwOJ0nWMyfiQ2-1W1&confirm=t&uuid=ccbccc40-59e3-493c-b701-1
To: /content/best.pth
100%|██████████| 48.1M/48.1M [00:00<00:00, 121MB/s]
💡 Загрузка весов в модель...
✓ Модель 'best' успешно загружена и готова к использованию!

```

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, '100% of test')

```

```
100%                                         1800/1800 [00:17<00:00, 114.89it/s]
metrics for 10% of test:
accuracy 0.9650:
balanced accuracy 0.9650:
```

Пример тестирования модели на полном наборе данных:

```
d_test = Dataset('test')

Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1RfPou3pFKpuHDJZ-D9XDFzgywpUBF1Dr
To: /content/test.npz
100%|██████████| 525M/525M [00:06<00:00, 75.2MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
```

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

100%                                         4500/4500 [01:39<00:00, 50.27it/s]
metrics for test:
accuracy 0.9542:
balanced accuracy 0.9542:
```

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 экспортить 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')
pred = final_model.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, 'test')

☒ Скачивание весов модели 'best' с Google Drive...
Downloading...
From (original): https://drive.google.com/uc?id=1nATCEJq6K06S8v1mw0J0nlWMyfiQ2-1W1
From (redirected): https://drive.google.com/uc?id=1nATCEJq6K06S8v1mw0J0nlWMyfiQ2-1W1&confirm=t&uuid=6c56f39f-17a9-4d65-9f70-1
To: /content/best.pth
100%|██████████| 48.1M/48.1M [00:00<00:00, 158MB/s]
☒ Загрузка весов в модель...
✓ Модель 'best' успешно загружена и готова к использованию!
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1RfPou3pFKpuHDJZ-D9XDFzgywpUBF1Dr
To: /content/test.npz
100%|██████████| 525M/525M [00:02<00:00, 241MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.

100%                                         4500/4500 [00:39<00:00, 125.80it/s]
metrics for test:
accuracy 0.9656:
balanced accuracy 0.9656:
```

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.')
```

▼ 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 питчу, 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)

```

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 турниров 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 турнир: <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:

- https://colab.research.google.com/github/cbernet/maldives/blob/master/numba/numba_cuda.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"
```

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"
```