

✓ Практическое задание №1

Установка необходимых пакетов:

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

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.12.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.16.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.6)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.6)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2024.8.3)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7
```

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': '1JFaqH7QY5PiWUcXzVkJ7H195P-EQVG',
    'train_small': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1I-2Z0uXLd4QwhZQ1tp817Kn3J0Xgbui',
    'test': '1UW309J4bP70EQZKRKPQvBuEqL_kjhadQ',
    'test_small': '1wbRsog0n7uG1HIPGLhyN-PMET2kdQ21I',
    '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=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.

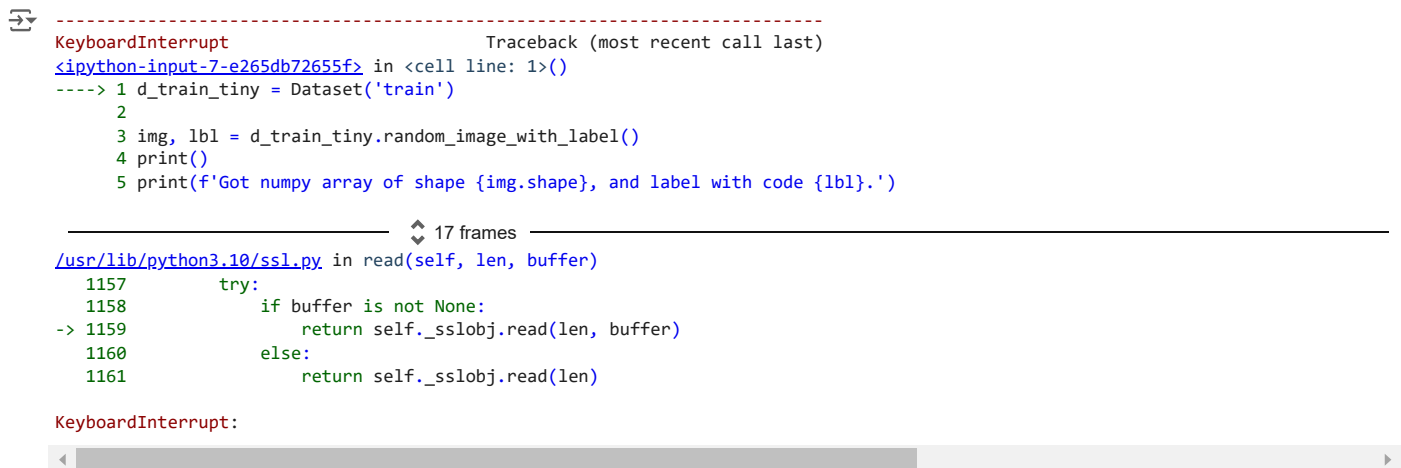
```

d_train_tiny = Dataset('train')

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)

```



```

-----
KeyboardInterrupt                                Traceback (most recent call last)
<ipython-input-7-e265db72655f> in <cell line: 1>()
----> 1 d_train_tiny = Dataset('train')
      2
      3 img, lbl = d_train_tiny.random_image_with_label()
      4 print()
      5 print(f'Got numpy array of shape {img.shape}, and label with code {lbl}.')

-----
17 frames
/usr/lib/python3.10/ssl.py in read(self, len, buffer)
    1157         try:
    1158             if buffer is not None:
-> 1159                 return self._sslobj.read(len, buffer)
    1160             else:
    1161                 return self._sslobj.read(len)

KeyboardInterrupt:

```

✓ 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 загрузки модели. Особенно актуально это будет во время тестирования на дополнительных наборах данных.

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

Также, Вы можете реализовать дополнительные функции, такие как:

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

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

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

```

import os
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms
from torchvision.models import MobileNet_V2_Weights
from torch.utils.tensorboard import SummaryWriter
from PIL import Image
import numpy as np
from torch.utils.data import DataLoader, Dataset as TorchDataset

class Model:
    def __init__(self, learning_rate=0.004, batch_size=32, num_epochs=10, weight_decay=1e-4):
        """
        Инициализация модели, гиперпараметров и преобразований данных.
        """
        # Гиперпараметры
        self.learning_rate = learning_rate
        self.batch_size = batch_size
        self.num_epochs = num_epochs
        self.weight_decay = weight_decay

        # Устройство (CPU/GPU)
        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

```

```

# Модель EfficientNet
self.model = models.efficientnet_b0(pretrained=True)
num_features = self.model.classifier[1].in_features
self.model.classifier[1] = nn.Linear(num_features, len(TISSUE_CLASSES)) # 9 классов
self.model.to(self.device)

# Функция потерь
self.criterion = nn.CrossEntropyLoss()

# Оптимизатор
self.optimizer = optim.Adam(self.model.parameters(), lr=self.learning_rate, weight_decay=self.weight_decay)

# Аугментации данных (тренировочные и тестовые)
self.train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.RandomRotation(15),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]), # Нормализация для ImageNet
])

self.test_transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

print("Model initialized with the following parameters:")
print(f"Learning rate: {self.learning_rate}, Batch size: {self.batch_size}, Epochs: {self.num_epochs}")

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

def load(self, name: str):
    self.model.load_state_dict(torch.load(f'/content/drive/MyDrive/{name}.pth', map_location=self.device))
    self.model.to(self.device)

def train(self, dataset: Dataset):
    """
    Основной обучающий цикл.
    """
    print("Preparing training data...")

    # Обернем наш `Dataset` в PyTorch-совместимый датасет
    class PyTorchDataset(TorchDataset):
        def __init__(self, dataset: Dataset, transform=None):
            """
            Инициализация PyTorch Dataset.
            :param dataset: Объект класса Dataset, который нужно адаптировать.
            :param transform: Трансформации для обработки изображений (например, аугментация).
            """
            self.dataset = dataset
            self.transform = transform

        def __len__(self):
            return self.dataset.n_files

        def __getitem__(self, idx):
            # Получаем изображение и метку
            image, label = self.dataset.image_with_label(idx)

            # Конвертируем numpy.ndarray в PIL.Image
            image = Image.fromarray(image)

            # Применяем трансформации, если они заданы
            if self.transform:
                image = self.transform(image)

            # Возвращаем изображение и метку
            return image, label

    # Создание PyTorch DataLoader для батчевого обучения
    train_data = PyTorchDataset(dataset, transform=self.train_transforms)
    train_loader = DataLoader(train_data, batch_size=self.batch_size, shuffle=True, num_workers=4)

    print("Training started...")

    self.model.train() # Перевод модели в режим обучения
    best_accuracy = 0.0

    for epoch in range(self.num_epochs):

```

```

epoch_loss = 0.0
correct = 0
total = 0

progress_bar = tqdm(train_loader, desc="Epoch {epoch+1}/{self.num_epochs}", unit="batch")
for images, labels in progress_bar:
    images, labels = images.to(self.device), labels.to(self.device)

    # Обнуление градиентов
    self.optimizer.zero_grad()

    # Прямой проход
    outputs = self.model(images)

    # Расчет потерь
    loss = self.criterion(outputs, labels)
    epoch_loss += loss.item()

    # Обратное распространение
    loss.backward()
    self.optimizer.step()

    # Обновление статистики
    _, predicted = torch.max(outputs, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()

    # Обновление прогресс-бара
    progress_bar.set_postfix(loss=loss.item(), accuracy=100 * correct / total)

# Эпохальная метрика
epoch_accuracy = 100 * correct / total
print(f"Epoch {epoch+1}: Loss = {epoch_loss:.4f}, Accuracy = {epoch_accuracy:.2f}%")

# Сохранение лучшей модели
if epoch_accuracy > best_accuracy:
    best_accuracy = epoch_accuracy
    self.save(f"best_model_epoch_{epoch+1}")
    print(f"New best model saved with accuracy: {best_accuracy:.2f}%")

print("Training complete!")

def test_on_dataset(self, dataset: Dataset, limit=None):
    # you can upgrade this code if you want to speed up testing using batches
    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):
    """
    Выполняет предсказание для одного изображения.
    :param img: Изображение в формате numpy array.
    :return: Индекс предсказанного класса.
    """
    self.model.eval() # Перевод модели в режим оценки (inference)

    with torch.no_grad(): # Отключаем вычисление градиентов для ускорения
        # Преобразуем изображение: numpy -> PyTorch Tensor
        transform = self.test_transforms
        img_tensor = transform(img).unsqueeze(0) # Добавляем batch dimension

        # Перемещаем данные на устройство (CPU/GPU)
        img_tensor = img_tensor.to(self.device)

        # Прогон изображения через модель
        output = self.model(img_tensor)

        # Получаем индекс предсказанного класса
        _, predicted_class = torch.max(output, 1)
        return predicted_class.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')
d_test = Dataset('test')
```

```

Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1JFaqah7QY5PiWUcXzVkQ17H195P-EQVG
To: /content/train.npz
100%|██████████| 2.10G/2.10G [00:21<00:00, 98.7MB/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=1UW309J4bP70EQZKRKPQvBuEgI\_kjhadQ
To: /content/test.npz
100%|██████████| 525M/525M [00:11<00:00, 46.1MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
```

```

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

```

/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/efficientnet\_b0\_rwightman-7f5810bc.pth" to /root/.cache/torch/hub/checkpoints/eff:
100%|██████████| 20.5M/20.5M [00:00<00:00, 66.8MB/s]
Model initialized with the following parameters:
Learning rate: 0.004, Batch size: 32, Epochs: 10
Preparing training data...
Training started...
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:617: UserWarning: This DataLoader will create 4 worker proces
warnings.warn(

Epoch 1/10: 100% 563/563 [01:59<00:00, 6.95batch/s, accuracy=84.7, loss=0.258]
Epoch 1: Loss = 275.0324, Accuracy = 84.73%
New best model saved with accuracy: 84.73%
Epoch 2/10: 100% 563/563 [01:59<00:00, 6.06batch/s, accuracy=90.9, loss=0.208]
Epoch 2: Loss = 162.1845, Accuracy = 90.86%
New best model saved with accuracy: 90.86%
Epoch 3/10: 100% 563/563 [02:02<00:00, 5.60batch/s, accuracy=92.3, loss=0.366]
Epoch 3: Loss = 140.7715, Accuracy = 92.29%
New best model saved with accuracy: 92.29%
Epoch 4/10: 100% 563/563 [02:00<00:00, 6.18batch/s, accuracy=93.1, loss=0.108]
Epoch 4: Loss = 124.1007, Accuracy = 93.08%
New best model saved with accuracy: 93.08%
Epoch 5/10: 100% 563/563 [02:00<00:00, 5.97batch/s, accuracy=93.4, loss=0.327]
Epoch 5: Loss = 119.3518, Accuracy = 93.43%
New best model saved with accuracy: 93.43%
Epoch 6/10: 100% 563/563 [02:02<00:00, 6.10batch/s, accuracy=93.8, loss=0.00944]
Epoch 6: Loss = 112.0043, Accuracy = 93.83%
New best model saved with accuracy: 93.83%
Epoch 7/10: 100% 563/563 [02:00<00:00, 6.19batch/s, accuracy=93.8, loss=0.127]
Epoch 7: Loss = 115.6526, Accuracy = 93.77%
Epoch 8/10: 100% 563/563 [02:05<00:00, 5.57batch/s, accuracy=94.7, loss=0.103]
Epoch 8: Loss = 99.4543, Accuracy = 94.73%
New best model saved with accuracy: 94.73%
Epoch 9/10: 100% 563/563 [02:00<00:00, 6.17batch/s, accuracy=94.4, loss=0.016]
Epoch 9: Loss = 98.7283, Accuracy = 94.38%
Epoch 10/10: 100% 563/563 [01:58<00:00, 6.19batch/s, accuracy=94.7, loss=0.359]
Epoch 10: Loss = 93.8583, Accuracy = 94.74%
New best model saved with accuracy: 94.74%
Training complete!
```

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 [00:08<00:00, 80.84it/s]
metrics for 10% of test:
  accuracy 0.9867:
  balanced accuracy 0.9867:
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:2480: UserWarning: y_pred contains classes not in y_true
  warnings.warn(f"The predicted labels {y_pred_classes} do not match the true labels {y_true_classes}")
```

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 [00:58<00:00, 90.69it/s]
metrics for test:
  accuracy 0.9687:
  balanced accuracy 0.9687:
```

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 = model.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, 'test-tiny')
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since
warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
warnings.warn(msg)
<ipython-input-9-b4fade21e4c5>:60: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value),
  self.model.load_state_dict(torch.load(f'/content/drive/MyDrive/{name}.pth', map_location=self.device))
Model initialized with the following parameters:
Learning rate: 0.004, Batch size: 32, Epochs: 10
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=lviiB0s041CNsAK4itvX8PnYthJ-MDnQc
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 58.5MB/s]
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
100% 90/90 [00:01<00:00, 60.17it/s]
metrics for test-tiny:
  accuracy 0.9667:
  balanced accuracy 0.9667:
```

Отмонтировать Google Drive.

```
drive.flush_and_unmount()
```

✓ Дополнительные "полезности"

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

✓ Измерение времени работы кода

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

```
import timeit

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

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

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

✓ Scikit-learn

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

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

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

# The digits dataset
digits = datasets.load_digits()

# The data that we are interested in is made of 8x8 images of digits, let's
# have a look at the first 4 images, stored in the `images` attribute of the
# dataset. If we were working from image files, we could load them using
# matplotlib.pyplot.imread. Note that each image must have the same size. For these
# images, we know which digit they represent: it is given in the 'target' of
# the dataset.
_, axes = plt.subplots(2, 4)
images_and_labels = list(zip(digits.images, digits.target))
for ax, (image, label) in zip(axes[0, :], images_and_labels[:4]):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Training: %i' % label)

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

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

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

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

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

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



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

plt.show()
```

✓ Scikit-image

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

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

from skimage import feature

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

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

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

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

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

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

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

fig.tight_layout()

plt.show()
```

✓ Tensorflow 2

Для создания и обучения нейросетевых моделей можно использовать фреймворк глубокого обучения Tensorflow 2. Ниже приведен пример простейшей 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 tutorиалов 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 tutorиал: <https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly>.

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/cbnet/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, внутри которой находятся 2 изображения.

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

