

Практическое задание №1

Установка необходимых пакетов:

In [1]:

```
!pip install -q libtiff
!pip install -q tqdm
```

Монтирование Вашего Google Drive к текущему окружению:

In [2]:

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

Mounted at /content/drive

В переменную PROJECT_DIR необходимо прописать путь к директории на Google Drive, в которую Вы загрузили zip архивы с предоставленными наборами данных.

In [3]:

```
# todo
PROJECT_DIR = 'nn/'
```

Константы, которые пригодятся в коде далее:

In [4]:

```
EVALUATE_ONLY = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
```

Импорт необходимых зависимостей:

In [5]:

```
from pathlib import Path
from libtiff import TIFF
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 tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.layers.normalization import BatchNormalization
```

Класс Dataset

Предназначен для работы с наборами данных, хранящихся на Google Drive, обеспечивает чтение изображений и соответствующих меток, а также формирование пакетов (батчей).

In [6]:

```
class Dataset:

    def __init__(self, name, gdrive_dir):
        self.name = name
        self.is_loaded = False
        p = Path("/content/drive/MyDrive/" + gdrive_dir + name + '.npz')
        if p.exists():
            print(f'Loading dataset {self.name} from npz.')
            np_obj = np.load(str(p))
            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.

In [7]:

```

d_train_tiny = Dataset('train_tiny', PROJECT_DIR)

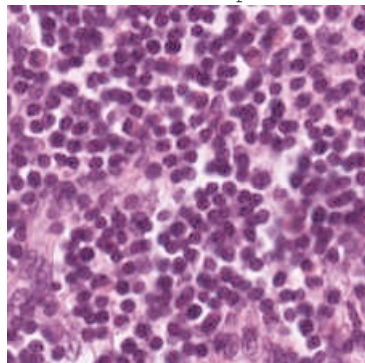
img, lbl = d_train_tiny.random_image_with_label()
print()
print(f'Got numpy array of shape {img.shape}, and label with code {lbl}.')
print(f'Label code corresponds to {TISSUE_CLASSES[lbl]} class.')

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

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 3.
Label code corresponds to LYM class.

```



Класс Metrics

Реализует метрики точности, используемые для оценивания модели:

1. точность,
2. сбалансированную точность.

In [8]:

```

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.

In [12]:

```
class Model:
```

```
    def __init__(self):
        self.classifier = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu', input_shape=(28, 28, 3)),
            tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'),
            tf.keras.layers.MaxPooling2D(pool_size=(4, 4)),
            tf.keras.layers.Dropout(0.1),
            tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'),
            tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'),
            tf.keras.layers.MaxPooling2D(pool_size=(4, 4)),
            tf.keras.layers.Dropout(0.1),
            tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same', activation='relu'),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.Dropout(0.1),
            tf.keras.layers.Conv2D(filters=128, kernel_size=3, padding='same', activation='relu'),
            tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dropout(0.1),
            tf.keras.layers.Dense(1024, activation='relu'),
            tf.keras.layers.Dropout(0.1),
            tf.keras.layers.Dense(9, activation='softmax')
        ])

    pass

    def save(self, name: str):
        # save model to PROJECT_DIR folder on gdrive with name 'name'
        self.classifier.save("/content/drive/MyDrive/" + PROJECT_DIR + name + '.h5')

    def load(self, name: str):
        # load model with name 'name' from PROJECT_DIR folder on gdrive
        p = Path("/content/drive/MyDrive/" + PROJECT_DIR + name + '.h5')
        if p.exists():
            print(f'Loading model {name}')
            self.classifier = tf.keras.models.load_model("/content/drive/MyDrive/" + PROJECT_DIR + name + '.h5')

    def train(self, dataset: Dataset):
```

```

# you can add some plots for better visualization,
# you can add model autosaving during training,
# etc.
print(f'training started')
self.classifier.compile(optimizer='adam',
                        loss='sparse_categorical_crossentropy',
                        metrics=['accuracy'])
#LBL1
x_train, x_test, y_train, y_test = train_test_split(dataset.images, dataset.labels, test_size=0.1)
self.history = self.classifier.fit(x_train, y_train, epochs=50, batch_size=64, validation_data=(x_test, y_test))
sleep(2)
print(f'training done')

def visual(self):
    #LBL2
    plt.plot(self.history.history['accuracy'])
    plt.plot(self.history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()

    plt.plot(self.history.history['loss'])
    plt.plot(self.history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()

def show_history(self):
    #LBL3
    return self.history.history

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):
    prediction = self.classifier.predict(img[np.newaxis, :]).argmax()
    return prediction

```

Классификация изображений

Используя введенные выше классы можем перейти уже непосредственно к обучению модели классификации изображений. Пример общего пайплайна решения задачи приведен ниже. Вы можете его расширять и улучшать. В данном примере используются наборы данных 'train_small' и 'test_small'.

In [10]:

```

d_train = Dataset('train', PROJECT_DIR)
d_test = Dataset('test', PROJECT_DIR)

Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.

model = Model()

model.train(d_train)

training started
Epoch 1/50
254/254 [=====] - 68s 267ms/step - loss: 4.4750 - accuracy: 0.2725 - val_loss: 1.0388 - val_accuracy: 0.5822
Epoch 2/50
254/254 [=====] - 67s 264ms/step - loss: 1.1047 - accuracy: 0.5597 - val_loss: 0.9648 - val_accuracy: 0.6333
Epoch 3/50
254/254 [=====] - 67s 265ms/step - loss: 0.8921 - accuracy: 0.6616 - val_loss: 0.8021 - val_accuracy: 0.6889

```

In [13]:

Epoch 4/50
254/254 [=====] - 67s 265ms/step - loss: 0.6274 - accuracy: 0.7758 - val_loss: 0.5532 - val_accuracy: 0.8017
Epoch 5/50
254/254 [=====] - 67s 265ms/step - loss: 0.4264 - accuracy: 0.8551 - val_loss: 0.5103 - val_accuracy: 0.8261
Epoch 6/50
254/254 [=====] - 67s 264ms/step - loss: 0.3832 - accuracy: 0.8720 - val_loss: 0.3623 - val_accuracy: 0.8761
Epoch 7/50
254/254 [=====] - 67s 264ms/step - loss: 0.3161 - accuracy: 0.8907 - val_loss: 0.2728 - val_accuracy: 0.9067
Epoch 8/50
254/254 [=====] - 67s 264ms/step - loss: 0.2800 - accuracy: 0.9001 - val_loss: 0.2878 - val_accuracy: 0.9067
Epoch 9/50
254/254 [=====] - 67s 263ms/step - loss: 0.2282 - accuracy: 0.9231 - val_loss: 0.4687 - val_accuracy: 0.8339
Epoch 10/50
254/254 [=====] - 67s 263ms/step - loss: 0.3757 - accuracy: 0.8691 - val_loss: 0.2153 - val_accuracy: 0.9272
Epoch 11/50
254/254 [=====] - 67s 263ms/step - loss: 0.1731 - accuracy: 0.9411 - val_loss: 0.2527 - val_accuracy: 0.9117
Epoch 12/50
254/254 [=====] - 67s 263ms/step - loss: 0.1945 - accuracy: 0.9337 - val_loss: 0.2605 - val_accuracy: 0.9106
Epoch 13/50
254/254 [=====] - 67s 263ms/step - loss: 0.2059 - accuracy: 0.9310 - val_loss: 0.1673 - val_accuracy: 0.9461
Epoch 14/50
254/254 [=====] - 67s 263ms/step - loss: 0.1355 - accuracy: 0.9540 - val_loss: 0.1434 - val_accuracy: 0.9472
Epoch 15/50
254/254 [=====] - 67s 263ms/step - loss: 0.1546 - accuracy: 0.9476 - val_loss: 0.2089 - val_accuracy: 0.9378
Epoch 16/50
254/254 [=====] - 67s 263ms/step - loss: 0.1520 - accuracy: 0.9495 - val_loss: 0.1980 - val_accuracy: 0.9367
Epoch 17/50
254/254 [=====] - 67s 263ms/step - loss: 0.1400 - accuracy: 0.9575 - val_loss: 0.2806 - val_accuracy: 0.9117
Epoch 18/50
254/254 [=====] - 67s 262ms/step - loss: 0.1642 - accuracy: 0.9449 - val_loss: 0.1614 - val_accuracy: 0.9450
Epoch 19/50
254/254 [=====] - 67s 262ms/step - loss: 0.1032 - accuracy: 0.9642 - val_loss: 0.1484 - val_accuracy: 0.9544
Epoch 20/50
254/254 [=====] - 67s 262ms/step - loss: 0.1118 - accuracy: 0.9631 - val_loss: 0.2246 - val_accuracy: 0.9222
Epoch 21/50
254/254 [=====] - 67s 262ms/step - loss: 0.1283 - accuracy: 0.9578 - val_loss: 0.1652 - val_accuracy: 0.9439
Epoch 22/50
254/254 [=====] - 67s 262ms/step - loss: 0.1298 - accuracy: 0.9593 - val_loss: 0.1894 - val_accuracy: 0.9539
Epoch 23/50
254/254 [=====] - 67s 263ms/step - loss: 0.1015 - accuracy: 0.9653 - val_loss: 0.3031 - val_accuracy: 0.9056
Epoch 24/50
254/254 [=====] - 67s 262ms/step - loss: 0.2508 - accuracy: 0.9210 - val_loss: 0.1893 - val_accuracy: 0.9433
Epoch 25/50
254/254 [=====] - 66s 262ms/step - loss: 0.0908 - accuracy: 0.9705 - val_loss: 0.1685 - val_accuracy: 0.9389
Epoch 26/50
254/254 [=====] - 67s 262ms/step - loss: 0.1132 - accuracy: 0.9655 - val_loss: 0.2467 - val_accuracy: 0.9261
Epoch 27/50
254/254 [=====] - 67s 262ms/step - loss: 0.1435 - accuracy: 0.9506 - val_loss: 0.1775 - val_accuracy: 0.9483
Epoch 28/50
254/254 [=====] - 67s 262ms/step - loss: 0.1095 - accuracy: 0.9607 - val_loss: 0.1544 - val_accuracy: 0.9522
Epoch 29/50
254/254 [=====] - 67s 262ms/step - loss: 0.0783 - accuracy: 0.9739 - val_loss:

```
0.3767 - val_accuracy: 0.8856
Epoch 30/50
254/254 [=====] - 66s 262ms/step - loss: 0.1233 - accuracy: 0.9574 - val_loss:
0.1655 - val_accuracy: 0.9522
Epoch 31/50
254/254 [=====] - 67s 262ms/step - loss: 0.0906 - accuracy: 0.9713 - val_loss:
0.1509 - val_accuracy: 0.9539
Epoch 32/50
254/254 [=====] - 66s 262ms/step - loss: 0.4410 - accuracy: 0.8641 - val_loss:
0.8105 - val_accuracy: 0.7294
Epoch 33/50
254/254 [=====] - 66s 260ms/step - loss: 0.4592 - accuracy: 0.8527 - val_loss:
0.1718 - val_accuracy: 0.9417
Epoch 34/50
254/254 [=====] - 66s 260ms/step - loss: 0.1544 - accuracy: 0.9520 - val_loss:
0.1991 - val_accuracy: 0.9461
Epoch 35/50
254/254 [=====] - 66s 261ms/step - loss: 0.1209 - accuracy: 0.9588 - val_loss:
0.2821 - val_accuracy: 0.9211
Epoch 36/50
254/254 [=====] - 66s 260ms/step - loss: 0.1245 - accuracy: 0.9565 - val_loss:
0.2229 - val_accuracy: 0.9378
Epoch 37/50
254/254 [=====] - 66s 261ms/step - loss: 0.1127 - accuracy: 0.9625 - val_loss:
0.1917 - val_accuracy: 0.9572
Epoch 38/50
254/254 [=====] - 66s 261ms/step - loss: 0.1260 - accuracy: 0.9626 - val_loss:
0.2290 - val_accuracy: 0.9400
Epoch 39/50
254/254 [=====] - 66s 261ms/step - loss: 0.0985 - accuracy: 0.9701 - val_loss:
0.1710 - val_accuracy: 0.9583
Epoch 40/50
254/254 [=====] - 66s 261ms/step - loss: 0.0977 - accuracy: 0.9715 - val_loss:
0.1611 - val_accuracy: 0.9522
Epoch 41/50
254/254 [=====] - 66s 261ms/step - loss: 0.0642 - accuracy: 0.9783 - val_loss:
0.1808 - val_accuracy: 0.9517
Epoch 42/50
254/254 [=====] - 66s 262ms/step - loss: 0.0767 - accuracy: 0.9764 - val_loss:
0.1355 - val_accuracy: 0.9544
Epoch 43/50
254/254 [=====] - 66s 261ms/step - loss: 0.0901 - accuracy: 0.9730 - val_loss:
0.2466 - val_accuracy: 0.9367
Epoch 44/50
254/254 [=====] - 66s 261ms/step - loss: 0.0968 - accuracy: 0.9699 - val_loss:
0.1389 - val_accuracy: 0.9578
Epoch 45/50
254/254 [=====] - 66s 261ms/step - loss: 0.0774 - accuracy: 0.9780 - val_loss:
0.1730 - val_accuracy: 0.9494
Epoch 46/50
254/254 [=====] - 66s 262ms/step - loss: 0.0958 - accuracy: 0.9702 - val_loss:
0.3170 - val_accuracy: 0.9394
Epoch 47/50
254/254 [=====] - 66s 261ms/step - loss: 0.0968 - accuracy: 0.9710 - val_loss:
0.1591 - val_accuracy: 0.9556
Epoch 48/50
254/254 [=====] - 66s 260ms/step - loss: 0.0992 - accuracy: 0.9689 - val_loss:
0.1736 - val_accuracy: 0.9522
Epoch 49/50
254/254 [=====] - 66s 261ms/step - loss: 0.1324 - accuracy: 0.9604 - val_loss:
0.2093 - val_accuracy: 0.9500
Epoch 50/50
254/254 [=====] - 66s 260ms/step - loss: 0.0610 - accuracy: 0.9801 - val_loss:
0.1933 - val_accuracy: 0.9561
training done
```

In [15]:

```
model.classifier.evaluate(d_test.images, d_test.labels)

141/141 [=====] - 4s 32ms/step - loss: 0.2345 - accuracy: 0.9507

[0.2344571053981781, 0.9506666660308838]
```

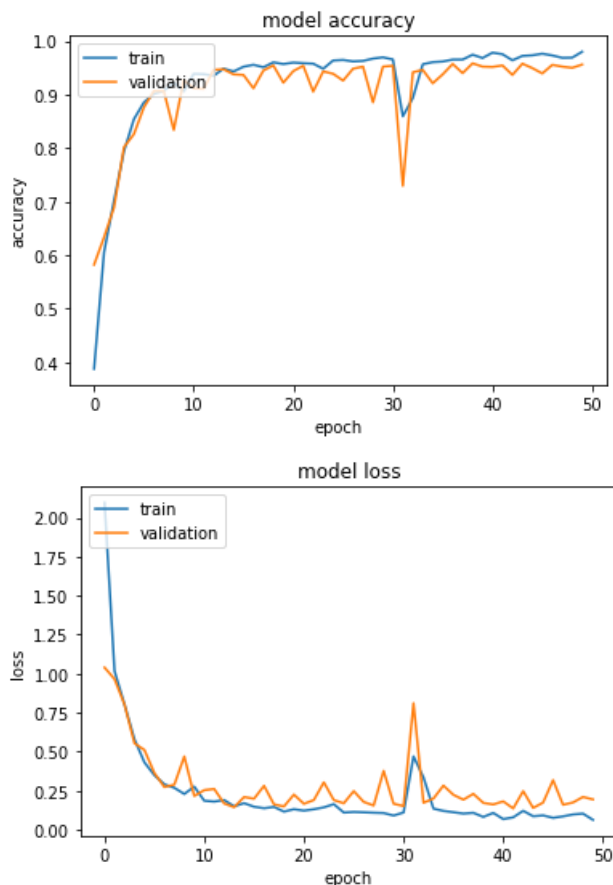
Out[15]:

In [19]:

```
model.save('best')
```

In [16]:

```
model.visual()
```



Пример тестирования модели на части набора данных:

In [17]:

```
# 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')

metrics for 10% of test:
  accuracy 0.9667:
  balanced accuracy 0.9667:
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1859: UserWarning: y_pred
contains classes not in y_true
  warnings.warn('y_pred contains classes not in y_true')
```

Пример тестирования модели на полном наборе данных:

In [18]:

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

metrics for test:
  accuracy 0.9507:
  balanced accuracy 0.9507:
```

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.

In [21]:

```
final_model = Model()
final_model.load('best')
d_test_tiny = Dataset('test_tiny-2', PROJECT_DIR)
pred = model.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, 'test-tiny')
```

```
Loading model best
Loading dataset test_tiny-2 from npz.
Done. Dataset test_tiny-2 consists of 90 images.
metrics for test-tiny:
  accuracy 0.9222:
  balanced accuracy 0.9222:
Отмонтировать Google Drive.
```

In []:

```
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:

In []:

```
import timeit

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

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

n_runs = 128
print(f'Function f is calculated {n_runs} times in {timeit.timeit(f, number=n_runs)}s.')
Function f is calculated 128 times in 0.03538683599981596s.
```

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:

In []:

```
# 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()

```

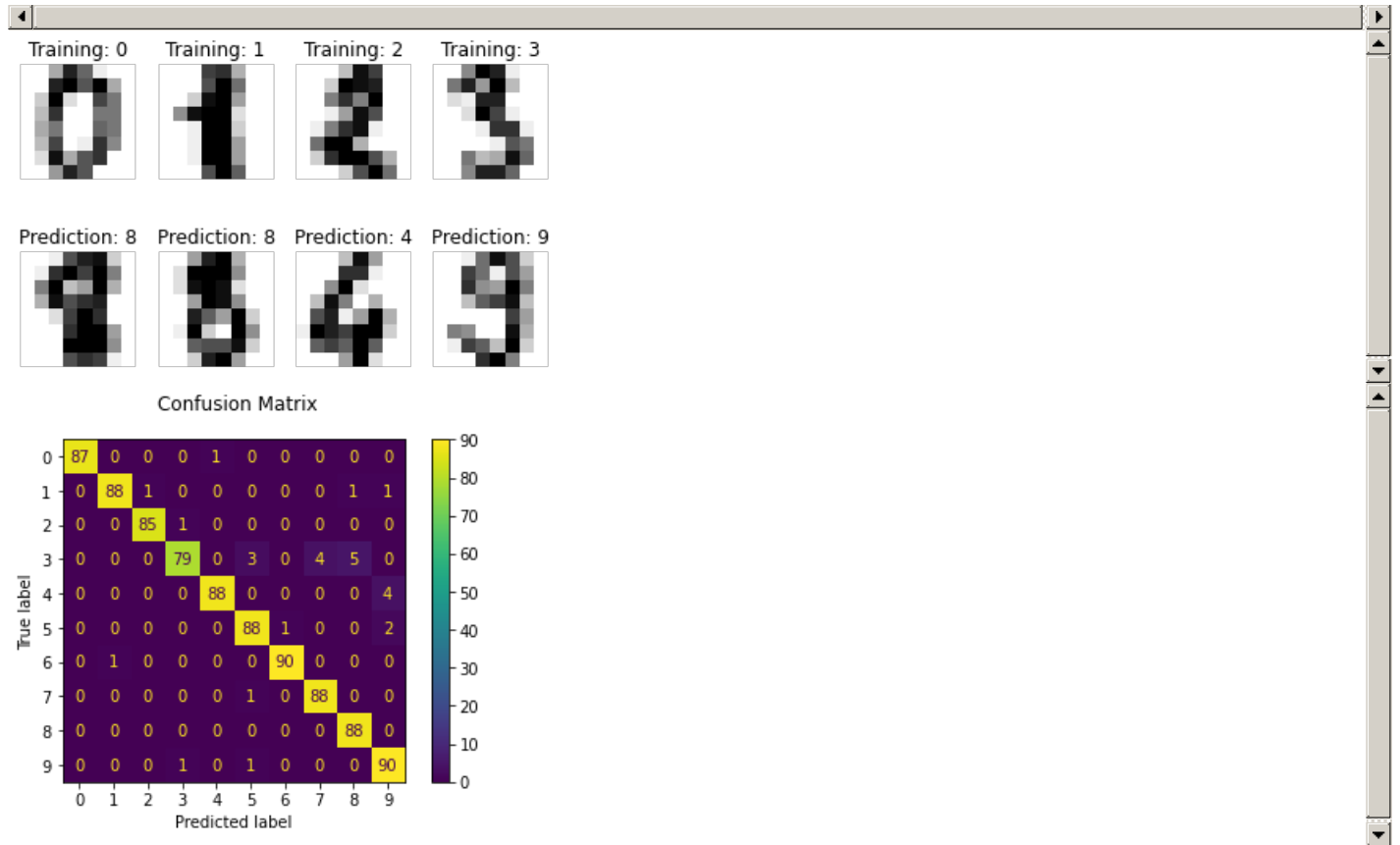
Classification report for classifier SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef=0.0,

decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
max_iter=-1, probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False):

	precision	recall	f1-score	support
0	1.00	0.99	0.99	88
1	0.99	0.97	0.98	91
2	0.99	0.99	0.99	86
3	0.98	0.87	0.92	91
4	0.99	0.96	0.97	92
5	0.95	0.97	0.96	91
6	0.99	0.99	0.99	91
7	0.96	0.99	0.97	89
8	0.94	1.00	0.97	88
9	0.93	0.98	0.95	92
accuracy			0.97	899
macro avg	0.97	0.97	0.97	899
weighted avg	0.97	0.97	0.97	899

Confusion matrix:

```
[[87  0  0  0  1  0  0  0  0  0]
 [ 0 88  1  0  0  0  0  0  1  1]
 [ 0  0 85  1  0  0  0  0  0  0]
 [ 0  0  0 79  0  3  0  4  5  0]
 [ 0  0  0  0 88  0  0  0  0  4]
 [ 0  0  0  0  0 88  1  0  0  2]
 [ 0  1  0  0  0  0 90  0  0  0]
 [ 0  0  0  0  0  1  0 88  0  0]
 [ 0  0  0  0  0  0  0  0 88  0]
 [ 0  0  0  1  0  1  0  0  0 90]]
```



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.

In []:

```
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.

In []:

```

# Install TensorFlow

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

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

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

```

```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [=====] - 0s 0us/step
Epoch 1/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.3014 - accuracy: 0.9136
Epoch 2/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.1446 - accuracy: 0.9571
Epoch 3/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.1077 - accuracy: 0.9671
Epoch 4/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.0864 - accuracy: 0.9735
Epoch 5/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.0731 - accuracy: 0.9773
313/313 - 0s - loss: 0.0787 - accuracy: 0.9760

```

Out[]:

```
[0.07866337150335312, 0.9760000109672546]
```

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 загрузке 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.

In []:

```

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"

/content/drive/MyDrive/dev/prak_nn_1_data
adding: tmp/ (stored 0%)
adding: tmp/img1.png (stored 0%)
adding: tmp/img2.png (stored 0%)

```

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

In []:

```

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

```

/content/drive/MyDrive/dev/prak_nn_1_data