Практическое задание №1

Установка необходимых пакетов:

In [1]:

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

```
     |████████████████████████████████| 133kB 10.5MB/s
  Building wheel for libtiff (setup.py) ... done
```

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 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.

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)
```

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 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.

In [9]:

```
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_shap
          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=32, 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=32, 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(512, activation='relu'),
          tf.keras.layers.Dropout(0.1),
          tf.keras.layers.Dense(128, activation='relu'),
          tf.keras.layers.Dropout(0.1),
          tf.keras.layers.Dense(10, 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 + '
```

```
    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
        sleep(2)
        print(f'training done')

    def visual(self):
        #LBL3
        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):
        #LBL2
        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
        n = dataset.n_files if not limit else int(dataset.n_files * limit)
        images, labels = d_test.random_batch_with_labels(n)
        predictions = self.classifier.predict(images).argmax(axis=1)
        return predictions, labels

    def test_on_image(self, img: np.ndarray):
        # todo: replace this code
        pass
```

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.
```

In [11]:

```
model = Model()

model.train(d_train);
```

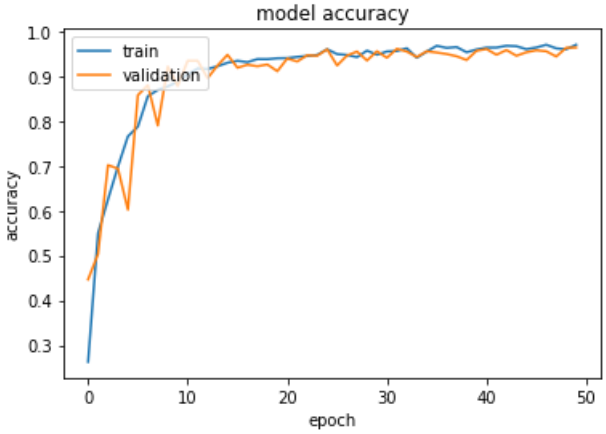
```
training started
Epoch 1/50
254/254 [==============================] - 130s 471ms/step - loss: 2.9841 - accuracy: 0.1754 - val_loss:
1.5973 - val_accuracy: 0.4467
Epoch 2/50
254/254 [==============================] - 118s 465ms/step - loss: 1.2656 - accuracy: 0.5123 - val_loss:
1.2656 - val_accuracy: 0.5033
Epoch 3/50
254/254 [==============================] - 118s 465ms/step - loss: 1.0334 - accuracy: 0.6026 - val_loss:
0.7972 - val_accuracy: 0.7017
```

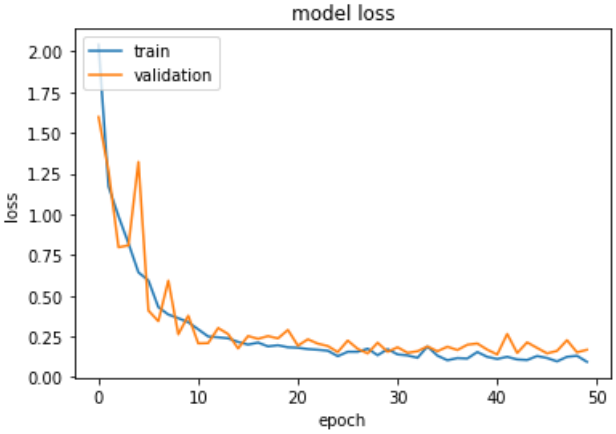
```
Epoch 4/50
254/254 [==============================] - 118s 464ms/step - loss: 0.8597 - accuracy: 0.6834 - val_loss:
0.8086 - val_accuracy: 0.6939
Epoch 5/50
254/254 [==============================] - 118s 464ms/step - loss: 0.6853 - accuracy: 0.7507 - val_loss:
1.3214 - val_accuracy: 0.6022
Epoch 6/50
254/254 [==============================] - 118s 463ms/step - loss: 0.7596 - accuracy: 0.7293 - val_loss:
0.4091 - val_accuracy: 0.8578
Epoch 7/50
254/254 [==============================] - 118s 463ms/step - loss: 0.4428 - accuracy: 0.8524 - val_loss:
0.3432 - val_accuracy: 0.8800
Epoch 8/50
254/254 [==============================] - 118s 463ms/step - loss: 0.3940 - accuracy: 0.8669 - val_loss:
0.5924 - val_accuracy: 0.7900
Epoch 9/50
254/254 [==============================] - 117s 462ms/step - loss: 0.4142 - accuracy: 0.8598 - val_loss:
0.2627 - val_accuracy: 0.9217
Epoch 10/50
254/254 [==============================] - 117s 462ms/step - loss: 0.3326 - accuracy: 0.8860 - val_loss:
0.3773 - val_accuracy: 0.8783
Epoch 11/50
254/254 [==============================] - 117s 462ms/step - loss: 0.3142 - accuracy: 0.8982 - val_loss:
0.2072 - val_accuracy: 0.9344
Epoch 12/50
254/254 [==============================] - 117s 462ms/step - loss: 0.2567 - accuracy: 0.9127 - val_loss:
0.2090 - val_accuracy: 0.9344
Epoch 13/50
254/254 [==============================] - 117s 462ms/step - loss: 0.2443 - accuracy: 0.9169 - val_loss:
0.3016 - val_accuracy: 0.8956
Epoch 14/50
254/254 [==============================] - 117s 462ms/step - loss: 0.2370 - accuracy: 0.9234 - val_loss:
0.2645 - val_accuracy: 0.9239
Epoch 15/50
254/254 [==============================] - 117s 462ms/step - loss: 0.2250 - accuracy: 0.9261 - val_loss:
0.1762 - val_accuracy: 0.9478
Epoch 16/50
254/254 [==============================] - 117s 461ms/step - loss: 0.1816 - accuracy: 0.9413 - val_loss:
0.2522 - val_accuracy: 0.9189
Epoch 17/50
254/254 [==============================] - 117s 462ms/step - loss: 0.2007 - accuracy: 0.9341 - val_loss:
0.2353 - val_accuracy: 0.9256
Epoch 18/50
254/254 [==============================] - 117s 461ms/step - loss: 0.1821 - accuracy: 0.9394 - val_loss:
0.2518 - val_accuracy: 0.9222
Epoch 19/50
254/254 [==============================] - 117s 461ms/step - loss: 0.2359 - accuracy: 0.9264 - val_loss:
0.2379 - val_accuracy: 0.9261
Epoch 20/50
254/254 [==============================] - 117s 461ms/step - loss: 0.1922 - accuracy: 0.9363 - val_loss:
0.2898 - val_accuracy: 0.9111
Epoch 21/50
254/254 [==============================] - 117s 461ms/step - loss: 0.2069 - accuracy: 0.9311 - val_loss:
0.1931 - val_accuracy: 0.9394
Epoch 22/50
254/254 [==============================] - 117s 461ms/step - loss: 0.1415 - accuracy: 0.9516 - val_loss:
0.2323 - val_accuracy: 0.9328
Epoch 23/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1866 - accuracy: 0.9382 - val_loss:
0.2055 - val_accuracy: 0.9467
Epoch 24/50
254/254 [==============================] - 117s 461ms/step - loss: 0.1686 - accuracy: 0.9453 - val_loss:
0.1910 - val_accuracy: 0.9456
Epoch 25/50
254/254 [==============================] - 117s 461ms/step - loss: 0.1286 - accuracy: 0.9596 - val_loss:
0.1538 - val_accuracy: 0.9600
Epoch 26/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1578 - accuracy: 0.9461 - val_loss:
0.2245 - val_accuracy: 0.9239
Epoch 27/50
254/254 [==============================] - 117s 461ms/step - loss: 0.1676 - accuracy: 0.9425 - val_loss:
0.1737 - val_accuracy: 0.9461
Epoch 28/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1966 - accuracy: 0.9342 - val_loss:
0.1457 - val_accuracy: 0.9550
Epoch 29/50
254/254 [==============================] - 117s 461ms/step - loss: 0.1180 - accuracy: 0.9631 - val_loss:
```

```
0.2114 - val_accuracy: 0.9350
Epoch 30/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1509 - accuracy: 0.9531 - val_loss:
0.1554 - val_accuracy: 0.9556
Epoch 31/50
254/254 [==============================] - 117s 461ms/step - loss: 0.1408 - accuracy: 0.9530 - val_loss:
0.1834 - val_accuracy: 0.9411
Epoch 32/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1329 - accuracy: 0.9570 - val_loss:
0.1494 - val_accuracy: 0.9611
Epoch 33/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1098 - accuracy: 0.9645 - val_loss:
0.1589 - val_accuracy: 0.9550
Epoch 34/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1581 - accuracy: 0.9481 - val_loss:
0.1884 - val_accuracy: 0.9428
Epoch 35/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1533 - accuracy: 0.9503 - val_loss:
0.1590 - val_accuracy: 0.9556
Epoch 36/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1066 - accuracy: 0.9656 - val_loss:
0.1868 - val_accuracy: 0.9528
Epoch 37/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1187 - accuracy: 0.9631 - val_loss:
0.1664 - val_accuracy: 0.9494
Epoch 38/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1183 - accuracy: 0.9625 - val_loss:
0.1990 - val_accuracy: 0.9444
Epoch 39/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1467 - accuracy: 0.9564 - val_loss:
0.2067 - val_accuracy: 0.9361
Epoch 40/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1390 - accuracy: 0.9558 - val_loss:
0.1686 - val_accuracy: 0.9567
Epoch 41/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1084 - accuracy: 0.9640 - val_loss:
0.1389 - val_accuracy: 0.9606
Epoch 42/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1071 - accuracy: 0.9700 - val_loss:
0.2652 - val_accuracy: 0.9478
Epoch 43/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1204 - accuracy: 0.9645 - val_loss:
0.1482 - val_accuracy: 0.9583
Epoch 44/50
254/254 [==============================] - 117s 460ms/step - loss: 0.0988 - accuracy: 0.9695 - val_loss:
0.2134 - val_accuracy: 0.9456
Epoch 45/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1281 - accuracy: 0.9585 - val_loss:
0.1794 - val_accuracy: 0.9533
Epoch 46/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1153 - accuracy: 0.9640 - val_loss:
0.1458 - val_accuracy: 0.9578
Epoch 47/50
254/254 [==============================] - 117s 460ms/step - loss: 0.0918 - accuracy: 0.9713 - val_loss:
0.1611 - val_accuracy: 0.9556
Epoch 48/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1092 - accuracy: 0.9666 - val_loss:
0.2268 - val_accuracy: 0.9439
Epoch 49/50
254/254 [==============================] - 117s 460ms/step - loss: 0.1283 - accuracy: 0.9617 - val_loss:
0.1525 - val_accuracy: 0.9633
Epoch 50/50
254/254 [==============================] - 117s 460ms/step - loss: 0.0992 - accuracy: 0.9692 - val_loss:
0.1680 - val_accuracy: 0.9639
training done
```

In [13]:

```
model.visual()
```





In [17]:

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

In [18]:

```
model.classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 56, 56, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 56, 64)        36928
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 32)        18464
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 32)          9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 288)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 288)               0
_________________________________________________________________
dense (Dense)                (None, 1024)              295936
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,027,978
Trainable params: 1,027,978
Non-trainable params: 0
_________________________________________________________________
```

Пример тестирования модели на части набора данных:

In [19]:

```
# evaluating model on 10% of test dataset
pred_1, labels = model.test_on_dataset(d_test, limit=0.1)
Metrics.print_all(labels, pred_1, '10% of test')
```

```
metrics for 10% of test:
  accuracy 0.9667:
  balanced accuracy 0.9641:
```

Пример тестирования модели на полном наборе данных:

In [20]:

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

```
metrics for test:
  accuracy 0.9551:
  balanced accuracy 0.9557:
```

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 [22]:

```
final_model = Model()
final_model.load('best')
d_test_tiny = Dataset('test_tiny-2', PROJECT_DIR)
pred, labels = model.test_on_dataset(d_test_tiny)
Metrics.print_all(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 0.9667:
  balanced accuracy 0.9607:
```

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

```
Function f is caluclated 128 times in 0.03538683599981596s.
```

Scikit-learn

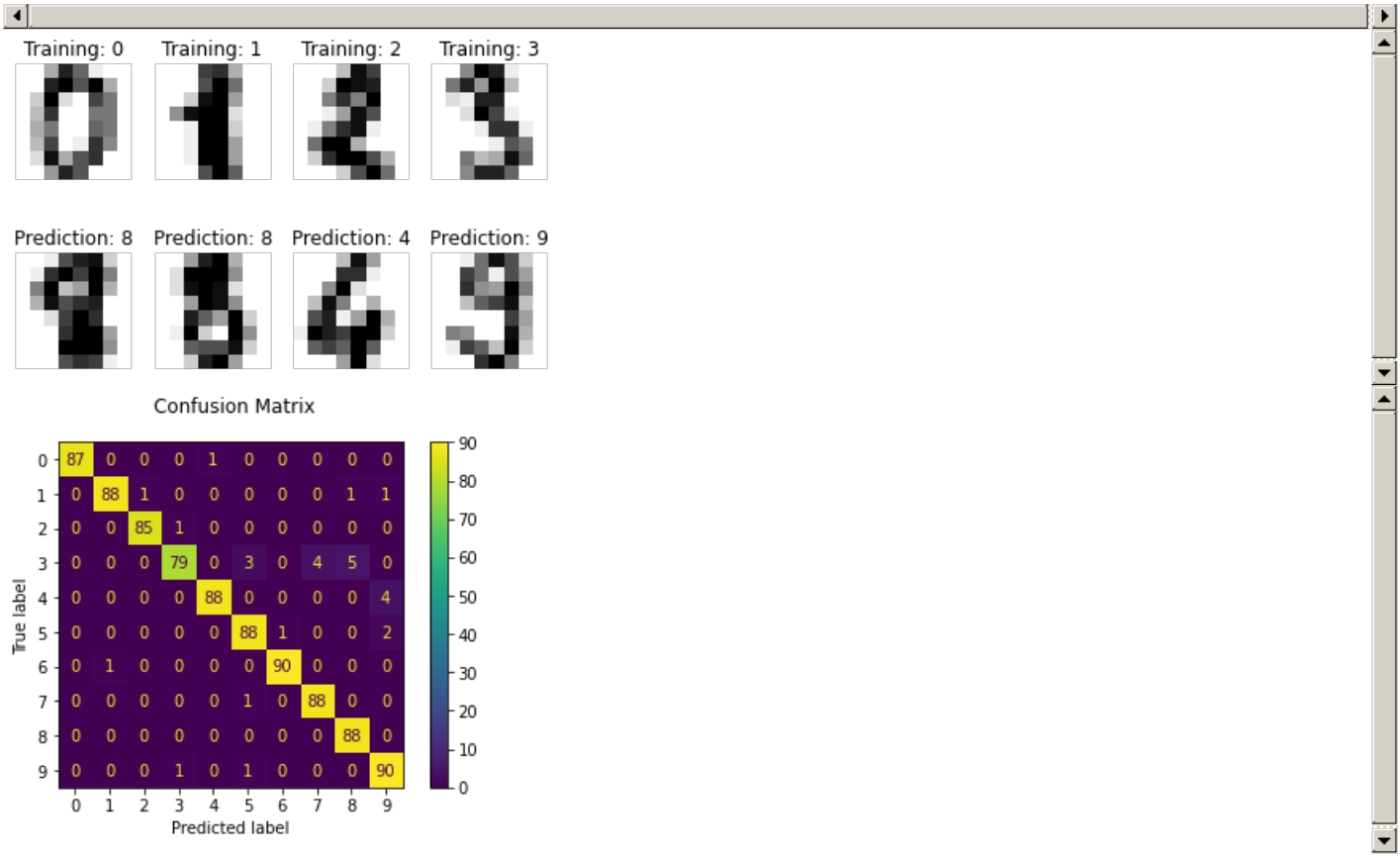
Для использования "классических" алгоритмов машинного обучения рекомендуется использовать библиотеку scikit-learn (https://scikit-learn.org/stable/). Пример классификации изображений цифр из набора данных MNIST при помощи классификатора SVM:

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=0.0,
    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]]
```



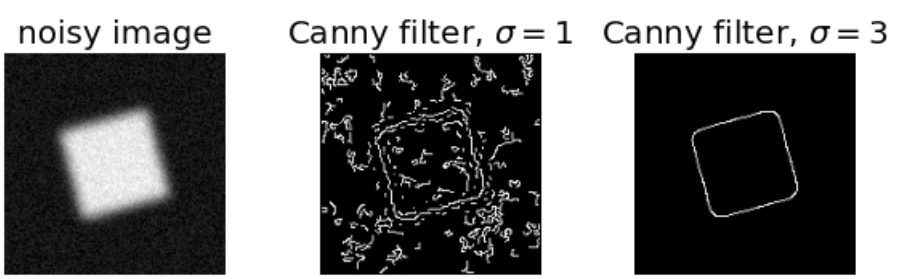
Scikit-image

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 нейроной 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 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.

Numba

В некоторых ситуациях, при ручных реализациях графовых алгоритмов, выполнение многократных вложенных циклов for в python можно существенно ускорить, используя JIT-компилятор Numba (https://numba.pydata.org/). Примеры использования Numba в Google Colab можно найти тут:

1. https://colab.research.google.com/github/cbernet/maldives/blob/master/numba/numba_cuda.ipynb
2. https://colab.research.google.com/github/evaneschneider/parallel-programming/blob/master/COMPASS_gpu_intro.ipynb

> Пожалуйста, если Вы решили использовать Numba для решения этого практического задания, еще раз подумайте, нужно ли это Вам, и есть ли возможность реализовать требуемую функциональность иным способом. Используйте Numba только при реальной необходимости.

Работа с zip архивами в Google Drive

Запаковка и распаковка zip архивов может пригодиться при сохранении и загрузки Вашей модели. Ниже приведен фрагмент кода, иллюстрирующий помещение нескольких файлов в zip архив с последующим чтением файлов из него. Все действия с директориями, файлами и архивами должны осущетвляться с примонтированным Google Drive.

Создадим 2 изображения, поместим их в директорию tmp внутри PROJECT_DIR, запакуем директорию tmp в архив tmp.zip.

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
```