Практическое задание №1

Установка необходимых пакетов:

In [1]:
```
!pip install -q tqdm
!pip install --upgrade --no-cache-dir gdown
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gdown in /usr/local/lib/python3.8/dist-packages (4.4.0)
Collecting gdown
  Downloading gdown-4.5.4-py3-none-any.whl (14 kB)
Requirement already satisfied: six in /usr/local/lib/python3.8/dist-packages (from gdown) (1.15.0)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.8/dist-packages (from gdown) (2.23.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from gdown) (4.64.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from gdown) (3.8.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.8/dist-packages (from gdown) (4.6.3)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests
[socks]->gdown) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown) (2022.
9.24)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.8/dist-packages (from requests[socks]->gdown)
(1.7.1)
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.4.0
    Uninstalling gdown-4.4.0:
      Successfully uninstalled gdown-4.4.0
Successfully installed gdown-4.5.4
```

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
```

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

In [3]:
```
EVALUATE_ONLY = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-cLi',
    'train_small': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui',
    'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr',
    'test_small': '1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ2lI',
    'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}
```

Импорт необходимых зависимостей:

In [4]:
```
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).

In [5]:
```
class Dataset:

    def __init__(self, name):
        self.name = name
        self.is_loaded = False
        url = f"https://drive.google.com/uc?export=download&confirm=pbef&id={DATASETS_LINKS[name]}"
        output = f'{name}.npz'
        gdown.download(url, output, quiet=False)
        print(f'Loading dataset {self.name} from npz.')
```

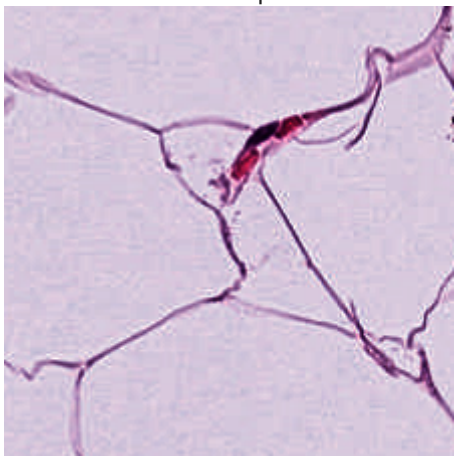
```
        np_obj = np.load(f'{name}.npz')
        self.images = np_obj['data']
        self.labels = np_obj['labels']
        self.n_files = self.images.shape[0]
        self.is_loaded = True
        print(f'Done. Dataset {name} consists of {self.n_files} images.')

    def image(self, i):
        # read i-th image in dataset and return it as numpy array
        if self.is_loaded:
            return self.images[i, :, :, :]

    def images_seq(self, n=None):
        # sequential access to images inside dataset (is needed for testing)
        for i in range(self.n_files if not n else n):
            yield self.image(i)

    def random_image_with_label(self):
        # get random image with label from dataset
        i = np.random.randint(self.n_files)
        return self.image(i), self.labels[i]

    def random_batch_with_labels(self, n):
        # create random batch of images with labels (is needed for training)
        indices = np.random.choice(self.n_files, n)
        imgs = []
        for i in indices:
            img = self.image(i)
            imgs.append(self.image(i))
        logits = np.array([self.labels[i] for i in indices])
        return np.stack(imgs), logits

    def image_with_label(self, i: int):
        # return i-th image with label from dataset
        return self.image(i), self.labels[i]
```

Пример использвания класса Dataset

Загрузим обучающий набор данных, получим произвольное изображение с меткой. После чего визуализируем изображение, выведем метку. В будущем, этот кусок кода можно закомментировать или убрать.

In [6]:
```
d_train_tiny = Dataset('train_tiny')

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

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

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 122MB/s]
Loading dataset train_tiny from npz.
Done. Dataset train_tiny consists of 900 images.

Got numpy array of shape (224, 224, 3), and label with code 0.
Label code corresponds to ADI class.
```



Класс Metrics

Реализует метрики точности, используемые для оценивания модели:

1. точность,
2. сбалансированную точность.

In [7]:
```python
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 [8]:
```
import keras
import tensorflow as tf
from keras import layers
from keras.applications import EfficientNetV2S
from sklearn.model_selection import train_test_split
from keras import callbacks
import matplotlib.pyplot as plt
```

In [9]:
```
from tensorflow.python.util.tf_export import kwarg_only
class Model:

    def __init__(self):
        # аугментация
        data_augmentation = keras.Sequential(
          [
            layers.RandomFlip(),
            layers.RandomRotation(0.2),
            layers.RandomZoom((-0.2, 0)),
          ]
        )

        self.model_1_epoch_num = 40

        self.model_1 = keras.models.Sequential()
```

```
        self.model_1.add(layers.Input(shape=(224, 224, 3)))

        # augmentation layer
        self.model_1.add(data_augmentation)

        self.model_1.add(layers.Normalization())
        self.model_1.add(EfficientNetV2S(include_top=False, input_shape=(224, 224, 3)))
        self.model_1.add(layers.Flatten())
        self.model_1.add(layers.Dense(9, activation='softmax'))

        self.model_1.compile(optimizer=keras.optimizers.Adam(),
                             loss=keras.losses.SparseCategoricalCrossentropy(from_logits=False),
                             metrics=['sparse_categorical_accuracy'])

    def save(self, name: str):
        print(f'saving started')
        self.model_1.save(f'/content/drive/MyDrive/{name}.hdf5')
        print(f'saving done')

    def load(self, name: str = 'best'):
        print(f'loading started')
        name_to_id_dict = {
            'best': '1-8StHduxeVrtZAEl-5AQIPT0YQj11Rso'
        }
        output = f'{name}.hdf5'
        gdown.download(f'https://drive.google.com/uc?id={name_to_id_dict[name]}', output, quiet=False)
        self.model_1 = tf.keras.models.load_model(f'{name}.hdf5')
        print(f'loading done')

    def loading_history(self, name: str = 'model_1_cur_logs'):
        print(f'loading started')
        name_to_id_dict = {
            'model_1_cur_logs': '1-49Mm_4ePquWz9ydjz6V54N8O7xVMc7X'
        }
        output = f'{name}.csv'
        gdown.download(f'https://drive.google.com/uc?id={name_to_id_dict[name]}', output, quiet=False)
        self.history_1 = pd.read_csv(f'{name}.csv')
        print(f'loading done')

    def train(self, dataset: Dataset, val_dataset = None):
        # you can add some plots for better visualization,
        # you can add model autosaving during training,
        # etc.
        print(f'training started')
```

```
        if val_dataset is None:
            self.train_1(dataset.images, dataset.labels)
        else:
            self.train_1(dataset.images, dataset.labels, (val_dataset.images, val_dataset.labels))
        print(f'training done')

    def train_1(self, images_train, labels_train, validation_data=None):

        # weights of different classes

        val, counts = np.unique(labels_train, return_counts=True)
        sort = np.argsort(val)
        val = val[sort]
        counts = 1 / (counts[sort] + 0.001) * labels_train.shape[0] / 2

        weights = dict(zip(val, counts))

        print("weights done")

        # splitting data

        if validation_data:
            images_train = images_train
            labels_train = labels_train
        else:
            # валидация
            images_train, images_val, labels_train, labels_val = train_test_split(images_train, labels_train,
                                                                                  test_size=0.05, random_state=42)
            validation_data = (images_val, labels_val)

        # callbacks

        initial_learning_rate = 0.0001
        def lr_exp_decay(epoch, lr):
            k = 0.1
            return initial_learning_rate * np.exp(-k*epoch)

        # сохранение модели
        checkpoint = keras.callbacks.ModelCheckpoint('/content/drive/MyDrive/model_1_cur.hdf5', monitor='val_loss', save_freq='e
                                                     verbose=1, save_best_only=True, save_weights_only=False, mode='auto')
        # выбор гиперпараметров
        # (количество эпох)
        earlystopping = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0.1, patience=10, verbose=1,
                                                      mode='auto', baseline=None)
```

```
        # (learning rate)
        lrscheduler = keras.callbacks.LearningRateScheduler(lr_exp_decay, verbose=1)
        reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=1,
                                                      mode='auto', min_delta=0.01, cooldown=0, min_lr=0)
        # сохранение показателей
        csvlogger = keras.callbacks.CSVLogger('/content/drive/MyDrive/model_1_cur_logs.csv', separator=',' , append=False)

        callbacks = [lrscheduler, checkpoint, earlystopping, reduce_lr, csvlogger]

        # train
        # вывод показателей
        # (fit выводит показатели для каждой эпохи)
        self.train = self.model_1.fit(images_train, labels_train,
                                      epochs=self.model_1_epoch_num,
                                      class_weight=weights,
                                      validation_data=validation_data,
                                      callbacks=callbacks)
        self.history_1 = self.train.history
        self.model_1.save(f'/content/drive/MyDrive/cur_trial_1.hdf5')

        print("model_1 done")

    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)
        predictions = self.test_on_dataset_1(dataset.images[:n])
        return predictions

    def test_on_dataset_1(self, images):
        predictions_1 = tf.argmax(self.model_1.predict(images), axis=1)
        return predictions_1

    def test_on_image(self, img: np.ndarray):
        # todo: replace this code
        img_array = tf.keras.utils.img_to_array(img)
        img_array = tf.expand_dims(img_array, 0)
        prediction = self.model.predict(img_array)
        return prediction

    def learning_plots(self):

          acc = self.history_1['sparse_categorical_accuracy']
          val_acc = self.history_1['val_sparse_categorical_accuracy']
```

```
        loss = self.history_1['loss']
        val_loss = self.history_1['val_loss']

        lr = self.history_1['lr']

        epochs_range = range(len(lr))

        plt.figure(figsize=(15, 5))
        plt.subplot(1, 3, 1)
        plt.plot(epochs_range, acc, label='Training Accuracy')
        plt.plot(epochs_range, val_acc, label='Validation Accuracy')
        plt.legend(loc='lower right')
        plt.title('Training and Validation Accuracy')
        plt.grid()

        plt.subplot(1, 3, 2)
        plt.plot(epochs_range, loss, label='Training Loss')
        plt.plot(epochs_range, val_loss, label='Validation Loss')
        plt.legend(loc='upper right')
        plt.title('Training and Validation Loss')
        plt.grid()

        plt.subplot(1, 3, 3)
        plt.plot(epochs_range, lr, label='Learning rate')
        plt.legend(loc='upper right')
        plt.title('Learning rate')
        plt.grid()
        plt.show()
```

In [ ]:
```
from sklearn.model_selection import KFold
from sklearn.metrics import balanced_accuracy_score
from collections import defaultdict
```

In [ ]:
```
# кросс-валидация
def cross_val(dataset: Dataset, n_splits=3):
    X = dataset.images
    y = dataset.labels

    scorer = balanced_accuracy_score
    cv = KFold(n_splits=n_splits, shuffle=True, random_state=42)
    res=[]

    t = cv.split(X)
```

```
    for train_i, test_i in t:
        X_train = X[train_i]
        y_train = y[train_i]
        X_test = X[test_i]
        y_test = y[test_i]
        model = Model()
        model.train_1(images_train=X_train, labels_train=y_train, validation_data=(X_test, y_test))
        pred = model.test_on_dataset_1(X_test)
        res.append(scorer(y_test, pred))

    splits_num = range(len(res))

    plt.figure(figsize=(4, 3))
    plt.plot(splits_num, res, label='accuracy')
    plt.legend(loc='lower right')
    plt.title('Cross validation accuracy')
    plt.ylim([0,1])
    plt.grid()
    plt.show()

    return res
```

Классификация изображений

Классификация изображений. Используем полные датасеты `train`, `test`.

In [ ]:
```
d_train = Dataset('train')
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=18bNP8R_k8FYq0Gx0DnC2j_UEOaGqVl14
To: /content/train.npz
100%|██████████| 2.10G/2.10G [00:09<00:00, 218MB/s]
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
```

Обучение модели

Используется аугментация данных стандартными функциями библиотеки `keras.layers`. Далее используется архитектура `EfficientNetV2S` с уже обученными на `ImageNet` слоями. После нее добавлен обычный полносвязный слой с 9 выходами и функцией активации `softmax` для окончательной классификации.

Модель обучена на 14 эпохах.

Использованы `callback`-функции:

- LearningRateScheduler - `learning rate` зависит от эпохи (экспоненциально)
- ModelCheckpoint - периодически сохраняет модель (только с лучшими весами)
- EarlyStopping - ранняя остановка для недопуска переобучения
- ReduceLROnPlateau - уменьшить `learning rate`, если лосс выходит на плато
- CSVLogger - сохранение информации об обучении (`loss`, `accuracy`, ...) в формате `.csv`

In [ ]:
```
model = Model()
if not EVALUATE_ONLY:
    model.train(d_train)
    model.save('best')
else:
    model.load('best')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-s_notop.h5
82420632/82420632 [==============================] - 5s 0us/step
training started
weights done

Epoch 1: LearningRateScheduler setting learning rate to 0.0001.
Epoch 1/40
535/535 [==============================] - ETA: 0s - loss: 1.4519 - sparse_categorical_accuracy: 0.8936
Epoch 1: val_loss improved from inf to 0.15565, saving model to /content/drive/MyDrive/model_1_cur.hdf5
535/535 [==============================] - 296s 497ms/step - loss: 1.4519 - sparse_categorical_accuracy: 0.8936 - val_loss: 0.15
56 - val_sparse_categorical_accuracy: 0.9544 - lr: 1.0000e-04

Epoch 2: LearningRateScheduler setting learning rate to 9.048374180359596e-05.
Epoch 2/40
535/535 [==============================] - ETA: 0s - loss: 0.5369 - sparse_categorical_accuracy: 0.9618
Epoch 2: val_loss improved from 0.15565 to 0.10841, saving model to /content/drive/MyDrive/model_1_cur.hdf5
535/535 [==============================] - 263s 491ms/step - loss: 0.5369 - sparse_categorical_accuracy: 0.9618 - val_loss: 0.10
84 - val_sparse_categorical_accuracy: 0.9633 - lr: 9.0484e-05

Epoch 3: LearningRateScheduler setting learning rate to 8.187307530779819e-05.
Epoch 3/40
535/535 [==============================] - ETA: 0s - loss: 0.3666 - sparse_categorical_accuracy: 0.9727
Epoch 3: val_loss improved from 0.10841 to 0.06697, saving model to /content/drive/MyDrive/model_1_cur.hdf5
535/535 [==============================] - 262s 490ms/step - loss: 0.3666 - sparse_categorical_accuracy: 0.9727 - val_loss: 0.06
70 - val_sparse_categorical_accuracy: 0.9767 - lr: 8.1873e-05

Epoch 4: LearningRateScheduler setting learning rate to 7.408182206817179e-05.
Epoch 4/40
535/535 [==============================] - ETA: 0s - loss: 0.2882 - sparse_categorical_accuracy: 0.9810
Epoch 4: val_loss improved from 0.06697 to 0.04043, saving model to /content/drive/MyDrive/model_1_cur.hdf5
535/535 [==============================] - 264s 494ms/step - loss: 0.2882 - sparse_categorical_accuracy: 0.9810 - val_loss: 0.04
04 - val_sparse_categorical_accuracy: 0.9856 - lr: 7.4082e-05

Epoch 5: LearningRateScheduler setting learning rate to 6.703200460356394e-05.
Epoch 5/40
535/535 [==============================] - ETA: 0s - loss: 0.2172 - sparse_categorical_accuracy: 0.9851
Epoch 5: val_loss did not improve from 0.04043
535/535 [==============================] - 258s 482ms/step - loss: 0.2172 - sparse_categorical_accuracy: 0.9851 - val_loss: 0.07
34 - val_sparse_categorical_accuracy: 0.9733 - lr: 6.7032e-05

Epoch 6: LearningRateScheduler setting learning rate to 6.065306597126335e-05.
Epoch 6/40
535/535 [==============================] - ETA: 0s - loss: 0.1813 - sparse_categorical_accuracy: 0.9874
Epoch 6: val_loss did not improve from 0.04043
```

```
535/535 [==============================] - 258s 482ms/step - loss: 0.1813 - sparse_categorical_accuracy: 0.9874 - val_loss: 0.07
29 - val_sparse_categorical_accuracy: 0.9767 - lr: 6.0653e-05

Epoch 7: LearningRateScheduler setting learning rate to 5.488116360940264e-05.
Epoch 7/40
535/535 [==============================] - ETA: 0s - loss: 0.1367 - sparse_categorical_accuracy: 0.9905
Epoch 7: val_loss did not improve from 0.04043
535/535 [==============================] - 258s 482ms/step - loss: 0.1367 - sparse_categorical_accuracy: 0.9905 - val_loss: 0.08
83 - val_sparse_categorical_accuracy: 0.9756 - lr: 5.4881e-05

Epoch 8: LearningRateScheduler setting learning rate to 4.965853037914095e-05.
Epoch 8/40
535/535 [==============================] - ETA: 0s - loss: 0.1493 - sparse_categorical_accuracy: 0.9899
Epoch 8: val_loss did not improve from 0.04043
535/535 [==============================] - 257s 481ms/step - loss: 0.1493 - sparse_categorical_accuracy: 0.9899 - val_loss: 0.07
38 - val_sparse_categorical_accuracy: 0.9822 - lr: 4.9659e-05

Epoch 9: LearningRateScheduler setting learning rate to 4.493289641172216e-05.
Epoch 9/40
535/535 [==============================] - ETA: 0s - loss: 0.1060 - sparse_categorical_accuracy: 0.9921
Epoch 9: val_loss improved from 0.04043 to 0.03137, saving model to /content/drive/MyDrive/model_1_cur.hdf5

Epoch 9: ReduceLROnPlateau reducing learning rate to 2.246644908154849e-05.
535/535 [==============================] - 263s 492ms/step - loss: 0.1060 - sparse_categorical_accuracy: 0.9921 - val_loss: 0.03
14 - val_sparse_categorical_accuracy: 0.9878 - lr: 4.4933e-05

Epoch 10: LearningRateScheduler setting learning rate to 4.0656965974059915e-05.
Epoch 10/40
535/535 [==============================] - ETA: 0s - loss: 0.0990 - sparse_categorical_accuracy: 0.9928
Epoch 10: val_loss did not improve from 0.03137
535/535 [==============================] - 259s 485ms/step - loss: 0.0990 - sparse_categorical_accuracy: 0.9928 - val_loss: 0.04
64 - val_sparse_categorical_accuracy: 0.9856 - lr: 4.0657e-05

Epoch 11: LearningRateScheduler setting learning rate to 3.678794411714424e-05.
Epoch 11/40
535/535 [==============================] - ETA: 0s - loss: 0.0922 - sparse_categorical_accuracy: 0.9932
Epoch 11: val_loss did not improve from 0.03137
535/535 [==============================] - 257s 481ms/step - loss: 0.0922 - sparse_categorical_accuracy: 0.9932 - val_loss: 0.03
68 - val_sparse_categorical_accuracy: 0.9911 - lr: 3.6788e-05

Epoch 12: LearningRateScheduler setting learning rate to 3.3287108369807955e-05.
Epoch 12/40
535/535 [==============================] - ETA: 0s - loss: 0.0587 - sparse_categorical_accuracy: 0.9953
Epoch 12: val_loss did not improve from 0.03137
```

```
535/535 [==============================] - 258s 481ms/step - loss: 0.0587 - sparse_categorical_accuracy: 0.9953 - val_loss: 0.05
00 - val_sparse_categorical_accuracy: 0.9889 - lr: 3.3287e-05

Epoch 13: LearningRateScheduler setting learning rate to 3.0119421191220204e-05.
Epoch 13/40
535/535 [==============================] - ETA: 0s - loss: 0.0620 - sparse_categorical_accuracy: 0.9954
Epoch 13: val_loss did not improve from 0.03137
535/535 [==============================] - 259s 485ms/step - loss: 0.0620 - sparse_categorical_accuracy: 0.9954 - val_loss: 0.03
23 - val_sparse_categorical_accuracy: 0.9878 - lr: 3.0119e-05

Epoch 14: LearningRateScheduler setting learning rate to 2.725317930340126e-05.
Epoch 14/40
535/535 [==============================] - ETA: 0s - loss: 0.0576 - sparse_categorical_accuracy: 0.9958
Epoch 14: val_loss did not improve from 0.03137

Epoch 14: ReduceLROnPlateau reducing learning rate to 1.3626589861814864e-05.
535/535 [==============================] - 260s 485ms/step - loss: 0.0576 - sparse_categorical_accuracy: 0.9958 - val_loss: 0.04
26 - val_sparse_categorical_accuracy: 0.9833 - lr: 2.7253e-05
Epoch 14: early stopping
model_1 done
training done
saving started
saving done
```

In [ ]:

Тестируем на датасете test

In [ ]:
```
d_test = Dataset('test')
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=14lLsHGRxcTlMkX2Y5j9KxH1W62ZjSiye
To: /content/test.npz
100%|██████████| 525M/525M [00:11<00:00, 44.6MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
```

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

```
141/141 [==============================] - 20s 124ms/step
metrics for full test:
         accuracy 0.9916:
         balanced accuracy 0.9916:
```

Чтение данных

Есть возможность считывать веса и данные обучения лучшей модели.

In [ ]:
```
model_1 = Model()
model_1.load()
```

```
loading started
Downloading...
From: https://drive.google.com/uc?id=1-8StHduxeVrtZAEl-5AQIPT0YQj11Rso
To: /content/best.hdf5
100%|██████████| 251M/251M [00:01<00:00, 152MB/s]
loading done
```

In [ ]:
```
pred_1 = model_1.test_on_dataset(d_test, limit=1)
Metrics.print_all(d_test.labels[:len(pred_1)], pred_1, 'full test')
```

```
141/141 [==============================] - 20s 125ms/step
metrics for full test:
         accuracy 0.9916:
         balanced accuracy 0.9916:
```

In [ ]:
```
import pandas as pd
```

In [ ]:
```
model_1.loading_history()
```

```
loading started
Downloading...
From: https://drive.google.com/uc?id=1-49Mm_4ePquWz9ydjz6V54N8O7xVMc7X
To: /content/model_1_cur_logs.csv
100%|██████████| 1.40k/1.40k [00:00<00:00, 2.91MB/s]
loading done
```

In [ ]:
```
model_1.history_1.head()
```

Out[ ]:

	epoch	loss	lr	sparse_categorical_accuracy	val_loss	val_sparse_categorical_accuracy
0	0	1.451855	0.000100	0.893626	0.155648	0.954444
1	1	0.536886	0.000090	0.961813	0.108412	0.963333
2	2	0.366590	0.000082	0.972749	0.066970	0.976667
3	3	0.288158	0.000074	0.980994	0.040435	0.985556
4	4	0.217177	0.000067	0.985146	0.073438	0.973333

Визуализации

Матрица ошибок для выборки `test`.

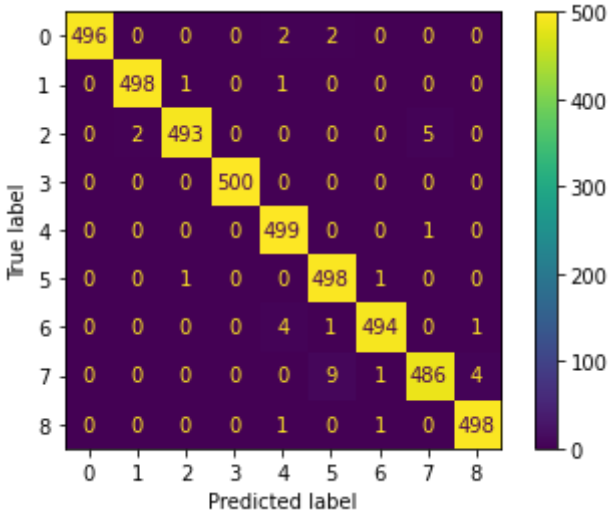
In [10]:
```
final_model = Model()
final_model.load('best')
d_test = Dataset('test')
pred_1 = final_model.test_on_dataset(d_test)
Metrics.print_all(d_test.labels, pred_1, 'test')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/efficientnet_v2/efficientnetv2-s_notop.h5
82420632/82420632 [==============================] - 1s 0us/step
loading started
Downloading...
From: https://drive.google.com/uc?id=1-8StHduxeVrtZAEl-5AQIPT0YQj11Rso
To: /content/best.hdf5
100%|██████████| 251M/251M [00:01<00:00, 127MB/s]
loading done
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr
To: /content/test.npz
100%|██████████| 525M/525M [00:04<00:00, 107MB/s]
```

```
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
141/141 [==============================] - 720s 5s/step
metrics for test:
         accuracy 0.9916:
         balanced accuracy 0.9916:
```

In [11]:
```
# матрица ошибок
import sklearn
conf_matr = sklearn.metrics.confusion_matrix(d_test.labels[:len(pred_1)], pred_1)
disp = sklearn.metrics.ConfusionMatrixDisplay(conf_matr)
disp.plot()
```

Out[11]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7feae57271c0>
```



In [ ]:
```
mcm = sklearn.metrics.multilabel_confusion_matrix(d_test.labels[:len(pred_1)], pred_1)
tn = mcm[:, 0, 0]
tp = mcm[:, 1, 1]
fn = mcm[:, 1, 0]
fp = mcm[:, 0, 1]
print("Sensitivity for each label:")
print(tp / (tp + fn))
print("Specificity for each label:")
print(tn / (tn + fp))
```

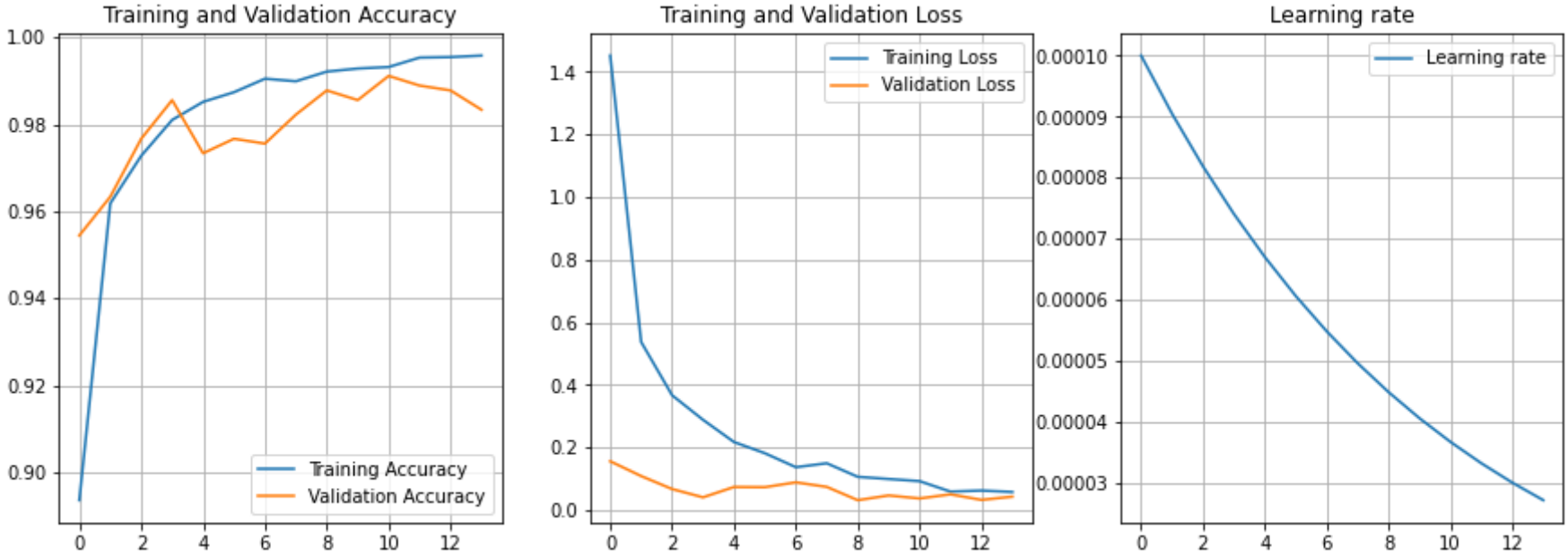
```
Sensitivity for each label:
[0.992 0.996 0.986 1.    0.998 0.996 0.988 0.972 0.996]
Specificity for each label:
[1.      0.9995  0.9995  1.      0.998   0.997   0.99925 0.9985  0.99875]
```

Параметры (`loss` , `accuracy` , `learning rate`) в процессе обучения

In [ ]:
```
final_model.loading_history()
```

```
loading started
Downloading...
From: https://drive.google.com/uc?id=1-49Mm_4ePquWz9ydjz6V54N8O7xVMc7X
To: /content/model_1_cur_logs.csv
100%|██████████| 1.40k/1.40k [00:00<00:00, 3.68MB/s]
loading done
```

In [ ]:
```
# графики обучения
final_model.learning_plots()
```



In [ ]:

Кросс-валидация

Написана также функция кросс-валидации для модели.

Ниже приведен пример работы функции. На полном датасете проводить кросс-валидацию слишком долго, и, в общем-то, не особо имеет смысл.

Модель обучается на 5 эпохах на датасете train_tiny.

In [ ]:
```
d_train_cross = Dataset('train_tiny')
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1zdQ4BoKXR-bRU0j0_ja5LeDjl_bPtbGO
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 112MB/s]
Loading dataset train_tiny from npz.
Done. Dataset train_tiny consists of 900 images.
```

In [ ]:
```
cross_val_scores = cross_val(dataset=d_train_cross, n_splits=5)
```

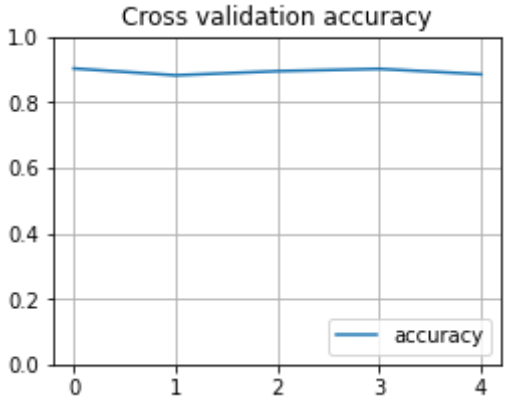
```
weights done

Epoch 1: LearningRateScheduler setting learning rate to 0.001.
Epoch 1/5
23/23 [==============================] - ETA: 0s - loss: 6.3189 - sparse_categorical_accuracy: 0.6583
Epoch 1: val_loss improved from inf to 1.20344, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 41s 907ms/step - loss: 6.3189 - sparse_categorical_accuracy: 0.6583 - val_loss: 1.2034
- val_sparse_categorical_accuracy: 0.8111 - lr: 0.0010

Epoch 2: LearningRateScheduler setting learning rate to 0.0009048374180359595.
Epoch 2/5
23/23 [==============================] - ETA: 0s - loss: 4.0421 - sparse_categorical_accuracy: 0.8139
Epoch 2: val_loss improved from 1.20344 to 0.46617, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 16s 696ms/step - loss: 4.0421 - sparse_categorical_accuracy: 0.8139 - val_loss: 0.4662
- val_sparse_categorical_accuracy: 0.8889 - lr: 9.0484e-04

Epoch 3: LearningRateScheduler setting learning rate to 0.0008187307530779819.
Epoch 3/5
23/23 [==============================] - ETA: 0s - loss: 2.7973 - sparse_categorical_accuracy: 0.8597
Epoch 3: val_loss improved from 0.46617 to 0.42818, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 16s 690ms/step - loss: 2.7973 - sparse_categorical_accuracy: 0.8597 - val_loss: 0.4282
- val_sparse_categorical_accuracy: 0.9056 - lr: 8.1873e-04

Epoch 4: LearningRateScheduler setting learning rate to 0.0007408182206817179.
Epoch 4/5
23/23 [==============================] - ETA: 0s - loss: 2.2126 - sparse_categorical_accuracy: 0.8972
Epoch 4: val_loss did not improve from 0.42818
23/23 [==============================] - 12s 519ms/step - loss: 2.2126 - sparse_categorical_accuracy: 0.8972 - val_loss: 1.0566
- val_sparse_categorical_accuracy: 0.8222 - lr: 7.4082e-04

Epoch 5: LearningRateScheduler setting learning rate to 0.0006703200460356394.
Epoch 5/5
23/23 [==============================] - ETA: 0s - loss: 2.3558 - sparse_categorical_accuracy: 0.9000
Epoch 5: val_loss did not improve from 0.42818
23/23 [==============================] - 12s 517ms/step - loss: 2.3558 - sparse_categorical_accuracy: 0.9000 - val_loss: 0.5252
- val_sparse_categorical_accuracy: 0.9056 - lr: 6.7032e-04
model_1 done
6/6 [==============================] - 3s 129ms/step
weights done

Epoch 1: LearningRateScheduler setting learning rate to 0.001.
Epoch 1/5
23/23 [==============================] - ETA: 0s - loss: 7.4492 - sparse_categorical_accuracy: 0.6431
Epoch 1: val_loss improved from inf to 1.53536, saving model to /content/drive/MyDrive/model_1_cur.hdf5
```

```
23/23 [==============================] - 43s 853ms/step - loss: 7.4492 - sparse_categorical_accuracy: 0.6431 - val_loss: 1.5354
- val_sparse_categorical_accuracy: 0.7167 - lr: 0.0010

Epoch 2: LearningRateScheduler setting learning rate to 0.0009048374180359595.
Epoch 2/5
23/23 [==============================] - ETA: 0s - loss: 6.6049 - sparse_categorical_accuracy: 0.7667
Epoch 2: val_loss improved from 1.53536 to 1.46090, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 16s 723ms/step - loss: 6.6049 - sparse_categorical_accuracy: 0.7667 - val_loss: 1.4609
- val_sparse_categorical_accuracy: 0.7167 - lr: 9.0484e-04

Epoch 3: LearningRateScheduler setting learning rate to 0.0008187307530779819.
Epoch 3/5
23/23 [==============================] - ETA: 0s - loss: 3.4674 - sparse_categorical_accuracy: 0.8292
Epoch 3: val_loss improved from 1.46090 to 0.70322, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 16s 708ms/step - loss: 3.4674 - sparse_categorical_accuracy: 0.8292 - val_loss: 0.7032
- val_sparse_categorical_accuracy: 0.8944 - lr: 8.1873e-04

Epoch 4: LearningRateScheduler setting learning rate to 0.0007408182206817179.
Epoch 4/5
23/23 [==============================] - ETA: 0s - loss: 4.0621 - sparse_categorical_accuracy: 0.8264
Epoch 4: val_loss did not improve from 0.70322
23/23 [==============================] - 12s 519ms/step - loss: 4.0621 - sparse_categorical_accuracy: 0.8264 - val_loss: 0.8855
- val_sparse_categorical_accuracy: 0.8833 - lr: 7.4082e-04

Epoch 5: LearningRateScheduler setting learning rate to 0.0006703200460356394.
Epoch 5/5
23/23 [==============================] - ETA: 0s - loss: 3.3155 - sparse_categorical_accuracy: 0.8653
Epoch 5: val_loss did not improve from 0.70322
23/23 [==============================] - 12s 515ms/step - loss: 3.3155 - sparse_categorical_accuracy: 0.8653 - val_loss: 0.8769
- val_sparse_categorical_accuracy: 0.8889 - lr: 6.7032e-04
model_1 done
6/6 [==============================] - 4s 133ms/step
weights done

Epoch 1: LearningRateScheduler setting learning rate to 0.001.
Epoch 1/5
23/23 [==============================] - ETA: 0s - loss: 7.0102 - sparse_categorical_accuracy: 0.6458
Epoch 1: val_loss improved from inf to 2.09458, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 41s 918ms/step - loss: 7.0102 - sparse_categorical_accuracy: 0.6458 - val_loss: 2.0946
- val_sparse_categorical_accuracy: 0.7167 - lr: 0.0010

Epoch 2: LearningRateScheduler setting learning rate to 0.0009048374180359595.
Epoch 2/5
23/23 [==============================] - ETA: 0s - loss: 5.1201 - sparse_categorical_accuracy: 0.7972
```

```
Epoch 2: val_loss improved from 2.09458 to 1.53172, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 16s 707ms/step - loss: 5.1201 - sparse_categorical_accuracy: 0.7972 - val_loss: 1.5317
- val_sparse_categorical_accuracy: 0.6389 - lr: 9.0484e-04

Epoch 3: LearningRateScheduler setting learning rate to 0.0008187307530779819.
Epoch 3/5
23/23 [==============================] - ETA: 0s - loss: 3.7074 - sparse_categorical_accuracy: 0.8194
Epoch 3: val_loss improved from 1.53172 to 1.25236, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 17s 728ms/step - loss: 3.7074 - sparse_categorical_accuracy: 0.8194 - val_loss: 1.2524
- val_sparse_categorical_accuracy: 0.8389 - lr: 8.1873e-04

Epoch 4: LearningRateScheduler setting learning rate to 0.0007408182206817179.
Epoch 4/5
23/23 [==============================] - ETA: 0s - loss: 3.6319 - sparse_categorical_accuracy: 0.8514
Epoch 4: val_loss improved from 1.25236 to 0.80076, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 17s 725ms/step - loss: 3.6319 - sparse_categorical_accuracy: 0.8514 - val_loss: 0.8008
- val_sparse_categorical_accuracy: 0.8278 - lr: 7.4082e-04

Epoch 5: LearningRateScheduler setting learning rate to 0.0006703200460356394.
Epoch 5/5
23/23 [==============================] - ETA: 0s - loss: 2.0340 - sparse_categorical_accuracy: 0.9208
Epoch 5: val_loss did not improve from 0.80076
23/23 [==============================] - 12s 514ms/step - loss: 2.0340 - sparse_categorical_accuracy: 0.9208 - val_loss: 0.8178
- val_sparse_categorical_accuracy: 0.8944 - lr: 6.7032e-04
model_1 done
6/6 [==============================] - 4s 131ms/step
weights done

Epoch 1: LearningRateScheduler setting learning rate to 0.001.
Epoch 1/5
23/23 [==============================] - ETA: 0s - loss: 6.7301 - sparse_categorical_accuracy: 0.6403
Epoch 1: val_loss improved from inf to 1.83547, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 42s 939ms/step - loss: 6.7301 - sparse_categorical_accuracy: 0.6403 - val_loss: 1.8355
- val_sparse_categorical_accuracy: 0.7056 - lr: 0.0010

Epoch 2: LearningRateScheduler setting learning rate to 0.0009048374180359595.
Epoch 2/5
23/23 [==============================] - ETA: 0s - loss: 4.8875 - sparse_categorical_accuracy: 0.8264
Epoch 2: val_loss did not improve from 1.83547
23/23 [==============================] - 12s 542ms/step - loss: 4.8875 - sparse_categorical_accuracy: 0.8264 - val_loss: 1.8554
- val_sparse_categorical_accuracy: 0.7889 - lr: 9.0484e-04

Epoch 3: LearningRateScheduler setting learning rate to 0.0008187307530779819.
Epoch 3/5
```

```
23/23 [==============================] - ETA: 0s - loss: 3.6090 - sparse_categorical_accuracy: 0.8458
Epoch 3: val_loss improved from 1.83547 to 1.31485, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 17s 749ms/step - loss: 3.6090 - sparse_categorical_accuracy: 0.8458 - val_loss: 1.3149
- val_sparse_categorical_accuracy: 0.8000 - lr: 8.1873e-04

Epoch 4: LearningRateScheduler setting learning rate to 0.0007408182206817179.
Epoch 4/5
23/23 [==============================] - ETA: 0s - loss: 2.7764 - sparse_categorical_accuracy: 0.8625
Epoch 4: val_loss improved from 1.31485 to 0.70417, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 16s 711ms/step - loss: 2.7764 - sparse_categorical_accuracy: 0.8625 - val_loss: 0.7042
- val_sparse_categorical_accuracy: 0.8722 - lr: 7.4082e-04

Epoch 5: LearningRateScheduler setting learning rate to 0.0006703200460356394.
Epoch 5/5
23/23 [==============================] - ETA: 0s - loss: 1.6747 - sparse_categorical_accuracy: 0.9097
Epoch 5: val_loss improved from 0.70417 to 0.55556, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 16s 702ms/step - loss: 1.6747 - sparse_categorical_accuracy: 0.9097 - val_loss: 0.5556
- val_sparse_categorical_accuracy: 0.9000 - lr: 6.7032e-04
model_1 done
6/6 [==============================] - 4s 134ms/step
weights done

Epoch 1: LearningRateScheduler setting learning rate to 0.001.
Epoch 1/5
23/23 [==============================] - ETA: 0s - loss: 6.6773 - sparse_categorical_accuracy: 0.6542
Epoch 1: val_loss improved from inf to 2.14505, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 40s 885ms/step - loss: 6.6773 - sparse_categorical_accuracy: 0.6542 - val_loss: 2.1450
- val_sparse_categorical_accuracy: 0.6889 - lr: 0.0010

Epoch 2: LearningRateScheduler setting learning rate to 0.0009048374180359595.
Epoch 2/5
23/23 [==============================] - ETA: 0s - loss: 5.5608 - sparse_categorical_accuracy: 0.7861
Epoch 2: val_loss did not improve from 2.14505
23/23 [==============================] - 12s 515ms/step - loss: 5.5608 - sparse_categorical_accuracy: 0.7861 - val_loss: 2.2032
- val_sparse_categorical_accuracy: 0.7333 - lr: 9.0484e-04

Epoch 3: LearningRateScheduler setting learning rate to 0.0008187307530779819.
Epoch 3/5
23/23 [==============================] - ETA: 0s - loss: 4.3022 - sparse_categorical_accuracy: 0.8028
Epoch 3: val_loss improved from 2.14505 to 0.59682, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 16s 722ms/step - loss: 4.3022 - sparse_categorical_accuracy: 0.8028 - val_loss: 0.5968
- val_sparse_categorical_accuracy: 0.8278 - lr: 8.1873e-04

Epoch 4: LearningRateScheduler setting learning rate to 0.0007408182206817179.
```

```
Epoch 4/5
23/23 [==============================] - ETA: 0s - loss: 2.3015 - sparse_categorical_accuracy: 0.8556
Epoch 4: val_loss improved from 0.59682 to 0.58581, saving model to /content/drive/MyDrive/model_1_cur.hdf5
23/23 [==============================] - 17s 725ms/step - loss: 2.3015 - sparse_categorical_accuracy: 0.8556 - val_loss: 0.5858
- val_sparse_categorical_accuracy: 0.9056 - lr: 7.4082e-04

Epoch 5: LearningRateScheduler setting learning rate to 0.0006703200460356394.
Epoch 5/5
23/23 [==============================] - ETA: 0s - loss: 2.0464 - sparse_categorical_accuracy: 0.8986
Epoch 5: val_loss did not improve from 0.58581
23/23 [==============================] - 12s 542ms/step - loss: 2.0464 - sparse_categorical_accuracy: 0.8986 - val_loss: 0.7394
- val_sparse_categorical_accuracy: 0.8722 - lr: 6.7032e-04
model_1 done
6/6 [==============================] - 4s 132ms/step
```



Использование аугментации данных

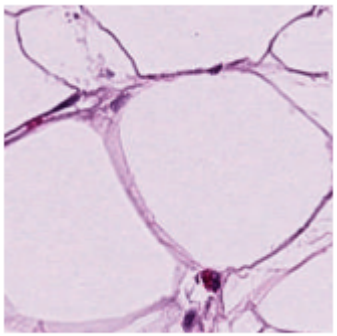
В нейронной сети используется слой аугментации данных (поворот вертикальный/горизонтальный, поворот на угол, увеличение масштаба изображения).

Выбраны параметры, не слишком сильно увеличивающие изображение (чтобы не терять в качестве). Zoom выбран исключительно отрицательным чтобы не дополнять изображение паддингом (т.к. тогда создаются зеркальные края, не всегда отражающие реальную структуру ткани).

In [ ]:
```
import keras
from keras import layers
```

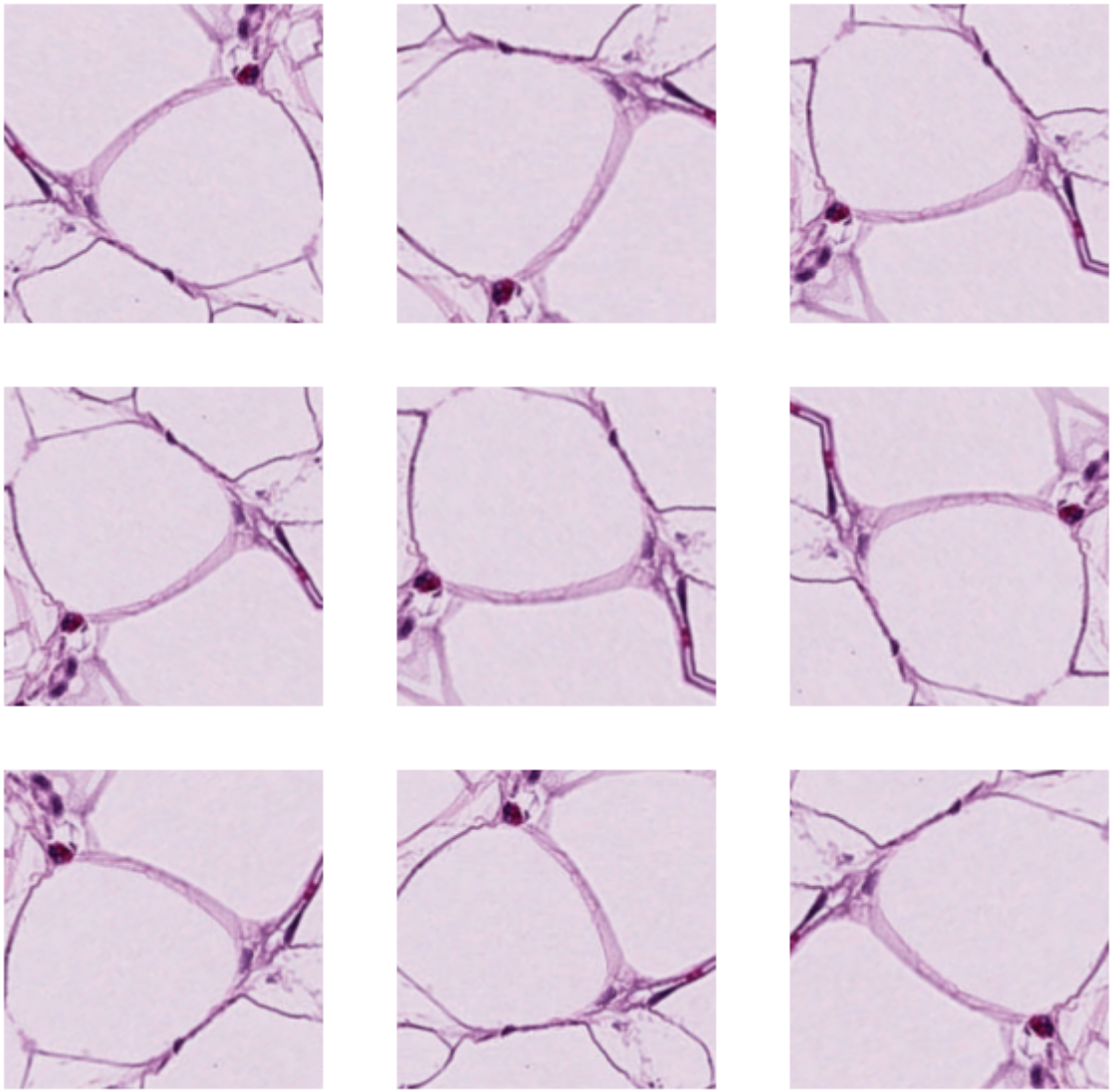
```
import matplotlib.pyplot as plt

plt.figure(figsize=(3, 3))
images, lbl = d_train.random_batch_with_labels(1)
print(images.shape)
ax = plt.subplot(1, 1, 1)
plt.imshow(images[0].astype("uint8"))
plt.axis("off")
```

Out[ ]:
```
(1, 224, 224, 3)
(-0.5, 223.5, 223.5, -0.5)
```



In [ ]:
```
data_augmentation = tf.keras.Sequential(
          [
            tf.keras.layers.RandomFlip(),
            tf.keras.layers.RandomRotation(0.2),
            tf.keras.layers.RandomZoom((-0.2, 0)),
          ]
        )
data_augmentation.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])
```

In [ ]:
```
plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_images = data_augmentation(images, 0.1)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_images[0].numpy().astype("uint8"))
  plt.axis("off")
```



In [ ]:

Пример тестирования модели на полном наборе данных:

In [ ]:
```
# evaluating model on full test dataset (may take time)
if TEST_ON_LARGE_DATASET:
```

```
    pred_2 = model.test_on_dataset(d_test)
    Metrics.print_all(d_test.labels, pred_2, 'test')
```

Результат работы пайплайна обучения и тестирования выше тоже будет оцениваться. Поэтому не забудьте присылать на проверку ноутбук с выполненными ячейками кода с демонстрациями метрик обучения, графиками и т.п. В этом пайплайне Вам необходимо продемонстрировать работу всех реализованных дополнений, улучшений и т.п.

Настоятельно рекомендуется после получения пайплайна с полными результатами обучения экспортировать ноутбук в pdf (файл -> печать) и прислать этот pdf вместе с самим ноутбуком.

Тестирование модели на других наборах данных

Ваша модель должна поддерживать тестирование на других наборах данных. Для удобства, Вам предоставляется набор данных test_tiny, который представляет собой малую часть (2% изображений) набора test. Ниже приведен фрагмент кода, который будет осуществлять тестирование для оценивания Вашей модели на дополнительных тестовых наборах данных.

Прежде чем отсылать задание на проверку, убедитесь в работоспособности фрагмента кода ниже.

In [ ]:
```
final_model = Model()
final_model.load('best')
d_test_tiny = Dataset('test_tiny')
pred = final_model.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, 'test_tiny')
```

```
loading started
Downloading...
From: https://drive.google.com/uc?id=1-8StHduxeVrtZAEl-5AQIPT0YQj11Rso
To: /content/best.hdf5
100%|██████████| 251M/251M [00:00<00:00, 293MB/s]
loading done
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1JcDlKmFQ-ohdFVMQDINGY2EauuSGSVgv
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 16.6MB/s]
```

```
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
3/3 [==============================] - 4s 484ms/step
metrics for test_tiny:
         accuracy 0.9889:
         balanced accuracy 0.9889:
```

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.')
```

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()
```

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
import cv2
from skimage import feature


# Generate noisy image of a square
im = d_train.images[np.random.randint(d_train.images.shape[0])][:,:,0]

# Compute the Canny filter for two values of sigma
edges1 = feature.canny(im)
edges2 = feature.canny(im, sigma=2)
ridge_filter = cv2.ximgproc.RidgeDetectionFilter_create()
edges3 = ridge_filter.getRidgeFilteredImage(im)
# display results
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(8, 3),
```

```
                                    sharex=True, sharey=True)

fig.set_size_inches(10, 10)

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, $\sigma=2$', fontsize=20)

ax4.imshow(edges3, cmap=plt.cm.gray)
ax4.axis('off')
ax4.set_title(r'Canny filter, $\sigma=3$', fontsize=20)

fig.tight_layout()

plt.show()
```

In [ ]:
```
plt.figure(figsize=(3, 3))
images, lbl = d_train.random_batch_with_labels(1)
print(images.shape)
ax = plt.subplot(1, 1, 1)
plt.imshow(images[0].astype("uint8"))
plt.axis("off")
```

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)
```

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 осуществляться с примонтированным Google Drive.

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

In [ ]:
```
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.

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