Практическое задание №1

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

```
    Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.7.1)
    Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.13.1)
    Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
    Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from gdown) (1.16.0)
    Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
    Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
    Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.4)
    Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.7
    Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.
```

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

```
    Mounted at /content/drive
```

```
EVALUATE_ONLY = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    # Закоментированные строки из оригинального ноутбука
    # К сожалению оригинальные ссылки не работают из-за большого кол-ва скачиваний


    # 'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-cLi',
    'train': '1ccAgGUs43hA6hf9rpV8fi84VLv_2uW8a',
    # 'train_small': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_small': '14bpdxgb55YzBuVGORq3imnLadTIuKTEo',
    # 'train_tiny': '1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui',
    'train_tiny': '18jKz6GfnilfIYZHT-sASvPfU1BH6p2OU',
    # 'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr',
    'test': '1brH5TzbTNUPKz3yoWS_RD4FW1xJc-dEK',
    # 'test_small': '1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ2lI',
    'test_small': '1FAULgTFgf-60lziVOGFABmveXcrOKFUH',
    # 'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
    'test_tiny': '1bOavoin0mTiBhx8AYZhIkAa3YhEinbLa'
}
IMG_HEIGHT = 224
IMG_WIDTH = 224

BATCH_SIZE = 16
SHUFFLE_BUFFER_SIZE = 100
TRAIN_RATIO = 0.8
```

```
from pathlib import Path
import numpy as np
from typing import List
from tqdm.notebook import tqdm
from time import sleep
from PIL import Image
import IPython.display
from sklearn.metrics import balanced_accuracy_score
import gdown
import cv2
import zipfile
import shutil

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
```

```
AUTOTUNE = tf.data.AUTOTUNE

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.')
        self.train_inds = np.random.choice(self.n_files, int(self.n_files * TRAIN_RATIO), replace=False)
        self.val_inds = np.setdiff1d(np.arange(self.n_files), self.train_inds)

    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 random_batch_from_train_val(self, n, set_name='train'):
        if set_name == 'train':
            indices = np.random.choice(self.train_inds, n)
        else:
            indices = np.random.choice(self.val_inds, 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]

    def train_val_index_split(self, train_ratio=0.8):
        self.train_inds = np.random.choice(self.n_files, int(self.n_files * train_ratio), replace=False)
        self.val_inds = np.setdiff1d(np.arange(self.n_files), self.train_inds)
```

```
class MySequence(tf.keras.utils.Sequence):
    def __init__(self, dataset: Dataset, set_name='train') -> None:
      super().__init__()
      self.dataset = dataset
      self.leny = dataset.n_files // BATCH_SIZE
      self.set_name = set_name

    def __len__(self):
      return self.leny

    def __getitem__(self, idx):
      return self.dataset.random_batch_from_train_val(BATCH_SIZE, self.set_name)


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)))
```

```python
class Model:

    def __init__(self):
        num_classes = len(TISSUE_CLASSES)

        data_augmentation = keras.Sequential(
            [
                layers.RandomFlip("horizontal",
                                input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
                layers.RandomRotation(0.1),
                layers.RandomZoom(0.1),
            ]
        )

        efficient_netb0 = keras.applications.EfficientNetB0(include_top=True, weights=None, input_shape=(IMG_HEIGHT, IMG_WIDT

        self.model = Sequential()
        self.model.add(data_augmentation) # LBL11
        self.model.add(efficient_netb0)
        self.model.add(layers.Dropout(0.2))
        self.model.add(layers.Dense(128, activation='relu', kernel_regularizer='l2')) # LBL13.1
        self.model.add(layers.Dense(num_classes))

        self.model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

        self.model.summary()

    def save(self, name: str):
        self.model.save(name)
        shutil.make_archive(name, 'zip', name)
        shutil.copy(f"{name}.zip", "/content/drive/MyDrive/")

    def load(self, name: str):
        name_to_id_dict = {
            'best': '1ZVknWsqBF7giQb0b-vJauG6hD_RjZcbP',
            'new_best': '1-A6HwKDCfGb7iHb0lNiTzyksM7gIEMTb'
        }
        output = f'{name}.zip'
        gdown.download(f'https://drive.google.com/uc?id={name_to_id_dict[name]}', output, quiet=False)
        zip_ref = zipfile.ZipFile(output, 'r')
        zip_ref.extractall()
        self.model = keras.models.load_model(f"{name}")
        self.model.summary()

    def train(self, dataset: Dataset):
        train_seq = MySequence(dataset, set_name = 'train')
        val_seq = MySequence(dataset, set_name = 'val') # LBL1

        checkpoint = ModelCheckpoint("new_model", monitor='loss', verbose=1, # LBL3
                    save_best_only=True, mode='auto', period=1)

        epochs=30
        self.history = self.model.fit( # LBL5
            train_seq,
            validation_data=val_seq,
            epochs=epochs,
            verbose=1,
            # steps_per_epoch = 15,
            # validation_steps = 7,
            callbacks=[checkpoint]
        )

    def continue_train(self, name: str, dataset: Dataset): # LBL12
        self.load(name) # LBL4
        self.train(dataset)

    def test_on_dataset(self, dataset: Dataset, limit=None):
        if limit is not None:
            return self.model.predict(dataset.images[:int(dataset.n_files*limit)]).argmax(axis=-1)
        return self.model.predict(dataset.images).argmax(axis=-1)


    def test_on_image(self, img: np.ndarray):
        prediction = self.model.predict(img)
        return prediction
```

```
d_train = Dataset('train')
d_test = Dataset('test')
```

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1ccAgGUs43hA6hf9rpV8fi84VLv_2uW8a
To: /content/train.npz
100%|██████████| 2.10G/2.10G [00:23<00:00, 88.3MB/s]
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1brH5TzbTNUPKz3yoWS_RD4FW1xJc-dEK
To: /content/test.npz
100%|██████████| 525M/525M [00:05<00:00, 101MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
```

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

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 efficientnetb0 (Functional  (None, 1000)              5330571
 )

 dropout (Dropout)           (None, 1000)              0

 dense (Dense)               (None, 128)               128128

 dense_1 (Dense)             (None, 9)                 1161

=================================================================
Total params: 5459860 (20.83 MB)
Trainable params: 5417837 (20.67 MB)
Non-trainable params: 42023 (164.16 KB)

_________________________________________________________________
Downloading...
From (uriginal): https://drive.google.com/uc?id=1ZVknWsqBF7giQb0b-vJauG6hD_RjZcbP
From (redirected): https://drive.google.com/uc?id=1ZVknWsqBF7giQb0b-vJauG6hD_RjZcbP&confirm=t&uuid=df864cf7-fb67-4ddd-b0dc-e9cd3af
To: /content/best.zip
100%|██████████| 60.6M/60.6M [00:00<00:00, 105MB/s]
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 efficientnetb0 (Functional  (None, 1000)              5330571
 )

 dropout (Dropout)           (None, 1000)              0

 dense (Dense)               (None, 128)               128128

 dense_1 (Dense)             (None, 9)                 1161

=================================================================
Total params: 5459860 (20.83 MB)
Trainable params: 5417837 (20.67 MB)
Non-trainable params: 42023 (164.16 KB)

_________________________________________________________________
```

```
# 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')
```

```
15/15 [==============================] - 12s 83ms/step
metrics for 10% of test:
         accuracy 0.9800:
         balanced accuracy 0.9800:
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:2184: UserWarning: y_pred contains classes not in y_tru
  warnings.warn("y_pred contains classes not in y_true")
```

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

```
141/141 [==============================] - 9s 61ms/step
metrics for test:
         accuracy 0.9600:
         balanced accuracy 0.9600:
```

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

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_2 (Sequential)   (None, 224, 224, 3)       0

 efficientnetb0 (Functional  (None, 1000)              5330571
 )

 dropout_1 (Dropout)         (None, 1000)              0

 dense_2 (Dense)             (None, 128)               128128

 dense_3 (Dense)             (None, 9)                 1161

=================================================================
Total params: 5459860 (20.83 MB)
Trainable params: 5417837 (20.67 MB)
Non-trainable params: 42023 (164.16 KB)
```