Практическое задание №1

Установка необходимых пакетов:

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

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (4.6.6)
Collecting gdown
  Downloading gdown-4.7.1-py3-none-any.whl (15 kB)
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) (3
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) (1.7
Installing collected packages: gdown
  Attempting uninstall: gdown
    Found existing installation: gdown 4.6.6
    Uninstalling gdown-4.6.6:
      Successfully uninstalled gdown-4.6.6
Successfully installed gdown-4.7.1
```

Монтирование Вашего Google Drive к текущему окружению:

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

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

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

```
EVALUATE_ONLY = False
TEST_ON_LARGE_DATASET = True
EPOCHS = 10
IMG_SIZE = 224
BATCH_SIZE = 20
NUM_CLASSES = 9
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '1spqKnyy9NjiyDRAxL1eVNc0CMAXA956W',
    'train_small': '1THDzHyqY6UTzCVWpYOEDShxJfv2SmcQS',
    'train_tiny': '14tBJvFpnrUFhWux7gNWRSRYi86uXwBOb',
    'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr',
    'test_small': '1RZuUu2QLA-BYorqngKmNphPcoVEzuP-0',
    'test_tiny': '1LsaisY6gZK9Y6VJRZyU3nVY-rVTMvrjr'
}
```

Импорт необходимых зависимостей:

```
from pathlib import Path
import numpy as np
from typing import List
from tqdm.notebook import tqdm
from time import sleep
from PIL import Image
import IPython.display
from sklearn.metrics import balanced_accuracy_score
import gdown
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import load_model
from keras.models import Sequential
from keras.layers import Dense, MaxPooling2D, Dropout, GlobalAveragePooling2D, BatchNormalization, Conv2D
from tensorflow import keras
from tensorflow.keras import layers
from keras import callbacks
```

```
from tensorflow.keras.applications import ResNet50V2
from tensorflow.keras.applications.resnet_v2 import preprocess_input, decode_predictions
```

Класс Dataset

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

```
class Dataset:

    def __init__(self, name):
        self.name = name
        self.is_loaded = False
        url = f"https://drive.google.com/uc?export=download&confirm=pbef&id={DATASETS_LINKS[name]}"
        output = f'{name}.npz'
        gdown.download(url, output, quiet=False)
        print(f'Loading dataset {self.name} from npz.')
        np_obj = np.load(f'{name}.npz')
        self.images = np_obj['data']
        self.labels = np_obj['labels']
        self.n_files = self.images.shape[0]
        self.is_loaded = True
        print(f'Done. Dataset {name} consists of {self.n_files} images.')

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

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

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

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

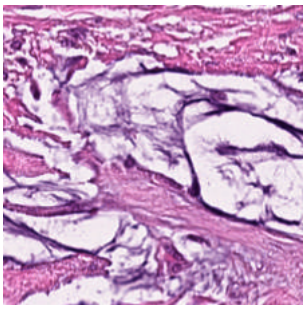
Загрузим обучающий набор данных, получим произвольное изображение с меткой. После чего визуализируем изображение, выведем метку. В будущем, этот кусок кода можно закомментировать или убрать.

```
d_train_tiny = Dataset('train_tiny')
d_test_tiny = Dataset('test_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=14tBJvFpnrUFhWux7gNWRSRYi86uXwBOb
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:00<00:00, 121MB/s]
Loading dataset train_tiny from npz.
Done. Dataset train_tiny consists of 900 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1LsaisY6gZK9Y6VJRZyU3nVY-rVTMvrjr
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 51.6MB/s]Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.

Got numpy array of shape (224, 224, 3), and label with code 7.
Label code corresponds to STR class.
```



Класс Metrics

Реализует метрики точности, используемые для оценивания модели:

1. точность,
2. сбалансированную точность.

```
class Metrics:

    @staticmethod
    def accuracy(gt: List[int], pred: List[int]):
        assert len(gt) == len(pred), 'gt and prediction should be of equal length'
        return sum(int(i[0] == i[1]) for i in zip(gt, pred)) / len(gt)

    @staticmethod
    def accuracy_balanced(gt: List[int], pred: List[int]):
        return balanced_accuracy_score(gt, pred)

    @staticmethod
    def print_all(gt: List[int], pred: List[int], info: str):
        print(f'metrics for {info}:')
        print('\t accuracy {:.4f}:'.format(Metrics.accuracy(gt, pred)))
        print('\t balanced accuracy {:.4f}:'.format(Metrics.accuracy_balanced(gt, pred)))
```

Класс Model

Класс, хранящий в себе всю информацию о модели.

Вам необходимо реализовать методы save, load для сохранения и 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.

```
class Model:
    def __init__(self):
        #base architecture
        model = ResNet50V2(include_top=False)
        model.trainable = False
        #LBL11 - augmentation
        augmentation = Sequential([layers.RandomFlip(),
                                   layers.RandomRotation((-0.2,0.2)),
                                   layers.RandomContrast(0.2)])

        input_data = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
        #pre-processing
        ds = augmentation(input_data)
        ds = preprocess_input(ds)
        #layers
        ds = model(ds, training=False)
        ds = Conv2D(16, 3, padding='same', activation='relu')(ds)
        ds = GlobalAveragePooling2D()(ds)
        ds = BatchNormalization()(ds)
        ds = Dropout(0.2)(ds)
        output_data = Dense(NUM_CLASSES, activation="softmax")(ds)

        self.model = tf.keras.Model(input_data, output_data)
        self.model.compile(loss='categorical_crossentropy',
                           optimizer='adam', metrics=['accuracy'])

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

    def load(self, name: str):
        self.model = load_model(f'/content/drive/MyDrive/{name}.npz')

    def train(self, dataset: Dataset):
        print(f'training started')
        #LBL1 - validation
        x_train, x_test, y_train, y_test = train_test_split(dataset.images,
                                           dataset.labels, test_size=0.3)
        y_train = keras.utils.to_categorical(y_train, NUM_CLASSES)
        y_test = keras.utils.to_categorical(y_test, NUM_CLASSES)
        #x_train_float = x_train.astype(float) / 255 - 0.5
        #x_test_float = x_test.astype(float) / 255 - 0.5

        #LBL5 - Output of the values of the loss function and intermediate accuracy
        a = self.model.fit(x_train, y_train, batch_size=BATCH_SIZE,
                           epochs=EPOCHS, validation_data = (x_test, y_test))
        sleep(2)
        print(f'training done')
        return a

    def test_on_dataset(self, dataset: Dataset, limit=None):
        # you can upgrade this code if you want to speed up testing using batches
        predictions = []
        n = dataset.n_files if not limit else int(dataset.n_files * limit)
        for img in tqdm(dataset.images_seq(n), total=n):
            predictions.append(self.test_on_image(img))
        return predictions

    def test_on_image(self, img: np.ndarray):
        pred = self.model.predict_on_batch(np.array([img]))
        decoded = np.argmax(pred, axis=1)
        return decoded[0]
```

Классификация изображений

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

данных 'train_small' и 'test_small'.

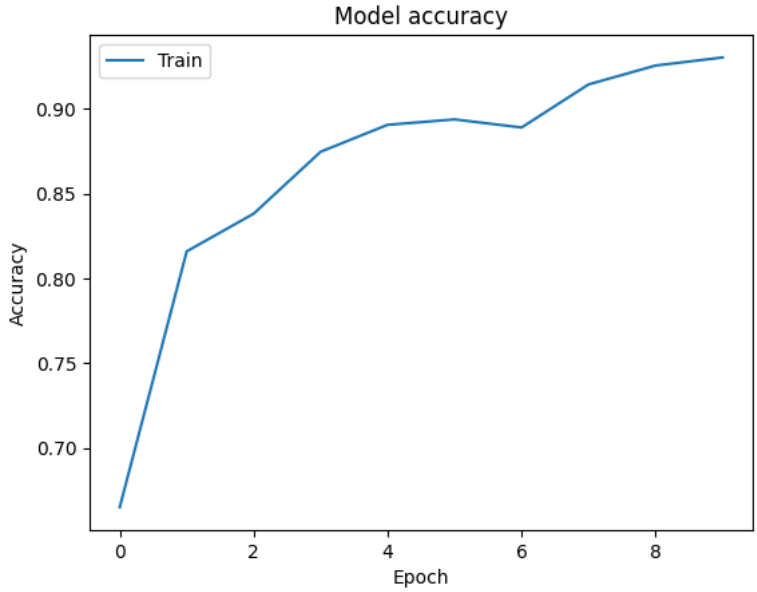
```
d_train = Dataset('train_tiny')
d_test = Dataset('test_small')
```

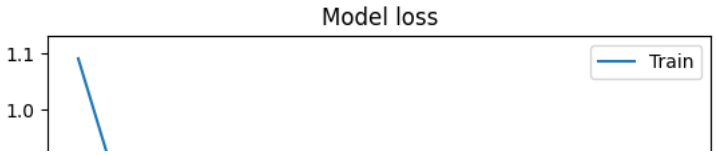
```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=14tBJvFpnrUFhWux7gNWRSRYi86uXwBOb
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:01<00:00, 77.4MB/s]
Loading dataset train_tiny from npz.
Done. Dataset train_tiny consists of 900 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1RZuUu2QLA-BYorqngKmNphPcoVEzuP-0
To: /content/test_small.npz
100%|██████████| 211M/211M [00:03<00:00, 61.7MB/s]
Loading dataset test_small from npz.
Done. Dataset test_small consists of 1800 images.
```

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

```
training started
Epoch 1/10
32/32 [==============================] - 204s 6s/step - loss: 1.0912 - accuracy: 0.6651 - val_loss: 0.4204 - val_accuracy: 0.8593
Epoch 2/10
32/32 [==============================] - 196s 6s/step - loss: 0.6991 - accuracy: 0.8159 - val_loss: 0.4540 - val_accuracy: 0.8704
Epoch 3/10
32/32 [==============================] - 196s 6s/step - loss: 0.6074 - accuracy: 0.8381 - val_loss: 0.3867 - val_accuracy: 0.8741
Epoch 4/10
32/32 [==============================] - 195s 6s/step - loss: 0.5199 - accuracy: 0.8746 - val_loss: 0.3722 - val_accuracy: 0.9074
Epoch 5/10
32/32 [==============================] - 170s 5s/step - loss: 0.4609 - accuracy: 0.8905 - val_loss: 0.3227 - val_accuracy: 0.9185
Epoch 6/10
32/32 [==============================] - 216s 7s/step - loss: 0.4297 - accuracy: 0.8937 - val_loss: 0.3241 - val_accuracy: 0.9000
Epoch 7/10
32/32 [==============================] - 197s 6s/step - loss: 0.3931 - accuracy: 0.8889 - val_loss: 0.3244 - val_accuracy: 0.9000
Epoch 8/10
32/32 [==============================] - 206s 7s/step - loss: 0.3510 - accuracy: 0.9143 - val_loss: 0.3389 - val_accuracy: 0.8926
Epoch 9/10
32/32 [==============================] - 197s 6s/step - loss: 0.3322 - accuracy: 0.9254 - val_loss: 0.2998 - val_accuracy: 0.9148
Epoch 10/10
32/32 [==============================] - 195s 6s/step - loss: 0.2939 - accuracy: 0.9302 - val_loss: 0.2937 - val_accuracy: 0.9037
training done
```

```
#LBL6
#Visualize the models accuracy
plt.plot(hisp.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper left')
plt.show()
print()
#Visualize the models loss
plt.plot(hisp.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper right')
plt.show()
```





Пример тестирования модели на части набора данных:

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

```
100%                                                180/180 [00:39<00:00, 5.37it/s]
metrics for 10% of test:
        accuracy 0.9778:
        balanced accuracy 0.9778:
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:2184: UserWarning: y_pred contains classes not in y_true
  warnings.warn("y_pred contains classes not in y_true")
```

Пример тестирования модели на полном наборе данных:

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

```
100%                                                1800/1800 [06:36<00:00, 5.88it/s]
metrics for test:
        accuracy 0.8783:
        balanced accuracy 0.8783:
```

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

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

Тестирование модели на других наборах данных

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

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

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

```
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1LsaisY6gZK9Y6VJRZ
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 163MB/s]
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
100%                                                  90/90 [00:23<00:00, 2.97it/s]
metrics for test-tiny:
        accuracy 0.8778:
```

Отмонтировать Google Drive.

```
drive.flush_and_unmount()
```

Дополнительные "полезности"

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

[ ] ↳ *Скрыто 17 ячеек.*