

▼ Практическое задание №1

Установка необходимых пакетов:

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

Requirement already satisfied: gdown in /usr/local/lib/python3.7/dist-packages (4.4.
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from g
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: urllib3!=1.25.0,!1.25.1,<1.26,>=1.21.1 in /usr/local
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.7/di
```

Монтирование Вашего Google Drive к текущему окружению:

```
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.
Это пригодится при сохранении модели.

```
# todo
PROJECT_DIR = 'dev/Prac1/'
```

Константы, которые пригодятся в коде далее:

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

Ссылки (gdrive идентификаторы) на предоставляемые наборы данных:

```
DATASETS_LINKS = {
    'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-cLi',
```

```

'train_small': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
'train_tiny': '1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui',
'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBF1Dr',
'test_small': '1wbRsog0n7uG1HIPGLhyN-PMET2kdQ21I',
'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}

```

Импорт необходимых зависимостей:

```

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 gdown

```

▼ Класс Dataset

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

```

class Dataset:

    def __init__(self, name):
        self.name = name
        self.is_loaded = False
        if not Path(f'{name}.npz').exists():
            url = f'https://drive.google.com/uc?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 использования класса Dataset

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

```

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?id=1I-2Z0uXLd4QwhZQ01tp817Kn3J0Xgbui
To: /content/train_tiny.npz
100%|██████████| 105M/105M [00:01<00:00, 101MB/s]
Loading dataset train_tiny from npz.

```

▼ 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 загрузки модели. Особенно актуально это будет во время тестирования на дополнительных наборах данных.

Пожалуйста, убедитесь, что сохранение и загрузка модели работает корректно. Для этого обучите модель, протестируйте, сохраните ее в файл, перезапустите среду выполнения, загрузите обученную модель из файла, вновь протестируйте ее на тестовой выборке и убедитесь в том, что получаемые метрики совпадают с полученными для тестовой выборки ранее.

Также, Вы можете реализовать дополнительные функции, такие как:

1. валидацию модели на части обучающей выборки;
2. использование кроссвалидации;

3. автоматическое сохранение модели при обучении;
4. загрузку модели с какой-то конкретной итерации обучения (если используется итеративное обучение);
5. вывод различных показателей в процессе обучения (например, значение функции потерь на каждой эпохе);
6. построение графиков, визуализирующих процесс обучения (например, график зависимости функции потерь от номера эпохи обучения);
7. автоматическое тестирование на тестовом наборе/наборах данных после каждой эпохи обучения (при использовании итеративного обучения);
8. автоматический выбор гиперпараметров модели во время обучения;
9. сохранение и визуализацию результатов тестирования;
10. Использование аугментации и других способов синтетического расширения набора данных (дополнительным плюсом будет обоснование необходимости и обоснование выбора конкретных типов аугментации)
11. и т.д.

Полный список опций и дополнений приведен в презентации с описанием задания.

При реализации дополнительных функций допускается добавление параметров в существующие методы и добавление новых методов в класс модели.

```
import tensorflow as tf
```

```
from matplotlib import use
class Model:
```

```
    def __init__(self):
        model = tf.keras.Sequential([
            tf.keras.layers.Rescaling(1./255, input_shape=(224, 224, 3)),
            tf.keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu'),
            tf.keras.layers.MaxPooling2D((4, 4)),
            tf.keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu'),
            tf.keras.layers.MaxPooling2D((4, 4)),
            tf.keras.layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
            tf.keras.layers.MaxPooling2D((4, 4)),
            tf.keras.layers.Dropout(0.2),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(128, activation='relu'),
            tf.keras.layers.Dense(9)
        ])

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

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

```

def load(self, name: str):
    name_to_id_dict = {
        'best': '1K3MwGEk1FqI2a3doSK-Po6oZwLj5rqNa',
        'trained_on_small_d': '1-g87BhIRe0xjjriQN8JOxHLuYqIasVbc'
    }
    url = f'https://drive.google.com/drive/folders/{name_to_id_dict[name]}'
    gdown.download_folder(url, quiet=True, output=name, use_cookies=False)
    self.model = tf.keras.models.load_model(name)

def train(self, dataset: Dataset):
    print(f'training started')
    self.model.fit(dataset.images, dataset.labels, epochs=10)
    self.model.evaluate(d_test.images, d_test.labels, verbose=2)
    print(f'training done')

def test_on_dataset(self, dataset: Dataset, limit=None):
    n = dataset.n_files if not limit else int(dataset.n_files * limit)
    imgs, true_labels = dataset.random_batch_with_labels(n)
    p_predictions = self.model.predict(imgs)
    predictions = []
    for i in range(0, n):
        predictions.append(np.argmax(p_predictions[i]))
    # 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, true_labels

def test_on_image(self, img: np.ndarray):
    #img_array = []
    #img_array.append(img)
    prediction = self.model.predict((img))

    return np.argmax(prediction[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')
d_test = Dataset('test')

```

```

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

```

```
model = Model()
if not EVALUATE_ONLY:
    model.train(d_train)
    model.save('best')
else:
    #todo: your link goes here
    model.load('best')
```

Пример тестирования модели на части набора данных:

```
# evaluating model on 10% of test dataset
pred_1, lbls_1 = model.test_on_dataset(d_test, limit=0.1)
Metrics.print_all(lbls_1, pred_1, '10% of test')

metrics for 10% of test:
    accuracy 0.9178:
    balanced accuracy 0.9160:
```

Пример тестирования модели на полном наборе данных:

```
# evaluating model on full test dataset (may take time)
if TEST_ON_LARGE_DATASET:
    pred_2, lbls_2 = model.test_on_dataset(d_test)
    Metrics.print_all(lbls_2, pred_2, 'test')

metrics for test:
    accuracy 0.8967:
    balanced accuracy 0.8987:
```

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, lbls = model.test_on_dataset(d_test_tiny)
Metrics.print_all(lbls, pred, 'test-tiny')

Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
metrics for test-tiny:
    accuracy 0.8778:
    balanced accuracy 0.8867:

```

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.

▼ Измерение времени работы кода

Измерять время работы какой-либо функции можно легко и непринужденно при помощи функции `timeit` из соответствующего модуля:

```

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

```
# 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.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi

from skimage import feature

# Generate noisy image of a square
im = np.zeros((128, 128))
im[32:-32, 32:-32] = 1

im = ndi.rotate(im, 15, mode='constant')
im = ndi.gaussian_filter(im, 4)
im += 0.2 * np.random.random(im.shape)

# Compute the Canny filter for two values of sigma
edges1 = feature.canny(im)
edges2 = feature.canny(im, sigma=3)

# display results
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(8, 3),
                                     sharex=True, sharey=True)

ax1.imshow(im, cmap=plt.cm.gray)
ax1.axis('off')
ax1.set_title('noisy image', fontsize=20)

ax2.imshow(edges1, cmap=plt.cm.gray)
ax2.axis('off')
ax2.set_title(r'Canny filter, $\sigma=1$', fontsize=20)

ax3.imshow(edges2, cmap=plt.cm.gray)
ax3.axis('off')
ax3.set_title(r'Canny filter, $\sigma=3$', fontsize=20)

fig.tight_layout()

plt.show()
```

▼ Tensorflow 2

Для создания и обучения нейросетевых моделей можно использовать фреймворк глубокого обучения Tensorflow 2. Ниже приведен пример простейшей нейронной сети, использующейся для классификации изображений из набора данных MNIST.

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

Loading dataset train_small from npz.
Done. Dataset train_small consists of 7200 images.
Loading dataset test_small from npz.
Done. Dataset test_small consists of 1800 images.

import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(224, 224, 3)),
    tf.keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D((4, 4)),
    tf.keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D((4, 4)),
    tf.keras.layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D((4, 4)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(9)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.fit(d_train.images, d_train.labels, epochs=10)

model.evaluate(d_test.images, d_test.labels, verbose=2)

Epoch 1/10
225/225 [=====] - 6s 25ms/step - loss: 1.5896 - accuracy: 0
Epoch 2/10
225/225 [=====] - 6s 25ms/step - loss: 1.0764 - accuracy: 0
Epoch 3/10
225/225 [=====] - 6s 25ms/step - loss: 0.8930 - accuracy: 0
Epoch 4/10
225/225 [=====] - 6s 25ms/step - loss: 0.7086 - accuracy: 0
Epoch 5/10
225/225 [=====] - 6s 25ms/step - loss: 0.5956 - accuracy: 0
Epoch 6/10
225/225 [=====] - 6s 25ms/step - loss: 0.5129 - accuracy: 0
Epoch 7/10
225/225 [=====] - 6s 25ms/step - loss: 0.4565 - accuracy: 0
Epoch 8/10
225/225 [=====] - 6s 25ms/step - loss: 0.4038 - accuracy: 0
```

```
Epoch 9/10
225/225 [=====] - 6s 27ms/step - loss: 0.3507 - accuracy: 0
Epoch 10/10
225/225 [=====] - 6s 25ms/step - loss: 0.3280 - accuracy: 0
57/57 - 1s - loss: 0.4538 - accuracy: 0.8344 - 993ms/epoch - 17ms/step
[0.45384442806243896, 0.8344444632530212]
```

Для эффективной работы с моделями глубокого обучения убедитесь в том, что в текущей среде Google Colab используется аппаратный ускоритель GPU или TPU. Для смены среды выберите "среда выполнения" -> "сменить среду выполнения".

Большое количество tutorиалов и примеров с кодом на Tensorflow 2 можно найти на официальном сайте <https://www.tensorflow.org/tutorials?hl=ru>.

Также, Вам может понадобиться написать собственный генератор данных для Tensorflow 2. Скорее всего он будет достаточно простым, и его легко можно будет реализовать, используя официальную документацию TensorFlow 2. Но, на всякий случай (если не удалось сразу разобраться или хочется вникнуть в тему более глубоко), можете посмотреть следующий отличный tutorиал: <https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly>.

Numba

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

1. https://colab.research.google.com/github/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.

```
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.

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

✓ 3 сек. выполнено в 01:51

