Notebook

Практическое задание №1

```
%matplotlib inline

from google.colab import drive
drive.mount('/content/drive')
```

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

Установка необходимых пакетов:

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

```
     |████████████████████████████████| 133kB 7.7MB/s
  Building wheel for libtiff (setup.py) ... done
```

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, в предоставленными наборами данных.

```
# todo
PROJECT_DIR = 'speccourse_task/'
```

Константы, которые пригодятся в коде далее:

```
EVALUATE_ONLY = False
TEST_ON_LARGE_DATASET = True
VAL_PROPORTION = 0.025
BATCH_SIZE = 128
EPOCHS = 60
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR',
```

Импорт необходимых зависимостей:

```
from pathlib import Path
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from libtiff import TIFF
from skimage.io import imsave, imread
import os
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 gc
```

Класс Dataset

Предназначен для работы с наборами данных, хранящихся на Google Drive, обеспечивае
меток, а также формирование пакетов (батчей).

```
def preprocess_img(img, label=0):…

class Dataset:

    def __init__(self, name, gdrive_dir):
        self.name = name
        self.is_loaded = False
        p = Path("/content/drive/MyDrive/" + gdrive_dir + name + '.npz')
        if p.exists():
            print(f'Loading dataset {self.name} from npz.')
            np_obj = np.load(str(p))
            self.images = np_obj['data']
            self.labels = np_obj['labels']
```

```
            self.n_files = self.images.shape[0]
            self.is_loaded = True
            self.dir_to_save = None
            print(f'Done. Dataset {name} consists of {self.n_files} images.

    def create_tf_dataset(self, pattern='*', preprocess=False):

        if preprocess:
            print('Preprocessing started')
            for i, img in enumerate(self.images):
                self.images[i], _ = preprocess_img(self.images[i])
            print('Preprocessing finished')

        return tf.data.Dataset.from_tensor_slices((self.images, self.labels

    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. Пос выведем метку. В будущем, этот кусок кода можно закомментировать или убрать.

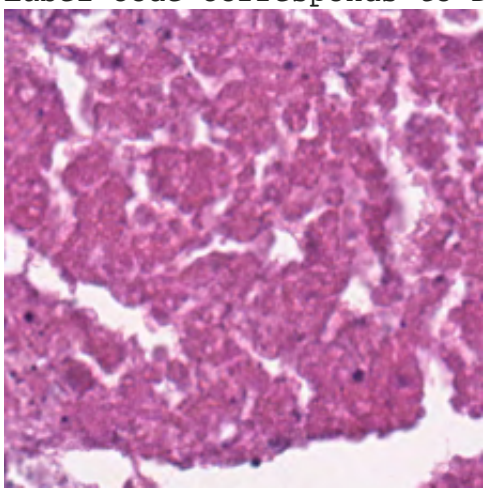
```
d_train_tiny = Dataset('test_small', PROJECT_DIR)

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

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

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

Got numpy array of shape (224, 224, 3), and label with code 2.
Label code corresponds to DEB 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
        return sum(int(i[0] == i[1]) for i in zip(gt, pred)) / len(gt)

    @staticmethod
    def accuracy_balanced(gt, pred):
        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_balanc
```

Класс Model

Класс, хранящий в себе всю информацию о модели.

Вам необходимо реализовать методы save, load для сохранения и заргрузки модели. Ос
тестирования на дополнительных наборах данных.

> *Пожалуйста, убедитесь, что сохранение и загрузка модели работает корректно. Для э
> сохраните ее в файл, перезапустите среду выполнения, загрузите обученную модель и
> тестовой выборке и убедитесь в том, что получаемые метрики совпадают с полученн*

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 ите
5. вывод различных показателей в процессе обучения (например, значение функции
6. построение графиков, визуализирующих процесс обучения (например, график зави
обучения);
7. автоматическое тестирование на тестовом наборе/наборах данных после каждой
итеративного обучения);
8. автоматический выбор гиперпараметров модели во время обучения;
9. сохранение и визуализацию результатов тестирования;
10. Использование аугментации и других способов синтетического расширения набор
обоснование необходимости и обоснование выбора конкретных типов аугментаци
11. и т.д.

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

При реализации дополнительных функций допускается добавление параметров в суще
методов в класс модели.

```
class MeanBalancedAccuracy(tf.keras.metrics.Metric): # LBL_ADDITIONAL_OUTPU

    def __init__(self, print_metric=True, print_test=False, name="mean_bala

        super(MeanBalancedAccuracy, self).__init__(name=name, **kwargs)
        self.print_metric = print_metric
        self.print_test = print_test
        self.accuracy_val = self.add_weight(name="mean_balanced_accuracy",
```

```
        self.count = self.add_weight(name="count", initializer='zeros', dty

    def update_state(self, y_true, y_pred, sample_weight=None):

        y_pred = np.argmax(y_pred.numpy(), axis=1)
        conf_matr = confusion_matrix(y_true, y_pred)
        true_positives = conf_matr.diagonal()
        amounts = np.sum(conf_matr, axis=1)
        amount_of_classes = amounts[amounts > 0].shape[0]
        # print(amount_of_classes)
        amounts[amounts == 0] = 1 # to avoid division by 0
        accuracies = true_positives / amounts / amount_of_classes
        accuracies = accuracies[np.logical_not(np.isnan(accuracies))]
        # print(conf_matr, conf_matr.sum())
        # print(accuracies, true_positives.sum() / conf_matr.sum())

        # if self.print_metric:
        #     print()
        #     print(conf_matr)
        #     print(amounts)
        #     print(accuracies)
        #     print(y_true)

        self.count.assign(self.count + 1)
        self.accuracy_val.assign(self.accuracy_val + tf.cast(tf.reduce_sum(

    def reset_states(self):

        super(MeanBalancedAccuracy, self).reset_states()
        if self.print_test: # LBL_AUTOTEST_DURING_TRAINING
            pred = model.test_on_dataset(d_test)
            Metrics.print_all(ds_labels, pred, 'test')

    def result(self):

        if self.print_metric:
            print("MeanBalancedAccuracy :", self.accuracy_val.value().numpy

        return self.accuracy_val / self.count
```

```
class MySparseCategoricalCrossentropy(tf.keras.losses.Loss): # LBL_ADDITION

    def __init__(self, print_loss=True):

        super(MySparseCategoricalCrossentropy, self).__init__()

        self.print_loss = print_loss

    def call(self, y_true, y_pred, eps=0.00000000001):

        global processed_batches

        indices = tf.where(y_pred == 0.)
        updates = tf.ones(tf.shape(indices)[0]) * eps
        y_pred = tf.tensor_scatter_nd_update(y_pred, indices, updates)
        y_true = tf.cast(tf.reshape(y_true, (-1, 1)), tf.int32)
        row_ind = tf.cast(tf.reshape(tf.range(0, y_pred.shape[0]), (-1, 1))
        indeces = tf.concat((row_ind, y_true), 1)
        loss = -tf.math.log(tf.gather_nd(y_pred, indeces))

        if self.print_loss:
            print('Loss :', (tf.reduce_sum(loss) / y_true.shape[0]).numpy()
            processed_batches += y_true.shape[0]


        return loss
```

```
class Model(tf.keras.Model):

    def __init__(self, amount_of_classes, directory='', L2_const=0.0005, dr

    def weight_branch(self, filters, kernel_size, stride=1, initializer=tf.

    def dense_layer(units, activation=None):…

    def conv2d_layer(self, filters, kernel_size=(3, 3),…

    def _set_training(self, training):…

    def call(self, img, training=True):…

    def predict(self, img):…

    # dont need it
    def save(self, name: str):…
```

```
385 >     def load(self, name: str): # LBL_START_FROM_A_PARTICULAR_EPOCH…
415
416 >     def compile_model(self, input_shape=(None, 224, 224, 3),…
428
429 >     def train(self, ds_train, val_prop,…
466
467 >     def show_training_results(self):…
474
475 >     def draw_plot(self, loss, val_loss): # LBL_SOME_GRAPHICS…
487
488       def test_on_dataset(self, dataset, limit=None,
489                           loss=MySparseCategoricalCrossentropy(print_loss=Fal
490                           metric=MeanBalancedAccuracy(print_metric=False)):
491           # you can upgrade this code if you want to speed up testing using b
492
493           if type(dataset) is Dataset:
494               dataset = dataset.create_tf_dataset(preprocess=True)
495
496           predictions = []
497           metric.reset_states()
498           # metr_obj = Metrics()
499           n_files = len(dataset)
500           print(n_files)
501           n = n_files if not limit else int(n_files * limit)
502           batched_dataset = dataset.take(n).batch(BATCH_SIZE)
503           for imgs, labels in tqdm(batched_dataset, total=len(batched_dataset
504               y_pred = self.predict(imgs)
505               # values = []
506               # y_true = tf.reshape(labels, (labels.shape[0], -1))
507               # if loss:
508               #     loss_val = np.mean(loss.call(y_true, y_pred).numpy())
509               #     values.append(loss_val)
510               # if metric:
511               #     metric.update_state(y_true, y_pred)
512
513               arg_max = np.argmax(y_pred.numpy(), axis=1).reshape(1, -1)
514               # values.append(Metrics.accuracy(labels, arg_max))
515
516               predictions.append(arg_max)
517           # loss, m = np.mean(np.array(predictions), axis=0)
518
519           return np.concatenate(predictions, axis=1).flatten()#loss, metric.r
520
521 >     def test_on_images(self, img):…
527
```

Классификация изображений

Используя введенные выше классы можем перейти уже непосредственно к обучению м
общего пайплайна решения задачи приведен ниже. Вы можете его расширять и улучша
данных 'train_small' и 'test_small'.

```
# get datasets
d_train = Dataset('train', PROJECT_DIR).create_tf_dataset(preprocess=True)
# d_test = Dataset('test_small', PROJECT_DIR).create_tf_dataset()
```

```
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
Preprocessing started
Preprocessing finished
0
```

```
ckpt_dir = '/content/drive/MyDrive/' + PROJECT_DIR + 'checkpoints'
model = Model(amount_of_classes=len(TISSUE_CLASSES), directory=ckpt_dir)

schedule = tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_
                                                          decay_steps=int(1
model.compile_model(optimizer=tf.keras.optimizers.Adam(learning_rate=schedu
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_1
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_2
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.rest
Model was compiled
```

```
processed_batches = 0
if EVALUATE_ONLY:
    model.train(d_train, val_prop=VAL_PROPORTION, batch_size=BATCH_SIZE, ep
    # model.save('best')
else:
    model.load('epoch_049_val_loss_0.616_.cpkt')
```

```
Loaded : epoch_049_val_loss_0.616_.cpkt
Weights loaded
```

```
d_test = Dataset('test', PROJECT_DIR)
```

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

Пример тестирования модели на части набора данных:

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

```
Preprocessing started
Preprocessing finished
4500
```

```
100%                                                36/36 [00:08<00:00, 4.03it/s]
```

```
metrics for test:
         accuracy 0.9371:
         balanced accuracy 0.9371:
```

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')
```

Результат работы пайплайна обучения и тестирования выше тоже будет оцениваться. П
ноутбук с выполненными ячейками кода с демонстрациями метрик обучения, графиками
продемонстрировать работу всех реализованных дополнений, улучшений и т.п.

Настоятельно рекомендуется после получения пайплайна с полными результатами обучения экспор
этот pdf вместе с самим ноутбуком.

Тестирование модели на других наборах данных

Ваша модель должна поддерживать тестирование на других наборах данных. Для удобс
test_tiny, который представляет собой малую часть (2% изображений) набора test. Ниже
осуществлять тестирование для оценивания Вашей модели на дополнительных тестовы

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

```
# final_model = Model(amount_of_classes=len(TISSUE_CLASSES), directory=ckpt
# final_model.load('best')
d_tiny = Dataset('test_tiny', PROJECT_DIR)
pred = model.test_on_dataset(d_tiny)
Metrics.print_all(d_tiny.labels, pred, 'test')
```

```
Loading dataset test_tiny from npz.
Done. Dataset test_tiny consists of 90 images.
Preprocessing started
Preprocessing finished
90
```

100% 1/1 [00:00<00:00, 3.85it/s]

```
metrics for test:
         accuracy 0.9444:
         balanced accuracy 0.9444:
```

Отмонтировать Google Drive.

```
drive.flush_and_unmount()
```

Дополнительные "полезности"

Ниже приведены примеры использования различных функций и библиотек, которые мо
практического задания.

Измерение времени работы кода

Измерять время работы какой-либо функции можно легко и непринужденно при помощ
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
```

Scikit-learn

Для использования "классических" алгоритмов машинного обучения рекомендуется ис
(https://scikit-learn.org/stable/). Пример классификации изображений цифр из набора да
SVM:

```
# 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 th
# 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.
# images, we know which digit they represent: it is given in the 'target' o
# 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 самостоя
используя специализированные библиотеки, например, scikit-image (https://scikit-image.
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 глубо
пример простейшей нейроной сети, использующейся для классификации изображений

```
# 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 Goo
или TPU. Для смены среды выберите "среда выполнения" -> "сменить среду выполнения".

Большое количество туториалов и примеров с кодом на Tensorflow 2 можно найти на оф
https://www.tensorflow.org/tutorials?hl=ru.

Также, Вам может понадобиться написать собственный генератор данных для Tensorflo
простым, и его легко можно будет реализовать, используя официальную документацию
удлось сразу разобраться или хочется вникнуть в тему более глубоко), можете посмотре
https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly.

Numba

В некоторых ситуациях, при ручных реализациях графовых алгоритмов, выполнение мн
можно существенно ускорить, используя JIT-компилятор Numba (https://numba.pydata.or
Colab можно найти тут:

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

> Пожалуйста, если Вы решили использовать Numba для решения этого практическо
> нужно ли это Вам, и есть ли возможность реализовать требуемую функциональнос
> Numba только при реальной необходимости.

Работа с zip архивами в Google Drive

Запаковка и распаковка zip архивов может пригодиться при сохранении и загрузки Ваш
иллюстрирующий помещение нескольких файлов в zip архив с последующим чтением ф
директориями, файлами и архивами должны осущетвляться с примонтированным Goog

Создадим 2 изображения, поместим их в директорию tmp внутри PROJECT_DIR, запакуе

```
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 которой находятся 2 изображения.

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