

▼ Практическое задание №1

Установка необходимых пакетов:

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

```
|██████████| 133kB 14.3MB/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 которую Вы загрузили zip архивы с предоставленными наборами данных.

```
# todo  
PROJECT_DIR = 'NN_course/NN_prac_1_data/'
```

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')
```

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 tensorflow as tf  
from tensorflow.keras import layers, models, optimizers, metrics, Input, callbacks  
import matplotlib.pyplot as plt
```

▼ 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, 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
            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]
```

▼ 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 Model:

```
class Model:

    def __init__(self):
        # todo
        self.model = models.Sequential()
        self.model.add(Input(shape=(224, 224, 3), dtype=np.float16))
        self.model.add(layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"))
        self.model.add(layers.experimental.preprocessing.RandomRotation(0.1)) #LBL11
        self.model.add(layers.experimental.preprocessing.RandomContrast(0.15)) #LBL11
        self.model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu', input_shape=(224, 224, 3)))
        self.model.add(layers.Conv2D(16, (3, 3), padding='same', activation='relu'))
        self.model.add(layers.MaxPooling2D(2, 2))
        self.model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu'))
        self.model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu'))
        self.model.add(layers.MaxPooling2D(2, 2))
        self.model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
        self.model.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
        self.model.add(layers.MaxPooling2D(2, 2))
        self.model.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
        self.model.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
        self.model.add(layers.MaxPooling2D(2, 2))
        self.model.add(layers.Conv2D(256, (3, 3), padding='same', activation='relu'))
        self.model.add(layers.Conv2D(256, (3, 3), padding='same', activation='relu'))
        self.model.add(layers.MaxPooling2D(2, 2)) # 7 * 7 * 256
        self.model.add(layers.Conv2D(2048, (7, 7), activation='relu'))
        self.model.add(layers.Dropout(0.3))
        self.model.add(layers.Conv2D(1024, (1, 1), activation='relu'))
        self.model.add(layers.Conv2D(256, (1, 1), activation='relu'))
        self.model.add(layers.Dropout(0.3))
        self.model.add(layers.Conv2D(9, (1, 1), activation='softmax'))
        self.model.add(layers.Flatten())
        pass

    def save(self, name: str):
        # save model to PROJECT_DIR folder on gdrive with name 'name'
        # todo
        p = Path("/content/drive/MvDrive/" + PROJECT_DIR + name)
```

```
self.model.save(str(p))
pass

def load(self, name: str):
    # load model with name 'name' from PROJECT_DIR folder on gdrive
    # todo
    p = Path("/content/drive/MyDrive/" + PROJECT_DIR + name)
    self.model = models.load_model(str(p))
    pass

def train(self, dataset: Dataset, val_dataset: Dataset):
    # you can add some plots for better visualization,
    # you can add model autosaving during training,
    # etc.
    print(f'training started')
    # to-do
    opt = optimizers.Adam(learning_rate=0.00015)
    reduce_lr = callbacks.ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.3,
        patience=1,
        verbose=0,
        mode='auto',
        min_delta=0.0001,
        cooldown=0,
        min_lr=0
    )
    p = Path("/content/drive/MyDrive/" + PROJECT_DIR + "best")
    save_model = callbacks.ModelCheckpoint( #LBL3
        filepath=str(p),
        save_best_only=True,
        save_weights_only=True,
        monitor='val_accuracy',
        verbose=1
    )
    self.model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=history = self.model.fit(dataset.images, dataset.labels, epochs=30, validation_dat
    print(f'training done')

#LBL6
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
pass

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):
    # todo: replace this code
    prediction = np.argmax(self.model.predict(img.reshape(1, 224, 224, 3)))
    return prediction
```

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

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()
model.model.summary()

Model: "sequential"

Layer (type)          Output Shape         Param #
=====
random_flip (RandomFlip)  (None, 224, 224, 3)      0
=====
random_rotation (RandomRotat (None, 224, 224, 3)      0
=====
random_contrast (RandomContr (None, 224, 224, 3)      0
=====
conv2d (Conv2D)        (None, 224, 224, 16)     448
=====
conv2d_1 (Conv2D)       (None, 224, 224, 16)     2320
=====
max_pooling2d (MaxPooling2D) (None, 112, 112, 16)    0
=====
conv2d_2 (Conv2D)       (None, 112, 112, 32)     4640
=====
conv2d_3 (Conv2D)       (None, 112, 112, 32)     9248
=====
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 32)      0
=====
conv2d_4 (Conv2D)       (None, 56, 56, 64)     18496
=====
conv2d_5 (Conv2D)       (None, 56, 56, 64)     36928
=====
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 64)      0
=====
conv2d_6 (Conv2D)       (None, 28, 28, 128)    73856
```

conv2d_7 (Conv2D)	(None, 28, 28, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 128)	0
conv2d_8 (Conv2D)	(None, 14, 14, 256)	295168
conv2d_9 (Conv2D)	(None, 14, 14, 256)	590080
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 256)	0
conv2d_10 (Conv2D)	(None, 1, 1, 2048)	25692160
dropout (Dropout)	(None, 1, 1, 2048)	0
conv2d_11 (Conv2D)	(None, 1, 1, 1024)	2098176
conv2d_12 (Conv2D)	(None, 1, 1, 256)	262400
dropout_1 (Dropout)	(None, 1, 1, 256)	0
conv2d_13 (Conv2D)	(None, 1, 1, 9)	2313
flatten (Flatten)	(None, 9)	0
<hr/>		
Total params: 29,233,817		
Trainable params: 29,233,817		
Non-trainable params: 0		

```
if EVALUATE_ONLY:  
    model.train(d_train, d_test)  
    model.save('best')  
else:  
    model.load('best')
```

```
training started
Epoch 1/30
563/563 [=====] - 110s 133ms/step - loss: 1.8082 - accuracy: 0.57933

Epoch 00001: val_accuracy improved from -inf to 0.57933, saving model to /content/dri...
Epoch 2/30
563/563 [=====] - 73s 130ms/step - loss: 1.0825 - accuracy: 0.71978

Epoch 00002: val_accuracy improved from 0.57933 to 0.71978, saving model to /content...
Epoch 3/30
563/563 [=====] - 75s 133ms/step - loss: 0.8082 - accuracy: 0.77067

Epoch 00003: val_accuracy improved from 0.71978 to 0.77067, saving model to /content...
Epoch 4/30
563/563 [=====] - 76s 135ms/step - loss: 0.5999 - accuracy: 0.83222

Epoch 00004: val_accuracy improved from 0.77067 to 0.83222, saving model to /content...
Epoch 5/30
563/563 [=====] - 76s 136ms/step - loss: 0.4594 - accuracy: 0.87467

Epoch 00005: val_accuracy improved from 0.83222 to 0.87467, saving model to /content...
Epoch 6/30
563/563 [=====] - 76s 135ms/step - loss: 0.3912 - accuracy: 0.87778

Epoch 00006: val_accuracy improved from 0.87467 to 0.87778, saving model to /content...
Epoch 7/30
563/563 [=====] - 76s 135ms/step - loss: 0.3731 - accuracy: 0.92356

Epoch 00007: val_accuracy did not improve from 0.87778
Epoch 8/30
563/563 [=====] - 76s 134ms/step - loss: 0.2484 - accuracy: 0.92956

Epoch 00008: val_accuracy improved from 0.87778 to 0.92356, saving model to /content...
Epoch 9/30
563/563 [=====] - 75s 134ms/step - loss: 0.2026 - accuracy: 0.94178

Epoch 00009: val_accuracy improved from 0.92356 to 0.92956, saving model to /content...
Epoch 10/30
563/563 [=====] - 75s 134ms/step - loss: 0.1907 - accuracy: 0.94667

Epoch 00010: val_accuracy improved from 0.92956 to 0.94178, saving model to /content...
Epoch 11/30
563/563 [=====] - 76s 134ms/step - loss: 0.1721 - accuracy: 0.94667

Epoch 00011: val_accuracy improved from 0.94178 to 0.94667, saving model to /content...
Epoch 12/30
563/563 [=====] - 76s 135ms/step - loss: 0.1599 - accuracy: 0.95378

Epoch 00012: val_accuracy did not improve from 0.94667
Epoch 13/30
563/563 [=====] - 76s 135ms/step - loss: 0.1469 - accuracy: 0.95378

Epoch 00013: val_accuracy improved from 0.94667 to 0.95378, saving model to /content...
Epoch 14/30
563/563 [=====] - 76s 135ms/step - loss: 0.1472 - accuracy: 0.95711

Epoch 00014: val_accuracy did not improve from 0.95378
Epoch 15/30
563/563 [=====] - 76s 135ms/step - loss: 0.1496 - accuracy: 0.95711

Epoch 00015: val_accuracy improved from 0.95378 to 0.95711, saving model to /content...
```

```
Epoch 16/30
563/563 [=====] - 76s 135ms/step - loss: 0.1294 - accuracy: 0.95711

Epoch 00016: val_accuracy did not improve from 0.95711
Epoch 17/30
563/563 [=====] - 76s 135ms/step - loss: 0.1025 - accuracy: 0.96156

Epoch 00017: val_accuracy improved from 0.95711 to 0.96156, saving model to /content
Epoch 18/30
563/563 [=====] - 76s 135ms/step - loss: 0.0997 - accuracy: 0.96911

Epoch 00018: val_accuracy improved from 0.96156 to 0.96911, saving model to /content
Epoch 19/30
563/563 [=====] - 76s 135ms/step - loss: 0.0828 - accuracy: 0.96911

Epoch 00019: val_accuracy did not improve from 0.96911
Epoch 20/30
563/563 [=====] - 76s 135ms/step - loss: 0.0796 - accuracy: 0.96933

Epoch 00020: val_accuracy did not improve from 0.96911
Epoch 21/30
563/563 [=====] - 76s 135ms/step - loss: 0.0716 - accuracy: 0.96956

Epoch 00021: val_accuracy improved from 0.96911 to 0.96933, saving model to /content
Epoch 22/30
563/563 [=====] - 76s 135ms/step - loss: 0.0708 - accuracy: 0.97022

Epoch 00022: val_accuracy did not improve from 0.96933
Epoch 23/30
563/563 [=====] - 76s 135ms/step - loss: 0.0694 - accuracy: 0.97022

Epoch 00023: val_accuracy improved from 0.96933 to 0.96956, saving model to /content
Epoch 24/30
563/563 [=====] - 76s 135ms/step - loss: 0.0755 - accuracy: 0.97022

Epoch 00024: val_accuracy improved from 0.96956 to 0.97022, saving model to /content
Epoch 25/30
563/563 [=====] - 76s 135ms/step - loss: 0.0669 - accuracy: 0.97022

Epoch 00025: val_accuracy did not improve from 0.97022
Epoch 26/30
563/563 [=====] - 76s 135ms/step - loss: 0.0749 - accuracy: 0.97022

Epoch 00026: val_accuracy did not improve from 0.97022
Epoch 27/30
563/563 [=====] - 76s 135ms/step - loss: 0.0674 - accuracy: 0.97022

Epoch 00027: val_accuracy did not improve from 0.97022
Epoch 28/30
563/563 [=====] - 76s 135ms/step - loss: 0.0731 - accuracy: 0.97022

Epoch 00028: val_accuracy did not improve from 0.97022
Epoch 29/30
```

Пример тестирования модели на части набора данных:

Epoch 00001: val_accuracy did not improve from 0.00000

```
model.load('best')
```

— 1 — 888888 — 7 — 888888 — 8 — 888888

```
# evaluating model on 10% of test dataset  
pred_1 = model_test_on_dataset(d_test, limit=-0.1)
```

```
пред_1 = model.test_on_dataset(д_тест, 100%)
Metrics.print_all(d_test.labels[:len(pred_1)], pred_1, '10% of test')
```

100%

450/450 [01:06<00:00, 6.81it/s]

```
metrics for 10% of test:
```

```
accuracy 0.9956:
```

```
balanced accuracy 0.9956:
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1859: UserWarning:
warnings.warn('y_pred contains classes not in y_true')
```

0.6 | / |

Пример тестирования модели на полном наборе данных:

0 5 10 15 20 25 30

```
# 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%

4500/4500 [03:30<00:00, 21.40it/s]

```
metrics for test:
```

```
accuracy 0.9696:
```

```
balanced accuracy 0.9696:
```

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 экспортить 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', PROJECT_DIR)
pred = final_model.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, 'test-tiny')
```



```
Loading dataset test_tiny from npz.  
Done. Dataset test_tiny consists of 90 images.  
100% 90/90 [00:04<00:00, 20.57it/s]
```

metrics for test-tiny:

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 calculated {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:

```
# 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.

```
# 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 удалось 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.

```
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"

/content/drive/MyDrive/NN_course/NN_prac_1_data
updating: tmp/ (stored 0%)
updating: tmp/img1.png (stored 0%)
updating: tmp/img2.png (stored 0%)
test.npz      test_tiny.npz  tmp.zip    train_small.npz
test_small.npz   tmp          train.npz  train_tiny.npz
```

Распакуем архив tmp.zip в директорию tmp2 в PROJECT_DIR. Теперь внутри директории tmp2 содержится директория tmp, внутри которой находятся 2 изображения.

22.03.2021

problem_1_starter.ipynb - Colaboratory

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