

▼ 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 (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.12.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.16.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.6)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.6)
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) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2024.8.1)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7
```

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 = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-cLi',
    'train_small': '1qd45xFDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1I-2Z0uXLd4QwhZQ0ltp817Kn3J0Xgbui',
    'test': '1RfPou3pFKpuHDJZ-D9XDFzgvvwpUBF1Dr',
    'test_small': '1wbRsog0n7uG1HIPGLhyN-PMeT2kdQ21I',
    'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}

MODELS = {
    'BestModel': '1-C180uIULdZADmLNNvfaK8kNf6ArHJ30',
    'BigVGG': '1-h0GeDl4Zh_8knrkoSERm01d5-VooLeB',
    'EfficientNetB0': '1-1qYvI510RNTxKP4bp4uffFKe_ZLVpv2s',
    'VGG16': '1-LQeX4L_KDhRU6KXgABfinOA_tsg2wOk',
}
```

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 keras
from sklearn import svm
import matplotlib.pyplot as plt
from google.colab import drive
from tensorflow.keras.models import load_model
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import load_model
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten
```

```

import tensorflow as tf
import joblib
from joblib import dump, load
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.applications import EfficientNetB0
from sklearn.metrics import confusion_matrix
import seaborn as sns
import albumentations as A
import random
from tensorflow.keras.preprocessing.image import ImageDataGenerator

```

▼ 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, drive = True, new = True):
        self.name = name
        output = f'{name}.npz'
        self.is_loaded = False
        if drive and new:
            url = f"https://drive.google.com/uc?export=download&confirm=pbef&id={DATASETS_LINKS[name]}"
            gdown.download(url, output, quiet=False)
        if new:
            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]

```

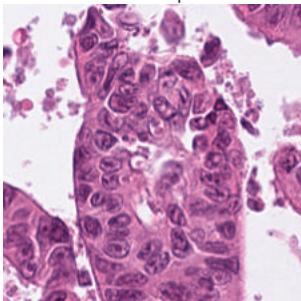
▼ Пример использования класса Dataset

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)
```

Got numpy array of shape (224, 224, 3), and label with code 8.
Label code corresponds to TUM 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 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.

```
pip install medmnist

→ Collecting medmnist
  Downloading medmnist-3.0.2-py3-none-any.whl.metadata (14 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from medmnist) (1.26.4)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from medmnist) (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from medmnist) (1.5.2)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-packages (from medmnist) (0.24.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from medmnist) (4.66.6)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from medmnist) (11.0.0)
Collecting fire (from medmnist)
  Downloading fire-0.7.0.tar.gz (87 kB)
    Preparing metadata (setup.py) ... done
      87.2/87.2 kB 2.4 MB/s eta 0:00:00
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from medmnist) (2.5.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (from medmnist) (0.20.1+cu121)
Requirement already satisfied: termcolor in /usr/local/lib/python3.10/dist-packages (from fire->medmnist) (2.5.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->medmnist) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->medmnist) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->medmnist) (2024.2)
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (1.13.1)
Requirement already satisfied: networkx>=2.8 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (3.4.2)
Requirement already satisfied: imageio>=2.33 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (2.36.0)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (2024.9)
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (24.2)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.10/dist-packages (from scikit-image->medmnist) (0.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->medmnist) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->medmnist) (3.5.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (4.12.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch->medmnist) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch->medmnist)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->medmnist)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->medmnist) (3.0.2)
Downloading medmnist-3.0.2-py3-none-any.whl (25 kB)
Building wheels for collected packages: fire
  Building wheel for fire (setup.py) ... done
    Created wheel for fire: filename=fire-0.7.0-py3-none-any.whl size=114249 sha256=27898693209b3c9dab0569aa736d4fa9011fe2892d9abd278
    Stored in directory: /root/.cache/pip/wheels/19/39/2f/2d3cadc408a8804103f1c34ddd4b9f6a93497b11fa96fe738e
Successfully built fire
Installing collected packages: fire, medmnist
Successfully installed fire-0.7.0 medmnist-3.0.2
```

Для начала обучил (EfficientNetB0, VGG16), чтобы потом использовать их как экстракторы

EfficientNetB0

```
def create_efficientnet_model(input_shape=(32, 32, 3), num_classes=10):
    base_model = EfficientNetB0(weights=None, include_top=False, input_shape=input_shape)
    base_model.trainable = True

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)

    predictions = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)

    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
return model
```

✓ data_small

```
d_train_small = Dataset('train_small', drive = True)
d_test_small = Dataset('test_small', drive = True)
```

⬇️ Downloading...

From: <https://drive.google.com/uc?export=download&confirm=pbef&id=1qd45xFxDwdZjktLFwQb-et-mAaFeCzOR>
 To: /content/train_small.npz
 100%|██████████| 841M/841M [00:07<00:00, 106MB/s]
 Loading dataset train_small from npz.
 Done. Dataset train_small consists of 7200 images.

Downloading...

From: <https://drive.google.com/uc?export=download&confirm=pbef&id=1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ2lI>
 To: /content/test_small.npz
 100%|██████████| 211M/211M [00:02<00:00, 87.4MB/s]
 Loading dataset test_small from npz.
 Done. Dataset test_small consists of 1800 images.

✓ tiny

```
d_train_tiny = Dataset('train_tiny', drive = True)
d_test_tiny = Dataset('test_tiny', drive = True)
```

⬇️ Downloading...

From: <https://drive.google.com/uc?export=download&confirm=pbef&id=1I-2Z0uXLd40whZQ0ltp817Kn3J0Xgbui>
 To: /content/train_tiny.npz
 100%|██████████| 105M/105M [00:01<00:00, 66.2MB/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=1viiB0s041CNsAK4itvX8PnYthJ-MDn0c>
 To: /content/test_tiny.npz
 100%|██████████| 10.6M/10.6M [00:00<00:00, 236MB/s] Loading dataset test_tiny from npz.
 Done. Dataset test_tiny consists of 90 images.

✓ обучение EfficientNetB0

Из дополнительного:

*Загрузка модели с какой-то конкретной итерации обучения (если используется итеративное обучение)
 *Построение матрицы ошибок, оценивание чувствительности и специфичности модели

```
model = create_efficientnet_model(input_shape=(224, 224, 3), num_classes=9)

batch_size = 60
epochs = 5
checkpoint_path = "/content/drive/My Drive/checkpoints/model_epoch_{epoch:02d}.weights.h5"

callbacks = [
    ModelCheckpoint(filepath=checkpoint_path, save_best_only=False, save_weights_only=True, monitor='val_loss', mode='min'),
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
]
y_train_one_hot = to_categorical(d_train_small.labels, num_classes=9)
y_test_one_hot = to_categorical(d_test_small.labels, num_classes=9)

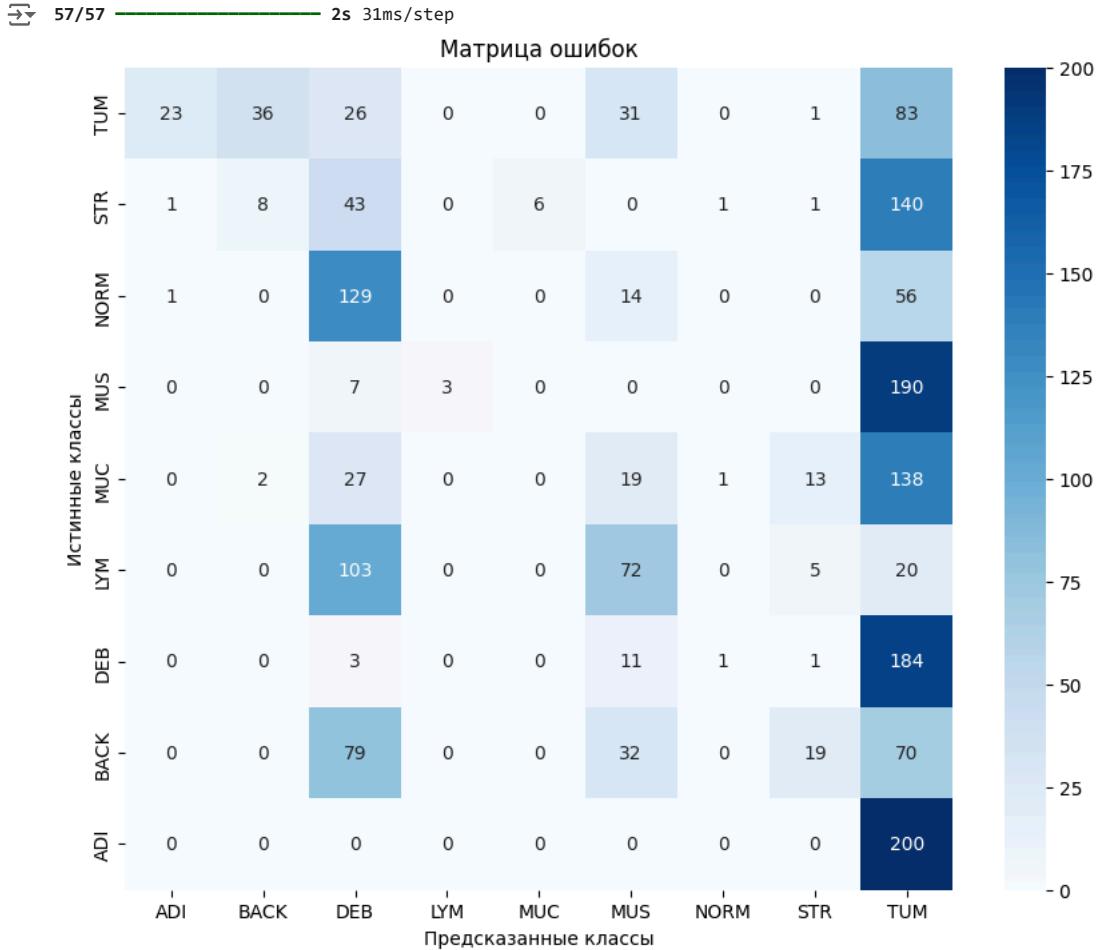
history = model.fit(
    d_train_small.images, y_train_one_hot,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(d_test_small.images, y_test_one_hot),
    callbacks=callbacks,
    shuffle=True
)
```

⬇️ Epoch 1/5
 120/120 ————— 93s 302ms/step - accuracy: 0.3447 - loss: 1.8673 - val_accuracy: 0.1111 - val_loss: 2.5193

```
Epoch 2/5
120/120 ————— 30s 250ms/step - accuracy: 0.6266 - loss: 0.9705 - val_accuracy: 0.1111 - val_loss: 4.3183
Epoch 3/5
120/120 ————— 30s 249ms/step - accuracy: 0.6795 - loss: 0.9161 - val_accuracy: 0.2161 - val_loss: 2.6683
Epoch 4/5
120/120 ————— 30s 249ms/step - accuracy: 0.7652 - loss: 0.6546 - val_accuracy: 0.3528 - val_loss: 2.3328
Epoch 5/5
120/120 ————— 30s 250ms/step - accuracy: 0.8000 - loss: 0.5867 - val_accuracy: 0.4289 - val_loss: 2.4729
```

```
y_true = d_test_small.labels
y_pred = np.argmax(model.predict(d_test_small.images), axis=1)

conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', xticklabels=TISSUE_CLASSES, yticklabels=TISSUE_CLASSES[::-1])
plt.xlabel("Предсказанные классы")
plt.ylabel("Истинные классы")
plt.title("Матрица ошибок")
plt.show()
```



Загружаем модель с 3 эпохи обучаем дальше

```
model.load_weights("/content/drive/My Drive/checkpoints/model_epoch_03.weights.h5")
model.fit(d_train_small.images, y_train_one_hot, epochs=5, initial_epoch=3, validation_data=(d_test_small.images, y_test_one_hot))

→ Epoch 4/5
225/225 ————— 100s 192ms/step - accuracy: 0.7218 - loss: 0.8251 - val_accuracy: 0.6628 - val_loss: 0.9801
Epoch 5/5
225/225 ————— 30s 133ms/step - accuracy: 0.7781 - loss: 0.6568 - val_accuracy: 0.2528 - val_loss: 6.7913
<keras.src.callbacks.History at 0x7904a4b19630>
```

✓ Оценка качества модели

```
test_loss, test_acc = model.evaluate(d_test.images, y_test_one_hot, verbose=2)

print(f'Test accuracy: {test_acc}')
print(f'Test loss: {test_loss}')
```

```
57/57 - 17s - 292ms/step - accuracy: 0.7511 - loss: 0.8308
Test accuracy: 0.751110901832581
Test loss: 0.830771267414093
```

- ↙ 57/57 - 17s - 292ms/step - accuracy: 0.7511 - loss: 0.8308
- Test accuracy: 0.751110901832581
- Test loss: 0.830771267414093
- ↙ Здесь, честно говоря, ради веселья добавил, чтобы модель переобучалась с параметров imagenet, но при этом ни один слой не замораживал. И результаты оказались получше, чем у прошлой модели))

Из дополнительного присутствует:

*Вывод различных показателей в процессе обучения (например, значение функции потерь на каждой эпохе)
 *Автоматическое тестирование на тестовом наборе/наборах данных после каждой эпохи обучения (при использовании итеративного обучения)
 *валидация модели на части обучающей выборки: я так понял это просто разделение на d_train.images и d_train.labels?

```
def create_vgg_model(input_shape=(224, 224, 3), num_classes=9):
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)
    base_model.trainable = True

    x = base_model.output

    x = GlobalAveragePooling2D()(x)

    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)

    predictions = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)

    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

```
modelVGG = create_vgg_model(input_shape=(224, 224, 3), num_classes=9)
```

```
batch_size = 60
epochs = 20

callbacks = [
    ModelCheckpoint('best_model.keras', save_best_only=True, monitor='val_loss', mode='min'),
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
]
y_train_one_hot = to_categorical(d_train_small.labels, num_classes=9)
y_test_one_hot = to_categorical(d_test_small.labels, num_classes=9)

history = modelVGG.fit(
    d_train_small.images, y_train_one_hot,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(d_test_small.images, y_test_one_hot),
    callbacks=callbacks,
    shuffle=True
)
```

```
drive.mount('/content/drive', force_remount=True)
```

```
→ Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_no\_top.h5 58889256/58889256 3s 0us/step
Epoch 1/20
120/120 ━━━━━━━━━━ 209s 1s/step - accuracy: 0.1155 - loss: 6.7437 - val_accuracy: 0.1433 - val_loss: 2.1405
Epoch 2/20
120/120 ━━━━━━━━━━ 114s 950ms/step - accuracy: 0.1618 - loss: 2.1122 - val_accuracy: 0.2472 - val_loss: 1.8901
Epoch 3/20
120/120 ━━━━━━━━━━ 114s 951ms/step - accuracy: 0.2730 - loss: 1.8478 - val_accuracy: 0.1822 - val_loss: 2.1443
Epoch 4/20
120/120 ━━━━━━━━━━ 116s 965ms/step - accuracy: 0.3180 - loss: 1.7179 - val_accuracy: 0.4894 - val_loss: 1.2310
Epoch 5/20
120/120 ━━━━━━━━━━ 116s 963ms/step - accuracy: 0.4964 - loss: 1.2650 - val_accuracy: 0.5517 - val_loss: 1.0963
Epoch 6/20
120/120 ━━━━━━━━━━ 115s 959ms/step - accuracy: 0.5581 - loss: 1.1089 - val_accuracy: 0.5539 - val_loss: 1.1346
Epoch 7/20
120/120 ━━━━━━━━━━ 115s 961ms/step - accuracy: 0.6193 - loss: 0.9977 - val_accuracy: 0.6483 - val_loss: 0.9156
Epoch 8/20
120/120 ━━━━━━━━━━ 115s 960ms/step - accuracy: 0.6424 - loss: 0.9223 - val_accuracy: 0.5983 - val_loss: 1.0467
```

```

Epoch 9/20          115s 959ms/step - accuracy: 0.6522 - loss: 0.9120 - val_accuracy: 0.6128 - val_loss: 1.0001
120/120          115s 963ms/step - accuracy: 0.6912 - loss: 0.8497 - val_accuracy: 0.7428 - val_loss: 0.6607
Epoch 11/20
120/120          115s 959ms/step - accuracy: 0.7440 - loss: 0.6818 - val_accuracy: 0.7206 - val_loss: 0.7242
Epoch 12/20
120/120          115s 961ms/step - accuracy: 0.7597 - loss: 0.6291 - val_accuracy: 0.7572 - val_loss: 0.6759
Epoch 13/20
120/120          116s 964ms/step - accuracy: 0.7924 - loss: 0.5751 - val_accuracy: 0.7989 - val_loss: 0.5817
Epoch 14/20
120/120          116s 969ms/step - accuracy: 0.8111 - loss: 0.5215 - val_accuracy: 0.8389 - val_loss: 0.4695
Epoch 15/20
120/120          115s 957ms/step - accuracy: 0.8313 - loss: 0.4846 - val_accuracy: 0.7589 - val_loss: 0.6353
Epoch 16/20
120/120          115s 959ms/step - accuracy: 0.8347 - loss: 0.4527 - val_accuracy: 0.8222 - val_loss: 0.5081
Epoch 17/20
120/120          116s 966ms/step - accuracy: 0.8671 - loss: 0.3877 - val_accuracy: 0.8550 - val_loss: 0.4144
Epoch 18/20
120/120          115s 963ms/step - accuracy: 0.8945 - loss: 0.3148 - val_accuracy: 0.8583 - val_loss: 0.4404
Epoch 19/20
120/120          116s 964ms/step - accuracy: 0.8908 - loss: 0.3400 - val_accuracy: 0.8800 - val_loss: 0.3669
Epoch 20/20
120/120          115s 960ms/step - accuracy: 0.8923 - loss: 0.3250 - val_accuracy: 0.8556 - val_loss: 0.4515
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```

- ✓ Здесь уже точность получилась более или менее приемлемая

```

model_VGG = load_model("/content/drive/My Drive/models/VGG16.keras")
test_loss, test_acc = model_VGG.evaluate(d_test.images, y_test_one_hot, verbose=2)

print(f'Test accuracy: {test_acc}')
print(f'Test loss: {test_loss}')

```

57/57 - 30s - 518ms/step - accuracy: 0.8800 - loss: 0.3669
Test accuracy: 0.879999952316284
Test loss: 0.36687713861465454

- ✓ Повторение модели VGG. Для основной модели построить график я забыл, а заново её обучать как-то не хотелось, в общем решил обучить такую же

```

def create_vgg_model(input_shape=(224, 224, 3), num_classes=9):
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)
    base_model.trainable = True

    x = base_model.output

    x = GlobalAveragePooling2D()(x)

    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)

    predictions = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=base_model.input, outputs=predictions)

    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

    return model

```

```
RepeatModel = create_vgg_model(input_shape=(224, 224, 3), num_classes=9)
```

```

batch_size = 30
epochs = 20

callbacks = [
    ModelCheckpoint('best_model.keras', save_best_only=True, monitor='val_loss', mode='min'),
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
]
y_train_one_hot = to_categorical(d_train_small.labels, num_classes=9)
y_test_one_hot = to_categorical(d_test_small.labels, num_classes=9)
# d_train_small.images = d_train_small.images / 255.0

```

```
RepeatHistory = RepeatModel.fit(
    d_train_small.images, y_train_one_hot,
```

```
batch_size=batch_size,
epochs=epochs,
validation_data=(d_test_small.images, y_test_one_hot),
callbacks=callbacks,
shuffle=True
)
```

Показать скрытые выходные данные

```
test_loss, test_acc = RepeatModel.evaluate(d_test_small.images, y_test_one_hot, verbose=2)

print(f'Test accuracy: {test_acc}')
print(f'Test loss: {test_loss}'
```

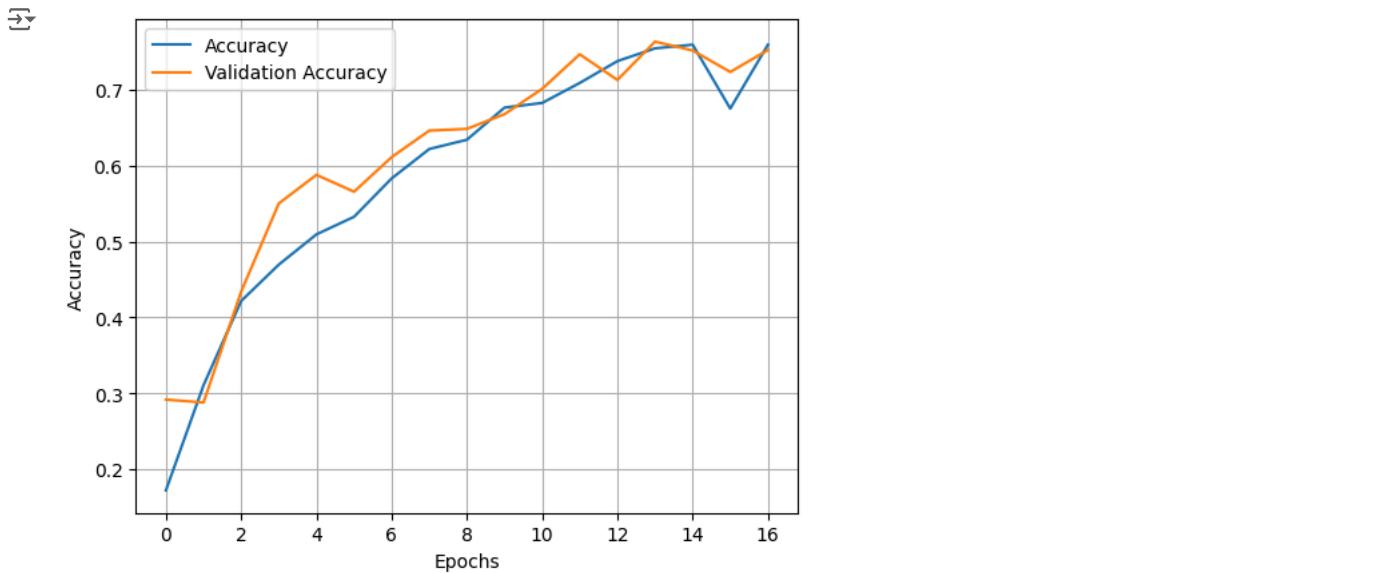
57/57 - 12s - 218ms/step - accuracy: 0.7633 - loss: 0.5985
Test accuracy: 0.763333206176758
Test loss: 0.5985029935836792

▼ График зависимости точности от эпохи

Из дополнительного присутствует:

*Построение графиков, визуализирующих процесс обучения (график зависимости функции потерь от номера эпохи обучения)

```
plt.plot(RepeatHistory.history["accuracy"], label="Accuracy")
plt.plot(RepeatHistory.history["val_accuracy"], label="Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



▼ data_big

```
d_test_big = Dataset('test')
d_train_big = Dataset('train')
```

Downloading...
From: <https://drive.google.com/uc?export=download&confirm=pbef&id=1RfPou3pFKpuHDJZ-D9XDFzgvwpUBF1Dr>
To: /content/test.npz
100% [██████████] 525M/525M [00:05<00:00, 98.5MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
Downloading...
From: <https://drive.google.com/uc?export=download&confirm=pbef&id=1Xt0zV05XbrfxpLHJuL0XBGJ5U7CS-cLi>
To: /content/train.npz
100% [██████████] 2.10G/2.10G [00:16<00:00, 125MB/s]
Loading dataset train from npz.

Done. Dataset train consists of 18000 images.

❖ Здесь я дообучаю экстрактор уже на всех данных

Из дополнительного пристутствует:

*Реализация возможности дообучения модели (на новом наборе данных, или, например, при экстренном закрытии Google Colab)

```
BigModelVGG = load_model("/content/drive/My Drive/models/VGG16.keras")

batch_size = 60
epochs = 50

callbacks = [
    ModelCheckpoint('best_model.keras', save_best_only=True, monitor='val_loss', mode='min'),
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
]
y_train_one_hot = to_categorical(d_train_big.labels, num_classes=9)
y_test_one_hot = to_categorical(d_test_big.labels, num_classes=9)

history = BigModelVGG.fit(
    d_train_big.images, y_train_one_hot,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(d_test_big.images, y_test_one_hot),
    callbacks=callbacks,
    shuffle=True
)

BigModelVGG.save('/content/drive/My Drive/models/BigVGG.keras')
BigModelVGG.save('/content/drive/My Drive/models/BigVGG.h5')

→ Epoch 1/50
300/300 ━━━━━━━━━━ 295s 970ms/step - accuracy: 0.8896 - loss: 0.3308 - val_accuracy: 0.8896 - val_loss: 0.3133
Epoch 2/50
300/300 ━━━━━━━━━━ 287s 957ms/step - accuracy: 0.8927 - loss: 0.3233 - val_accuracy: 0.8931 - val_loss: 0.3393
Epoch 3/50
300/300 ━━━━━━━━━━ 287s 958ms/step - accuracy: 0.9059 - loss: 0.2817 - val_accuracy: 0.8789 - val_loss: 0.3466
Epoch 4/50
300/300 ━━━━━━━━━━ 288s 962ms/step - accuracy: 0.9194 - loss: 0.2381 - val_accuracy: 0.9267 - val_loss: 0.2313
Epoch 5/50
300/300 ━━━━━━━━━━ 287s 958ms/step - accuracy: 0.9406 - loss: 0.1969 - val_accuracy: 0.8998 - val_loss: 0.2964
Epoch 6/50
300/300 ━━━━━━━━━━ 287s 956ms/step - accuracy: 0.9353 - loss: 0.2034 - val_accuracy: 0.9367 - val_loss: 0.1958
Epoch 7/50
300/300 ━━━━━━━━━━ 286s 953ms/step - accuracy: 0.9391 - loss: 0.1800 - val_accuracy: 0.9311 - val_loss: 0.2256
Epoch 8/50
300/300 ━━━━━━━━━━ 286s 953ms/step - accuracy: 0.9483 - loss: 0.1583 - val_accuracy: 0.9204 - val_loss: 0.2352
Epoch 9/50
300/300 ━━━━━━━━━━ 286s 953ms/step - accuracy: 0.9451 - loss: 0.1636 - val_accuracy: 0.9196 - val_loss: 0.2604
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

❖ Уже конкурентоспособная точность

```
model_VGG_bigboy = load_model("/content/drive/My Drive/models/BigVGG.keras")
test_loss, test_acc = model_VGG_bigboy.evaluate(d_test_big.images, y_test_one_hot, verbose=2)

print(f'Test accuracy: {test_acc}')
print(f'Test loss: {test_loss}')

→ 141/141 - 34s - 238ms/step - accuracy: 0.9367 - loss: 0.1958
Test accuracy: 0.9366666674613953
Test loss: 0.1958276778459549
```

❖ Здесь я цепляю SVM на экстрактор

Из дополнительного:

*не знаю считается ли это улучшением не из списка, но...

```
modelBigVGG_SVM = load_model("/content/drive/My Drive/models/BigVGG.keras")
```

```
# x = modelBigVGG_SVM.output
x = modelBigVGG_SVM.layers[-2].output

feature_extractor = Model(inputs=modelBigVGG_SVM.input, outputs=x)

train_features = feature_extractor.predict(d_train.images)
test_features = feature_extractor.predict(d_test.images)

param_grid = {
    'kernel': ['linear', 'rbf', 'poly'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto']
}

svm = SVC()

grid_search = GridSearchCV(estimator=svm, param_grid=param_grid, cv=5, verbose=2, n_jobs=-1)

grid_search.fit(train_features, d_train.labels)

print("Best parameters found: ", grid_search.best_params_)

y_pred_SVM = grid_search.predict(test_features)

→ 225/225 ━━━━━━━━ 26s 116ms/step
57/57 ━━━━━━━━ 7s 127ms/step
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Best parameters found: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

▼ Добавка к качеству есть)

```
accuracy = accuracy_score(d_test_big.labels, y_pred_SVM)
print(f'Accuracy of SVM with EfficientNetB0 features: {accuracy}')
```

→ Accuracy of SVM with EfficientNetB0 features: 0.9388888888888889

▼ Попробовал вместо SVM случайный лес

Здесь из дополнительного присутствует:

*Автоматическая кросс-валидация в grid_search
 *Автоматический выбор гиперпараметров модели во время обучения
 *Автоматическое сохранение и визуализация результатов тестирования: в best_params_ сохранены лучшие гиперпараметры модели

```
x = modelBigVGG_SVM.layers[-2].output

feature_extractor = Model(inputs=modelBigVGG_SVM.input, outputs=x)

train_features = feature_extractor.predict(d_train_big.images)
test_features = feature_extractor.predict(d_test_big.images)

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_features': ['log2', 'sqrt'],
    'max_depth': [None, 10, 20, 30],
}

rf = RandomForestClassifier()

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose=2, n_jobs=-1)

grid_search.fit(train_features, d_train_big.labels)

print("Best parameters found: ", grid_search.best_params_)

y_pred_RandomForest = grid_search.predict(test_features)

→ 225/225 ━━━━━━━━ 29s 129ms/step
57/57 ━━━━━━━━ 7s 121ms/step
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best parameters found: {'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 200}

accuracy = accuracy_score(d_test.labels, y_pred_RandomForest)
print(f'Accuracy of SVM with RandomForestClassifier features: {accuracy}')
```

→ Accuracy of SVM with RandomForestClassifier features: 0.9361111111111111

✓ Для случайного леса

Из дополнительного пристутствует:

*Автоматическое сохранение модели при обучении

```
class My_Model:

    def __init__(self, SVM_or_RandomForest = "SVM", base_model = "BestModel", onlyNN = True):
        # todo
        self.onlyNN = onlyNN
        model = MODELS[base_model]
        try:
            url = f"https://drive.google.com/uc?export=download&id={model}"
            output = f'{base_model}.keras'
            gdown.download(url, output, quiet=False)
            model_path = f'/content/{output}'
            modelBigVGG = load_model(f"/content/{output}")
        except Exception as e:
            print(f"Произошла ошибка при скачивании файла с диска: {e}")
            print("Скачайте файл к себе в окружение")
            modelBigVGG = load_model('/content/sample_data/BigVGG.keras')
        if not onlyNN:
            x = modelBigVGG.layers[-2].output
            self.feature_extractor = Model(inputs=modelBigVGG.input, outputs=x)

        if SVM_or_RandomForest == "SVM":
            self.model = SVC(kernel="rbf", C=10, gamma="scale")
        elif SVM_or_RandomForest == "Forest":
            self.model = RandomForestClassifier(max_depth=None, max_features="sqrt", n_estimators=200)
        else:
            self.model = modelBigVGG
        pass

    def save(self, name: str):
        if not self.onlyNN:
            dump(self.model, f'/content/{name}.joblib')
        else:
            self.model.save(f"/content/{name}.keras")
        pass

    def load(self, name = "BestModel"):
        # todo
        model = MODELS[name]
        url = f"https://drive.google.com/uc?export=download&id={model}"
        try:
            if not self.onlyNN:
                output = f'{name}.joblib'
                gdown.download(url, output, quiet=False)
                model_path = f'/content/{output}'
                self.model = load(model_path)
            else:
                output = f'{name}.keras'
                gdown.download(url, output, quiet=False)
                model_path = f'/content/{output}'
                self.model = load_model(f"/content/{output}")
        except Exception as e:
            print(f"Не получилось загрузить: {e}")
        print(f"Модель {name} загружена")
        pass

    def train(self, dataset: Dataset):
        self.train_features = self.feature_extractor.predict(dataset.images)
        self.model.fit(self.train_features, dataset.labels)
        try:
            dump(self.model, '/content/mymodel.joblib')
            print("Модель сохранена: mymodel")
        except Exception as e:
            print(f"Не удалось сохранить модель: {e}")
        pass

    def trainNN(self, train: Dataset, test: Dataset):
        batch_size = 20
        epochs = 100

        callbacks = [
            ModelCheckpoint('best_model.keras', save_best_only=True, monitor='val_loss', mode='min'),
            EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True),
            ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-6, verbose=1)
        ]
```

```

y_train_one_hot = to_categorical(train.labels, num_classes=9)
y_test_one_hot = to_categorical(test.labels, num_classes=9)

for layer in BestModel.layers[:-4]:
    layer.trainable = False

history = BestModel.fit(
    train.images, y_train_one_hot,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(test.images, y_test_one_hot),
    callbacks=callbacks,
    shuffle=True
)
try:
    self.model.save("mymodel")
    print("Модель сохранена: mymodel")
except Exception as e:
    print(f"Не удалось сохранить модель: {e}")

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:
    batched_image = np.expand_dims(img, axis=0)
    if not self.onlyNN:
        features = self.feature_extractor.predict(batched_image)
        prediction = self.model.predict(features)
    else:
        prob = self.model.predict(batched_image)
        prediction = np.argmax(prob, axis=1)[0]
    return prediction

my_model = My_Model(SVM_or_RandomForest="Forest")

my_model.train(d_train_big)

my_model.save("verybest")

my_model_SVM = My_Model(SVM_or_RandomForest="SVM")
my_model_SVM.train(d_train_big)
my_model_SVM.save("verybestSVM")


```

▼ 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'.

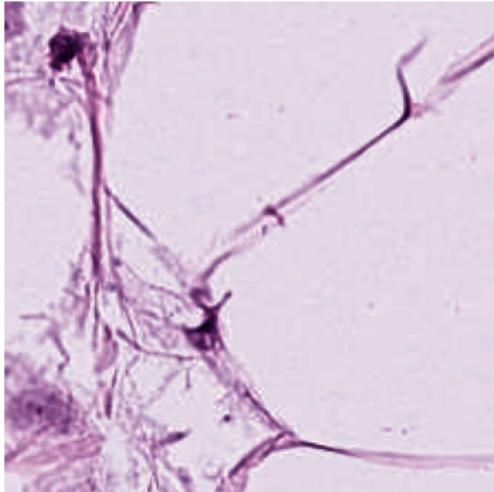
```

def view(dataset, num = 0):
    plt.imshow(dataset.images[num])
    plt.axis('off')
    plt.title(f"Image with label {dataset.labels[num]}")
    plt.show()
view(d_train_big, 10)

```



Image with label 0



Я, так понял, здесь просто проверяется работа самого класса Model(который у меня назван My_Model,
✓ чтобы не было разногласий с Model из Keras), поэтому у меня есть model, которая я обучил на
небольшом датасете и my_model, которую я обучил на большом датасете

```
model = My_Model()  
model.train(d_train_small)  
model.save('best')  
model.load('best')  
  
→ 225/225 ━━━━━━━━ 29s 128ms/step  
Модель best загружена
```

Пример тестирования модели на части набора данных:

```
pred_1 = model.test_on_dataset(d_test_small, limit=0.1)  
Metrics.print_all(d_test_small.labels[:len(pred_1)], pred_1, '10% of test')
```

→ Показать скрытые выходные данные

metrics for 10% of test:

```
accuracy 0.9778:  
balanced accuracy 0.9778:
```

Пример тестирования модели на полном наборе данных:

✓ my_model(Случайный лес) на d_test_big

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

→ Показать скрытые выходные данные

metrics for test:

```
accuracy 0.9467:  
balanced accuracy 0.9467:
```

✓ my_model_SVM на d_test_big

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

➡️ Показать скрытые выходные данные

Здесь такие значения, но слишком много step(by step))

metrics for test:

```
accuracy 0.9487:
balanced accuracy 0.9487:
```

▼ Здесь я n-раз запускал код, чтобы хотя бы одна картинка была предсказана неверно

```
images = d_test_tiny.images
labels = d_test_tiny.labels

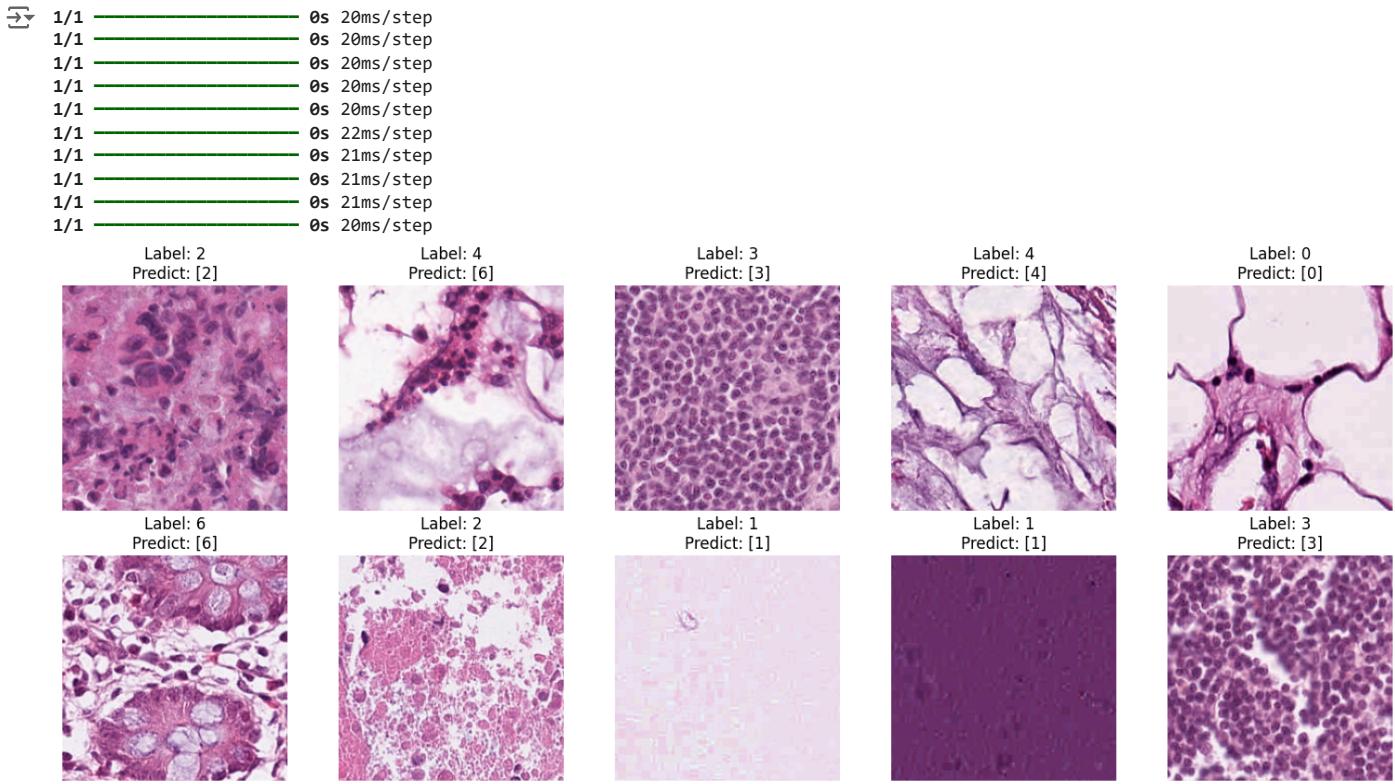
n_rows = 2
n_cols = 5
n_images = n_rows * n_cols

if len(images) > n_images:
    random_indices = random.sample(range(len(images)), n_images)
    sampled_images = [images[i] for i in random_indices]
    sampled_labels = [labels[i] for i in random_indices]
else:
    sampled_images = images
    sampled_labels = labels

fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 3, n_rows * 3))

for i, ax in enumerate(axes.ravel()):
    if i < len(sampled_images):
        ax.imshow(sampled_images[i], cmap='gray')
        ax.axis('off')
        prediction = my_model_SVM.test_on_image(sampled_images[i])
        ax.set_title(f"Label: {sampled_labels[i]}\nPredict: {prediction}")
    else:
        ax.axis('off')

plt.tight_layout()
plt.show()
```



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

Показать скрытые выходные данные

metrics for test-tiny:

```
accuracy 0.9000:
balanced accuracy 0.9000:
```

```
final_model_SVM = My_Model()
final_model_SVM.load('verybestSVM')
d_test_tiny = Dataset('test_tiny')
pred = final_model_SVM.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, 'test-tiny')
```

Показать скрытые выходные данные

metrics for test-tiny:

```
accuracy 0.9000:
```

```
balanced accuracy 0.9000:
```

Здесь я пытаюсь обучить самую качественную модель с помощью аугментации

Посмотрев на изображения, мне показалось, имеет смысл использовать различные геометрические преобразования, возможно поможет размытие ...

```
import albumentations as A

def get_augmentation_pipeline():
    return A.Compose(
        [
            # Геометрические преобразования
            A.HorizontalFlip(p=0.5),
            A.VerticalFlip(p=0.3),
            A.ShiftScaleRotate(
                shift_limit=0.1, scale_limit=0.1, rotate_limit=10, p=0.5
            ),
            A.ElasticTransform(alpha=1, sigma=50, alpha_affine=50, p=0.5),
            A.GridDistortion(num_steps=5, distort_limit=0.3, p=0.4),
            A.OpticalDistortion(distort_limit=0.05, shift_limit=0.05, p=0.3),

            # Добавление шума и размытия
            A.OneOf(
                [
                    A.GaussianBlur(blur_limit=(3, 5), p=0.4),
                    A.MotionBlur(blur_limit=3, p=0.4),
                    A.GaussNoise(var_limit=(10, 30), p=0.4),
                ],
                p=0.5
            ),

            # Изменение цветовых каналов
            A.HueSaturationValue(hue_shift_limit=10, sat_shift_limit=15, val_shift_limit=10, p=0.5),
            A.RGBShift(r_shift_limit=15, g_shift_limit=15, b_shift_limit=15, p=0.4),
            A.RandomBrightnessContrast(brightness_limit=0.2, contrast_limit=0.2, p=0.5),

            # Обрезка и маскирование
            A.RandomResizedCrop(height=224, width=224, scale=(0.8, 1.0), ratio=(0.9, 1.1), p=0.5),
            A.CoarseDropout(max_holes=4, max_height=20, max_width=20, fill_value=0, p=0.5)
        ]
    )

def apply_batch_augmentations(batch_images):
    aug_pipeline = get_augmentation_pipeline()
    augmented_images = []

    for i in range(batch_images.shape[0]):
        augmented = aug_pipeline(image=batch_images[i])
        augmented_images.append(augmented["image"])

    return np.array(augmented_images)

augmented_tensor = apply_batch_augmentations(d_train_tiny.images)
```

```
→ <ipython-input-45-4b665b856090>:12: UserWarning: Argument 'alpha_affine' is not valid and will be ignored.  
A.ElasticTransform(alpha=1, sigma=50, alpha_affine=50, p=0.5),
```

```

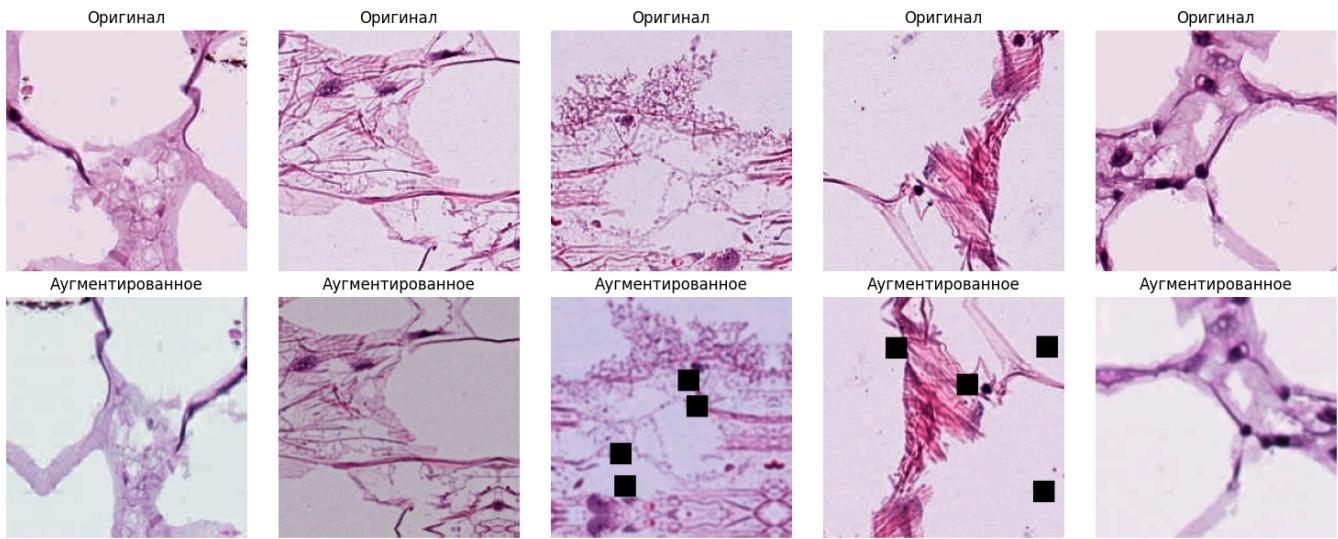
n_rows = 2
n_cols = 5
fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 3, n_rows * 3))
for i in range(n_cols):
    axes[0, i].imshow(d_train_tiny.images[i], cmap='gray')
    axes[0, i].axis('off')
    axes[0, i].set_title("Оригинал")

    augmented_image = get_augmentation_pipeline()(image=d_train_tiny.images[i])['image']
    axes[1, i].imshow(augmented_image, cmap='gray')
    axes[1, i].axis('off')
    axes[1, i].set_title("Аугментированное")

plt.tight_layout()
plt.show()

```

→ <ipython-input-45-4b665b856090>:12: UserWarning: Argument 'alpha_affine' is not valid and will be ignored.
A.ElasticTransform(alpha=1, sigma=50, alpha_affine=50, p=0.5),



▼ Аугментация больших данных

```

augmented_tensor = apply_batch_augmentations(d_train_big.images)

→ <ipython-input-48-f45bed83c1fe>:12: UserWarning: Argument 'alpha_affine' is not valid and will be ignored.  
A.ElasticTransform(alpha=1, sigma=50, alpha_affine=50, p=0.5),

```

```

augmented_tensor_test = apply_batch_augmentations(d_test_big.images)

→ <ipython-input-48-f45bed83c1fe>:12: UserWarning: Argument 'alpha_affine' is not valid and will be ignored.  
A.ElasticTransform(alpha=1, sigma=50, alpha_affine=50, p=0.5),

```

```
BestModel = load_model("/content/drive/My Drive/models/BigVGG.keras")
```

```

batch_size = 20
epochs = 100

callbacks = [
    ModelCheckpoint('best_model.keras', save_best_only=True, monitor='val_loss', mode='min'),
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-6, verbose=1)
]

```

```
y_train_one_hot = to_categorical(d_train_big.labels, num_classes=9)
y_test_one_hot = to_categorical(d_test_big.labels, num_classes=9)
```

```

def random_hsv(image):
    hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    h, s, v = cv2.split(hsv)

    s = cv2.add(s, np.random.randint(-40, 40))
    v = cv2.add(v, np.random.randint(-30, 30))
    s = np.clip(s, 0, 255)
    v = np.clip(v, 0, 255)

```

```

hsv = cv2.merge([h, s, v])
image = cv2.cvtColor(hsv, cv2.COLOR_HSV2RGB)
return image

datagen = ImageDataGenerator(
    rotation_range=15,           # Небольшие повороты на 15 градусов
    width_shift_range=0.2,        # Горизонтальные сдвиги до 20%
    height_shift_range=0.2,       # Вертикальные сдвиги до 20%
    shear_range=0.1,             # Искривление изображения
    zoom_range=[0.8, 1.2],        # Масштабирование (80% - 120%)
    horizontal_flip=True,         # Отражение по горизонтали
    vertical_flip=True,           # Отражение по вертикали
    brightness_range=[0.8, 1.2],   # Изменение яркости изображения
    channel_shift_range=20.0,      # Изменение цветов в канале RGB
    fill_mode='nearest',          # Заполнение пустых мест
    preprocessing_function=random_hsv # Собственная обработка цвета, если необходимо
)

for layer in BestModel.layers[:-4]:
    layer.trainable = False

history = BestModel.fit(
    d_train_big.images, y_train_one_hot,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(d_test_big.images, y_test_one_hot),
    callbacks=callbacks,
    shuffle=True
)

Epoch 1/100
900/900 —————— 126s 126ms/step - accuracy: 0.9619 - loss: 0.1213 - val_accuracy: 0.9453 - val_loss: 0.1755 - learning_
Epoch 2/100
900/900 —————— 110s 122ms/step - accuracy: 0.9611 - loss: 0.1226 - val_accuracy: 0.9444 - val_loss: 0.1732 - learning_
Epoch 3/100
900/900 —————— 109s 121ms/step - accuracy: 0.9638 - loss: 0.1127 - val_accuracy: 0.9442 - val_loss: 0.1809 - learning_
Epoch 4/100
900/900 —————— 0s 97ms/step - accuracy: 0.9654 - loss: 0.1134
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.000500000237487257.
900/900 —————— 109s 121ms/step - accuracy: 0.9654 - loss: 0.1134 - val_accuracy: 0.9471 - val_loss: 0.1774 - learning_
Epoch 5/100
900/900 —————— 110s 122ms/step - accuracy: 0.9640 - loss: 0.1131 - val_accuracy: 0.9469 - val_loss: 0.1703 - learning_
Epoch 6/100
900/900 —————— 109s 122ms/step - accuracy: 0.9720 - loss: 0.0918 - val_accuracy: 0.9458 - val_loss: 0.1691 - learning_
Epoch 7/100
900/900 —————— 109s 121ms/step - accuracy: 0.9704 - loss: 0.0952 - val_accuracy: 0.9469 - val_loss: 0.1714 - learning_
Epoch 8/100
900/900 —————— 0s 97ms/step - accuracy: 0.9687 - loss: 0.0990
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
900/900 —————— 109s 121ms/step - accuracy: 0.9687 - loss: 0.0990 - val_accuracy: 0.9473 - val_loss: 0.1725 - learning_
Epoch 9/100
900/900 —————— 109s 121ms/step - accuracy: 0.9714 - loss: 0.0918 - val_accuracy: 0.9462 - val_loss: 0.1706 - learning_

```

```
BestModel.save("/content/drive/My Drive/models/BestModel.keras")
```

```
BestModelSVM = My_Model(onlyNN = True)
```

```
→ /usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:713: UserWarning: Skipping variable loading for optimizer 'ac
saveable.load_own_variables(weights_store.get(inner_path))
```

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

[Показать скрытые выходные данные](#)

metrics for test-tiny:

```
accuracy 0.9111:
balanced accuracy 0.9111:
```

Здесь опять нацепил SVM

```
BestModel_SVM = My_Model()
```

```
↳ /usr/local/lib/python3.10/dist-packages/keras/src/saving/saving_lib.py:713: UserWarning: Skipping variable loading for optimizer 'ac
saveable.load_own_variables(weights_store.get(inner_path))
```

```
BestModel_SVM.train(d_train_big)
pred = BestModel_SVM.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, 'test-tiny')
```

Показать скрытые выходные данные

metrics for test-tiny:

```
accuracy 0.9000:
balanced accuracy 0.9000:
```

```
NN = My_Model()
pred = NN.test_on_dataset(d_test_big)
Metrics.print_all(d_test_big.labels, pred, 'test-big')
```

Показать скрытые выходные данные

```
svm = My_Model(onlyNN = False)
svm.train(d_train_big)
pred = svm.test_on_dataset(d_test_big)
Metrics.print_all(d_test_big.labels, pred, 'test-big')
```

Показать скрытые выходные данные

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.')

 Function f is calculated 128 times in 0.0290673549998246s.
```

▼ 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)

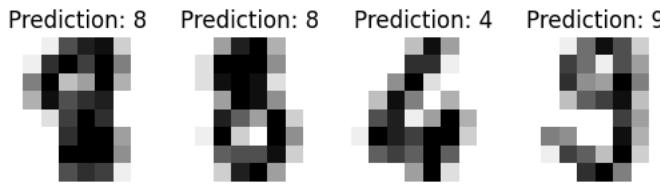
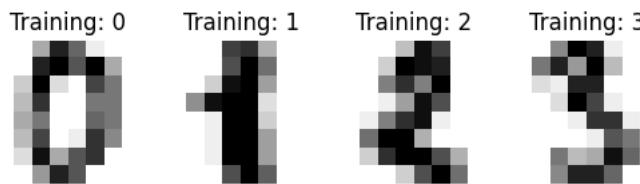
images_and_predictions = list(zip(digits.images[n_samples // 2:], predicted))
for ax, (image, prediction) in zip(axes[1, :], images_and_predictions[:4]):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Prediction: %i' % prediction)

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()
```

Classification report for classifier SVC(gamma=0.001):

	precision	recall	f1-score	support
0	1.00	0.99	0.99	88
1	0.99	0.97	0.98	91
2	0.99	0.99	0.99	86
3	0.98	0.87	0.92	91
4	0.99	0.96	0.97	92
5	0.95	0.97	0.96	91
6	0.99	0.99	0.99	91
7	0.96	0.99	0.97	89
8	0.94	1.00	0.97	88
9	0.93	0.98	0.95	92
accuracy			0.97	899
macro avg	0.97	0.97	0.97	899
weighted avg	0.97	0.97	0.97	899



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)

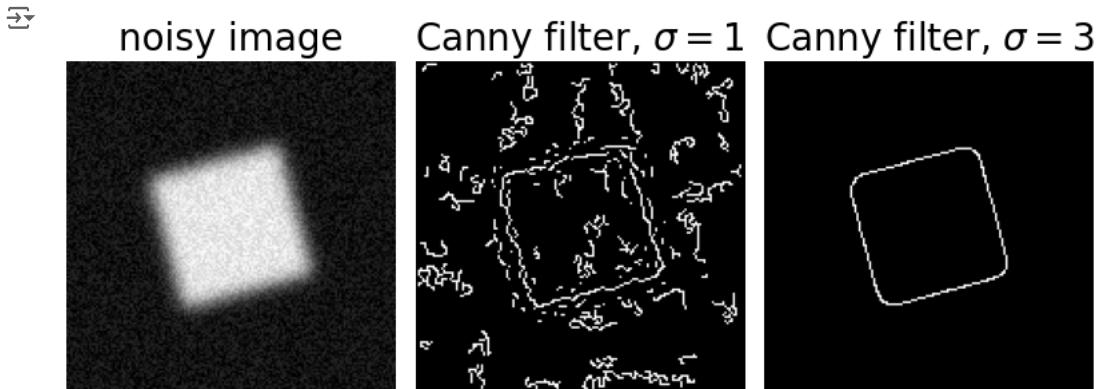
ax1.imshow(im, cmap=plt.cm.gray)
ax1.axis('off')
ax1.set_title('noisy image', fontsize=20)

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

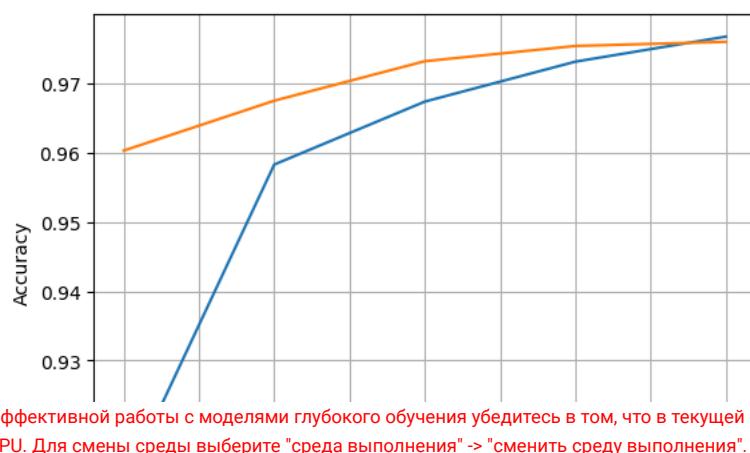
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

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'])
history = model.fit(
    x_train,
    y_train,
    validation_data=(x_test, y_test),
    epochs=5
)
model.evaluate(x_test, y_test, verbose=2)

Epoch 1/5
1875/1875 ━━━━━━━━ 5s 2ms/step - accuracy: 0.8569 - loss: 0.4894 - val_accuracy: 0.9603 - val_loss: 0.1376
Epoch 2/5
1875/1875 ━━━━━━ 3s 1ms/step - accuracy: 0.9558 - loss: 0.1529 - val_accuracy: 0.9675 - val_loss: 0.1077
Epoch 3/5
1875/1875 ━━━━ 3s 1ms/step - accuracy: 0.9661 - loss: 0.1111 - val_accuracy: 0.9732 - val_loss: 0.0863
Epoch 4/5
1875/1875 ━━━━ 3s 1ms/step - accuracy: 0.9727 - loss: 0.0882 - val_accuracy: 0.9754 - val_loss: 0.0826
Epoch 5/5
1875/1875 ━━━━ 3s 1ms/step - accuracy: 0.9777 - loss: 0.0721 - val_accuracy: 0.9760 - val_loss: 0.0800
313/313 - 0s - 1ms/step - accuracy: 0.9760 - loss: 0.0800
[0.08003123104572296, 0.9760000109672546]

plt.plot(history.history["accuracy"], label="Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```



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".