

✓ Практическое задание №1

Установка необходимых пакетов:

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

Downloading nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
99.1/99.1 kB 13.0 MB/s eta 0:00:00
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch==2.1.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch==2.1.0->torchvision)
Downloading nvidia_nvjitlink_cu12-12.3.101-py3-none-manylinux1_x86_64.whl (20.5 MB)
20.5/20.5 MB 59.3 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from torch==2.1.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->torch==2.1.0)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->torch==2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->torch==2.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2==3.1.2->torch==2.1.0)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy==1.11.1->torch==2.1.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand
Attempting uninstall: torch
Found existing installation: torch 2.1.0+cu118
Uninstalling torch-2.1.0+cu118:
Successfully uninstalled torch-2.1.0+cu118
Attempting uninstall: torchvision
Found existing installation: torchvision 0.16.0+cu118
Uninstalling torchvision-0.16.0+cu118:
Successfully uninstalled torchvision-0.16.0+cu118
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed
torchdata 2.1.0+cu118 requires torch==2.1.0, but you have torch 2.1.1 which is incompatible.
torchdata 0.7.0 requires torch==2.1.0, but you have torch 2.1.1 which is incompatible.
torchtext 0.16.0 requires torch==2.1.0, but you have torch 2.1.1 which is incompatible.
Successfully installed nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.1.1)
Collecting efficientnet_pytorch
Downloading efficientnet_pytorch-0.7.1.tar.gz (21 kB)
Preparing metadata (setup.py) ... done
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.11.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.2.1)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==8.9.2.26 in /usr/local/lib/python3.10/dist-packages (from torch) (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.10/dist-packages (from torch) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.10/dist-packages (from torch) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.10/dist-packages (from torch) (11.4.5.107)
Requirement already satisfied: nvidia-cusparselt-cu12==1.0.106 in /usr/local/lib/python3.10/dist-packages (from torch) (1.0.106)
Requirement already satisfied: nvidia-nccl-cu12==2.18.1 in /usr/local/lib/python3.10/dist-packages (from torch) (2.18.1)
Requirement already satisfied: nvidia-nvtx-cu12==12.1.105 in /usr/local/lib/python3.10/dist-packages (from torch) (12.1.105)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.1.0)
Requirement already satisfied: nvidia-nvjitlink-cu12 in /usr/local/lib/python3.10/dist-packages (from torch) (12.3.101)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2==3.1.2->torch==2.1.0)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy==1.11.1->torch==2.1.0)
Building wheels for collected packages: efficientnet_pytorch
Building wheel for efficientnet_pytorch (setup.py) ... done
Created wheel for efficientnet_pytorch: filename=efficientnet_pytorch-0.7.1-py3-none-any.whl size=164
Stored in directory: /root/.cache/pip/wheels/03/3f/e9/911b1bc46869644912bda90a56bcf7b960f20b5187f6ea3
Successfully built efficientnet_pytorch
Installing collected packages: efficientnet_pytorch
Successfully installed efficientnet_pytorch-0.7.1
```

Монтирование Вашего Google Drive к текущему окружению:

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

Mounted at /content/drive

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

```
EVALUATE_ONLY = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '1LghRTnXjI4uFS9vA6d18PqiEXmD5jMX3&export',
    'train_small': '1CCqpp8vt6UP2057cJKVa0Blv1EqwCHZ5&export',
    'train_tiny': '1aXJPY3RsPDGbUtx4gecZxoGfHZow6A00',
    'test': '1XQT3dJwx0y4kwWJjuLMHu2XWDxjh8b09',
    'test_small': '1inPVVNrRcpA9nqPwtGGRZDyS-7qLl-Ba',
    'test_tiny': '15LQAjWgNC4wZWjj0AMJV4D2HnGARnyc0'
}
```

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 torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import numpy as np
import os
```

✓ 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, transform=None):
        self.name = name
        self.is_loaded = False
        url = f"https://drive.google.com/u/0/uc?id={DATASETS_LINKS[name]}&export=download"
        output = f'{name}.npz'
        gdown.download(url, output, quiet=False)
        print(f'Loading dataset {self.name} from npz.')
        np_obj = np.load(f'{name}.npz')
        self.images = np_obj['data']
        self.labels = np_obj['labels']
        self.n_files = self.images.shape[0]
        self.is_loaded = True
        self.transform = transform
        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]

    def __len__(self):
        # Возвращаем количество изображений в датасете
        return self.n_files

    def __getitem__(self, index):
        img, label = self.image_with_label(index)
        # Преобразования изображения, если необходимо
        # img = transform_function(img)
        if self.transform:
            img = self.transform(img)

        return img, label

```

✓ Пример использования класса Dataset

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

```
d_train_tiny = Dataset('train_tiny')

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

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

Downloading...

From (original): <https://drive.google.com/u/0/uc?id=1aXJPY3RsPDGbUtx4gecZxoGfHZow6A00&export=download>

From (redirected): <https://drive.google.com/uc?id=1aXJPY3RsPDGbUtx4gecZxoGfHZow6A00&export=download&conf>

To: /content/train_tiny.npz

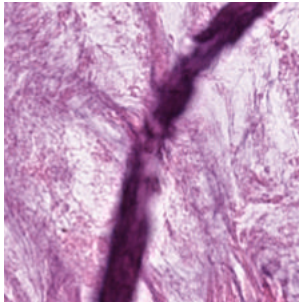
100%|██████████| 105M/105M [00:02<00:00, 38.6MB/s]

Loading dataset train_tiny from npz.

Done. Dataset train_tiny consists of 900 images.

Got numpy array of shape (224, 224, 3), and label with code 4.

Label code corresponds to MUC 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.

```

import cv2
import torch
import torch.nn as nn
from efficientnet_pytorch import EfficientNet
from torch.utils.data import DataLoader

class Model:

    def __init__(self):
        model_name = 'efficientnet-b6'
        self.model = EfficientNet.from_pretrained(model_name, num_classes=9)

    def save(self, name: str):
        # todo
        # pass
        # example demonstrating saving the model to PROJECT_DIR folder on gdrive with name 'name'
        arr = np.array([1, 2, 3, 4, 5], dtype=np.float32)
        np.savez(f'/content/drive/MyDrive/{name}.npz', data=arr)

    def load(self, name: str):
        # todo
        pass
        # example demonstrating loading the model with name 'name' from gdrive using link
        name_to_id_dict = {
            'best': '1S8bwrVgvtSzadEX2aLlyb3VTlD31UI4R'
        }
        output = f'{name}.npz'
        gdown.download(f'https://drive.google.com/uc?id={name_to_id_dict[name]}', output, quiet=False)
        np_obj = np.load(f'{name}.npz')
        print(np_obj['data'])

    def train(self, dataset: Dataset):
        print(f'training started')

        train_dataset = dataset
        test_dataset = Dataset('test_small', transforms.Compose([transforms.ToTensor()]))
        train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
        test_loader = DataLoader(test_dataset, batch_size=32, shuffle=True)
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        model_name = 'efficientnet-b6'
        model = EfficientNet.from_pretrained(model_name, num_classes=9)

        model.to(device)

        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

        num_epochs = 10
        for epoch in range(num_epochs):
            model.train()
            #print(type(train_loader))
            for inputs, labels in train_loader:
                inputs, labels = torch.tensor(inputs).to(device), torch.tensor(labels).to(device)

                optimizer.zero_grad()
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()

            model.eval()
            correct = 0
            total = 0
            with torch.no_grad():
                for inputs, labels in test_loader:
                    inputs, labels = torch.tensor(inputs).to(device), torch.tensor(labels).to(device)

```

```

        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    accuracy = correct / total
    print(f'Epoch {epoch+1}/{num_epochs}, Loss: {loss.item():.4f}, Accuracy: {accuracy:.4f}')

    # Save the trained model if needed
    model_path = '/content/drive/MyDrive/efficientnet_model.pth'
    torch.save(model.state_dict(), 'efficientnet_model.pth')
    torch.save(model.state_dict(), model_path)

# Rest of the training loop...

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):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model_path = '/content/drive/MyDrive/efficientnet_model.pth'
    state_dict = torch.load(model_path, map_location=torch.device(device))
    self.model.load_state_dict(state_dict)
    self.model.eval()

    # Преобразование изображения в тензор
    transform = transforms.Compose([transforms.ToTensor()])
    input_tensor = transform(img).unsqueeze(0) # добавляем размерность батча

    # Передача тензора через модель для предсказания
    with torch.no_grad():
        output = self.model(input_tensor)

    # Получение индекса предсказанного класса
    predicted_class = torch.argmax(output).item()

    return predicted_class

```

✓ 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_small', transforms.Compose([transforms.ToTensor()]))
d_test = Dataset('test', transforms.Compose([transforms.ToTensor()]))

Downloading...
From (uriginal): https://drive.google.com/u/0/uc?id=1CCqpp8vt6UP2057cJKVa0Blv1EqwCHZ5&export=download
From (redirected): https://drive.google.com/uc?id=1CCqpp8vt6UP2057cJKVa0Blv1EqwCHZ5&export=download
To: /content/train_small.npz
100%|██████████| 841M/841M [00:13<00:00, 61.7MB/s]
Loading dataset train_small from npz.
Done. Dataset train_small consists of 7200 images.
Downloading...
From (uriginal): https://drive.google.com/u/0/uc?id=1X0T3dJwx0y4kwWJjuLMHu2XWDxjh8b09&export=download
From (redirected): https://drive.google.com/uc?id=1X0T3dJwx0y4kwWJjuLMHu2XWDxjh8b09&export=download&confirm=1

```

```

To: /content/test.npz
100%|██████████| 525M/525M [00:06<00:00, 80.8MB/s]
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.

model = Model()

if EVALUATE_ONLY:
    model.train(d_train)
    #model.save('best')
else:
    #todo: your link goes here
    model.load('best')

    training started
    Downloading...
    From (uriginal): https://drive.google.com/u/0/uc?id=1inPVVNrRcpA9ngPwtGGRZDyS-7qLL-Ba&export=download
    From (redirected): https://drive.google.com/uc?id=1inPVVNrRcpA9ngPwtGGRZDyS-7qLL-Ba&export=download&con
    To: /content/test_small.npz
    100%|██████████| 211M/211M [00:00<00:00, 234MB/s]
    Loading dataset test_small from npz.
    Done. Dataset test_small consists of 1800 images.
    Loaded pretrained weights for efficientnet-b6
    <ipython-input-7-e9d670f694ab>:55: UserWarning: To copy construct from a tensor, it is recommended to u
    inputs, labels = torch.tensor(inputs).to(device), torch.tensor(labels).to(device)
    <ipython-input-7-e9d670f694ab>:68: UserWarning: To copy construct from a tensor, it is recommended to u
    inputs, labels = torch.tensor(inputs).to(device), torch.tensor(labels).to(device)
    Epoch 1/10, Loss: 0.1173, Accuracy: 0.8300
    Epoch 2/10, Loss: 0.0475, Accuracy: 0.9583
    Epoch 3/10, Loss: 0.1131, Accuracy: 0.9600
    Epoch 4/10, Loss: 0.0041, Accuracy: 0.9589
    Epoch 5/10, Loss: 0.0374, Accuracy: 0.9022
    Epoch 6/10, Loss: 0.0037, Accuracy: 0.9594
    Epoch 7/10, Loss: 0.0115, Accuracy: 0.9656
    Epoch 8/10, Loss: 0.0479, Accuracy: 0.9367
    Epoch 9/10, Loss: 0.4143, Accuracy: 0.9372
    Epoch 10/10, Loss: 0.0059, Accuracy: 0.9644

```

Пример тестирования модели на части набора данных:

```

# evaluating model on 10% of test dataset
model = Model()
pred_1 = model.test_on_dataset(d_test, limit=0.01)
Metrics.print_all(d_test.labels[:len(pred_1)], pred_1, '10% of test')

↳ Loaded pretrained weights for efficientnet-b6
100% 45/45 [00:37<00:00, 1.42it/s]

metrics for 10% of test:
    accuracy 0.9778:
    balanced accuracy 0.9778:
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:2184: UserWarning: y_pred co
warnings.warn("y_pred contains classes not in y_true")

```

+ Код

+ Текст

Пример тестирования модели на полном наборе данных:

```

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

100% 4500/4500 [58:28<00:00, 1.18it/s]

metrics for test:
    accuracy 0.9640:
    balanced accuracy 0.9640:

```


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

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

✓ Тестирование модели на других наборах данных

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

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

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

Loaded pretrained weights for efficientnet-b6
Access denied with the following error:

Cannot retrieve the public link of the file. You may need to change the permission to 'Anyone with the link', or have had many accesses.

You may still be able to access the file from the browser:

<https://drive.google.com/uc?id=1S8bwrVgvtSzadEX2aLlyb3VTlD31UI4R>

```
-----
FileNotFoundError                                Traceback (most recent call last)
<ipython-input-22-701859591eeb> in <cell line: 2>()
      1 final_model = Model()
----> 2 final_model.load('best')
      3 d_test_tiny = Dataset('test_tiny')
      4 pred = model.test_on_dataset(d_test_tiny)
      5 Metrics.print_all(d_test_tiny.labels, pred, 'test-tiny')
```

```
----- 1 frames -----
/usr/local/lib/python3.10/dist-packages/numpy/lib/npvio.py in load(file, mmap_mode, allow_pickle, fix_in
    403         own_fid = False
    404     else:
--> 405         fid = stack.enter_context(open(os_fspath(file), "rb"))
    406         own_fid = True
    407
```

FileNotFoundError: [Errno 2] No such file or directory: 'best.npz'

ИСКАТЬ НА STACK OVERFLOW

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
```

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".

Узнайте больше о TensorFlow 2.x и о том, как использовать Google Colab.

Большое количество tutorиалов и примеров с кодом на Tensorflow 2 можно найти на официальном сайте

<https://www.tensorflow.org/tutorials?hl=ru>.