

✓ 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) (3.10)
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) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2024.7.4)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
```

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 = False
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM')
DATASETS_LINKS = {
    'train': '1JFaqaH7QY5PiWUcXzVkJ17H195P-EQVG',
    'train_small': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1I-2Z0uXLd4QwhZQQLtp817Kn3J0Xgbui',
    'test': '1UW309J4bP70EQZKRKPQvBuEqL_kjhadQ',
    'test_small': '1wbRsog0n7uG1HIPGLhyN-PMET2kdQ2lI',
    'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}
```

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
import torch.nn as nn
import torchvision
import torch.nn.functional as F
from torch.optim import Adam
from torch.autograd import Variable
```

✓ 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):
        self.name = name
        self.is_loaded = False
        url = f"https://drive.google.com/uc?export=download&confirm=pbef&id={DATASETS_LINKS[name]}"
```

```

output = f'{name}.npz'
gdown.download(url, output, quiet=False)
print(f'Loading dataset {self.name} from npz.')
np_obj = np.load(f'{name}.npz')
self.images = np_obj['data']
self.labels = np_obj['labels']
self.n_files = self.images.shape[0]
self.is_loaded = True
print(f'Done. Dataset {name} consists of {self.n_files} images.')

def image(self, i):
    # read i-th image in dataset and return it as numpy array
    if self.is_loaded:
        return self.images[i, :, :, :]

def images_seq(self, n=None):
    # sequential access to images inside dataset (is needed for testing)
    for i in range(self.n_files if not n else n):
        yield self.image(i)

def random_image_with_label(self):
    # get random image with label from dataset
    i = np.random.randint(self.n_files)
    return self.image(i), self.labels[i]

def random_batch_with_labels(self, n):
    # create random batch of images with labels (is needed for training)
    indices = np.random.choice(self.n_files, n)
    imgs = []
    for i in indices:
        img = self.image(i)
        imgs.append(self.image(i))
    logits = np.array([self.labels[i] for i in indices])
    return np.stack(imgs), logits

def image_with_label(self, i: int):
    # return i-th image with label from dataset
    return self.image(i), self.labels[i]

def __len__(self):
    return self.n_files

```

✓ Пример использования класса Dataset

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

```

d_train_tiny = Dataset('train_tiny')

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

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

```



```

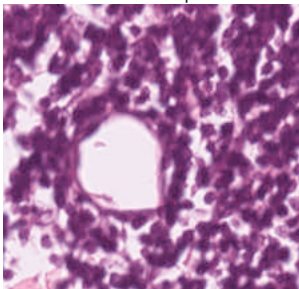
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1I-2Z0uXld4QwhZQ0ltp817Kn3J0Xgbui
To: /content/train_tiny.npz
100%[██████████] 105M/105M [00:00<00:00, 133MB/s]
Loading dataset train_tiny from npz.
Done. Dataset train_tiny consists of 900 images.

```

```

Got numpy array of shape (224, 224, 3), and label with code 3.
Label code corresponds to LYM 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 torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, transforms
from torch.utils.data import DataLoader
from tqdm import tqdm
from PIL import Image
import numpy as np
import logging
import matplotlib.pyplot as plt

class Model:
    def __init__(self, learning_rate=0.002, batch_size=32, num_epochs=15, weight_decay=1e-4, dropout_rate=0.5):
```

```

self.learning_rate = learning_rate
self.batch_size = batch_size
self.num_epochs = num_epochs
self.weight_decay = weight_decay
self.dropout_rate = dropout_rate

self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

self.model = models.mobilenet_v2(weights=models.MobileNet_V2_Weights.IMAGENET1K_V1)
in_features = self.model.classifier[1].in_features
self.model.classifier[1] = nn.Sequential(
    nn.Dropout(dropout_rate),
    nn.BatchNorm1d(in_features),
    nn.Linear(in_features, 9)
)
self.model = self.model.to(self.device)

self.criterion = nn.CrossEntropyLoss()
self.optimizer = optim.SGD(
    self.model.parameters(),
    lr=self.learning_rate,
    momentum=0.9,
    weight_decay=self.weight_decay
)
self.scheduler = torch.optim.lr_scheduler.StepLR(self.optimizer, step_size=5, gamma=0.1)

self.transforms = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

logging.basicConfig(level=logging.INFO)
self.logger = logging.getLogger("Model")

self.epochs = []
self.losses = []
self accuracies = []

def save(self, name: str):
    torch.save(self.model.state_dict(), f"{name}.pth")
    self.logger.info(f"Model saved to {name}.pth")

def load(self, name: str):
    self.model.load_state_dict(torch.load(f"{name}.pth", map_location=self.device))
    self.model.to(self.device)
    self.logger.info(f"Model loaded from {name}.pth")

def train(self, dataset):
    class PyTorchDataset:
        def __init__(self, dataset, transform=None):
            self.dataset = dataset
            self.transform = transform

        def __len__(self):
            return self.dataset.n_files

        def __getitem__(self, idx):
            image, label = self.dataset.image_with_label(idx)
            image = Image.fromarray(image)
            if self.transform:
                image = self.transform(image)
            return image, label

    train_data = PyTorchDataset(dataset, transform=self.transforms)
    train_loader = DataLoader(train_data, batch_size=self.batch_size, shuffle=True, num_workers=4)

    self.model.train()
    best_accuracy = 0.0

    for epoch in range(self.num_epochs):
        epoch_loss = 0.0
        correct = 0
        total = 0

        progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{self.num_epochs}", unit="batch")
        for images, labels in progress_bar:
            images, labels = images.to(self.device), labels.to(self.device)

            self.optimizer.zero_grad()
            outputs = self.model(images)
            loss = self.criterion(outputs, labels)
            loss.backward()

```

```

        self.optimizer.step()

        epoch_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

        progress_bar.set_postfix(loss=loss.item(), accuracy=100 * correct / total)

    epoch_accuracy = 100 * correct / total
    self.logger.info(f"Epoch {epoch+1}: Loss = {epoch_loss:.4f}, Accuracy = {epoch_accuracy:.2f}%")

    if epoch_accuracy > best_accuracy:
        best_accuracy = epoch_accuracy
        self.save("best_model")
        self.logger.info(f"New best model saved with accuracy: {best_accuracy:.2f}%")

    self.scheduler.step()
    self.epochs.append(epoch + 1)
    self.losses.append(epoch_loss)
    self accuracies.append(epoch_accuracy)

    self.plot_metrics()

def plot_metrics(self):
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(self.epochs, self.losses, label="Training Loss")
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training Loss per Epoch')
    plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(self.epochs, self accuracies, label="Training Accuracy", color='green')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy (%)')
    plt.title('Training Accuracy per Epoch')
    plt.grid(True)

    plt.tight_layout()
    plt.show()

def test_on_dataset(self, dataset, limit=None):
    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):
    self.model.eval()
    img_tensor = self.transforms(img).unsqueeze(0).to(self.device)

    with torch.no_grad():
        output = self.model(img_tensor)
        _, predicted_class = torch.max(output, 1)

    return predicted_class.item()

```

✓ Классификация изображений

Используя введенные выше классы можем перейти уже непосредственно к обучению модели классификации изображений. Пример общего пайплайна решения задачи приведен ниже. Вы можете его расширять и улучшать. В данном примере используются наборы данных 'train_small' и 'test_small'.

```

d_train = Dataset('train')
d_test = Dataset('test')

```



```

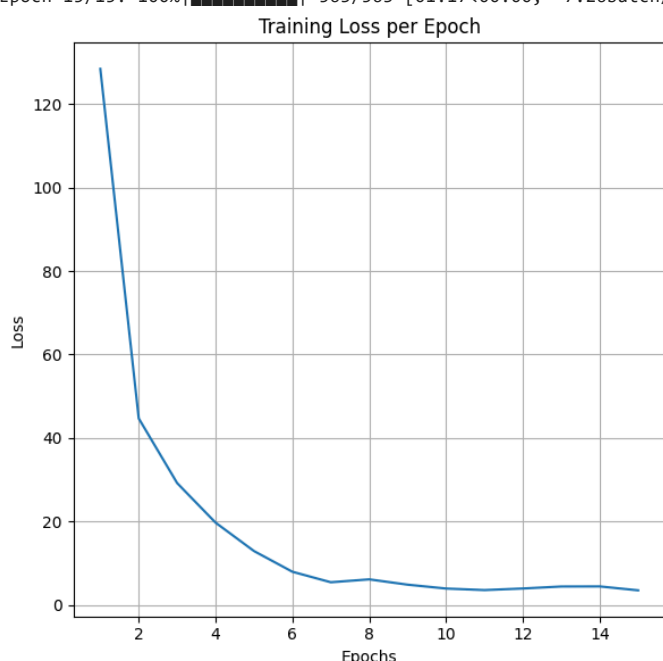
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1JFagaH7QY5PiWUcXzVkQ17H195P-EQVG
To: /content/train.npz
100%|██████████| 2.10G/2.10G [00:39<00:00, 53.8MB/s]
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
Downloading...
From: https://drive.google.com/uc?export=download&confirm=pbef&id=1UW309J4bP70EQZKRKPQvBuEgL\_kjhadQ
To: /content/test.npz

```

100%|██████████| 525M/525M [00:12<00:00, 42.8MB/s]
 Loading dataset test from npz.
 Done. Dataset test consists of 4500 images.

```
model = Model()
if not EVALUATE_ONLY:
    model.train(d_train)
    model.save('best')
else:
    model.load('best')
```

```
Epoch 1/15: 100%|██████████| 563/563 [01:22<00:00, 6.81batch/s, accuracy=93, loss=0.0524]
Epoch 2/15: 100%|██████████| 563/563 [01:16<00:00, 7.32batch/s, accuracy=97.5, loss=0.00838]
Epoch 3/15: 100%|██████████| 563/563 [01:16<00:00, 7.37batch/s, accuracy=98.4, loss=0.0119]
Epoch 4/15: 100%|██████████| 563/563 [01:16<00:00, 7.39batch/s, accuracy=98.9, loss=0.0487]
Epoch 5/15: 100%|██████████| 563/563 [01:16<00:00, 7.34batch/s, accuracy=99.3, loss=0.00478]
Epoch 6/15: 100%|██████████| 563/563 [01:16<00:00, 7.35batch/s, accuracy=99.7, loss=0.124]
Epoch 7/15: 100%|██████████| 563/563 [01:16<00:00, 7.36batch/s, accuracy=99.8, loss=0.0439]
Epoch 8/15: 100%|██████████| 563/563 [01:16<00:00, 7.39batch/s, accuracy=99.7, loss=0.00733]
Epoch 9/15: 100%|██████████| 563/563 [01:16<00:00, 7.39batch/s, accuracy=99.8, loss=0.0159]
Epoch 10/15: 100%|██████████| 563/563 [01:16<00:00, 7.39batch/s, accuracy=99.8, loss=0.0504]
Epoch 11/15: 100%|██████████| 563/563 [01:16<00:00, 7.33batch/s, accuracy=99.9, loss=0.0337]
Epoch 12/15: 100%|██████████| 563/563 [01:21<00:00, 6.90batch/s, accuracy=99.9, loss=0.00889]
Epoch 13/15: 100%|██████████| 563/563 [01:20<00:00, 7.03batch/s, accuracy=99.8, loss=0.00922]
Epoch 14/15: 100%|██████████| 563/563 [01:16<00:00, 7.35batch/s, accuracy=99.8, loss=0.289]
Epoch 15/15: 100%|██████████| 563/563 [01:17<00:00, 7.26batch/s, accuracy=99.9, loss=0.00292]
```



```
for i in range(5):
    img, label = d_test.random_image_with_label()
    pred = model.test_on_image(img)
    print(f'Real label: {TISSUE_CLASSES[label]}, Predicted: {TISSUE_CLASSES[pred]}')
```

```
Real label: ADI, Predicted: ADI
Real label: BACK, Predicted: BACK
Real label: ADI, Predicted: ADI
Real label: ADI, Predicted: ADI
Real label: MUC, Predicted: MUC
```

Пример тестирования модели на части набора данных:

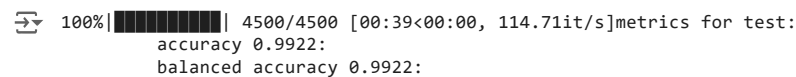
```
# evaluating model on 10% of test dataset
pred_1 = model.test_on_dataset(d_test, limit=0.1)
Metrics.print_all(d_train.labels[:len(pred_1)], pred_1, '10% of test')
```

```
100%|██████████| 450/450 [00:04<00:00, 100.23it/s]metrics for 10% of test:
accuracy 1.0000:
balanced accuracy 1.0000:
```

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:409: UserWarning: A single label was found in 'y_train' warnings.warn()

Пример тестирования модели на полном наборе данных:

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



Результат работы пайплайна обучения и тестирования выше тоже будет оцениваться. 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.

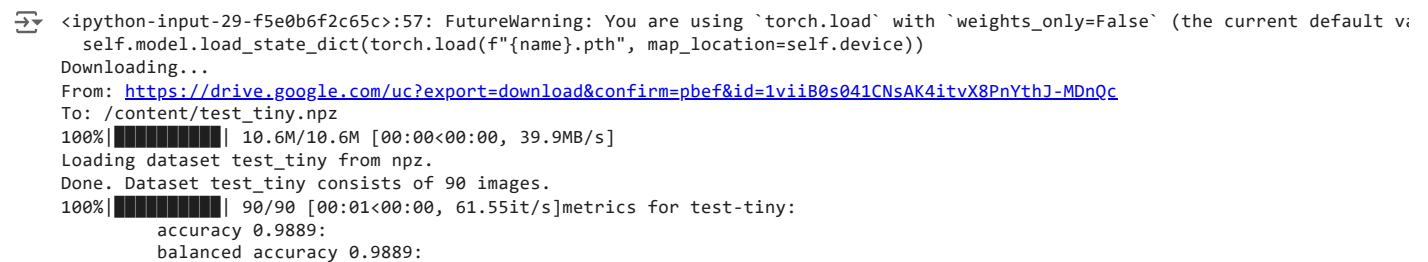
Настоятельно рекомендуется после получения пайплайна с полными результатами обучения экспортировать ноутбук в pdf (файл -> печать) и прислать этот pdf вместе с самим ноутбуком.

✓ 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_model')
d_test_tiny = Dataset('test_tiny')
pred = model.test_on_dataset(d_test_tiny)
Metrics.print_all(d_test_tiny.labels, pred, '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 tutorиалов и примеров с кодом на Tensorflow 2 можно найти на официальном сайте

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

Также, Вам может понадобиться написать собственный генератор данных для Tensorflow 2. Скорее всего он будет достаточно простым, и его легко можно будет реализовать, используя официальную документацию TensorFlow 2. Но, на всякий случай (если не удалось сразу разобраться или хочется вникнуть в тему более глубоко), можете посмотреть следующий отличный tutorial:

<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/cbnet/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.

```
PROJECT_DIR = "/dev/prak_nn_1/"
arr1 = np.random.rand(100, 100, 3) * 255
arr2 = np.random.rand(100, 100, 3) * 255

img1 = Image.fromarray(arr1.astype('uint8'))
img2 = Image.fromarray(arr2.astype('uint8'))

p = "/content/drive/MyDrive/" + PROJECT_DIR

if not (Path(p) / 'tmp').exists():
    (Path(p) / 'tmp').mkdir()

img1.save(str(Path(p) / 'tmp' / 'img1.png'))
img2.save(str(Path(p) / 'tmp' / 'img2.png'))

%cd $p
!zip -r "tmp.zip" "tmp"
```

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

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