

## ✓ 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) (2024.8.3)
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': '1qd45xXfDwdZjktLFwQb-et-mAaFeCzOR',
    'train_tiny': '1I-2Z0uXLd4QwhZQQ1tp817Kn3J0Xgbui',
    'test': '1RfPou3pFKpuHDJZ-D9XDFzgvvpUBF1Dr',
    'test_small': '1wbRsog0n7uG1HIPGLhyN-PMET2kdQ21I',
    'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}
```

Импорт необходимых зависимостей:

```
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
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 torch.optim as optim
from torch.utils.data import Dataset as TorchDataset, DataLoader
import torchvision.transforms as transforms
from torch.utils.tensorboard import SummaryWriter
from torchvision import models
```

## ✓ 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, gdrive_path):
        self.name = name
        self.is_loaded = False
```

```

self.file_path = Path(gdrive_path) / f"{name}.npz"

if not self.file_path.exists():
    raise FileNotFoundError(f"Dataset file not found at {self.file_path}")

print(f"Loading dataset {self.name} from {self.file_path}.")
np_obj = np.load(self.file_path)
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 {self.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.

```

gdrive_path = "/content/drive/MyDrive/"
d_train_tiny = Dataset('train_tiny', gdrive_path)

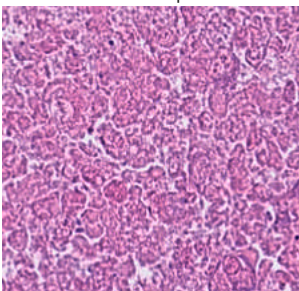
img, lbl = d_train_tiny.random_image_with_label()
print()
print(f'Got numpy array of shape {img.shape}, and label with code {lbl}.')
print(f'Label code corresponds to {TISSUE_CLASSES[lbl]} class.')

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

```

↗ Loading dataset train\_tiny from /content/drive/MyDrive/train\_tiny.npz.  
Done. Dataset train\_tiny consists of 900 images.

Got numpy array of shape (224, 224, 3), and label with code 2.  
Label code corresponds to DEB class.



## ✓ Класс Metrics

Реализует метрики точности, используемые для оценивания модели:

1. точность,
2. сбалансированную точность.

```
class Metrics:

    @staticmethod
    def accuracy(gt: List[int], pred: List[int]):
        assert len(gt) == len(pred), 'gt and prediction should be of equal 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.

```
class Model:
    def __init__(self):

        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = models.resnet50(pretrained=True)

        for param in self.model.parameters():
            param.requires_grad = False

        for param in self.model.layer4.parameters():
            param.requires_grad = True

        num_fts = self.model.fc.in_features
        self.model.fc = nn.Sequential(
```

```

self.model.to(self.device)
nn.Dropout(0.5),
nn.Linear(num_fts, len(TISSUE_CLASSES))
)

self.model.to(self.device)
device_name = torch.cuda.get_device_name(0) if torch.cuda.is_available() else "CPU"
print(f"Device name: {device_name}")
#print(f"Model architecture:\n{self.model}")

def save(self, name: str):
    torch.save(self.model.state_dict(), f'/content/drive/MyDrive/{name}.pth')
    print(f"Model saved as {name}.pth")

def load(self, name: str):
    name_to_id_dict = {
        'best': '1I18-4QFcoj07svUf5EcwHlU8L4xBeCwX',
        'best2': '1YxHtvVjr1nJYdRHeUsfJswgYgDfWYsUA'
    }

    if name in name_to_id_dict:
        file_id = name_to_id_dict[name]
        output = f'{name}.pth'

        gdown.download(f'https://drive.google.com/uc?id={file_id}', output, quiet=False)

        state_dict = torch.load(output, map_location=self.device)
        self.model.load_state_dict(state_dict)
        print(f"Model {name}.pth uploaded")
    else:
        print(f"Model '{name}' not found")

def train(self, dataset: Dataset):
    from torch.utils.tensorboard import SummaryWriter

    writer = SummaryWriter()
    global_step = 0

    epoch_losses = []
    epoch accuracies = []

    dummy_input = torch.randn(1, 3, 224, 224).to(self.device)
    writer.add_graph(self.model, dummy_input)

    transform = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(20),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.AdamW(filter(lambda p: p.requires_grad, self.model.parameters()), lr=1e-4, weight_decay=1e-4)
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

    num_epochs = 10
    batch_size = 64
    self.model.train()

    for epoch in range(num_epochs):
        running_loss = 0.0
        correct = 0
        total = 0

        n_batches = dataset.n_files // batch_size

        with tqdm(total=n_batches, desc=f'Epoch {epoch + 1}/{num_epochs}') as pbar:
            for i in range(n_batches):
                imgs, labels = dataset.random_batch_with_labels(batch_size)

                inputs = torch.stack([transform(Image.fromarray(img)) for img in imgs])
                inputs = inputs.to(self.device)
                labels = torch.from_numpy(labels).long().to(self.device)

                optimizer.zero_grad()
                outputs = self.model(inputs)

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

```

```

        running_loss += loss.item()

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

        batch_loss = loss.item()
        batch_accuracy = (predicted == labels).sum().item() / labels.size(0)
        writer.add_scalar('Loss/train', batch_loss, global_step)
        writer.add_scalar('Accuracy/train', batch_accuracy, global_step)
        global_step += 1

        pbar.set_postfix({'Loss': f'{running_loss / (i + 1):.4f}', 'Accuracy': f'{(correct / total):.4f}'})
        pbar.update(1)

    scheduler.step()

    epoch_loss = running_loss / n_batches
    epoch_accuracy = correct / total
    writer.add_scalar('Loss/epoch', epoch_loss, epoch)
    writer.add_scalar('Accuracy/epoch', epoch_accuracy, epoch)

    epoch_losses.append(epoch_loss)
    epoch_accuracies.append(epoch_accuracy)

    print(f'\nEpoch {epoch + 1} finished. Accuracy: {epoch_accuracy:.4f}')

writer.close()

epochs = range(1, num_epochs + 1)
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(epochs, epoch_losses, 'b-', label='Training Loss')
plt.title('Training Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.xticks(epochs)

plt.subplot(1, 2, 2)
plt.plot(epochs, epoch_accuracies, 'r-', label='Training Accuracy')
plt.title('Training Accuracy per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.xticks(epochs)

plt.tight_layout()
plt.show()

def test_on_image(self, img: np.ndarray):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])

    img_tensor = transform(Image.fromarray(img)).unsqueeze(0).to(self.device)

    self.model.eval()
    with torch.no_grad():
        outputs = self.model(img_tensor)
        _, predicted = torch.max(outputs, 1)
    return predicted.item()

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

```

## ✓ 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'.

```
gdrive_path = "/content/drive/MyDrive/"
d_train = Dataset('train', gdrive_path)
d_test = Dataset('test', gdrive_path)
```

↗ Loading dataset train from /content/drive/MyDrive/train.npz.  
Done. Dataset train consists of 18000 images.  
Loading dataset test from /content/drive/MyDrive/test.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')
```

↗ Device name: NVIDIA L4

Epoch 1/10: 100% 281/281 [02:36<00:00, 1.83it/s, Loss=0.2956, Accuracy=0.9067]

Epoch 1 finished. Accuracy: 0.9067

Epoch 2/10: 100% 281/281 [02:36<00:00, 1.79it/s, Loss=0.1430, Accuracy=0.9533]

Epoch 2 finished. Accuracy: 0.9533

Epoch 3/10: 100% 281/281 [02:36<00:00, 1.79it/s, Loss=0.1081, Accuracy=0.9654]

Epoch 3 finished. Accuracy: 0.9654

Epoch 4/10: 100% 281/281 [02:36<00:00, 1.82it/s, Loss=0.0937, Accuracy=0.9693]

Epoch 4 finished. Accuracy: 0.9693

Epoch 5/10: 100% 281/281 [02:36<00:00, 1.75it/s, Loss=0.0793, Accuracy=0.9739]

Epoch 5 finished. Accuracy: 0.9739

Epoch 6/10: 100% 281/281 [02:36<00:00, 1.81it/s, Loss=0.0589, Accuracy=0.9812]

Epoch 6 finished. Accuracy: 0.9812

Epoch 7/10: 100% 281/281 [02:37<00:00, 1.81it/s, Loss=0.0497, Accuracy=0.9828]

Epoch 7 finished. Accuracy: 0.9828

Epoch 8/10: 100% 281/281 [02:37<00:00, 1.81it/s, Loss=0.0466, Accuracy=0.9844]

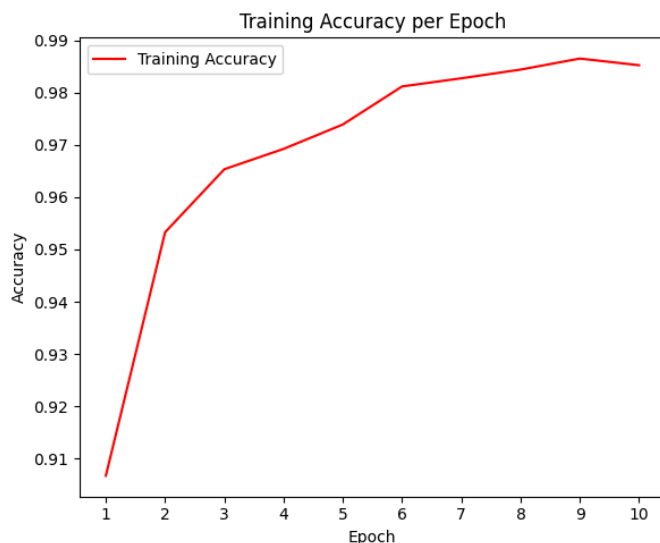
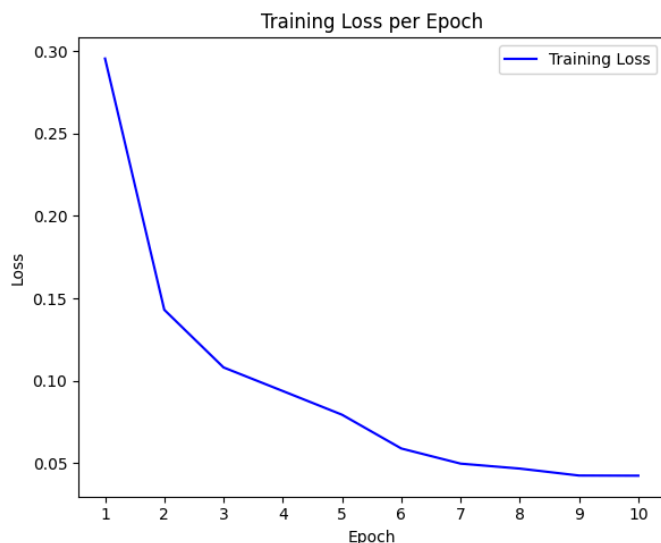
Epoch 8 finished. Accuracy: 0.9844

Epoch 9/10: 100% 281/281 [02:37<00:00, 1.70it/s, Loss=0.0424, Accuracy=0.9865]

Epoch 9 finished. Accuracy: 0.9865


Epoch 10/10: 100% 281/281 [02:36<00:00, 1.80it/s, Loss=0.0423, Accuracy=0.9853]

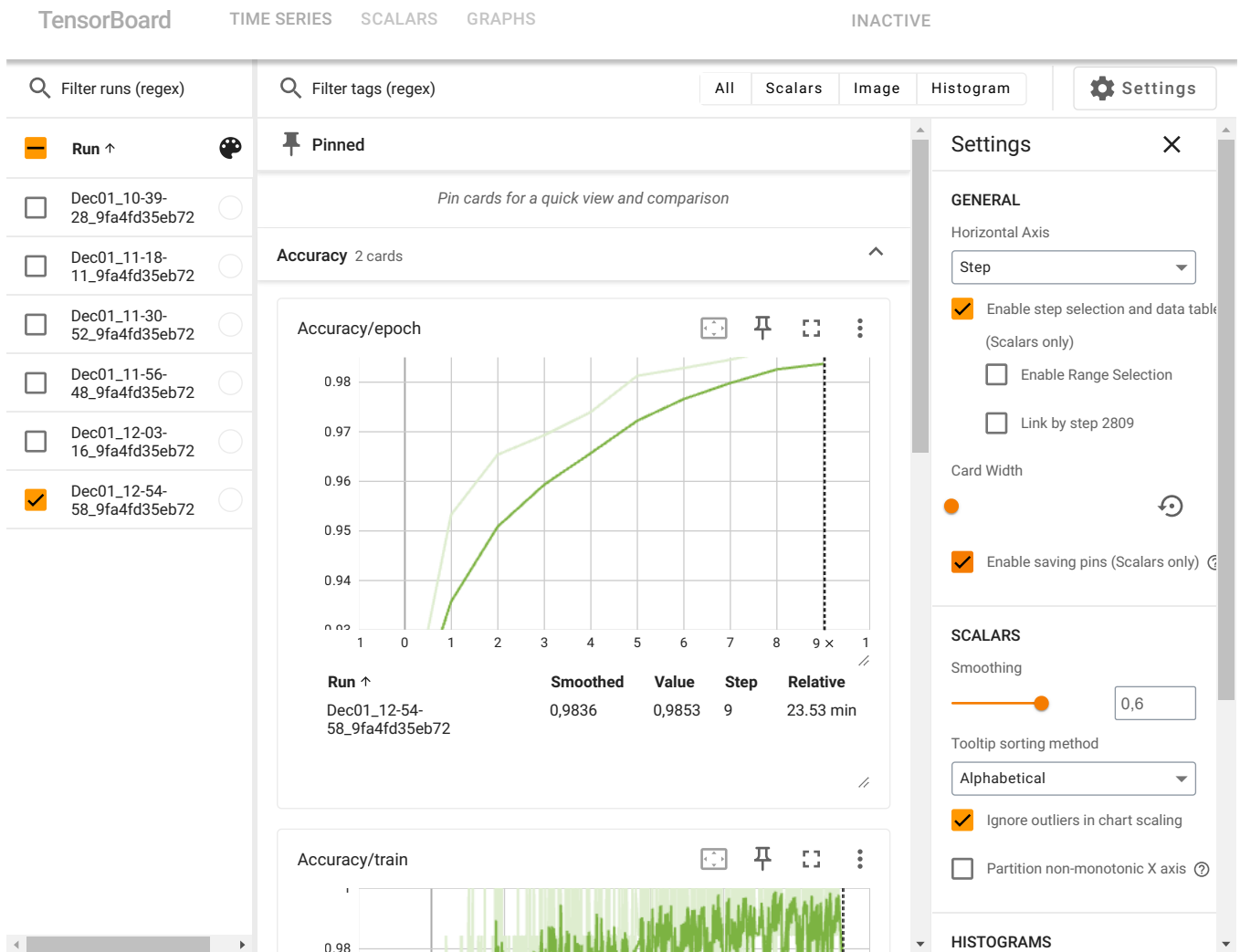
Epoch 10 finished. Accuracy: 0.9853



Model saved as best.pth


```
%load_ext tensorboard
%tensorboard --logdir=runs
```

 The tensorboard extension is already loaded. To reload it, use:  
`%reload_ext tensorboard`  
 Reusing TensorBoard on port 6006 (pid 48238), started 1:46:49 ago. (Use '!kill 48238' to kill it.)



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_test.labels[:len(pred_1)], pred_1, '10% of test')
```

 100% 450/450 [00:03<00:00, 119.69it/s]

metrics for 10% of test:

accuracy 0.9978:


balanced accuracy 0.9978:

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/\_classification.py:2480: UserWarning: y\_pred contains classes not in y\_true

warnings.warn("y\_pred contains classes not in y\_true")

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 [00:38<00:00, 117.70it/s]

metrics for test:

accuracy 0.9873:

balanced accuracy 0.9873:

✓ Тестирование модели на других наборах данных

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

```
⚡ /usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)
Device name: NVIDIA L4
Downloading...
From (original): https://drive.google.com/uc?id=1I18-4QFcoj07svUf5EcwHlU8L4xBeCwX
From (redirected): https://drive.google.com/uc?id=1I18-4QFcoj07svUf5EcwHlU8L4xBeCwX&confirm=t&uuid=fc4181e3-56b7-4143-bff4-827333b0f
To: /content/best.pth
100%|██████████| 94.4M/94.4M [00:01<00:00, 59.3MB/s]
<ipython-input-143-ebf4383352a6>:40: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value
  state_dict = torch.load(output, map_location=self.device)
Model best.pth uploaded
Loading dataset test_tiny from /content/drive/MyDrive/test_tiny.npz.
Done. Dataset test_tiny consists of 90 images.

100%                               90/90 [00:00<00:00, 118.69it/s]

metrics for test-tiny:
  accuracy 0.9667:
  balanced accuracy 0.9667:
```

Отмонтировать Google Drive.

```
drive.flush_and_unmount()
```