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

```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.12/dist-packages (5.2.0)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.12/dist-packages (from gdown) (4.13.5)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from gdown) (3.20.0)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.12/dist-packages (from gdown) (2.32.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from gdown) (4.67.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4->gdown) (2.8)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.12/dist-packages (from beautifulsoup4->gdo
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gd
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown) (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdown) (
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.12/dist-packages (from requests[socks]->gdow
```

```
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, confusion_matrix, ConfusionMatrixDisplay
import gdown

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset as TorchDataset, DataLoader
from torchvision import models, transforms

EVALUATE_ONLY = False
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-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui',
    'test': '1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr',
    'test_small': '1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ2lI',
    'test_tiny': '1viiB0s041CNsAK4itvX8PnYthJ-MDnQc'
}

```

```
class Dataset:
    def __init__(self, name):
        self.name = name
        self.is_loaded = False
        file_name = f"{name}.npz"
        if not os.path.exists(file_name):
            raise FileNotFoundError("нет файла " + file_name)
        print("загружаю датасет", name, "из", file_name)
        np_obj = np.load(file_name)
        self.images = np_obj["data"]
        self.labels = np_obj["labels"]
        self.n_files = self.images.shape[0]
        self.is_loaded = True
        print("готово, картинок:", self.n_files)

    def image(self, i):
        if not self.is_loaded:
            raise RuntimeError("датасет не загружен")
        return self.images[i, :, :, :]

    def images_seq(self, n=None):
        total = self.n_files if n is None else n
        for i in range(total):
            yield self.image(i)

    def random_image_with_label(self):
        i = np.random.randint(self.n_files)
        return self.image(i), self.labels[i]

    def random_batch_with_labels(self, n):
        idx = np.random.choice(self.n_files, n)
```

```
        imgs = [self.image(i) for i in idx]
        labels = np.array([self.labels[i] for i in idx])
        return np.stack(imgs), labels

    def image_with_label(self, i):
        return self.image(i), self.labels[i]

```

```
class Metrics:
    @staticmethod
    def accuracy(gt, pred):
        if len(gt) != len(pred):
            raise ValueError("gt and pred different length")
        s = 0
        for a, b in zip(gt, pred):
            if a == b:
                s += 1
        return s / len(gt)

    @staticmethod
    def accuracy_balanced(gt, pred):
        return balanced_accuracy_score(gt, pred)

    @staticmethod
    def print_all(gt, pred, info):
        print("metrics", info)
        acc = Metrics.accuracy(gt, pred)
        bacc = Metrics.accuracy_balanced(gt, pred)
        print("acc", acc)
        print("balanced_acc", bacc)

```

```
class HistologyTorchDataset(TorchDataset):
    def __init__(self, base_dataset, indices, transform=None):
        self.base = base_dataset
        self.indices = np.array(indices)
        self.transform = transform

    def __len__(self):
        return len(self.indices)

    def __getitem__(self, idx):
        real_idx = int(self.indices[idx])
        img = self.base.images[real_idx]
        label = int(self.base.labels[real_idx])
        img = Image.fromarray(img)
        if self.transform is not None:
            img = self.transform(img)
        else:
            img = transforms.ToTensor()(img)
        return img, label


class Model:
    def __init__(self, num_classes=len(TISSUE_CLASSES), project_name="histology_cnn", save_dir=None):
        self.num_classes = num_classes
        self.project_name = project_name
        self.save_dir = save_dir or os.path.join(BASE_DIR, "histology_models")
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        print("device:", self.device)

        #LBL11 аугментации изображений
        self.train_transform = transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.RandomRotation(15),
            transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5, 0.5, 0.5],
                                 std=[0.5, 0.5, 0.5]),
        ])

        self.eval_transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.5, 0.5, 0.5],
                                 std=[0.5, 0.5, 0.5]),
        ])

        self.model = models.resnet18(pretrained=True)
        in_features = self.model.fc.in_features
        self.model.fc = nn.Linear(in_features, num_classes)
```

```
        self.model = self.model.to(self.device)

        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.model.parameters(), lr=1e-4, weight_decay=1e-4)
        self.history = {"train_loss": [], "val_loss": [], "val_acc": []}

    #LBL1 валидация на части обучающей выборки
    def _make_dataloaders(self, dataset, batch_size=64, val_split=0.2, seed=42):
        np.random.seed(seed)
        idx = np.arange(dataset.n_files)
        np.random.shuffle(idx)
        split = int(len(idx) * (1.0 - val_split))
        train_idx = idx[:split]
        val_idx = idx[split:]
        train_ds = HistologyTorchDataset(dataset, train_idx, transform=self.train_transform)
        val_ds = HistologyTorchDataset(dataset, val_idx, transform=self.eval_transform)
        train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True, num_workers=2)
        val_loader = DataLoader(val_ds, batch_size=batch_size, shuffle=False, num_workers=2)
        print("train samples:", len(train_ds), "val samples:", len(val_ds))
        return train_loader, val_loader

    #LBL3 сохранение модели по эпохам
    def save(self, name):
        os.makedirs(self.save_dir, exist_ok=True)
        path = os.path.join(self.save_dir, f"{name}.pt")
        torch.save({"model_state_dict": self.model.state_dict(), "num_classes": self.num_classes}, path)
        print("saved model to", path)

    #LBL4 загрузка модели по имени чекпоинта
    def load(self, name):
        os.makedirs(self.save_dir, exist_ok=True)
        path = os.path.join(self.save_dir, f"{name}.pt")
        if not os.path.exists(path):
            file_id = "PUT_YOUR_FILE_ID_HERE"
            if file_id is None or file_id == "" or "PUT_YOUR_FILE_ID_HERE" in file_id:
                raise FileNotFoundError("нет файла с весами " + path + " и не задан file_id")
            url = "https://drive.google.com/uc?id=" + file_id
            print("скачиваю веса из", url)
            gdown.download(url, path, quiet=False)
        checkpoint = torch.load(path, map_location=self.device)
        self.model.load_state_dict(checkpoint["model_state_dict"])
        self.model.to(self.device)
        self.model.eval()
        print("loaded model from", path)

    def train(self, dataset, epochs=10, batch_size=64, lr=1e-4, val_split=0.2):
        for g in self.optimizer.param_groups:
            g["lr"] = lr

        train_loader, val_loader = self._make_dataloaders(dataset, batch_size=batch_size, val_split=val_split)
        best_val_acc = 0.0

        for epoch in range(1, epochs + 1):
            self.model.train()
            running_loss = 0.0
            correct = 0
            total = 0

            for images, labels in train_loader:
                images = images.to(self.device)
                labels = labels.to(self.device)

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

                running_loss += loss.item() * labels.size(0)
                _, preds = torch.max(outputs, 1)
                correct += (preds == labels).sum().item()
                total += labels.size(0)

            train_loss = running_loss / total
            train_acc = correct / total

            self.model.eval()
            val_loss = 0.0
            val_correct = 0
            val_total = 0
            all_val_preds = []
            all_val_labels = []
```

```
132
133             with torch.no_grad():
134                 for images, labels in val_loader:
135                     images = images.to(self.device)
136                     labels = labels.to(self.device)
137
138                     outputs = self.model(images)
139                     loss = self.criterion(outputs, labels)
140
141                     val_loss += loss.item() * labels.size(0)
142                     _, preds = torch.max(outputs, 1)
143                     val_correct += (preds == labels).sum().item()
144                     val_total += labels.size(0)
145
146                     all_val_preds.append(preds.cpu().numpy())
147                     all_val_labels.append(labels.cpu().numpy())
148
149             val_loss = val_loss / val_total
150             val_acc = val_correct / val_total
151             all_val_preds = np.concatenate(all_val_preds)
152             all_val_labels = np.concatenate(all_val_labels)
153
154             self.history["train_loss"].append(train_loss)
155             self.history["val_loss"].append(val_loss)
156             self.history["val_acc"].append(val_acc)
157
158             #LBL5 вывод метрик обучения по эпохам
159             print("epoch", epoch, "/", epochs)
160             print("train_loss", float(train_loss), "train_acc", float(train_acc))
161             print("val_loss", float(val_loss), "val_acc", float(val_acc))
162             Metrics.print_all(all_val_labels, all_val_preds, "val epoch " + str(epoch))
163
164             ckpt_name = f"{self.project_name}_epoch_{epoch}"
165             self.save(ckpt_name)
166             if val_acc > best_val_acc:
167                 best_val_acc = val_acc
168                 self.save("best")
169                 print("new best model, val_acc", best_val_acc)
170
171     def test_on_dataset(self, dataset, limit=None):
172         predictions = []
173         n = dataset.n_files if limit is None else int(dataset.n_files * limit)
174         for img in tqdm(dataset.images_seq(n), total=n):
175             predictions.append(self.test_on_image(img))
176         return predictions
177
178     def test_on_image(self, img):
179         self.model.eval()
180         with torch.no_grad():
181             pil_img = Image.fromarray(img)
182             tensor = self.eval_transform(pil_img).unsqueeze(0).to(self.device)
183             outputs = self.model(tensor)
184             _, pred = torch.max(outputs, 1)
185             return int(pred.item())
186
187     #LBL8 матрица ошибок и визуализация результатов теста
188     #LBL10 сохранение предсказаний и матрицы ошибок в npz
189     def evaluate_and_visualize(self, dataset, dataset_name="test", save_prefix=None):
190         y_pred = self.test_on_dataset(dataset)
191         y_true = dataset.labels[:len(y_pred)]
192         Metrics.print_all(y_true, y_pred, dataset_name)
193         cm = confusion_matrix(y_true, y_pred)
194         disp = ConfusionMatrixDisplay(cm, display_labels=TISSUE_CLASSES)
195         import matplotlib.pyplot as plt
196         plt.figure(figsize=(6, 6))
197         disp.plot(include_values=True, cmap="Blues", xticks_rotation=45, colorbar=False)
198         plt.title("confusion " + dataset_name)
199         plt.tight_layout()
200         plt.show()
201         if save_prefix is not None:
202             os.makedirs(self.save_dir, exist_ok=True)
203             out_path = os.path.join(self.save_dir, save_prefix + "_results.npz")
204             np.savez(out_path, y_true=y_true, y_pred=y_pred, cm=cm)
205             print("saved test results to", out_path)
206
207     #LBL6 построение графиков потерь и точности
208     def plot_history(self):
209         if not self.history["train_loss"]:
210             print("нет истории, сперва train()")
211             return
212
213         import matplotlib.pyplot as plt
```

```
        epochs = range(1, len(self.history["train_loss"]) + 1)
        plt.figure(figsize=(12, 4))
        plt.subplot(1, 2, 1)
        plt.plot(epochs, self.history["train_loss"], label="train_loss")
        plt.plot(epochs, self.history["val_loss"], label="val_loss")
        plt.xlabel("epoch")
        plt.ylabel("loss")
        plt.legend()
        plt.title("Loss")
        plt.subplot(1, 2, 2)
        plt.plot(epochs, self.history["val_acc"], label="val_acc")
        plt.xlabel("epoch")
        plt.ylabel("accuracy")
        plt.legend()
        plt.title("Validation accuracy")
        plt.show()

```

```
!cp '/content/drive/MyDrive/train_small.npz' 'train_small.npz'
!cp '/content/drive/MyDrive/test_small.npz' 'test_small.npz'
!cp '/content/drive/MyDrive/train_tiny.npz' 'train_tiny.npz'
!cp '/content/drive/MyDrive/test_tiny.npz' 'test_tiny.npz'

```

```
d_train = Dataset('train_small')
d_test = Dataset('test_small')

model = Model()

if not EVALUATE_ONLY:
    model.train(d_train, epochs=10, batch_size=64, lr=1e-4, val_split=0.2)
    model.save('best')
else:
    model.load('best')

```

```
saved model to /content/drive/MyDrive/histology_models/best.pt
new best model, val_acc 0.9701388888888889
epoch 4 / 10
train_loss 0.07005240021066533 train_acc 0.9777777777777777
val_loss 0.08641760117477841 val_acc 0.9708333333333333
metrics val epoch 4
acc 0.9708333333333333
balanced_acc 0.9710445398857919
saved model to /content/drive/MyDrive/histology_models/histology_cnn_epoch_4.pt
saved model to /content/drive/MyDrive/histology_models/best.pt
new best model, val_acc 0.9708333333333333
epoch 5 / 10
train_loss 0.055242568937440714 train_acc 0.9840277777777777
val_loss 0.09785338511897458 val_acc 0.9694444444444444
```

```
val_loss 0.06625294434941477 val_acc 0.9833333333333333
metrics val epoch 10
acc 0.9833333333333333
balanced_acc 0.9835439289954278
saved model to /content/drive/MyDrive/histology_models/histology_cnn_epoch_10.pt
saved model to /content/drive/MyDrive/histology_models/best.pt
new best model, val_acc 0.9833333333333333
saved model to /content/drive/MyDrive/histology_models/best.pt
```

```
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')

if TEST_ON_LARGE_DATASET:
    pred_2 = model.test_on_dataset(d_test)
    Metrics.print_all(d_test.labels, pred_2, 'test')

```

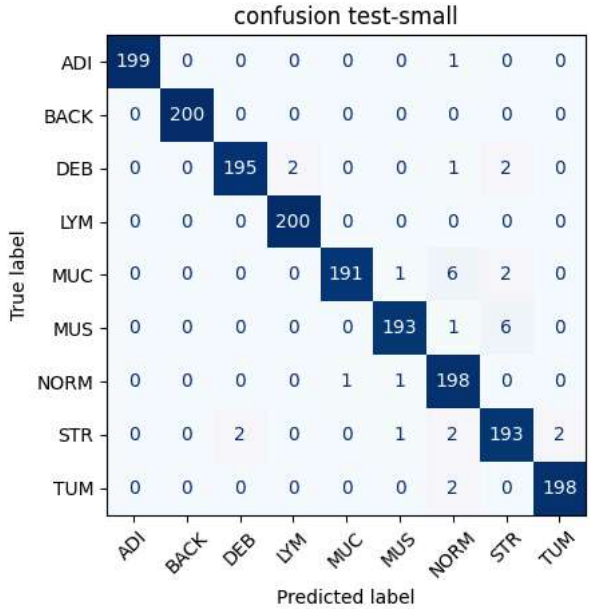
```
100%  180/180 [00:01<00:00, 173.08it/s]
metrics 10% of test
acc 0.9944444444444445
balanced_acc 0.9944444444444445
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:2524: UserWarning: y_pred contains classes not ir
  warnings.warn("y_pred contains classes not in y_true")
100%  1800/1800 [00:07<00:00, 250.33it/s]
metrics test
acc 0.9816666666666667
balanced_acc 0.9816666666666668
```

```
final_model = Model()
final_model.load('best')

d_test_tiny = Dataset('test_small')
final_model.evaluate_and_visualize(d_test_tiny, dataset_name='test-small', save_prefix='test_small')

```

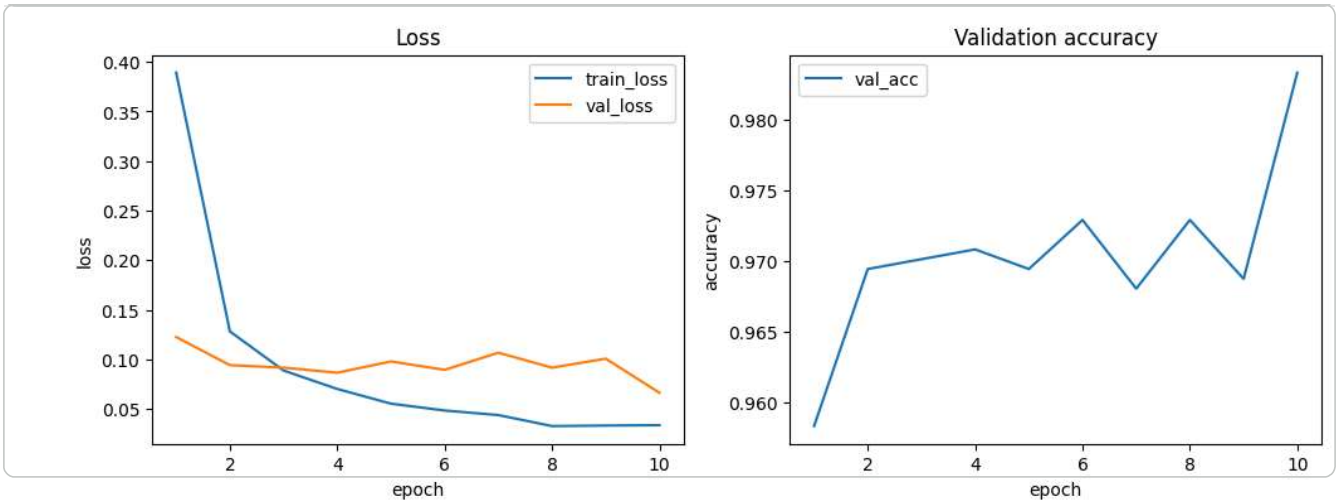
```
device: cuda
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecа
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or
  warnings.warn(msg)
loaded model from /content/drive/MyDrive/histology_models/best.pt
загружаю датасет test_small из test_small.npz
готово, картинок: 1800
100%  1800/1800 [00:10<00:00, 247.01it/s]
metrics test-small
acc 0.9816666666666667
balanced_acc 0.9816666666666668
<Figure size 600x600 with 0 Axes>
```




```
saved test results to /content/drive/MyDrive/histology_models/test_small_results.npz
```

```
model.plot_history()

```


Loss
train_loss
val_loss
loss
epoch
Validation accuracy
val_acc
accuracy
epoch