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.
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
```

```
cd /content/drive/MyDrive/competition
```

```
/content/drive/MyDrive/competition
```

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': '1kT3GtZWcLqeuPECEGNXpMNi3jp9LZY_B',
    'train_small': '1IMMjP6oUFNmrDnmPRpuFt5vzHDGfELMY',
    'train_tiny': '1Ay1nq2ngf4XyVgE-d8S_z19dfD5PNszy',
    'test': '1EdCDUOB3bJLSkt117v-eSbIXiERur1Xr',
    'test_small': '1vF2YdizwJmsuzEY0hasrFnpE-uPoGAO_',
    'test_tiny': '1-2mhSO6nMSkm1Q61pbQb_QAwsGE_FuIu'
}
'''


"""
import gdown

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'
}
"""
'''for name, file_id in DATASETS_LINKS.items():
    url = f"https://drive.google.com/uc?export=download&id={file_id}"
    output = f"{name}.zip"  # Название файла для сохранения
    print(f"Скачивание {name} из {url}...")
    gdown.download(url, output, quiet=False)

print("Скачивание завершено!")
'''
```

```
"""
```

```
'\nDATASETS_LINKS = {\n    \'train\': \'1XtQzVQ5XbrfxpLHJuL0XBGJ5U7CS-cLi\',\n    \'train_small\': \'1qd45xXfDwdZjktLFwQb-et-mAaFeC
zOR\',\n    \'train_tiny\': \'1I-2ZOuXLd4QwhZQQltp817Kn3J0Xgbui\',\n    \'test\': \'1RfPou3pFKpuHDJZ-D9XDFzgvwpUBFlDr\',\n    \'tes
t_small\': \'1wbRsog0n7uGlHIPGLhyN-PMeT2kdQ2lI\',\n    \'test_tiny\': \'1viiB0s041CNsAK4itvX8PnYthJ-MDnQc\'\n}\n\n\'\'\'for name, f
ile_id in DATASETS_LINKS.items():\n    url = f"https://drive.google.com/uc?export=download&id={file_id}"\n    output = f"{name}.zi
```

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 matplotlib.pyplot as plt
from collections import Counter

import random
import torch
import torchvision.transforms as transforms
'''
```

(ЗАМЕНИЛ НА СОБСТВЕННЫЙ)

Класс Dataset(ЗАМЕНИЛ НА СОБСТВЕННЫЙ)

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
        self.transform = transform  # Добавляем трансформации
        ''' Если уже лежат в локальной директории '''
        #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 __len__(self):
        # Возвращаем количество файлов в датасете
        return self.n_files

    def __getitem__(self, idx):
        # Получаем i-е изображение и метку
        img, lbl = self.images[idx], self.labels[idx]

        # Преобразуем изображение из numpy в PIL для использования transforms
        img = Image.fromarray(img)

        # Применяем трансформации, если они указаны
```

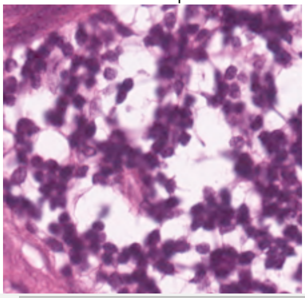
```
        if self.transform:
            img = self.transform(img)

        return img, lbl
    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 использвания 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)
'''
```

```
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.
```



Модель

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as data
from torchvision import transforms, models
from torchvision.models import EfficientNet_B3_Weights
from sklearn.model_selection import KFold, train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import wandb
import os
from PIL import Image
import matplotlib.pyplot as plt


#LBL12
# Инициализация WANDB(крутой сервис для визуализации и подбора гипер-параметров)
wandb.init(project="histology-classification", config={
    "epochs": 20,
    "batch_size": 32,
    "learning_rate": 1e-4,
    "num_folds": 5,
    "augmentation_epochs": 12,  # Эпохи с аугментацией(меньше чем количество эпох обучения)
    "optimizer": "adam"
})

# Получаем гиперпараметры из wandb
config = wandb.config
```

```
wandb: Using wandb-core as the SDK backend.  Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit: ··········
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

Tracking run with wandb version 0.18.7
Run data is saved locally in /content/drive/MyDrive/competition/wandb/run-20241202_201056-w8ghmiso
Syncing run **woven-sponge-7** to Weights & Biases (docs)
View project at https://wandb.ai/vortex_d11-m-v-lomonosovmoscow-state-university/histology-classification
View run at https://wandb.ai/vortex_d11-m-v-lomonosovmoscow-state-university/histology-classification/runs/w8ghmiso

```
class Dataset(data.Dataset):
    def __init__(self, images, labels, transform=None):
        self.images = images
        self.labels = labels
        self.transform = transform
        self.n_files = self.images.shape[0]

    def __len__(self):
        return self.n_files

    def __getitem__(self, idx):
        img, lbl = self.images[idx], self.labels[idx]
        img = Image.fromarray(img)
        if self.transform:
            img = self.transform(img)
        return img, lbl


class Trainer:
    def __init__(self, train_dataset=None, val_dataset=None, num_classes=9, config=None):
        self.train_dataset = train_dataset
        self.val_dataset = val_dataset
        self.num_classes = num_classes
        self.config = config or {}
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = self._initialize_model()
```

```
        self.criterion = nn.CrossEntropyLoss()

        if self.train_dataset is not None and self.val_dataset is not None:
            self.optimizer = self._initialize_optimizer()
            self.scheduler = optim.lr_scheduler.ReduceLROnPlateau(self.optimizer, 'min')

        self.checkpoint_dir = "/content/drive/MyDrive/checkpoints" #Гугл-диск
        os.makedirs(self.checkpoint_dir, exist_ok=True)

    def _initialize_model(self):
        weights = EfficientNet_B3_Weights.IMAGENET1K_V1
        model = models.efficientnet_b3(weights=weights)
        model.classifier[1] = nn.Linear(model.classifier[1].in_features, self.num_classes)
        return model.to(self.device)

    def _initialize_optimizer(self):
        if self.config.optimizer == "adam":
            optimizer = optim.Adam(self.model.parameters(), lr=self.config.learning_rate)
        elif self.config.optimizer == "sgd":
            optimizer = optim.SGD(self.model.parameters(), lr=self.config.learning_rate, momentum=0.9)
        else:
            raise ValueError(f"Unsupported optimizer: {self.config.optimizer}")
        return optimizer

    def train(self):
        train_loader = data.DataLoader(self.train_dataset, batch_size=self.config.batch_size, shuffle=Tru
        val_loader = data.DataLoader(self.val_dataset, batch_size=self.config.batch_size, shuffle=False)

        best_val_loss = float('inf')  # Чтобы сохранить лучшую модель
        for epoch in range(self.config.epochs):
            print(f'Epoch {epoch + 1}/{self.config.epochs}')
            self._train_one_epoch(train_loader, epoch)
            val_loss = self._validate(val_loader)
            self.scheduler.step(val_loss)
            wandb.log({"epoch": epoch, "val_loss": val_loss})

            # Сохраняем лучшую модель
            if val_loss < best_val_loss:
                best_val_loss = val_loss
                self._save_checkpoint(epoch, is_best=True)

        # Сохраняем модель после окончания обучения
        self._save_model()
        # Тест модели
        self._test(val_loader)

    def _train_one_epoch(self, loader, epoch):
        self.model.train()
        running_loss = 0.0
        for inputs, labels in loader:
            inputs, labels = inputs.to(self.device), labels.to(self.device)

            # Так как модель предобученная, то применяется лишь аугментация до "средних" эпох, ибо слишком
            if epoch >= self.config.epochs - self.config.augmentation_epochs:
                inputs = self._augment_data(inputs)

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

            running_loss += loss.item()
            wandb.log({"train_loss": loss.item()})

        epoch_loss = running_loss / len(loader)
        print(f'Training Loss: {epoch_loss:.4f}')
```

```
    def _validate(self, loader):
        self.model.eval()
        running_loss = 0.0
        all_preds = []
        all_labels = []
        with torch.no_grad():
            for inputs, labels in loader:
                inputs, labels = inputs.to(self.device), labels.to(self.device)
                outputs = self.model(inputs)
                loss = self.criterion(outputs, labels)
                running_loss += loss.item()

                preds = torch.argmax(outputs, dim=1)
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())
        #LBL5
        val_loss = running_loss / len(loader)
        val_accuracy = np.mean(np.array(all_preds) == np.array(all_labels))
        print(f'Validation Loss: {val_loss:.4f}, Accuracy: {val_accuracy:.4f}')
        wandb.log({"val_accuracy": val_accuracy})
        return val_loss

    def _test(self, loader):
        self.model.eval()
        all_preds = []
        all_labels = []
        with torch.no_grad():
            for inputs, labels in loader:
                inputs, labels = inputs.to(self.device), labels.to(self.device)
                outputs = self.model(inputs)
                preds = torch.argmax(outputs, dim=1)
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

        # Cinfussion matrix(почти как в лекциях)
        cm = confusion_matrix(all_labels, all_preds)
        cr = classification_report(all_labels, all_preds, output_dict=True)
        wandb.log({"confusion_matrix": wandb.plot.confusion_matrix(probs=None,
                                                           y_true=all_labels,
                                                           preds=all_preds,
                                                           class_names=[str(i) for i in range(sel
        #LBL8
        # Чувствительность и специфичность
        sensitivity = np.diag(cm) / np.sum(cm, axis=1)
        specificity = np.diag(cm) / np.sum(cm, axis=0)
        print(f'Sensitivity: {sensitivity}')
        print(f'Specificity: {specificity}')
        wandb.log({"sensitivity": sensitivity.mean(), "specificity": specificity.mean()})

        # Выгружаем результаты тестирования, нужные для первичной "оценки" модели
        #LBL9
        np.savez("val_results.npz", preds=all_preds, labels=all_labels)

    def _augment_data(self, inputs):
        #LBL10
        augmentation = transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.RandomRotation(15)
        ])
        augmented_inputs = []
        for img in inputs:
            img = transforms.ToPILImage()(img.cpu())
            img = augmentation(img)
            img = transforms.ToTensor()(img)
            augmented_inputs.append(img)
        return torch.stack(augmented_inputs).to(self.device)
    #LBL3
```

```
    def _save_checkpoint(self, epoch, is_best=False):
        checkpoint_path = os.path.join(self.checkpoint_dir, f'model_epoch_{epoch}.pth')
        torch.save({
            'epoch': epoch,
            'model_state_dict': self.model.state_dict(),
            'optimizer_state_dict': self.optimizer.state_dict(),
        }, checkpoint_path)
        print(f'Checkpoint saved at {checkpoint_path}')

        if is_best:
            best_model_path = os.path.join(self.checkpoint_dir, f'best_model.pth')
            torch.save(self.model.state_dict(), best_model_path)
            print(f'Best model saved at {best_model_path}')

        # WANDB moment
        artifact = wandb.Artifact('model', type='model')
        artifact.add_file(checkpoint_path)
        wandb.log_artifact(artifact)

    def _save_model(self):
        # Сохранение финальных весов модели для дальнейшей работы
        final_model_path = os.path.join(self.checkpoint_dir, f'final_model.pth')
        torch.save(self.model.state_dict(), final_model_path)
        print(f'Final model saved at {final_model_path}')

        # Нужно для wandb
        artifact = wandb.Artifact('final_model', type='model')
        artifact.add_file(final_model_path)
        wandb.log_artifact(artifact)

    def load_model(self, model_path):
        #LBL11
        # Загрузка весов модели для инференса или дообучения
        self.model.load_state_dict(torch.load(model_path, map_location=self.device))
        self.model.to(self.device)
        print(f'Model loaded from {model_path}')
    #LBL4
    def load_checkpoint(self, checkpoint_path):
        checkpoint = torch.load(checkpoint_path, map_location=self.device)
        self.model.load_state_dict(checkpoint['model_state_dict'])
        self.optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        start_epoch = checkpoint['epoch'] + 1
        print(f'Checkpoint loaded, resuming training from epoch {start_epoch}')
        # Если есть чекпоинты, то можем стартовать с последнего. Нужно, если вдруг аварийно все сломается
        return start_epoch

    def test_on_dataset(self, dataset, limit=None):
        self.model.eval()
        predictions = []
        true_labels = []
        n = len(dataset) if limit is None else int(len(dataset) * limit)
        loader = data.DataLoader(dataset, batch_size=1, shuffle=False)
        for i, (img, label) in enumerate(loader):
            if i >= n:
                break
            img = img.to(self.device)
            with torch.no_grad():
                output = self.model(img)
                pred = torch.argmax(output, dim=1).cpu().item()
                predictions.append(pred)
                true_labels.append(label.item())
        return predictions, true_labels

    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)
```

```
            pred = torch.argmax(output, dim=1).cpu().item()
        return pred


def inference(model, dataloader, device):
    model.eval()
    predictions = []
    with torch.no_grad():
        for inputs, _ in dataloader:
            inputs = inputs.to(device)
            outputs = model(inputs)
            preds = torch.argmax(outputs, dim=1)
            predictions.extend(preds.cpu().numpy())
    return predictions


# Основная функция
def main():
    # Приводим к адекватному представлению, для того чтобы модель "скушала" наши данные
    data_transforms = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
    ])

    # Обучаем на самом большом(из предложенных) датасете - train
    np_obj = np.load('train.npz')
    images = np_obj['data']
    labels = np_obj['labels']
                                                                    #LBL1
    train_images, test_images, train_labels, test_labels = train_test_split(
        images, labels, test_size=0.1, stratify=labels, random_state=42)

    test_dataset = Dataset(test_images, test_labels, transform=data_transforms)
    test_loader = data.DataLoader(test_dataset, batch_size=config.batch_size, shuffle=False)

    kfold_images = train_images
    kfold_labels = train_labels
    #LBL2
    kfold = KFold(n_splits=config.num_folds, shuffle=True, random_state=42)
    for fold, (train_idx, val_idx) in enumerate(kfold.split(kfold_images)):
        print(f'Fold {fold + 1}/{config.num_folds}')

        train_images_fold = kfold_images[train_idx]
        train_labels_fold = kfold_labels[train_idx]
        val_images_fold = kfold_images[val_idx]
        val_labels_fold = kfold_labels[val_idx]

        train_dataset = Dataset(train_images_fold, train_labels_fold, transform=data_transforms)
        val_dataset = Dataset(val_images_fold, val_labels_fold, transform=data_transforms)

        trainer = Trainer(train_dataset, val_dataset, config=config)

        latest_checkpoint = None
        if os.path.exists(trainer.checkpoint_dir):
            checkpoints = [ckpt for ckpt in os.listdir(trainer.checkpoint_dir) if 'model_epoch' in ckpt a
            if checkpoints:
                latest_checkpoint = max(checkpoints, key=lambda x: int(x.split('_')[-1].split('.')[0]))
                latest_checkpoint = os.path.join(trainer.checkpoint_dir, latest_checkpoint)

        if latest_checkpoint:
            start_epoch = trainer.load_checkpoint(latest_checkpoint)
        else:
            start_epoch = 0

        trainer.train()

        print("Testing on independent test set:")
        trainer.load_model(os.path.join(trainer.checkpoint_dir, 'best_model.pth'))
        test_results(trainer.model, test_loader, trainer.device)
```

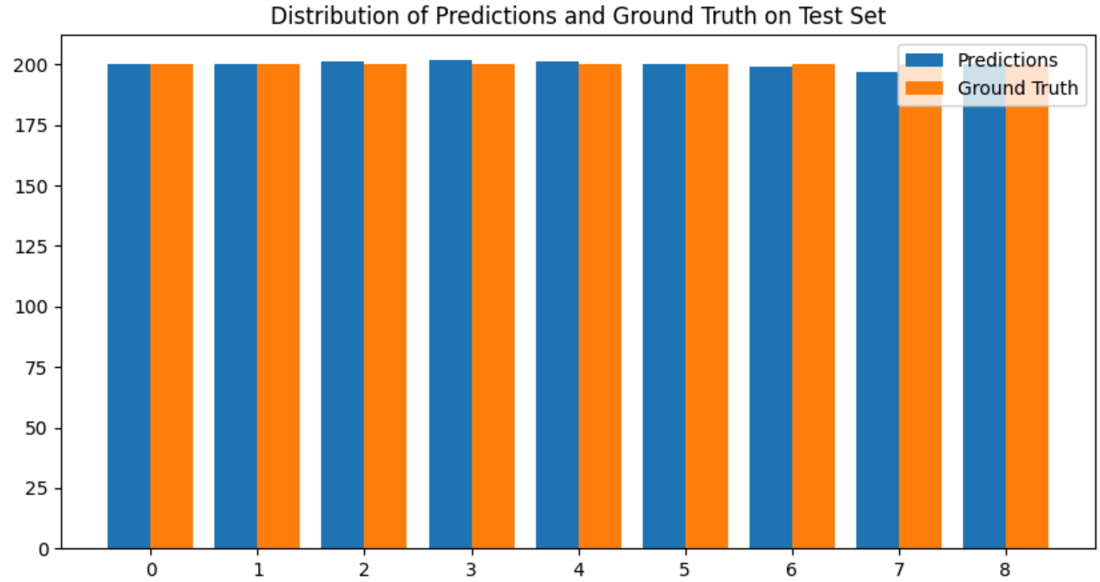
```
        # Визуализация результатов тестирования
        visualize_results()

        # если не хотим тратить ресурсы(а модель хорошо обучается и без дополнительных fold'ов)
        break
        # если нужны фолды(к-в), то убрать break

def test_results(model, loader, device):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            preds = torch.argmax(outputs, dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    cm = confusion_matrix(all_labels, all_preds)
    cr = classification_report(all_labels, all_preds, output_dict=True)
    #LBL6
    wandb.log({"test_confusion_matrix": wandb.plot.confusion_matrix(probs=None,
                                                                    y_true=all_labels,
                                                                    preds=all_preds,
                                                                    class_names=[str(i) for i in range(mo

    sensitivity = np.diag(cm) / np.sum(cm, axis=1)
    specificity = np.diag(cm) / np.sum(cm, axis=0)
    print(f'Test Sensitivity: {sensitivity}')
    print(f'Test Specificity: {specificity}')
    wandb.log({"test_sensitivity": sensitivity.mean(), "test_specificity": specificity.mean()})
    #LBL7
    np.savez("test_results.npz", preds=all_preds, labels=all_labels)

def visualize_results():
    results = np.load("test_results.npz")
    preds = results['preds']
    labels = results['labels']

    # Построение графиков
    plt.figure(figsize=(10, 5))
    plt.hist([preds, labels], label=['Predictions', 'Ground Truth'], bins=np.arange(10)-0.5, rwidth=0.8)
    plt.xticks(range(9))
    plt.legend()
    plt.title('Distribution of Predictions and Ground Truth on Test Set')
    plt.savefig('test_results.png')
    wandb.log({"test_results": wandb.Image('test_results.png')})
```

ОБУЧЕНИЕ

```
if __name__ == "__main__":
    main()
```

```
Fold 1/5
Epoch 1/20
Training Loss: 0.4861
Validation Loss: 0.0824, Accuracy: 0.9772
Checkpoint saved at /content/drive/MyDrive/checkpoints/model_epoch_0.pth
Best model saved at /content/drive/MyDrive/checkpoints/best_model.pth
Epoch 2/20
Training Loss: 0.0968
Validation Loss: 0.0531, Accuracy: 0.9833
Checkpoint saved at /content/drive/MyDrive/checkpoints/model_epoch_1.pth
Best model saved at /content/drive/MyDrive/checkpoints/best_model.pth
Epoch 3/20
Training Loss: 0.0509
Validation Loss: 0.0371, Accuracy: 0.9907
Checkpoint saved at /content/drive/MyDrive/checkpoints/model_epoch_2.pth
Best model saved at /content/drive/MyDrive/checkpoints/best_model.pth
Epoch 4/20
Training Loss: 0.0320
Validation Loss: 0.0371, Accuracy: 0.9892
Checkpoint saved at /content/drive/MyDrive/checkpoints/model_epoch_3.pth
Best model saved at /content/drive/MyDrive/checkpoints/best_model.pth
Epoch 5/20
Training Loss: 0.0195
Validation Loss: 0.0398, Accuracy: 0.9886
Epoch 6/20
Training Loss: 0.0168
Validation Loss: 0.0397, Accuracy: 0.9886
Epoch 7/20
Training Loss: 0.0154
Validation Loss: 0.0843, Accuracy: 0.9855
Epoch 8/20
Training Loss: 0.0107
Validation Loss: 0.1359, Accuracy: 0.9907
Epoch 9/20
Training Loss: 0.0549
Validation Loss: 0.0330, Accuracy: 0.9889
Checkpoint saved at /content/drive/MyDrive/checkpoints/model_epoch_8.pth
Best model saved at /content/drive/MyDrive/checkpoints/best_model.pth
Epoch 10/20
Training Loss: 0.0274
Validation Loss: 0.0275, Accuracy: 0.9910
Checkpoint saved at /content/drive/MyDrive/checkpoints/model_epoch_9.pth
Best model saved at /content/drive/MyDrive/checkpoints/best_model.pth
Epoch 11/20
Training Loss: 0.0240
Validation Loss: 0.0290, Accuracy: 0.9923
Epoch 12/20
Training Loss: 0.0208
Validation Loss: 0.0222, Accuracy: 0.9917
Checkpoint saved at /content/drive/MyDrive/checkpoints/model_epoch_11.pth
Best model saved at /content/drive/MyDrive/checkpoints/best_model.pth
Epoch 13/20
Training Loss: 0.0176
Validation Loss: 0.0229, Accuracy: 0.9941
Epoch 14/20
Training Loss: 0.0143
Validation Loss: 0.0266, Accuracy: 0.9910
Epoch 15/20
Training Loss: 0.0138
Validation Loss: 0.0222, Accuracy: 0.9929
Epoch 16/20
Training Loss: 0.0141
Validation Loss: 0.0266, Accuracy: 0.9929
Epoch 17/20
Training Loss: 0.0106
Validation Loss: 0.0209, Accuracy: 0.9938
Checkpoint saved at /content/drive/MyDrive/checkpoints/model_epoch_16.pth
Best model saved at /content/drive/MyDrive/checkpoints/best_model.pth
Epoch 18/20
Training Loss: 0.0145
Validation Loss: 0.0228, Accuracy: 0.9951
Epoch 19/20
Training Loss: 0.0076
Validation Loss: 0.0206, Accuracy: 0.9938
Checkpoint saved at /content/drive/MyDrive/checkpoints/model_epoch_18.pth
Best model saved at /content/drive/MyDrive/checkpoints/best_model.pth
Epoch 20/20
Training Loss: 0.0094
Validation Loss: 0.0271, Accuracy: 0.9920
Final model saved at /content/drive/MyDrive/checkpoints/final_model.pth
Sensitivity: [0.99430199 0.99702381 0.99197861 0.9921671  0.99152542 0.99711816
 0.99455041 0.97574124 0.99439776]
Specificity: [1.         0.99406528 0.99197861 1.         1.         0.98295455
 0.99455041 0.98102981 0.9833795 ]
Testing on independent test set:
<ipython-input-35-a0ef99328ed6>:177: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default val
  self.model.load_state_dict(torch.load(model_path, map_location=self.device))
Model loaded from /content/drive/MyDrive/checkpoints/best_model.pth
Test Sensitivity: [1.    1.    0.995 1.    0.99  0.99  0.99  0.97  0.995]
Test Specificity: [1.         1.         0.99004975 0.99009901 0.98507463 0.99
```

```
0.99497487 0.98477157 0.995     ]
```



МЕТРИКИ

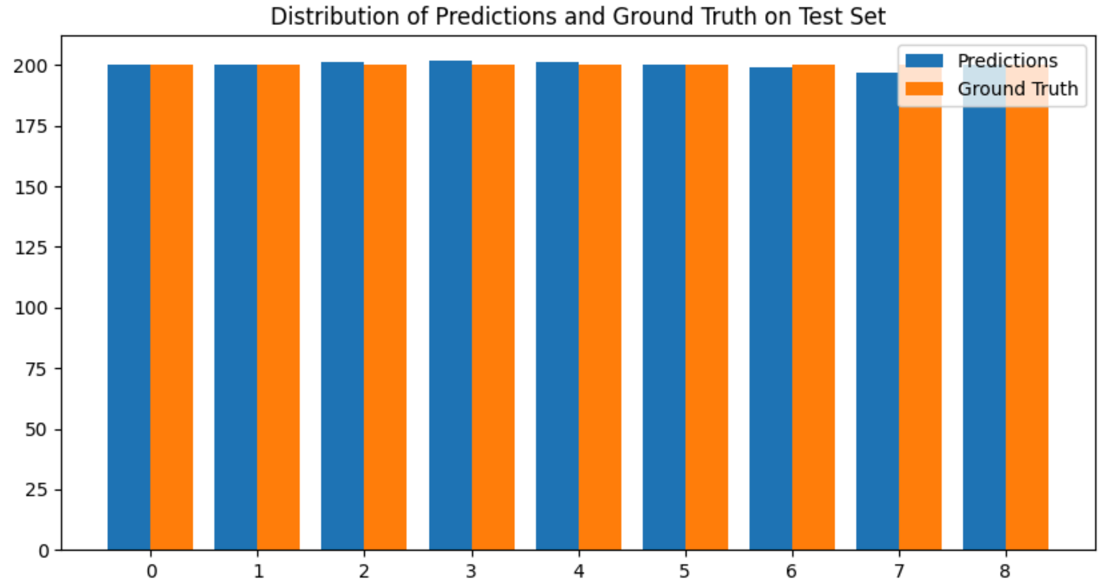
```
data_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

np_obj = np.load('test.npz')
test_images = np_obj['data']
test_labels = np_obj['labels']
test_dataset = Dataset(test_images, test_labels, transform=data_transforms)

trainer = Trainer(config=config)
trainer.transforms = data_transforms

best_model_path = os.path.join(trainer.checkpoint_dir, 'best_model.pth')
trainer.load_model(best_model_path)

predictions, true_labels = trainer.test_on_dataset(test_dataset)

from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(true_labels, predictions)
print("Confusion Matrix:")
print(cm)
cr = classification_report(true_labels, predictions, digits=4)
print("Classification Report:")
print(cr)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-10-323abd388847> in <cell line: 7>()
      5 
      6 np_obj = np.load('test.npz')
----> 7 test_images = np_obj['data']
      8 test_labels = np_obj['labels']
      9 test_dataset = Dataset(test_images, test_labels, transform=data_transforms)

4 frames

/usr/lib/python3.10/zipfile.py in _read1(self, n)
   1004         elif self._compress_type == ZIP_DEFLATED:
   1005             n = max(n, self.MIN_READ_SIZE)
-> 1006             data = self._decompressor.decompress(data, n)
   1007             self._eof = (self._decompressor.eof or
   1008                          self._compress_left <= 0 and

KeyboardInterrupt: 
```

```python
from typing import List
from sklearn.metrics import balanced_accuracy_score


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)))


def test_results1(model, loader, device, class_names):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in loader:
            inputs = inputs.to(device)
            labels = labels.to(device)
            outputs = model(inputs)
            preds = torch.argmax(outputs, dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())


    Metrics.print_all(all_labels, all_preds, 'test')


def main():
    data_transforms = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
    ])

    trainer = Trainer(config=config)
    trainer.model.to(trainer.device)


    best_model_path = os.path.join(trainer.checkpoint_dir, 'best_model.pth')
    trainer.load_model(best_model_path)
```

```
    test_data = np.load('test_tiny.npz')
    test_images = test_data['data']
    test_labels = test_data['labels']
    test_dataset = Dataset(test_images, test_labels, transform=data_transforms)
    test_loader = data.DataLoader(test_dataset, batch_size=config.batch_size, shuffle=False)

    class_names = ['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM']

    print("Evaluating model:")
    test_results1(trainer.model, test_loader, trainer.device, class_names)

    visualize_results()

main()
```

```
<ipython-input-13-b9849e0b1108>:175: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value
  self.model.load_state_dict(torch.load(model_path, map_location=self.device))
Model loaded from /content/drive/MyDrive/checkpoints/best_model.pth
Evaluating model:
metrics for test:
        accuracy 0.9889:
        balanced accuracy 0.9889:
```



ПРИМЕР С СОЗДАНИЕМ НЕСБАЛАНСИРОВАННОГО ДАТАСЕТА

```
from collections import Counter

def create_unbalanced_dataset(images, labels, class_ratios):

    unbalanced_images = []
    unbalanced_labels = []
    unique_classes = np.unique(labels)
    for cls in unique_classes:
        cls_indices = np.where(labels == cls)[0]
        n_samples = int(len(cls_indices) * class_ratios.get(cls, 1.0))
        selected_indices = np.random.choice(cls_indices, n_samples, replace=False)
        unbalanced_images.append(images[selected_indices])
        unbalanced_labels.append(labels[selected_indices])
    unbalanced_images = np.concatenate(unbalanced_images)
    unbalanced_labels = np.concatenate(unbalanced_labels)
    return unbalanced_images, unbalanced_labels


test_data = np.load('test.npz')
test_images = test_data['data']
test_labels = test_data['labels']
```

```
class_ratios = {
    1: 0.15, # 15% от 1класса(забыл какой там)
    3: 0.5, # Оставить 50% от класса LYM
    4: 0.10, # Оставить 10% от класса MUC
    5: 0.05, # Оставить 5% от класса MUS
}

unbalanced_images, unbalanced_labels = create_unbalanced_dataset(test_images, test_labels, class_ratios)


unbalanced_dataset = Dataset(unbalanced_images, unbalanced_labels, transform=data_transforms)
unbalanced_loader = data.DataLoader(unbalanced_dataset, batch_size=config.batch_size, shuffle=False)


print("Evaluating model on unbalanced test dataset:")
test_results1(trainer.model, unbalanced_loader, trainer.device, class_names)
```

```
Evaluating model on unbalanced test dataset:
metrics for test:
	 accuracy 0.9914:
	 balanced accuracy 0.9944:
```

ЗАГРУЗКА МОДЕЛИ С ДРАЙВА

```
import gdown

# Ссылка на модель в Google Drive

url_model1 = "https://drive.google.com/uc?export=download&id=117ANVkbp1Ktn4iQllV7JOEdXHL0wk8N6"

#Точно ссылка на модель(продублирую еще в README ссылку на директорию и /content/drive/MyDrive/checkpoints/

model_w = "model_weights.pth"

gdown.download(url_model1, model_w, quiet=False)

print(f"Файл {model_w} успешно скачан.")
```

```
Downloading...
From (original): https://drive.google.com/uc?export=download&id=117ANVkbp1Ktn4iQllV7JOEdXHL0wk8N6
From (redirected): https://drive.google.com/uc?export=download&id=117ANVkbp1Ktn4iQllV7JOEdXHL0wk8N6&confirm=t&uuid=5ffff98b-e680-497
To: /content/drive/MyDrive/competition/model_weights.pth
100%|██████████| 43.4M/43.4M [00:00<00:00, 96.3MB/s]
Файл model_weights.pth успешно скачан.
```

```
#Ссылки на представленные датасеты на моем(!) gd(из-за проблем с диском)
import gdown

DATASETS_LINKS = { #Возможно где-то ошибься с ссылками...
    'train': '1kT3GtZWcLqeuPECEGNXpMNi3jp9LZY_B',
    'train_small': '1IMMjP6oUFNmrDnmPRpuFt5vzHDGfELMY',
    'train_tiny': '1Ay1nq2ngf4XyVgE-d8S_z19dfD5PNszy',
    'test': '1EdCDUOB3bJLSkt117v-eSbIXiERur1Xr',
    'test_small': '1vF2YdizwJmsuzEY0hasrFnpE-uPoGAO_',
    'test_tiny': '1pxwyXm4bSsAtnlfujjfUrHldkHeJg76A'
}

name = "test_tiny"

file_id = DATASETS_LINKS[name]
output = f'{name}.npz'

gdown.download(id=file_id, output=output, quiet=False)

print(f'Загружаем датасет {name} из файла npz.')
```

```
Downloading...
From: https://drive.google.com/uc?id=1pxwyXm4bSsAtnlfujjfUrHldkHeJg76A
To: /content/drive/MyDrive/competition/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 17.2MB/s]Загружаем датасет test_tiny из файла npz.
```

```
def main():
    data_transforms = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
    ])

    trainer = Trainer(config=config)
    trainer.model.to(trainer.device)


    trainer.load_model(model_w)


    test_data = np.load(f'{name}.npz')
    test_images = test_data['data']
    test_labels = test_data['labels']
    test_dataset = Dataset(test_images, test_labels, transform=data_transforms)
    test_loader = data.DataLoader(test_dataset, batch_size=config.batch_size, shuffle=False)

    class_names = ['ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM']

    print("Evaluating model:")
    test_results1(trainer.model, test_loader, trainer.device, class_names)

    visualize_results()

main()
```

```
<ipython-input-7-0877451826ea>:178: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value),
  self.model.load_state_dict(torch.load(model_path, map_location=self.device))
Model loaded from model_weights.pth
Evaluating model:
metrics for test:
         accuracy 0.9889:
         balanced accuracy 0.9889:
```