Практическое задание №1

Установка необходимых пакетов:

```
!pip install -q tqdm
!pip install --upgrade --no-cache-dir gdown
!pip install --upgrade --no-cache-dir opencv-python-headless==4.1.2.30
!pip install albumentations==1.1.0
```

Монтирование Вашего Google Drive к текущему окружению:

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

```
    Mounted at /content/drive
```

В переменную PROJECT_DIR необходимо прописать путь к директории на Google Drive. Это пригодится при сохранении модели.

```
PROJECT_DIR = 'drive/MyDrive/nn_prak1/'
```

Константы, которые пригодятся в коде далее:

```
EVALUATE_ONLY = True
TEST_ON_LARGE_DATASET = True
TISSUE_CLASSES = ('ADI', 'BACK', 'DEB', 'LYM', 'MUC', 'MUS', 'NORM', 'STR', 'TUM
```

Ссылки (gdrive идентификаторы) на предоставляемые наборы данных:

```
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'
}
```

Импорт необходимых зависимостей:

```
import IPython.display
import os
import numpy as np
import gdown
from pathlib import Path
from typing import List
from time import sleep
from tqdm.notebook import tqdm
from collections import namedtuple

import albumentations as A
import albumentations.pytorch.transforms as T
from sklearn.metrics import classification_report

from PIL import Image
from sklearn.metrics import balanced_accuracy_score

import torch
from torch import nn
from torch.utils.tensorboard import SummaryWriter
from torchvision.transforms.functional import to_pil_image
```

```
%load_ext tensorboard
```

Класс Dataset

Предназначен для работы с наборами данных, хранящихся на Google Drive, обеспечивает чтение изображений и соответствующих меток, а также формирование пакетов (батчей).

```
class Dataset:
    def __init__(self, name):
        self.name = name
        self.is_loaded = False
        if not Path(f'{name}.npz').exists():
            url = f'https://drive.google.com/uc?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]
```

```
# Загрузка датасетов в ОЗУ
ds_train_numpy = Dataset("train")
ds_test_numpy = Dataset("test")
```

```
Loading dataset train from npz.
Done. Dataset train consists of 18000 images.
Loading dataset test from npz.
Done. Dataset test consists of 4500 images.
```

Обертка для класса Dataset

1. Для разбиения обучающей выборки на train и val
2. Для последующего использования DataLoader из torch

```
class DatasetTorch(torch.utils.data.Dataset):
    def __init__(self, dataset: Dataset, mode: str, transforms=None, train_part=
        if mode not in ['train', 'val', 'all']:
            raise ValueError(f"Bad mode: {mode}")

        images, labels = dataset.images, dataset.labels

        if mode == 'all':
            self.samples = list(zip(images, labels))
        else:
            # LBL1
            # разбиение выполняется так, чтобы не возникло дисбаланса классов
            result_idx = []
            for c in np.unique(labels):
                indexes = np.argwhere(labels == c)

                split_part = int(len(indexes) * train_part)
                start, end = None, split_part
                if mode == 'val':
                    start, end = end, start

                result_idx += list(indexes[start: end])

            result_idx = np.array(result_idx).ravel()
            self.samples = list(zip(images[result_idx], labels[result_idx]))

        self.transforms = transforms

    def __getitem__(self, idx: int):
        image, label = self.samples[idx]

        # apply transforms and convert to tensor
        if self.transforms:
            image = self.transforms(image=image)['image']
        image = torch.tensor(image.transpose(2, 0, 1), dtype=torch.float32) / 25
        return image, label

    def __len__(self):
        return len(self.samples)
```

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 lengt
        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)

    # LBL6
    @staticmethod
    def cs_report(gt: List[int], pred: List[int]):
        return classification_report(gt, pred, target_names=TISSUE_CLASSES)

    @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
```

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 ConvBlock(nn.Module):
    def __init__(self, ch_in, ch_out):
        super(ConvBlock, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(ch_in, ch_out, kernel_size=3, stride=1, padding=1, bias=Tr
            nn.BatchNorm2d(ch_out),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        x = self.conv(x)
        return x


class BaseCNN(nn.Module):
```

```
    def __init__(self, img_ch, num_classes):
        super().__init__()

        self.Maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.Conv1 = ConvBlock(img_ch, 16)
        self.Conv2 = ConvBlock(16, 32)
        self.Conv3 = ConvBlock(32, 64)
        self.Conv4 = ConvBlock(64, 128)
        self.Conv5 = ConvBlock(128, 256)

        self.classification_head = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256 * 7 * 7, 128),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.Conv1(x)
        x = self.Maxpool(x)

        x = self.Conv2(x)
        x = self.Maxpool(x)

        x = self.Conv3(x)
        x = self.Maxpool(x)

        x = self.Conv4(x)
        x = self.Maxpool(x)

        x = self.Conv5(x)
        x = self.Maxpool(x)

        return self.classification_head(x)


class Model:
    def __init__(self, cfg):
        self.cfg = cfg

        self.device = cfg.device
        self.out_dir = os.path.join(cfg.out_dir, cfg.model_name)
        os.makedirs(self.out_dir, exist_ok=True)

        log_dir = os.path.join(self.out_dir, "runs", cfg.unique_name)
```

```
        os.makedirs(log_dir, exist_ok=True)

        self.writer = SummaryWriter(log_dir=log_dir)
        self.writer.add_text("Config", str(self.cfg), global_step=None, walltime

        self.model = self.get_model(self.cfg.model_name)
        self.criterion = nn.CrossEntropyLoss()


    def save(self, path: str):
        torch.save(self.model.state_dict(), path)


    def load(self, name: str):
        name_to_id_dict = {
            'base': '1-drIhe8o4nA5wReKDcFvC9gkLN6beuvD',
            'base_full': '1-jbIAuuA1l9RzFZGAtSvaKwPWauSmyPx',
            'best': '1-jbIAuuA1l9RzFZGAtSvaKwPWauSmyPx'
        }
        output = f'{name}.pth'
        gdown.download(f'https://drive.google.com/uc?id={name_to_id_dict[name]}'
        self.model.load_state_dict(torch.load(output, map_location='cpu'))


    def train(self, train_ds: Dataset):
        self.model.to(self.device)
        params = [p for p in self.model.parameters() if p.requires_grad]
        optimizer = self._get_optimizer(params, self.cfg.optimizer, self.cfg.lr,
        lr_scheduler = self._get_scheduler(
            optimizer,
            self.cfg.scheduler,
            self.cfg.milestones,
            self.cfg.sch_gamma,
            self.cfg.patience
        )
        train_dl, val_dl = self.get_train_dataloaders(
            train_ds,
            self.cfg.batch_train, self.cfg.batch_val, self.cfg.num_workers
        )

        # creating directory for model weights
        weights_dir = os.path.join(self.out_dir, "weights")
        os.makedirs(weights_dir, exist_ok=True)

        for epoch in range(1, self.cfg.epochs + 1):
            print(f"Epoch: {epoch}")

            # training and logging
```

```
            logs = self.train_one_epoch(optimizer, train_dl, epoch)
            for key, value in logs.items():
                self.writer.add_scalar(key, value, epoch)

            if lr_scheduler is not None:
                lr_scheduler.step()

            # LBL1
            # LBL5
            # evaluation and logging
            metrics = self.evaluate(val_dl)
            for key, value in metrics.items():
                self.writer.add_scalar(f"val/{key}", value, epoch)

        # LBL2
        # saving weights
        self.save(os.path.join(weights_dir, f"{self.cfg.unique_name}.pth"))


    def retrain(self, train_ds: Dataset):
        """
        Retrain on full train set
        """
        self.model.to(self.device)
        params = [p for p in self.model.parameters() if p.requires_grad]
        optimizer = self._get_optimizer(params, self.cfg.optimizer, self.cfg.lr,
        lr_scheduler = self._get_scheduler(
            optimizer,
            self.cfg.scheduler,
            self.cfg.milestones,
            self.cfg.sch_gamma,
            self.cfg.patience
        )

        train_ds = DatasetTorch(train_ds, "train", train_part=1.0)
        train_dl = torch.utils.data.DataLoader(
            train_ds,
            batch_size=self.cfg.batch_train,
            num_workers=self.cfg.num_workers,
            shuffle=True
        )

        # creating directory for model weights
        weights_dir = os.path.join(self.out_dir, "weights")
        os.makedirs(weights_dir, exist_ok=True)

        for epoch in range(1, self.cfg.epochs + 1):
            print(f"Epoch: {epoch}")
```

```
            # training
            self.train_one_epoch(optimizer, train_dl, epoch)

            if lr_scheduler is not None:
                lr_scheduler.step()

        # saving weights
        self.save(os.path.join(weights_dir, f"{self.cfg.unique_name}_full.pth"))


    def train_one_epoch(self, optimizer, train_dl, epoch, verbose=True):
        self.model.train()

        lr_scheduler = None
        if epoch == 1:
            warmup_factor = 1.0 / 1000
            warmup_iters = min(1000, len(train_dl) - 1)

            lr_scheduler = torch.optim.lr_scheduler.LinearLR(
                optimizer, start_factor=warmup_factor, total_iters=warmup_iters
            )

        for images, targets in tqdm(train_dl):
            optimizer.zero_grad()
            loss = self.criterion(self.model(images.to(self.device)), targets.to

            loss.backward()
            optimizer.step()

            if lr_scheduler is not None:
                lr_scheduler.step()

        # LBL3
        if verbose:
            print(f"train loss: {loss}")

        return {"loss": loss.detach(), "lr": optimizer.param_groups[0]["lr"]}


    def evaluate(self, val_dl, verbose=True):
        self.model.eval()

        prediction, target = [], []
        for images, targets in tqdm(val_dl):
            outputs = self.model(images.to(self.device)).to('cpu')
            target += list(targets.numpy())
            _, predicted = torch.max(outputs, 1)
```

```
            prediction += list(predicted.detach().numpy())

        acc = Metrics.accuracy(target, prediction)
        bc_acc = Metrics.accuracy_balanced(target, prediction)

        # LBL3
        if verbose:
            print(f"val accuracy: {acc}")

        return {"accuracy": acc, "bc_accuracy": bc_acc}


    @staticmethod
    def get_model(model_name):
        if model_name == "base":
            return BaseCNN(img_ch=3, num_classes=9)
        raise ValueError(f"{model_name} model isn't supported")

    @staticmethod
    def _get_optimizer(parameters, optimizer_name, lr, weight_decay):
        if optimizer_name == "Adam":
            return torch.optim.Adam(
                parameters, lr=lr, weight_decay=weight_decay
            )
        elif optimizer_name == "SGD":
            return torch.optim.SGD(
                parameters, lr=lr, momentum=0.9, weight_decay=weight_decay
            )
        raise ValueError(f"{optimizer_name} optimizer isn't supported")

    @staticmethod
    def _get_scheduler(optimizer, scheduler_name, milestones, sch_gamma, patienc
        if scheduler_name == "ReduceLROnPlateau":
            return torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, patienc
        elif scheduler_name == "MultiStepLR":
            return torch.optim.lr_scheduler.MultiStepLR(optimizer, milestones, g
        elif scheduler_name == "None":
            return None

    def get_train_dataloaders(self, ds_numpy, batch_train, batch_val, num_worker
        train_ds = DatasetTorch(ds_numpy, "train")
        val_ds = DatasetTorch(ds_numpy, "val")

        train_dl = torch.utils.data.DataLoader(
            train_ds,
            batch_size=batch_train,
            num_workers=num_workers,
            shuffle=True
```

```
        )

        val_dl = torch.utils.data.DataLoader(
            val_ds,
            batch_size=batch_val,
            num_workers=num_workers,
            shuffle=False,
        )

        return train_dl, val_dl

    def test_on_dataset(self, dataset: Dataset, limit=None):
        self.model.eval()
        self.model.to('cpu')
        predictions = []
        n = dataset.n_files if not limit else int(dataset.n_files * limit)
        for img in tqdm(dataset.images_seq(n), total=n):
            predictions.append(self.test_on_image(img))
        return predictions

    def test_on_image(self, img: np.ndarray):
        img = torch.tensor(img.transpose(2, 0, 1), dtype=torch.float32) / 255
        _, prediction = torch.max(self.model(img.unsqueeze(0)), 1)
        return prediction.numpy()
```

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'.

Обучение модели

```
# hyperparameters
cfg_dict = {
    "out_dir": Path(PROJECT_DIR),
    "model_name": "base",
    "unique_name": "base",

    "batch_train": 64,
    "batch_val": 128,
    "num_workers": 2,

    "device": "cuda",  # training device
    "epochs": 12,
    "optimizer": "Adam",
    "lr": 0.001,
    "weight_decay": 0.0001,

    "scheduler": "MultiStepLR",
    "milestones": [8],
    "sch_gamma": 0.1,
    "patience": None
}

cfg = namedtuple("Config", cfg_dict.keys())(**cfg_dict)


if not EVALUATE_ONLY:
    model = Model(cfg)

    # обучение с валидацией
    # каждую эпоху логируется лосс на обучении и точность на валидации
    model.train(ds_train_numpy)
```

```
Epoch: 1
100%                                                225/225 [00:35<00:00, 6.61it/s]
train loss: 0.6984010338783264
100%                                                29/29 [00:05<00:00, 7.11it/s]
val accuracy: 0.6058333333333333
Epoch: 2
100%                                                225/225 [00:34<00:00, 6.56it/s]
train loss: 0.4925207793712616
100%                                                29/29 [00:05<00:00, 7.35it/s]
val accuracy: 0.69
Epoch: 3
100%                                                225/225 [00:35<00:00, 6.49it/s]
train loss: 0.3764006197452545
```

```
100%                                                  29/29 [00:05<00:00, 6.87it/s]

val accuracy: 0.7366666666666667
Epoch: 4
100%                                                  225/225 [00:35<00:00, 6.54it/s]

train loss: 0.2263260781764984
100%                                                  29/29 [00:05<00:00, 7.10it/s]

val accuracy: 0.5483333333333333
Epoch: 5
100%                                                  225/225 [00:35<00:00, 6.57it/s]

train loss: 0.13592074811458588
100%                                                  29/29 [00:05<00:00, 6.91it/s]

val accuracy: 0.6136111111111111
Epoch: 6
100%                                                  225/225 [00:35<00:00, 6.55it/s]

train loss: 0.13036738336086273
100%                                                  29/29 [00:05<00:00, 7.19it/s]

val accuracy: 0.7222222222222222
Epoch: 7
100%                                                  225/225 [00:34<00:00, 6.59it/s]

train loss: 0.21652568876743317
100%                                                  29/29 [00:05<00:00, 6.91it/s]

val accuracy: 0.7266666666666667
Epoch: 8
100%                                                  225/225 [00:34<00:00, 6.61it/s]

train loss: 0.17136777937412262
100%                                                  29/29 [00:05<00:00, 6.98it/s]

val accuracy: 0.7947222222222222
Epoch: 9
100%                                                  225/225 [00:34<00:00, 6.58it/s]

train loss: 0.02141961269080639
100%                                                  29/29 [00:05<00:00, 6.99it/s]

val accuracy: 0.9758333333333333
Epoch: 10
100%                                                  225/225 [00:35<00:00, 6.53it/s]

train loss: 0.04547959566116333
100%                                                  29/29 [00:05<00:00, 7.11it/s]

val accuracy: 0.9716666666666667
Epoch: 11
100%                                                  225/225 [00:35<00:00, 6.46it/s]

train loss: 0.028606321662664413
```

```
100%                                                 29/29 [00:05<00:00, 7.10it/s]
val accuracy: 0.975
Epoch: 12
100%                                                 225/225 [00:35<00:00, 6.51it/s]
train loss: 0.03376896306872368
100%                                                 29/29 [00:05<00:00, 7.05it/s]
val accuracy: 0.9794444444444445
```

```
if not EVALUATE_ONLY:
    model = Model(cfg)

    # обучение на полной обучающей выборке с найденными гиперпараметрами без лог
    model.retrain(ds_train_numpy)
```

```
Epoch: 1
```

100% 282/282 [00:44<00:00, 7.01it/s]

```
train loss: 0.8149434924125671
Epoch: 2
```

100% 282/282 [00:44<00:00, 7.06it/s]

```
train loss: 1.059192180633545
Epoch: 3
```

100% 282/282 [00:44<00:00, 7.00it/s]

```
train loss: 0.3788038492202759
Epoch: 4
```

100% 282/282 [00:43<00:00, 7.05it/s]

```
train loss: 0.37819769978523254
Epoch: 5
```

100% 282/282 [00:43<00:00, 7.10it/s]

```
train loss: 0.15828156471252441
Epoch: 6
```

100% 282/282 [00:44<00:00, 7.10it/s]

```
train loss: 0.537818193435669
Epoch: 7
```

100% 282/282 [00:44<00:00, 7.08it/s]

```
train loss: 0.18117311596870422
Epoch: 8
```

100% 282/282 [00:43<00:00, 7.09it/s]

```
train loss: 0.027162831276655197
Epoch: 9
```

100% 282/282 [00:43<00:00, 7.09it/s]

```
train loss: 0.09539880603551865
Epoch: 10
```

100% 282/282 [00:44<00:00, 7.09it/s]

```
train loss: 0.12581753730773926
Epoch: 11
```

100% 282/282 [00:44<00:00, 7.09it/s]

```
train loss: 0.21720291674137115
Epoch: 12
```

100% 282/282 [00:44<00:00, 7.13it/s]

```
train loss: 0.10715653002262115
```

```
if EVALUATE_ONLY:
    model = Model(cfg)
    model.load('best')
```

```
Downloading...
From: https://drive.google.com/uc?id=1-jbIAuuA1l9RzFZGAtSvaKwPWauSmyPx
To: /content/best.pth
100%|██████████| 8.02M/8.02M [00:00<00:00, 145MB/s]
```

```
# логи во время обучения с валидацией
# здесь необходимо указать директорию до логов Path(PROJECT_DIR) / cfg.unique_na
# LBL4
%tensorboard --logdir drive/MyDrive/nn_prak1/base/runs
```

TensorBoard SCALARS TEXT INACTIVE

Show data download links

Ignore outliers in chart scalin

Tooltip sorting method: default

Smoothing

0,6

Horizontal Axis

STEP RELATIVE

WALL

Runs

Write a regex to filter runs

base

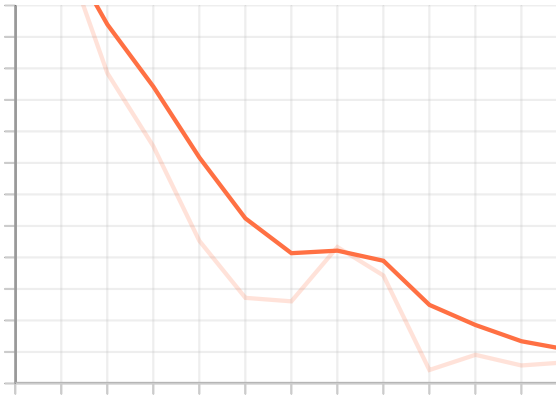
TOGGLE ALL RUNS

drive/MyDrive/nn_prak1/base/runs

Filter tags (regular expressions supported)

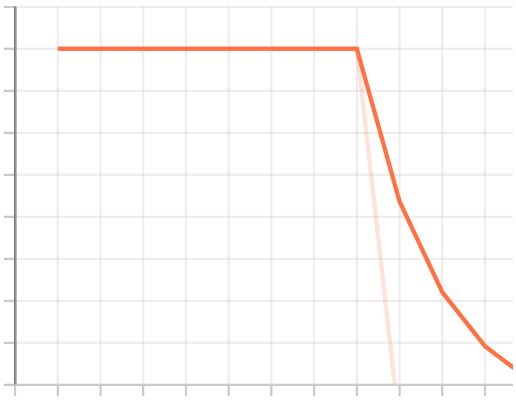
loss

loss
tag: loss



lr

lr
tag: lr



```
# сохранение логов в tensorboard dev
# понадобится войти в аккаунт
# !tensorboard dev upload --logdir drive/MyDrive/nn_prak1/base/runs \
#   --name "Histological patches classification"
```

Ссылка на загруженные логи:

https://tensorboard.dev/experiment/3h7pm4iVRBWDRbNxGknlnA/

Пример тестирования модели на части набора данных:

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

```
100%                                                 450/450 [00:12<00:00, 38.04it/s]
metrics for 10% of test:
         accuracy 0.9956:
         balanced accuracy 0.9956:
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1
  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(ds_test_numpy)
    Metrics.print_all(ds_test_numpy.labels, pred_2, 'test')
    print('\n classification report:', Metrics.cs_report(ds_test_numpy.labels, p
```

```
100%                                                4500/4500 [02:10<00:00, 24.20it/s]
metrics for test:
        accuracy 0.9800:
        balanced accuracy 0.9800:

 classification report:
              precision    recall  f1-score   support

         ADI       1.00      1.00      1.00       500
        BACK       1.00      1.00      1.00       500
         DEB       0.98      0.96      0.97       500
         LYM       1.00      1.00      1.00       500
         MUC       0.97      0.98      0.98       500
         MUS       0.97      0.98      0.98       500
        NORM       0.98      0.97      0.97       500
         STR       0.94      0.97      0.95       500
         TUM       0.98      0.97      0.97       500

    accuracy                           0.98      4500
   macro avg       0.98      0.98      0.98      4500
weighted avg       0.98      0.98      0.98      4500
```

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

```
Downloading...
From: https://drive.google.com/uc?id=1-jbIAuuA1l9RzFZGAtSvaKwPWauSmyPx
To: /content/best.pth
100%|██████████| 8.02M/8.02M [00:00<00:00, 142MB/s]
Downloading...
From: https://drive.google.com/uc?id=1viiB0s041CNsAK4itvX8PnYthJ-MDnQc
To: /content/test_tiny.npz
100%|██████████| 10.6M/10.6M [00:00<00:00, 45.2MB/s]Loading dataset test_ti
Done. Dataset test_tiny consists of 90 images.

100%                                                  90/90 [00:02<00:00, 36.58it/s]
metrics for test-tiny:
         accuracy 0.9333:
         balanced accuracy 0.9333:
```

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 caluclated {n_runs} times in {timeit.timeit(f, number=n_ru
```

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 t
# 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 нейроной 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 туториалов 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 удлось сразу разобраться или хочется вникнуть в тему более глубоко), можете посмотреть следующий отличный туториал: 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/cbernet/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 осущетвляться с примонтированным Google Drive.

Создадим 2 изображения, поместим их в директорию tmp внутри PROJECT_DIR, запакуем директорию tmp в архив tmp.zip.

```
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"
```