

✓ Практическое задание №2

Общая терминология по используемым данным

Предоставляемые данные для разработки моделей и алгоритмов трекинга мяча в теннисе представляют собой набор игр (game), состоящих из нескольких клипов (clip), каждый из которых состоит из набора кадров (frame). Обратите внимание на структуру организации файлов внутри предоставляемого датасета для полного понимания.

Большинство алгоритмов трекинга объектов работают с несколькими последовательными кадрами, и в данном задании также подразумевается использование этого приема. Последовательность нескольких кадров будем именовать стопкой (stack), размер стопки (stack_s) является гиперпараметром разрабатываемого алгоритма.

✓ Заготовка решения

Загрузка датасета

Для работы с данными в ноутбуке kaggle необходимо подключить датасет. File -> Add or upload data, далее в поиске написать tennis-tracking-assignment и выбрать датасет. Если поиск не работает, то можно добавить датасет по url:

<https://www.kaggle.com/xubiker/tennistackingassignment>. После загрузки данные датасета будут примонтированы в `../input/tennistackingassignment`.

✓ Установка и импорт зависимостей

Установка необходимых пакетов (не забудьте "включить интернет" в настройках ноутбука kaggle):

```
!pip install moviepy --upgrade
!pip install gdown

Requirement already satisfied: moviepy in /opt/conda/lib/python3.10/site-packages (1.0.3)
Requirement already satisfied: decorator<5.0,>=4.0.2 in /opt/conda/lib/python3.10/site-packages (from moviepy) (4.4.2)
Requirement already satisfied: tqdm<5.0,>=4.11.2 in /opt/conda/lib/python3.10/site-packages (from moviepy) (4.66.1)
Requirement already satisfied: requests<3.0,>=2.8.1 in /opt/conda/lib/python3.10/site-packages (from moviepy) (2.31.0)
Requirement already satisfied: proglog<1.0.0 in /opt/conda/lib/python3.10/site-packages (from moviepy) (0.1.10)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/lib/python3.10/site-packages (from moviepy) (1.24.3)
Requirement already satisfied: imageio<3.0,>=2.5 in /opt/conda/lib/python3.10/site-packages (from moviepy) (2.31.1)
Requirement already satisfied: imageio-ffmpeg>=0.2.0 in /opt/conda/lib/python3.10/site-packages (from moviepy) (0.4.9)
Requirement already satisfied: pillow>=8.3.2 in /opt/conda/lib/python3.10/site-packages (from imageio<3.0,>=2.5->moviepy) (10.1.0)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.10/site-packages (from imageio-ffmpeg>=0.2.0->moviepy) (68.1.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.10/site-packages (from requests<3.0,>=2.8.1->moviepy) (3.4)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-packages (from requests<3.0,>=2.8.1->moviepy) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/site-packages (from requests<3.0,>=2.8.1->moviepy) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-packages (from requests<3.0,>=2.8.1->moviepy) (2023.11.17)
Requirement already satisfied: gdown in /opt/conda/lib/python3.10/site-packages (4.7.1)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-packages (from gdown) (3.12.2)
Requirement already satisfied: requests[socks] in /opt/conda/lib/python3.10/site-packages (from gdown) (2.31.0)
Requirement already satisfied: six in /opt/conda/lib/python3.10/site-packages (from gdown) (1.16.0)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.10/site-packages (from gdown) (4.66.1)
Requirement already satisfied: beautifulsoup4 in /opt/conda/lib/python3.10/site-packages (from gdown) (4.12.2)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/lib/python3.10/site-packages (from beautifulsoup4->gdown) (2.3.2.post1)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.10/site-packages (from requests[socks]->gdown) (3.4)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-packages (from requests[socks]->gdown) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/site-packages (from requests[socks]->gdown) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-packages (from requests[socks]->gdown) (2023.11.17)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /opt/conda/lib/python3.10/site-packages (from requests[socks]->gdown) (1.7.0)
```

После установки пакетов для корректной работы надо обязательно перезагрузить ядро. Run -> Restart and clear cell outputs. Без сего действия будет ошибка при попытке обращения к библиотеке moviepy при сохранении визуализации в виде видео. Может когда-то авторы библиотеки это починят...

Импорт необходимых зависимостей:

```
from pathlib import Path
from typing import List, Tuple, Sequence

import numpy as np
from numpy import unravel_index
from PIL import Image, ImageDraw, ImageFont
```

```

from tqdm import tqdm, notebook

from moviepy.video.io.ImageSequenceClip import ImageSequenceClip

import math
from scipy.ndimage import gaussian_filter

import cv2 as cv

import gdown

import IPython.display

import torch
from torch import nn
from torch.nn import functional as F
import torchvision
from torchvision.transforms import v2

import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

import gc
import time
import random
import csv

```

✓ Набор функций для загрузки данных из датасета

Функция `load_clip_data` загружает выбранный клип из выбранной игры и возвращает его в виде numpy массива `[n_frames, height, width, 3]` типа `uint8`. Для ускорения загрузки используется кэширование - однажды загруженные клипы хранятся на диске в виде `prz` архивов, при последующем обращении к таким клипам происходит загрузка `prz` архива.

Также добавлена возможность чтения клипа в половинном разрешении `640x360`, вместо оригинального `1280x720` для упрощения и ускорения разрабатываемых алгоритмов.

Функция `load_clip_labels` загружает референсные координаты мяча в клипе в виде numpy массива `[n_frames, 4]`, где в каждой строке массива содержатся значения `[code, x, y, q]`. `x, y` соответствуют координате центра мяча на кадре, `q` не используется в данном задании, `code` описывает статус мяча:

- `code = 0` - мяча в кадре нет
- `code = 1` - мяч присутствует в кадре и легко идентифицируем
- `code = 2` - мяч присутствует в кадре, но сложно идентифицируем
- `code = 3` - мяч присутствует в кадре, но заслонен другими объектами.

При загрузке в половинном разрешении координаты `x, y` делятся на 2.

Функция `load_clip` загружает выбранный клип и соответствующий массив координат и возвращает их в виде пары.

```

def get_num_clips(path: Path, game: int) -> int:
    return len(list((path / f'game{game}').iterdir()))

def get_game_clip_pairs(path: Path, games: List[int]) -> List[Tuple[int, int]]:
    return [(game, c) for game in games for c in range(1, get_num_clips(path, game) + 1)]

def load_clip_data(path: Path, game: int, clip: int, downscale: bool, quiet=False) -> np.ndarray:
    if not quiet:
        suffix = 'downscaled' if downscale else ''
        print(f'loading clip data (game {game}, clip {clip}) {suffix}')
    cache_path = path / 'cache'
    cache_path.mkdir(exist_ok=True)
    resize_code = '_ds2' if downscale else ''
    cached_data_name = f'{game}_{clip}{resize_code}.npz'
    if (cache_path / cached_data_name).exists():
        clip_data = np.load(cache_path / cached_data_name)['clip_data']
    else:
        clip_path = path / f'game{game}/clip{clip}'
        n_imgs = len(list(clip_path.iterdir())) - 1
        imgs = [None] * n_imgs
        for i in notebook.tqdm(range(n_imgs)):
            img = Image.open(clip_path / f'{i:04d}.jpg')
            if downscale:
                img = img.resize((img.width // 2, img.height // 2),)

```

```

        imgs[i] = np.array(img, dtype=np.uint8)
        clip_data = np.stack(imgs)
        cache_path.mkdir(exist_ok=True, parents=True)
        np.savez_compressed(cache_path / cached_data_name, clip_data=clip_data)
    return clip_data

def load_clip_labels(path: Path, game: int, clip: int, downscale: bool, quiet=False):
    if not quiet:
        print(f'loading clip labels (game {game}, clip {clip})')
    clip_path = path / f'game{game}/clip{clip}'
    labels = []
    with open(clip_path / 'labels.csv') as csvfile:
        lines = list(csv.reader(csvfile))
        for line in lines[1:]:
            values = np.array([-1 if i == '' else int(i) for i in line[1:]])
            if downscale:
                values[1] //= 2
                values[2] //= 2
            labels.append(values)
    return np.stack(labels)

def load_clip(path: Path, game: int, clip: int, downscale: bool, quiet=False):
    data = load_clip_data(path, game, clip, downscale, quiet)
    labels = load_clip_labels(path, game, clip, downscale, quiet)
    return data, labels

```

✓ Набор дополнительных функций

Еще несколько функций, немного облегчающих выполнение задания:

- `prepare_experiment` создает новую директорию в `out_path` для хранения результатов текущего эксперимента. Нумерация выполняется автоматически, функция возвращает путь к созданной директории эксперимента;
- `ball_gauss_template` - создает "шаблон" мяча, может быть использована в алгоритмах поиска мяча на изображении по корреляции;
- `create_masks` - принимает набор кадров и набор координат мяча, и генерирует набор масок, в которых помещает шаблон мяча на заданные координаты. Может быть использована при обучении нейронной сети семантической сегментации;

```

def prepare_experiment(out_path: Path) -> Path:
    out_path.mkdir(parents=True, exist_ok=True)
    dirs = [d for d in out_path.iterdir() if d.is_dir() and d.name.startswith('exp_')]
    experiment_id = max(int(d.name.split('_')[1]) for d in dirs) + 1 if dirs else 1
    exp_path = out_path / f'exp_{experiment_id}'
    exp_path.mkdir()
    return exp_path

def ball_gauss_template(rad, sigma):
    x, y = np.meshgrid(np.linspace(-rad, rad, 2 * rad + 1), np.linspace(-rad, rad, 2 * rad + 1))
    dst = np.sqrt(x * x + y * y)
    gauss = np.exp(-(dst ** 2 / (2.0 * sigma ** 2)))
    return gauss

def create_masks(data: np.ndarray, labels: np.ndarray, resize):
    rad = 64 #25
    sigma = 10
    if resize:
        rad //= 2
    ball = ball_gauss_template(rad, sigma)
    n_frames = data.shape[0]
    sh = rad
    masks = []
    for i in range(n_frames):
        label = labels[i, ...]
        frame = data[i, ...]
        if 0 < label[0] < 3:
            x, y = label[1:3]
            mask = np.zeros((frame.shape[0] + 2 * rad + 2 * sh, frame.shape[1] + 2 * rad + 2 * sh), np.float32)
            mask[y + sh : y + sh + 2 * rad + 1, x + sh : x + sh + 2 * rad + 1] = ball
            mask = mask[rad + sh : -rad - sh, rad + sh : -rad - sh]
            masks.append(mask)
        else:
            masks.append(np.zeros((frame.shape[0], frame.shape[1]), dtype=np.float32))

```

```
return np.stack(masks)
```

✓ Набор функций, предназначенных для визуализации результатов

Функция `visualize_prediction` принимает набор кадров, набор координат детекции мяча (можно подавать как референсные значения, так и предсказанные) и создает видеоклип, в котором отрисовывается положение мяча, его трек, номер кадра и метрика качества трекинга (если она была передана в функцию). Видеоклип сохраняется в виде `mp4` файла. Кроме того данная функция создает текстовый файл, в который записывает координаты детекции мяча и значения метрики качества трекинга.

Функция `visualize_prob` принимает набор кадров и набор предсказанных карт вероятности и создает клип с наложением предсказанных карт вероятности на исходные карты. Области "подсвечиваются" желтым, клип сохраняется в виде `mp4` видеофайла. Данная функция может быть полезна при наличии в алгоритме трекинга сети, осуществляющей семантическую сегментацию.

```
def _add_frame_number(frame: np.ndarray, number: int) -> np.ndarray:
    fnt = ImageFont.load_default() # ImageFont.truetype("arial.ttf", 25)
    img = Image.fromarray(frame)
    draw = ImageDraw.Draw(img)
    draw.text((10, 10), f'frame {number}', font=fnt, fill=(255, 0, 255))
    return np.array(img)

def _vis_clip(data: np.ndarray, lbls: np.ndarray, metrics: List[float] = None, ball_rad=5, color=(255, 0, 0), track_length=10):
    print('performing clip visualization')
    n_frames = data.shape[0]
    frames_res = []
    fnt = ImageFont.load_default() # ImageFont.truetype("arial.ttf", 25)
    for i in range(n_frames):
        img = Image.fromarray(data[i, ...])
        draw = ImageDraw.Draw(img)
        txt = f'frame {i}'
        if metrics is not None:
            txt += f', SiBaTrAcc: {metrics[i]:.3f}'
        draw.text((10, 10), txt, font=fnt, fill=(255, 0, 255))
        label = lbls[i]
        if label[0] != 0: # the ball is clearly visible
            px, py = label[1], label[2]
            draw.ellipse((px - ball_rad, py - ball_rad, px + ball_rad, py + ball_rad), outline=color, width=2)
            for q in range(track_length):
                if lbls[i-q-1][0] == 0:
                    break
                if i - q > 0:
                    draw.line((lbls[i - q - 1][1], lbls[i - q - 1][2], lbls[i - q][1], lbls[i - q][2]), fill=color)
        frames_res.append(np.array(img))
    return frames_res

def _save_clip(frames: Sequence[np.ndarray], path: Path, fps):
    assert path.suffix in ('.mp4', '.gif')
    clip = ImageSequenceClip(frames, fps=fps)
    if path.suffix == '.mp4':
        clip.write_videofile(str(path), fps=fps, logger=None)
    else:
        clip.write_gif(str(path), fps=fps, logger=None)

def _to_yellow_heatmap(frame: np.ndarray, pred_frame: np.ndarray, alpha=0.4):
    img = Image.fromarray((frame * alpha).astype(np.uint8))
    maskR = (pred_frame * (1 - alpha) * 255).astype(np.uint8)
    maskG = (pred_frame * (1 - alpha) * 255).astype(np.uint8)
    maskB = np.zeros_like(maskG, dtype=np.uint8)
    mask = np.stack([maskR, maskG, maskB], axis=-1)
    return img + mask

def _vis_pred_heatmap(data_full: np.ndarray, pred_prob: np.ndarray, display_frame_number):
    n_frames = data_full.shape[0]
    v_frames = []
    for i in range(n_frames):
        frame = data_full[i, ...]
        pred = pred_prob[i, ...]
        hm = _to_yellow_heatmap(frame, pred)
        if display_frame_number:
            hm = _add_frame_number(hm, i)
        v_frames.append(hm)
    return v_frames
```

```
def visualize_prediction(data_full: np.ndarray, labels_pr: np.ndarray, save_path: Path, name: str, metrics=None, fps=15):
    with open(save_path / f'{name}.txt', mode='w') as f:
        if metrics is not None:
            f.write(f'SiBaTrAcc: {metrics[-1]} \n')
        for i in range(labels_pr.shape[0]):
            f.write(f'frame {i}: {labels_pr[i, 0]}, {labels_pr[i, 1]}, {labels_pr[i, 2]} \n')

    v = _vis_clip(data_full, labels_pr, metrics)
    _save_clip(v, save_path / f'{name}.mp4', fps=fps)

def visualize_prob(data: np.ndarray, pred_prob: np.ndarray, save_path: Path, name: str, frame_number=True, fps=15):
    v_pred = _vis_pred_heatmap(data, pred_prob, frame_number)
    _save_clip(v_pred, save_path / f'{name}_prob.mp4', fps=fps)
```

✓ Класс DataGenerator

Класс, отвечающий за генерацию данных для обучения модели. Принимает на вход путь к директории с играми, индексы игр, используемые для генерации данных, и размер стопки. Хранит в себе автоматически обновляемый пул с клипами игр.

В пуле содержится pool_s клипов. DataGenerator позволяет генерировать батч из стопок (размера stack_s) последовательных кадров. Выбор клипа для извлечения данных взвешенно-случайный: чем больше длина клипа по сравнению с другими клипами в пуле, тем вероятнее, что именно из него будет сгенерирована стопка кадров. Выбор стопки кадров внутри выбранного клипа полностью случаен. Кадры внутри стопки конкатенируются по последнему измерению (каналам).

После генерирования количества кадров равного общему количеству кадров, хранимых в пуле, происходит автоматическое обновление пула: из пула извлекаются pool_update_s случайных клипов, после чего в пул загружаются pool_update_s случайных клипов, не присутствующих в пуле. В случае, если размер пула pool_s больше или равен суммарному количеству клипов в играх, переданных в конструктор, все клипы сразу загружаются в пул, и автообновление не производится.

Использование подобного пула позволяет работать с практически произвольным количеством клипов, без необходимости загружать их всех в оперативную память.

Для вашего удобства функция извлечения стопки кадров из пула помимо самой стопки также создает и возвращает набор сгенерированных масок с мячом исходя из референсных координат мяча в клипе.

Функция random_g принимает гиперпараметр размера стопки кадров и предоставляет генератор, возвращающий стопки кадров и соответствующие им маски. Данный генератор может быть использован при реализации решения на tensorflow. Обновление пула происходит автоматически, об этом беспокоиться не нужно.

```
class DataGenerator:

    def __init__(self, path: Path, games: List[int], stack_s, downscale, pool_s=30, pool_update_s=10, pool_autoupdate=True, quiet=False):
        self.path = path
        self.stack_s = stack_s
        self.downscale = downscale
        self.pool_size = pool_s
        self.pool_update_size = pool_update_s
        self.pool_autoupdate = pool_autoupdate
        self.quiet = quiet
        self.data = []
        self.masks = []

        self.frames_in_pool = 0
        self.produced_frames = 0
        self.game_clip_pairs = get_game_clip_pairs(path, list(set(games)))
        self.game_clip_pairs_loaded = []
        self.game_clip_pairs_not_loaded = list.copy(self.game_clip_pairs)
        self.pool = {}

        self._first_load()

    def _first_load(self):
        # --- if all clips can be placed into pool at once, there is no need to refresh pool at all ---
        if len(self.game_clip_pairs) <= self.pool_size:
            for gcp in self.game_clip_pairs:
                self._load(gcp)
            self.game_clip_pairs_loaded = list.copy(self.game_clip_pairs)
            self.game_clip_pairs_not_loaded.clear()
            self.pool_autoupdate = False
        else:
            self._load_to_pool(self.pool_size)
            self._update_clip_weights()
```

```

def _load(self, game_clip_pair):
    game, clip = game_clip_pair
    data, labels = load_clip(self.path, game, clip, self.downscale, quiet=self.quiet)
    masks = create_masks(data, labels, self.downscale)
    weight = data.shape[0] if data.shape[0] >= self.stack_s else 0
    self.pool[game_clip_pair] = (data, labels, masks, weight)
    self.frames_in_pool += data.shape[0] - self.stack_s + 1
    # print(f'items in pool: {len(self.pool)} - {self.pool.keys()}')

def _remove(self, game_clip_pair):
    value = self.pool.pop(game_clip_pair)
    self.frames_in_pool -= value[0].shape[0] - self.stack_s + 1
    del value
    # print(f'items in pool: {len(self.pool)} - {self.pool.keys()}')

def _update_clip_weights(self):
    weights = [self.pool[pair][-1] for pair in self.game_clip_pairs_loaded]
    tw = sum(weights)
    self.clip_weights = [w / tw for w in weights]
    # print(f'clip weights: {self.clip_weights}')

def _remove_from_pool(self, n):
    # --- remove n random clips from pool ---
    if len(self.game_clip_pairs_loaded) >= n:
        remove_pairs = random.sample(self.game_clip_pairs_loaded, n)
        for pair in remove_pairs:
            self._remove(pair)
            self.game_clip_pairs_loaded.remove(pair)
            self.game_clip_pairs_not_loaded.append(pair)
        gc.collect()

def _load_to_pool(self, n):
    # --- add n random clips to pool ---
    gc.collect()
    add_pairs = random.sample(self.game_clip_pairs_not_loaded, n)
    for pair in add_pairs:
        self._load(pair)
        self.game_clip_pairs_not_loaded.remove(pair)
        self.game_clip_pairs_loaded.append(pair)

def update_pool(self):
    self._remove_from_pool(self.pool_update_size)
    self._load_to_pool(self.pool_update_size)
    self._update_clip_weights()

def get_random_stack(self):
    pair_idx = np.random.choice(len(self.game_clip_pairs_loaded), 1, p=self.clip_weights)[0]
    game_clip_pair = self.game_clip_pairs_loaded[pair_idx]
    d, _, m, _ = self.pool[game_clip_pair]
    start = np.random.choice(d.shape[0] - self.stack_s, 1)[0]
    frames_stack = d[start : start + self.stack_s, ...]
    frames_stack = np.squeeze(np.split(frames_stack, indices_or_sections=self.stack_s, axis=0))
    frames_stack = np.concatenate(frames_stack, axis=-1)
    mask = m[start + self.stack_s - 1, ...]
    return frames_stack, mask

def get_random_batch(self, batch_s):
    imgs, masks = [], []
    while len(imgs) < batch_s:
        frames_stack, mask = self.get_random_stack()
        imgs.append(frames_stack)
        masks.append(mask)
    if self.pool_autoupdate:
        self.produced_frames += batch_s
        # print(f'produced frames: {self.produced_frames} from {self.frames_in_pool}')
        if self.produced_frames >= self.frames_in_pool:
            self.update_pool()
            self.produced_frames = 0
    return np.stack(imgs), np.stack(masks)

def random_g(self, batch_s):
    while True:
        imgs_batch, masks_batch = self.get_random_batch(batch_s)
        yield imgs_batch, masks_batch

```

✓ Пример использования DataGenerator

Рекомендованный размер пула pool_s=10 в случае использования уменьшенных вдвое изображений. При большем размере пула есть большая вероятность нехватки имеющихся 13G оперативной памяти. Используйте параметр quiet=True в конструкторе DataGenerator, если хотите скрыть все сообщения о чтении данных и обновлении пула.

```
stack_s = 3
batch_s = 4
train_gen = DataGenerator(Path('../input/tennistackingassignment/train/'), [1, 2, 3, 4], stack_s=stack_s, downscale=True, pool_s=10, p
for i in range(10):
    imgs, masks = train_gen.get_random_batch(batch_s)
    print(imgs.shape, imgs.dtype, masks.shape, masks.dtype)

imgs, masks = train_gen.get_random_batch(batch_s)
print(imgs.shape, imgs.dtype, masks.shape, masks.dtype)
fig, axes = plt.subplots(2, 2, figsize=(12,8))
axes[0, 0].imshow(imgs[0, :, :, 0:3])
axes[0, 1].imshow(imgs[0, :, :, 3:6])
axes[1, 0].imshow(imgs[0, :, :, 6:9])
axes[1, 1].imshow(masks[0, :, :])

import matplotlib.pyplot as plt

stack_s = 3
train_gen = DataGenerator(Path('../input/tennistackingassignment/train/'), [1], stack_s=stack_s, downscale=True, pool_s=10, pool_update
stack, mask = train_gen.get_random_stack()
print(stack.shape, mask.shape)

for i in range(stack_s):
    plt.figure()
    plt.imshow(stack[:, :, 3 * i: 3 * i + 3])
```

✓ Класс Metrics

Класс для вычисления метрики качества трекинга SiBaTrAcc. Функция evaluate_predictions принимает массив из референсных и предсказанных координат мяча для клипа и возвращает массив аккумулированных значений SiBaTrAcc (может быть полезно для визуализации результатов предсказания) и итоговое значение метрики SiBaTrAcc.

```
class Metrics:

    @staticmethod
    def position_error(label_gt: np.ndarray, label_pr: np.ndarray, step=8, alpha=1.5, e1=5, e2=5):
        # gt codes:
        # 0 - the ball is not within the image
        # 1 - the ball can easily be identified
        # 2 - the ball is in the frame, but is not easy to identify
        # 3 - the ball is occluded
        if label_gt[0] != 0 and label_pr[0] == 0:
            return e1
        if label_gt[0] == 0 and label_pr[0] != 0:
            return e2
        dist = math.sqrt((label_gt[1] - label_pr[1]) ** 2 + (label_gt[2] - label_pr[2]) ** 2)
        pe = math.floor(dist / step) ** alpha
        pe = min(pe, 5)
        return pe

    @staticmethod
    def evaluate_predictions(labels_gt, labels_pr) -> Tuple[List[float], float]:
        pe = [Metrics.position_error(labels_gt[i, ...], labels_pr[i, ...]) for i in range(len(labels_gt))]
        SIBATRACC = []
        for i, _ in enumerate(pe):
            SIBATRACC.append(1 - sum(pe[: i + 1]) / ((i + 1) * 5))
        SIBATRACC_total = 1 - sum(pe) / (len(labels_gt) * 5)
        return SIBATRACC, SIBATRACC_total
```

✓ Основной класс модели SuperTrackingModel

Реализует всю логику обучения, сохранения, загрузки и тестирования разработанной модели трекинга. Этот класс можно и нужно расширять.

В качестве примера вам предлагается заготовка модели, в которой трекинг осуществляется за счет предсказания маски по входному батчу и последующему предсказанию координат мяча по полученной маски. В данном варианте вызов функции

предсказания координат по клипу (predict) повлечет за собой разбиение клипа на батчи, вызов предсказания маски для каждого батча, склеивание результатов в последовательность масок, вызов функции по вычислению координат мяча по маскам и возвращения результата. Описанные действия уже реализованы, вам остается только написать функции predict_on_bath и get_labels_from_prediction. Эта же функция predict используется и в вызове функции test, дополнительно вычисляя метрику качества трекинга и при необходимости визуализируя результат тестирования. Обратите внимание, что в результирующем пипру массиве с координатами помимо значений x и y первым значением в каждой строке должно идти значение code (0, если мяча в кадре нет и > 0, если мяч в кадре есть) для корректного вычисления качества трекинга.

Вам разрешается менять логику работы класса модели, (например, если решение не подразумевает использование масок), но при этом логика и работа функций load и test должна остаться неизменной!

```
def encoder_block(in_channels, out_channels, kernel_size, padding):
    block = nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size, padding=padding),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(),
        nn.MaxPool2d(2),
    )

    return block

def decoder_block(in_channels, out_channels, kernel_size, padding):
    block = nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size, padding=padding),
        nn.BatchNorm2d(out_channels),
        nn.ReLU(),
        nn.Upsample(scale_factor=2, mode='bilinear'),
    )

    return block

class UNet(nn.Module):
    def __init__(self, in_channels, size, autoencoder=True):
        super().__init__()

        self.enc1_block = encoder_block(in_channels, 32, 7, 3)
        self.enc2_block = encoder_block(32, 64, 3, 1)
        self.enc3_block = encoder_block(64, 128, 3, 1)

        self.adec1_block = decoder_block(128, 64, 3, 1)
        self.adec2_block = decoder_block(64, 32, 3, 1)
        self.adec3_block = decoder_block(32, in_channels, 3, 1)

        self.dec1_block = decoder_block(128, 64, 3, 1)
        self.dec2_block = decoder_block(128, 32, 3, 1)
        self.dec3_block = decoder_block(64, 1, 3, 1)

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(size[0] * size[1] // 64, 2)
        self.fc = nn.Linear(size[0] * size[1], 3)
        self.autoencoder = autoencoder

    def forward(self, x):
        enc1 = self.enc1_block(x)
        enc2 = self.enc2_block(enc1)
        enc3 = self.enc3_block(enc2)

        if self.autoencoder:
            dec1 = self.adec1_block(enc3)
            dec2 = self.adec2_block(dec1)
            dec3 = self.adec3_block(dec2)

            return dec3
        else:
            dec1 = self.dec1_block(enc3)
            dec2 = self.dec2_block(torch.cat([dec1, enc2], 1))
            dec3 = self.dec3_block(torch.cat([dec2, enc1], 1))
            return torch.nn.functional.softmax((dec3 * x[:, -1, :, :].reshape(dec3.shape)).permute(0, 2, 3, 1).flatten(1, 2))

class VNet(nn.Module):
    def __init__(self, in_channels, size):
        super().__init__()

        self.enc1_block = encoder_block(in_channels, 64, 7, 3)
        self.enc2_block = encoder_block(64, 128, 3, 1)
        self.enc3_block = encoder_block(128, 256, 3, 1)

        self.dec1_block = decoder_block(256, 128, 3, 1)
        self.dec2_block = decoder_block(256, 64, 3, 1)
```



```

self.dec3_block = decoder_block(128, 1, 3, 1)

def forward(self, x):
    enc1 = self.enc1_block(x)
    enc2 = self.enc2_block(enc1)
    enc3 = self.enc3_block(enc2)

    dec1 = self.dec1_block(enc3)
    dec2 = self.dec2_block(torch.cat([dec1, enc2], 1))
    dec3 = self.dec3_block(torch.cat([dec2, enc1], 1))
    return (torch.nn.functional.sigmoid(dec3)).permute(0, 2, 3, 1).flatten(1, 2)

def create_model(stack_s, img_channels=3, downscale=True):
    shape = None
    if downscale:
        shape = (360, 640)
    else:
        shape = (720, 1280)
    return VNet(stack_s * img_channels + more_channels, shape)

extra_channels = 2
more_channels = 1

def line(point1, point2):

    x1, y1 = point1
    x2, y2 = point2

    A = y2 - y1
    B = x1 - x2
    C = x2 * y1 - x1 * y2

    return A, B, C

def distance_to_line(point, coefficients):
    x0, y0 = point
    A, B, C = coefficients

    # Вычисляем расстояние от точки до прямой
    distance = abs(A * x0 + B * y0 + C) / math.sqrt(A**2 + B**2)

    return distance

#LBL1
def regularise(coords): # (n_frames, 3) (y/n, w, h) (720, 1280)
    n_frames = coords.shape[0]
    i = 2
    while i < n_frames - 2:

        if coords[i - 2, 0] != 0 and coords[i - 1, 0] != 0 and coords[i + 2, 0] != 0 and coords[i + 1, 0] != 0:

            A, B, C = line(coords[i - 1, 1: 3], coords[i - 2, 1: 3])
            A1, B1, C1 = line(coords[i + 1, 1: 3], coords[i + 2, 1: 3])

            vx = coords[i - 1, 1] - coords[i - 2, 1]
            vy = coords[i - 1, 2] - coords[i - 2, 2]

            px = coords[i - 1, 1] + vx
            py = coords[i - 1, 2] + vy

            if coords[i, 0] == 0:
                if py < 1280 and py > 0 and px < 720 and px > 0:
                    coords[i] = [1, px, py]
            else:
                if (distance_to_line(coords[i, 1:3], (A, B, C)) > 30 and distance_to_line(coords[i, 1:3], (A1, B1, C1)) > 30
                    or distance_to_line(coords[i, 1:3], (A, B, C)) > 100 or distance_to_line(coords[i, 1:3], (A1, B1, C1)) > 100):
                    coords[i] = [1, px, py]
            i += 1

    return coords

def get_diff(X):
    grey1 = cv.cvtColor(X[:, :, 0:3], cv.COLOR_RGB2GRAY)
    grey2 = cv.cvtColor(X[:, :, 3:6], cv.COLOR_RGB2GRAY)
    grey3 = cv.cvtColor(X[:, :, 6:9], cv.COLOR_RGB2GRAY)
    diff = (grey1 - grey2) ** 2 + (grey2 - grey3) ** 2
    diff = diff / (diff.max()) > 0.25
    return diff

def get_bins(X):
    grey = cv.cvtColor(X, cv.COLOR_RGB2GRAY)
    threshold = cv.adaptiveThreshold(grey, 255, cv.ADAPTIVE_THRESH_MEAN_C, \

```

```

        cv.THRESH_BINARY, 5, -5)
edges = cv.Canny(X,100, 200)
return np.concatenate((threshold[:, :, None], edges[:, :, None]), axis=2)

def expand_channels(X):
    shape = X.shape
    ans = np.zeros((shape[0], shape[1], (shape[2] // 3) * extra_channels + more_channels))
    for i in range(X.shape[2] // 3):
        ans[:, :, i * extra_channels: (i + 1) * extra_channels] = get_bins(X[:, :, i * 3: (i + 1) * 3])

    ans[:, :, -1] = get_diff(X)

    return ans

def prepare_batch(batch):
    X_batch, y_batch = batch
    shape = X_batch.shape
    X_add = np.zeros((shape[0], shape[1], shape[2], (shape[3] // 3) * extra_channels + more_channels))
    for i in range(X_batch.shape[0]):
        X_add[i] = expand_channels(X_batch[i])
    X_add = torch.Tensor(X_add).permute(0, 3, 1, 2).type(torch.FloatTensor)
    X_batch = torch.Tensor(X_batch).permute(0, 3, 1, 2).type(torch.FloatTensor)
    X_batch = torch.cat([X_batch, X_add], 1)
    y_batch = torch.Tensor(y_batch).type(torch.FloatTensor)[: , : , : , None]

    return X_batch, y_batch

#LBL2
class MyLoss(nn.Module):
    def __init__(self, shape, batch_s):
        super(MyLoss, self).__init__()
        self.ce = nn.CrossEntropyLoss(weight=torch.Tensor([0.1, 1.]).to(device))
        self.mse = nn.MSELoss()
        self.shape = shape
        self.bs = batch_s

    def forward(self, inputs, targets):
        relu = nn.functional.relu
        inputs = inputs.to(device)

        targets = targets.to(device)

        ninputs = inputs.flatten(1, 2).to(device)
        ntargets = targets.flatten(1, 3).to(device)

        is_ball = torch.sign(relu(ntargets.max(dim=1)[0])).to(device)

        noball = torch.sign(relu(ninputs.max(dim=1)[0] - 0.5)) - is_ball
        noball = torch.sign(noball) * noball

        target_w = (ntargets * 3 + 0.1).to(device)

        mse = (ntargets - ninputs) ** 2
        coord = (1 - (torch.sign(relu(ninputs - ninputs.max(dim=1)[0]).reshape(self.bs, 1).repeat(1, self.shape[0] * self.shape[1]) + 1e-5
#         mul_max = (relu(-coord) * 2).mean()
        return torch.mean(mse.to(device) * target_w) + noball.mean() * 2 + torch.mean(coord).abs() * 1.5

class SuperTrackingModel:

    def __init__(self, batch_s, stack_s, out_path, downscale):
        self.batch_s = batch_s
        self.stack_s = stack_s
        self.out_path = out_path
        self.downscale = downscale
        if self.downscale:
            self.shape = (360, 640)
        else:
            self.shape = (720, 1280)
        self.model = create_model(stack_s, img_channels=extra_channels + 3, downscale=downscale).to(device)

    def load(self, name='best', from_disk=True):
        output = f'/kaggle/working/exp_1/{name}.txt'
        if from_disk:
            name_to_id_dict = {
                'best': '1ZkWK3WUuf3g56fXHHYwJkZxGIjURF1Au'
            }
            output = f'{name}.npz'
            gdown.download(f'https://drive.google.com/uc?id={name_to_id_dict[name]}', output, quiet=False)

        self.model.load_state_dict(torch.load(output))
        self.model.to(device)
        self.model.eval()

```

```

def save(self, name: str):
    torch.save(self.model.state_dict(), f'/kaggle/working/exp_1/{name}.txt')

def predict_on_batch(self, batch: np.ndarray) -> np.ndarray:
    # todo: add code for batch mask prediction here
    X_batch = batch

    shape = X_batch.shape
    X_add = np.zeros((shape[0], shape[1], shape[2], (shape[3] // 3) * extra_channels + more_channels))
    for i in range(X_batch.shape[0]):
        X_add[i] = expand_channels(X_batch[i])
    X_add = torch.Tensor(X_add).permute(0, 3, 1, 2).type(torch.FloatTensor)
    X_batch = torch.Tensor(X_batch).permute(0, 3, 1, 2).type(torch.FloatTensor)
    X_batch = torch.cat([X_batch, X_add], 1)

    logits = self.model(X_batch.to(device))
    return logits[:, :, 0].reshape((self.batch_s, self.shape[0], self.shape[1])).detach().cpu().numpy()

def _predict_prob_on_clip(self, clip: np.ndarray) -> np.ndarray:
    print('doing predictions')
    n_frames = clip.shape[0]
    # --- get stacks ---
    stacks = []
    for i in range(n_frames - self.stack_s + 1):
        stack = clip[i : i + self.stack_s, ...]
        stack = np.squeeze(np.split(stack, self.stack_s, axis=0))
        stack = np.concatenate(stack, axis=-1)
        stacks.append(stack)
    # --- round to batch size ---
    add_stacks = 0
    while len(stacks) % self.batch_s != 0:
        stacks.append(stacks[-1])
        add_stacks += 1
    # --- group into batches ---
    batches = []
    for i in range(len(stacks) // self.batch_s):
        batch = np.stack(stacks[i * self.batch_s : (i + 1) * self.batch_s])
        batches.append(batch)
    stacks.clear()
    # --- perform predictions ---
    predictions = []
    for batch in batches:
        pred = np.squeeze(self.predict_on_batch(batch))
        predictions.append(pred)
    # --- crop back to source length ---
    predictions = np.concatenate(predictions, axis=0)
    if (add_stacks > 0):
        predictions = predictions[:-add_stacks, ...]
    batches.clear()
    # --- add (stack_s - 1) null frames at the begining ---
    start_frames = np.zeros((stack_s - 1, predictions.shape[1], predictions.shape[2]), dtype=np.float32)
    predictions = np.concatenate((start_frames, predictions), axis=0)
    print('predictions are made')
    return predictions

def get_labels_from_prediction(self, pred_prob: np.ndarray, upscale_coords: bool) -> np.ndarray:
    # todo: get ball coordinates from predicted masks
    # remember to upscale predicted coords if you use downscaled images

    if self.downscale:
        shape = (360, 640)
    else:
        shape = (720, 1280)

    n_frames = pred_prob.shape[0]
    coords = np.zeros([n_frames, 3])
    for i in range(n_frames):
        argmax = pred_prob[i].argmax()

        l = np.round(np.max(pred_prob[i]))
        coords[i, 2] = argmax // shape[1]
        coords[i, 1] = argmax % shape[1]
        coords[i, 0] = 1

    if upscale_coords:
        coords *= 2
    coords = regularise(coords)

    return coords

def predict(self, clip: np.ndarray, upscale_coords=True) -> np.ndarray:
    prob_pr = self._predict_prob_on_clip(clip)

```

```

labels_pr = self.get_labels_from_prediction(prob_pr, upscale_coords)
return labels_pr, prob_pr

def test(self, data_path: Path, games: List[int], do_visualization=False, test_name='test'):
    game_clip_pairs = get_game_clip_pairs(data_path, games)
    SIBATRACC_vals = []
    for game, clip in game_clip_pairs:
        data = load_clip_data(data_path, game, clip, downscale=self.downscale)
        if do_visualization:
            data_full = load_clip_data(data_path, game, clip, downscale=False) if self.downscale else data
            labels_gt = load_clip_labels(data_path, game, clip, downscale=False)
            labels_pr, prob_pr = self.predict(data)
            SIBATRACC_per_frame, SIBATRACC_total = Metrics.evaluate_predictions(labels_gt, labels_pr)
            print(SIBATRACC_total)
            SIBATRACC_vals.append(SIBATRACC_total)
        if do_visualization:
            visualize_prediction(data_full, labels_pr, self.out_path, f'{test_name}_g{game}_c{clip}', SIBATRACC_per_frame)
            visualize_prob(data, prob_pr, self.out_path, f'{test_name}_g{game}_c{clip}')
            del data_full
        del data, labels_gt, labels_pr, prob_pr
        gc.collect()
    SIBATRACC_final = sum(SIBATRACC_vals) / len(SIBATRACC_vals)
    return SIBATRACC_final

def evaluate(self, val_gen, loss_fn, shape):

    losses = []
    acc = []

    batch_size = self.batch_s

    for i, batch in enumerate(val_gen(batch_size)):

        if i >= 40:
            break

        X_batch, y_batch = prepare_batch(batch)

        with torch.no_grad():
            logits = self.model(X_batch.to(device))
            loss = loss_fn(logits, y_batch)

            losses.append(loss.item())

            model_answers = self.get_labels_from_prediction(logits[:, :, 0].detach().cpu().numpy(), False)
            _, accuracy = Metrics.evaluate_predictions(self.get_labels_from_prediction(y_batch.flatten(1, 2)[ :, :, 0].numpy(), False)

            acc.append(accuracy)

    return np.mean(acc), np.mean(losses)

def train(self, train_gen, val_gen, autoencoder=True):

    n_epoch = 10
    eval_every = 40
    epoch_len = 200

    shape = None
    if self.downscale:
        shape = (360, 640)
    else:
        shape = (720, 1280)

    optim = torch.optim.Adamax
    loss_fn = None
    if autoencoder:
        loss_fn = nn.MSELoss()
    else:
        loss_fn = MyLoss(shape, self.batch_s)

    learning_rate = 1e-3

    optimizer = optim(self.model.parameters(), lr=learning_rate)

    augmenting_transforms = v2.Compose([
        v2.ColorJitter(brightness=0.12, contrast=0.3)
    ])

    history = ''

```

```

losses = []
acc = []
batch_size = self.batch_s

for epoch in range(n_epoch):

    print("Epoch:", epoch+1)
    history += f"Epoch: {epoch+1}\n"

    self.model.train(True)

    for i, batch in enumerate(train_gen(batch_size)):

        if i >= epoch_len:
            break

        if autoencoder:

            X_batch, _ = prepare_batch(batch)

            logits = self.model(X_batch.to(device))
            loss = loss_fn(logits, X_batch.to(device))

            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

            train_accuracy = 0.

        else:

            X_batch, y_batch = prepare_batch(batch)
            logits = self.model(X_batch.to(device))

            loss = loss_fn(logits, y_batch)

            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

            model_answers = self.get_labels_from_prediction(logits[:, :, 0].detach().cpu().numpy(), False)
            _, accuracy = Metrics.evaluate_predictions(self.get_labels_from_prediction(y_batch.flatten(1, 2)[ :, :, 0].numpy(), Fa

        losses.append(loss.item())
        acc.append(accuracy)

        if (i + 1) % eval_every == 0:
            history += f"Loss: {np.mean(losses)} Acc: {np.mean(acc)} \n"
            print(f"Loss: {np.mean(losses)} Acc: {np.mean(acc)}")

    self.model.train(False)

    val_accuracy, val_loss = self.evaluate(val_gen, loss_fn, shape)

    X, y = prepare_batch(next(train_gen(self.batch_s)))

    with torch.no_grad():

        logits = model.model(X.to(device))
        logits = logits[0, :, 0].detach().cpu().reshape(self.shape)

        figure, axis = plt.subplots(2, 2, figsize=(16,10))

        IPython.display.clear_output(wait=True)

        axis[0, 0].imshow(y[0, :, :, 0])
        axis[0, 1].imshow(logits)
        axis[1, 0].imshow((X[0, 6:9, :, :] / 255).permute(1, 2, 0))
        plt.show()

        history += f"Epoch {epoch+1}/{n_epoch}: Loss: {val_loss} Acc: {val_accuracy} \n"
        print(history)

    print(f'training done')

for i, layer in enumerate(model.model.children()):
    print(i, ":", layer)
    if i < 3:
        for param in layer.parameters():
            param.requires_grad = False
    else:

```

```

        for param in layer.parameters():
            param.requires_grad = True

X, y = prepare_batch(train_gen.get_random_batch(1))
figure, axis = plt.subplots(6, 3, figsize=(16,20))
for i in range(16):
    axis[i // 3, i % 3].imshow(X[0, i, :, :], cmap='gray')

model.save('best')

model.load('best', from_disk=False)

```

Пример пайплайна для обучения модели:

```

batch_s = 4
stack_s = 3
downscale = True

output_path = prepare_experiment(Path('/kaggle/working'))

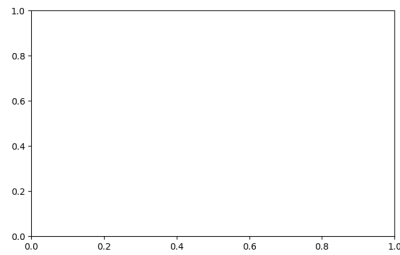
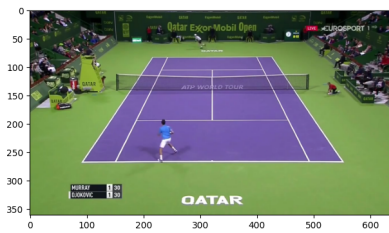
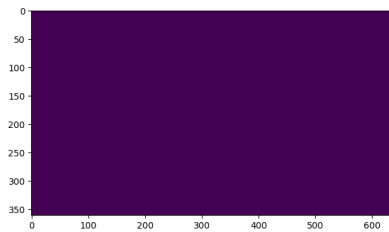
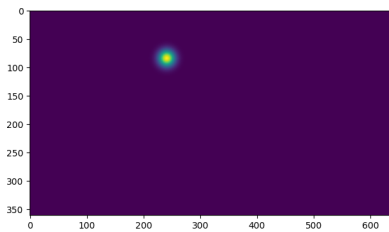
train_gen = DataGenerator(Path('/kaggle/input/train'), [1, 5, 3, 4, 6], stack_s=stack_s, downscale=downscale, pool_s=10, pool_update_s=4)
val_gen = DataGenerator(Path('/kaggle/input/train'), [2], stack_s=stack_s, downscale=downscale, pool_s=4, pool_update_s=2, quiet=True)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SuperTrackingModel(batch_s, stack_s, out_path=output_path, downscale=downscale)

model.model.autoencoder = False

model.train(train_gen.random_g, val_gen.random_g, autoencoder=False)
#model.save('best')

```



Epoch: 1

Loss: 0.3543236811645329 Acc: 0.8214644660940673

Loss: 0.28092161738313737 Acc: 0.8607322330470335

Loss: -1.688036478481566 Acc: 0.8671548220313559

Loss: -1.202464579686057 Acc: 0.8659911165235169

Loss: -2.0945697473175824 Acc: 0.871835786437627

Epoch 1/20: Loss: 0.5610452015884221 Acc: 0.72875

Epoch: 2

Loss: -1.7038529750037317 Acc: 0.8731964886980225

Loss: -2.2649542050768754 Acc: 0.8747041331697335

Loss: -3.429519001947483 Acc: 0.8800536165235169

Loss: -3.0228675352823404 Acc: 0.8810437109202446

Loss: -2.697432678034529 Acc: 0.882335786437627

Epoch 2/20: Loss: 0.456254833313316 Acc: 0.77125

Epoch: 3

Loss: -2.4323164445390417 Acc: 0.8828052603978426

Loss: -2.2131695347566467 Acc: 0.8859226942058614

Loss: -2.027599595540848 Acc: 0.8870055638823335

Loss: -1.868980029850666 Acc: 0.8895561997546003

Loss: -1.737550807725638 Acc: 0.8931691197709603

Epoch 3/20: Loss: -5.516117243748158 Acc: 0.8275

Epoch: 4

Loss: -1.9831823664222612 Acc: 0.8942210497852752

Loss: -2.200182966367506 Acc: 0.8955903997979061

Loss: -2.0719280709745362 Acc: 0.8988214886980223

Loss: -2.2607816237809235 Acc: 0.8995414103454948

Loss: -2.1450934308674188 Acc: 0.9032518398282201

Epoch 4/20: Loss: 0.35525051057338713 Acc: 0.8479289321881346

Epoch: 5

Loss: -2.3154799757636195 Acc: 0.904525561741162

Loss: -2.4680356057953428 Acc: 0.904603945298382

Loss: -2.609350635058692 Acc: 0.9062515998506263

Loss: -2.487741583007543 Acc: 0.9038036165235168

Loss: -3.3159862634819 Acc: 0.902401471862576

Epoch 5/20: Loss: -34.963067554309966 Acc: 0.8952144660940672

Epoch: 6

Loss: -3.857209843216249 Acc: 0.9020288947176336

Loss: -4.570213992248669 Acc: 0.8985741208392026

Loss: -4.608133614495663 Acc: 0.8985880617411621

Loss: -4.8452450369837985 Acc: 0.897997592925745

Loss: -4.6761763814960915 Acc: 0.8979810064948868

Epoch 6/20: Loss: -5.581751419603824 Acc: 0.8791789321881346

Epoch: 7

Loss: -4.8960055921407 Acc: 0.8976164234527335

Loss: -4.737509267448331 Acc: 0.8981987227198356

Loss: -4.765691896039767 Acc: 0.8982533068798406

Loss: -4.963979759431608 Acc: 0.8982006939155531

Loss: -5.6509654328533045 Acc: 0.8966449598036801

Epoch 7/20: Loss: -23.248455607984216 Acc: 0.9112500000000001

Epoch: 8

Loss: -5.648902583834974 Acc: 0.8957659331424667

Loss: -5.487603814974175 Acc: 0.8940618394384561

Loss: -5.799465818912722 Acc: 0.8935658559293931

Loss: -5.792104592681743 Acc: 0.891941461318231

Loss: -5.639543488976779 Acc: 0.8909554247852752

Epoch 8/20: Loss: -40.61218493282795 Acc: 0.8989644660940673

Epoch: 9

Loss: -5.7812308576196525 Acc: 0.8902613900344148

Loss: -5.7781301460905174 Acc: 0.8905927600856939

Loss: -5.774276926564486 Acc: 0.890375486595329

Loss: -5.770380931257651 Acc: 0.8894010437181624

Loss: -6.028559394247002 Acc: 0.8897199094133142

Epoch 9/20: Loss: 0.2894818698056042 Acc: 0.8550000000000001

Epoch: 10

Loss: -6.274958742037415 Acc: 0.8898393562976785

Loss: -6.509709978945791 Acc: 0.8891246891424088

Loss: -6.491447603148602 Acc: 0.8890647997852753

Loss: -6.5928272129473635 Acc: 0.8885780582813729

```

Loss: -6.693571846424602 Acc: 0.8898064971157456
Epoch 10/20: Loss: -34.9487230528146 Acc: 0.8889644660940672
Epoch: 11
Loss: -6.67334790355252 Acc: 0.8900798991330838
Loss: -6.655109327420807 Acc: 0.8908997946515643
Loss: -6.747090727744519 Acc: 0.8911894211675726
Loss: -6.617140807005301 Acc: 0.890935913368173
Loss: -6.493336900494654 Acc: 0.891168896761479
Epoch 11/20: Loss: -29.06926059545949 Acc: 0.8927144660940673
Epoch: 12
Loss: -6.373237858312703 Acc: 0.8911033807478812
Loss: -6.2570015126308265 Acc: 0.8911059530154622
Loss: -6.145676353222145 Acc: 0.8911983411719898
Loss: -6.2361316912689135 Acc: 0.8910158088163314
Loss: -6.130126679642126 Acc: 0.8918738786693924
Epoch 12/20: Loss: -29.185416036238895 Acc: 0.9264644660940672
Epoch: 13
Loss: -6.315241675021616 Acc: 0.8923554544289107
Loss: -6.211774372170499 Acc: 0.8934868180671539
Loss: -6.2976925689268795 Acc: 0.894483059050215
Loss: -6.3802688754884 Acc: 0.8952137925025554
Loss: -6.370536608077729 Acc: 0.8961335803102084
Epoch 13/20: Loss: 0.3324136408045888 Acc: 0.8377144660940672
Epoch: 14
Loss: -6.432864679970468 Acc: 0.897044435153993
Loss: -6.422444547653726 Acc: 0.8977415331367692
Loss: -6.4111839009579 Acc: 0.897762671358113
Loss: -6.486044015233284 Acc: 0.8981786393977644
Loss: -6.391094959072237 Acc: 0.898582722636283
Epoch 14/20: Loss: -5.542278569471091 Acc: 0.8502144660940673
Epoch: 15
Loss: -6.382537692071031 Acc: 0.8994155640934349
Loss: -6.5849270780932985 Acc: 0.8998086043989992
Loss: -6.492612336789679 Acc: 0.9002906783113419
Loss: -6.79936590272669 Acc: 0.9007935069828102
Loss: -6.706986922483892 Acc: 0.9011357864376268
Epoch 15/20: Loss: 0.2442538533359766 Acc: 0.8864644660940673
Epoch: 16
Loss: -6.694147966570515 Acc: 0.9017787366160791
Loss: -6.681830130688024 Acc: 0.9024049867898963
Loss: -6.670785735160089 Acc: 0.903319666446436
Loss: -6.585108786184765 Acc: 0.9040529618078736
Loss: -6.648072134314571 Acc: 0.9045491747852752
Epoch 16/20: Loss: -11.473455048166215 Acc: 0.8814644660940673
Epoch: 17
Loss: -6.70897969755482 Acc: 0.9047862220101485
Loss: -6.767902987944985 Acc: 0.9048829079136107
Loss: -6.685565685988572 Acc: 0.9057126319146516
Loss: -6.6040670704115405 Acc: 0.9059200767770106

```

Пример пайплайна для тестирования обученной модели:

```

Epoch: 18
output_path = prepare_experiment(Path('/kaggle/working'))
print(output_path)
new_model = SuperTrackingModel(batch_s, stack_s, out_path=output_path, downscale=downscale)
new_model.load()
sibatracc_final = new_model.test(Path('../input/test/'), [1, 2], do_visualization=False, test_name='test')
print(f'SiBaTrAcc final value: {sibatracc_final}')

```

```

/kaggle/working/exp_25
Downloading...
From: https://drive.google.com/uc?id=1ZkKw3WUuf3g56fXHHYwJkZxGIjURF1Au
To: /kaggle/working/best.npz
100%|██████████| 3.48M/3.48M [00:00<00:00, 167MB/s]
loading clip data (game 1, clip 1) downscaled
loading clip labels (game 1, clip 1)
doing predictions
predictions are made
0.9197789529788236
loading clip data (game 1, clip 2) downscaled
loading clip labels (game 1, clip 2)
doing predictions
predictions are made
0.9444083331668495
loading clip data (game 1, clip 3) downscaled
loading clip labels (game 1, clip 3)
doing predictions
predictions are made
0.8888888888888888
/tmp/ipykernel_161/3020300260.py:20: RuntimeWarning: invalid value encountered in scalar divide
  distance = abs(A * x0 + B * y0 + C) / math.sqrt(A**2 + B**2)
loading clip data (game 1, clip 4) downscaled
loading clip labels (game 1, clip 4)
doing predictions
predictions are made
0.9037473144467005
loading clip data (game 1, clip 5) downscaled
loading clip labels (game 1, clip 5)
doing predictions

```



```

predictions are made
0.9481342580359733
loading clip data (game 1, clip 6) downscaled
loading clip labels (game 1, clip 6)
doing predictions
predictions are made
0.9452012476412062
loading clip data (game 1, clip 7) downscaled
loading clip labels (game 1, clip 7)
doing predictions
predictions are made
0.816054574471168
loading clip data (game 1, clip 8) downscaled
loading clip labels (game 1, clip 8)
doing predictions
predictions are made
0.9593889727911699
loading clip data (game 2, clip 1) downscaled
loading clip labels (game 2, clip 1)
doing predictions
predictions are made
0.9425821033138646
loading clip data (game 2, clip 2) downscaled
loading clip labels (game 2, clip 2)
doing predictions
predictions are made
0.9264895913761657
loading clip data (game 2, clip 3) downscaled

new_model.save('cabbages')
# 0.9124608824397608

```

Во время самостоятельного тестирования попробуйте хотя бы раз сделать тестирование с визуализацией (`do_visualization=True`), чтобы визуально оценить качество трекинга разработанной моделью.

Загрузка модели через функцию `load` должна происходить полностью автоматически без каких-либо действий со стороны пользователя! Один из вариантов подобной реализации с использованием `google drive` и пакета `gdown` приведен в разделе с дополнениями.

▼ Дополнения

Иногда при записи большого количества файлов в `output` директорию `kaggle` может "тупить" и не отображать корректно структуру дерева файлов в `output` и не показывать кнопки для скачивания выбранного файла. В этом случае удобно будет запаковать директорию с экспериментом и выкачать ее вручную. Пример для выкачивания директории с первым экспериментом приведен ниже:

```

%cd /kaggle/working/
!zip -r "exp_24.zip" "exp_24"
from IPython.display import FileLink
FileLink(r'exp_24.zip')

```

```

/kaggle/working
adding: exp_24/ (stored 0%)
adding: exp_24/test_g1_c6.mp4 (deflated 0%)
adding: exp_24/test_g1_c4.mp4 (deflated 0%)
adding: exp_24/test_g1_c2.mp4 (deflated 0%)
adding: exp_24/test_g2_c5.mp4 (deflated 0%)
adding: exp_24/test_g1_c3_prob.mp4 (deflated 1%)
adding: exp_24/test_g2_c8.txt (deflated 76%)
adding: exp_24/test_g1_c1_prob.mp4 (deflated 1%)
adding: exp_24/test_g1_c7.mp4 (deflated 0%)
adding: exp_24/test_g1_c8.txt (deflated 78%)
adding: exp_24/test_g1_c8.mp4 (deflated 0%)
adding: exp_24/test_g2_c8.mp4 (deflated 0%)
adding: exp_24/test_g2_c6.txt (deflated 76%)
adding: exp_24/test_g2_c3.txt (deflated 78%)
adding: exp_24/test_g2_c9_prob.mp4 (deflated 2%)
adding: exp_24/test_g2_c7_prob.mp4 (deflated 1%)
adding: exp_24/test_g2_c2.mp4 (deflated 0%)
adding: exp_24/test_g2_c7.txt (deflated 77%)
adding: exp_24/test_g2_c6_prob.mp4 (deflated 1%)
adding: exp_24/test_g2_c8_prob.mp4 (deflated 1%)
adding: exp_24/test_g2_c4.txt (deflated 78%)

```

удалить лишние директории или файлы в output тоже легко:

```

adding: exp_24/test_g2_c4.txt (deflated 78%)
...
!rm -r /kaggle/working/exp_1
adding: exp_24/test_g2_c7.mp4 (deflated 0%)
...

```

Для реализации загрузки данных рекомендуется использовать облачное хранилище google drive и пакет gdown для скачивания файлов. Пример подобного использования приведен ниже:

1. загружаем файл в google drive (в данном случае, это npz архив, содержащий один numpy массив по ключу 'w')
2. в интерфейсе google drive открываем доступ на чтение к файлу по ссылке и извлекаем из ссылки id файла
3. формируем url для скачивания файла
4. с помощью gdown скачиваем файл
5. распаковываем npz архив и пользуемся numpy массивом

Обратите внимание, что для корректной работы нужно правильно определить id файла. В частности, в ссылке https://drive.google.com/file/d/1kZ8CC-zfkB_TlwtBjuPcEfsPV0Jz7IPA/view?usp=sharing id файла заключен между ...d/ b /view?... и равен 1kZ8CC-zfkB_TlwtBjuPcEfsPV0Jz7IPA

```

adding: exp_24/test_g1_c6_prob.mp4 (deflated 1%)
import gdown

id = '1kZ8CC-zfkB_TlwtBjuPcEfsPV0Jz7IPA'
url = f'https://drive.google.com/uc?id={id}'
output = 'sample-weights.npz'
gdown.download(url, output, quiet=False)

import numpy as np

weights = np.load('/kaggle/working/sample-weights.npz')['w']
print(weights)

```