

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/tennistrackingassignment>. После загрузки данные датасета будут примонтированы в ../input/tennistrackingassignment.

Установка и импорт зависимостей

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

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

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...

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

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)
(3.2.0)
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)
(1.7.1)
```

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 gc
import time
import random
import csv

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.24.3)
```

```
warnings.warn(f"A NumPy version >={np_minversion} and  
<{np_maxversion}")
```

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 `npz` архивов, при последующем обращении к таким клипам происходит загрузка `npz` 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) -> None:
        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, pool_update_s=4,
quiet=False)
for i in range(10):
    imgs, masks = train_gen.get_random_batch(batch_s)
    print(imgs.shape, imgs.dtype, masks.shape, masks.dtype)

loading clip data (game 4, clip 15) downscaled
loading clip labels (game 4, clip 15)
loading clip data (game 1, clip 1) downscaled
loading clip labels (game 1, clip 1)
loading clip data (game 2, clip 3) downscaled
loading clip labels (game 2, clip 3)
loading clip data (game 2, clip 8) downscaled
loading clip labels (game 2, clip 8)
loading clip data (game 2, clip 7) downscaled
loading clip labels (game 2, clip 7)
loading clip data (game 1, clip 9) downscaled
loading clip labels (game 1, clip 9)
loading clip data (game 1, clip 11) downscaled
loading clip labels (game 1, clip 11)
loading clip data (game 3, clip 1) downscaled
loading clip labels (game 3, clip 1)
loading clip data (game 1, clip 2) downscaled
loading clip labels (game 1, clip 2)
loading clip data (game 1, clip 4) downscaled
loading clip labels (game 1, clip 4)

```

```

(4, 360, 640, 9) uint8 (4, 360, 640) float32
(4, 360, 640, 9) uint8 (4, 360, 640) float32
(4, 360, 640, 9) uint8 (4, 360, 640) float32
(4, 360, 640, 9) uint8 (4, 360, 640) float32
(4, 360, 640, 9) uint8 (4, 360, 640) float32
(4, 360, 640, 9) uint8 (4, 360, 640) float32
(4, 360, 640, 9) uint8 (4, 360, 640) float32
(4, 360, 640, 9) uint8 (4, 360, 640) float32
(4, 360, 640, 9) uint8 (4, 360, 640) float32
(4, 360, 640, 9) uint8 (4, 360, 640) float32

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_s=4,
quiet=False)
stack, mask = train_gen.get_random_stack()
print(stack.shape, mask.shape)

for i in range(stack_s):
    plt.figure()
    plt.imshow(stack[:, :, 0: 3])

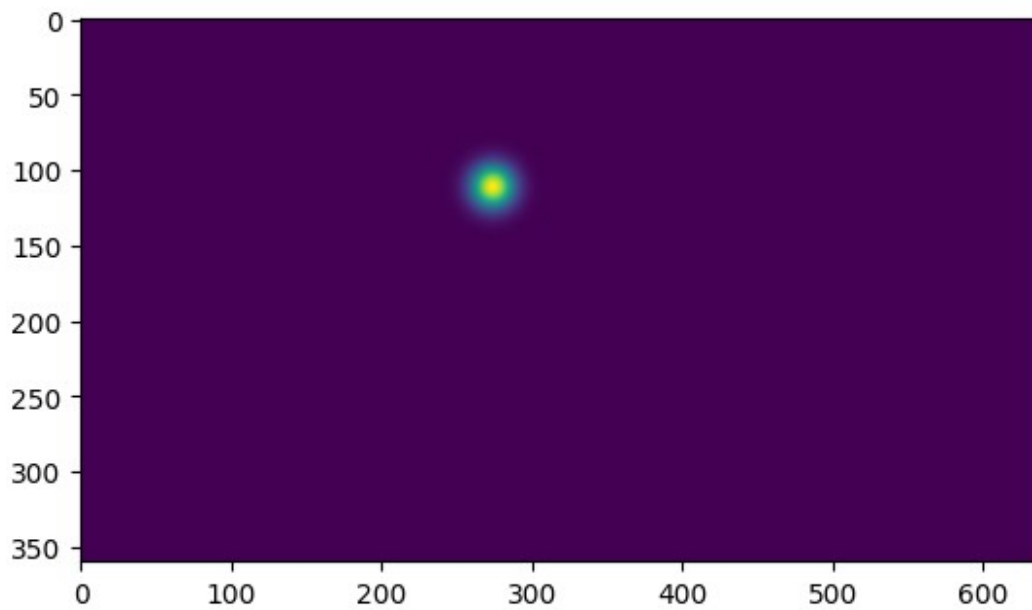
loading clip data (game 1, clip 6) downscaled
loading clip labels (game 1, clip 6)
loading clip data (game 1, clip 10) downscaled
loading clip labels (game 1, clip 10)
loading clip data (game 1, clip 1) downscaled
loading clip labels (game 1, clip 1)
loading clip data (game 1, clip 8) downscaled
loading clip labels (game 1, clip 8)
loading clip data (game 1, clip 13) downscaled
loading clip labels (game 1, clip 13)
loading clip data (game 1, clip 2) downscaled
loading clip labels (game 1, clip 2)
loading clip data (game 1, clip 9) downscaled
loading clip labels (game 1, clip 9)
loading clip data (game 1, clip 11) downscaled
loading clip labels (game 1, clip 11)
loading clip data (game 1, clip 3) downscaled
loading clip labels (game 1, clip 3)
loading clip data (game 1, clip 4) downscaled
loading clip labels (game 1, clip 4)
(360, 640, 9) (360, 640)

```





```
plt.imshow(mask)
#plt.imshow(stack[:, :, 3 * i: 3 * i + 3])
<matplotlib.image.AxesImage at 0x7d31a8504fa0>
```



```
mask.shape
mask[244, 561]
flat_index = np.argmax(mask)
```

```
# Преобразуем плоский индекс в координаты
max_coordinates = np.unravel_index(flat_index, mask.shape)
max_coordinates

(112, 274)
```

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 аккумулятированных 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 маске. 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_batch 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 numpy 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!

```
import matplotlib.pyplot as plt
import torch.nn.functional as F
import torch.optim as optim
import torch.nn as nn
import numpy as np

import torch
import glob
import cv2
import os

from torch.utils.data import Dataset, DataLoader

import torchvision.transforms as transforms
import torchvision.models as models
import torch.utils.data as data
import torch.utils.data as data

import albumentations as A
import albumentations.pytorch

import gdown

from torchvision.models import resnet18, resnet34

import torch
import torch.nn as nn

def double_conv(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1,
```



```

padding=1),
    nn.BatchNorm2d(out_channels),
    nn.ReLU(inplace=True),
    nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1,
padding=1),
    nn.BatchNorm2d(out_channels),
    nn.ReLU(inplace=True)
)

def up_conv(in_channels, out_channels):
    return nn.ConvTranspose2d(
        in_channels, out_channels, kernel_size=(1,2), stride=2
    )

def up_conv2(in_channels, out_channels):
    return nn.ConvTranspose2d(
        in_channels, out_channels, kernel_size=(2,2), stride=2
    )

class SResUnet(nn.Module):
    """Shallow Unet with ResNet18 or ResNet34 encoder.
    """

    def __init__(self, encoder, *, pretrained=True, out_channels=1):
        super().__init__()
        self.first_layer = nn.Conv2d(9, 3, kernel_size=1, stride=1,
padding=0)
        self.encoder = encoder(pretrained=pretrained)
        self.encoder_layers = list(self.encoder.children())

        self.block1 = nn.Sequential(*self.encoder_layers[:3])
        self.block2 = nn.Sequential(*self.encoder_layers[3:5])
        self.block3 = self.encoder_layers[5]
        self.block4 = self.encoder_layers[6]
        self.block5 = self.encoder_layers[7]

        self.up_conv6 = up_conv(512, 512)
        self.conv6 = double_conv(512 + 256, 512)
        self.up_conv7 = up_conv(512, 256)
        self.conv7 = double_conv(256 + 128, 256)
        self.up_conv8 = up_conv2(256, 128)
        self.conv8 = double_conv(128 + 64, 128)
        self.up_conv9 = up_conv2(128, 64)
        self.conv9 = double_conv(64 + 64, 64)
        self.up_conv10 = up_conv2(64, 32)
        self.conv10 = nn.Conv2d(32, out_channels, kernel_size=1)

        if not pretrained:

```

```

        self._weights_init()

    def _weights_init(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out',
nonlinearity='relu')
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

    def forward(self, x):
        block1 = self.first_lay(x)
        block1 = self.block1(block1)
        block2 = self.block2(block1)
        block3 = self.block3(block2)
        block4 = self.block4(block3)
        block5 = self.block5(block4)

        x = self.up_conv6(block5)

        x = torch.cat([x, block4], dim=1)
        x = self.conv6(x)

        x = self.up_conv7(x)
        x = torch.cat([x, block3], dim=1)
        x = self.conv7(x)

        x = self.up_conv8(x)

        x = torch.cat([x, block2], dim=1)
        x = self.conv8(x)

        x = self.up_conv9(x)

        x = torch.cat([x, block1], dim=1)
        x = self.conv9(x)

        x = self.up_conv10(x)
        x = self.conv10(x)

        return x

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = SResUnet(resnet34)
model.to(device)

tmp = np.ndarray((4, 9, 360, 640))

```

```

images = torch.tensor(tmp, dtype=torch.float32).to(device)
print(images.shape)
print(model(images).size())

Downloading: "https://download.pytorch.org/models/resnet34-
b627a593.pth" to /root/.cache/torch/hub/checkpoints/resnet34-
b627a593.pth
100%|██████████| 83.3M/83.3M [00:01<00:00, 61.7MB/s]

torch.Size([4, 9, 360, 640])
torch.Size([4, 1, 360, 640])

import torchvision.transforms.v2 as A

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
tmp_a = 0

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
        self.model = SResUnet(resnet34)
        self.model.to(device)

    def save(self):
        with open('/kaggle/working/checkpoint.pth', "wb") as fp:
            torch.save(self.model.state_dict(), fp)

    def load(self):
        # todo: add code for loading model here
        name_to_id_dict = {
            'best': '1_R4wSql8UpXICK_nyyXdx0mxhMM6CuDQ'
        }
        output = 'checkpoint2.pth'
        gdown.download(f'https://drive.google.com/uc?
id=1_R4wSql8UpXICK_nyyXdx0mxhMM6CuDQ', output, quiet=False)
        with open('/kaggle/working/checkpoint2.pth', "rb") as fp:
            state_dict = torch.load(fp)
            self.model.load_state_dict(state_dict)

    def predict_on_batch(self, batch: np.ndarray) -> np.ndarray:
        self.model.eval()
        # todo: add code for batch mask prediction here
        #print(batch.shape[:3])
        #print(batch.shape)
        batch1 = batch

```

```

        #batch2 = batch[:, :, :, 3:6]
        #batch3 = batch[:, :, :, 6:9]

        batch1 = torch.tensor(batch1, dtype=torch.float32).permute(0,
3, 1, 2).to(device)
        #batch2 = torch.tensor(batch2, dtype=torch.float32).permute(0,
3, 1, 2).to(device)
        #batch3 = torch.tensor(batch3, dtype=torch.float32).permute(0,
3, 1, 2).to(device)

        model_answers1 = self.model(batch1)
        #model_answers2 = self.model(batch2)
        #model_answers3 = self.model(batch3)

        #tensor_list = [model_answers1, model_answers2,
model_answers3]
        #result_tensor = torch.cat((model_answers1, model_answers2,
model_answers3), dim=1)
        #print(result_tensor.shape)
        return model_answers1.cpu().detach().numpy()
        #return torch.mean(torch.stack(tensor_list),
dim=0).cpu().detach().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 beginning ---
    start_frames = np.zeros((self.stack_s - 1,
predictions.shape[1], predictions.shape[2]), dtype=np.float32)
#         print("QQQQQ")
#         print(start_frames.shape)
#         print(predictions.shape)
    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
    n_frames = pred_prob.shape[0]
    coords = np.zeros([n_frames, 3])
    #print(n_frames)
    for i in range(n_frames):
        curr_mask = pred_prob[i]
        #print(curr_mask)
        if len(np.unique(curr_mask)) > 1:
            coords[i, 0] = 1
            flat_index = np.argmax(curr_mask)
            tmp_a = curr_mask
            #print(flat_index)
            # Преобразуем плоский индекс в координаты
            max_coordinates = np.unravel_index(flat_index,
curr_mask.shape)
            #print(curr_mask.shape)
            coords[i] = [1, max_coordinates[1] + 1,
max_coordinates[0] + 8]
            #print(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=True) if self.downscale else data
                labels_gt = load_clip_labels(data_path, game, clip,
downscale=True)
                labels_pr, prob_pr = self.predict(data)
                SIBATRACC_per_frame, SIBATRACC_total =
Metrics.evaluate_predictions(labels_gt, labels_pr)
                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 train(self, param_1=None, param_2=None, param_3=None,
param_4=None, param_5=None, param_6=None):
        # todo: implement model training here
        print('Running stub for training model...')
        optimizer = optim.Adam(self.model.parameters(), lr=3e-4)
        criterion = nn.BCEWithLogitsLoss()

        for epoch in range(10):
            self.model.train()

            cnt = 0
            total_loss = 0
            for batch_train, batch_answers in param_1(3):
                batch_train = torch.tensor(batch_train,
dtype=torch.float32)
                #print(batch_train.shape)
                batch_answers = torch.tensor(batch_answers,

```

```

dtype=torch.float32)
    batch_train = batch_train.to(device)
    batch_answers = batch_answers.to(device)

    batch_answers = batch_answers.unsqueeze(1)
    batch_train = batch_train.permute(0, 3, 1, 2)
    #batch_train1 = batch_train[:, 0:3]
    #batch_train2 = batch_train[:, 3:6]
    #batch_train3 = batch_train[:, 6:9]

    optimizer.zero_grad()
    model_answers = self.model(batch_train)
    #model_answers2 = self.model(batch_train2)
    #model_answers3 = self.model(batch_train3)

    #tensor_list = [model_answers, model_answers2,
model_answers3]
    new_loss = criterion(model_answers, batch_answers)
    new_loss.backward()
    optimizer.step()
    cnt += 1
    total_loss += new_loss
    if cnt % 50 == 0:
        print(total_loss/50)
        total_loss = 0

    self.model.eval()
    print('training done.')

batch_s = 4
stack_s = 3
downscale = True
output_path = prepare_experiment(Path('/kaggle/working/'))
train_gen =
DataGenerator(Path('../input/tennistackingassignment/train/'), [1, 2,
3], stack_s=stack_s, downscale=True, pool_s=10, pool_update_s=4,
quiet=False)
val_gen =
DataGenerator(Path('../input/tennistackingassignment/train/'), [4],
stack_s=stack_s, downscale=True, pool_s=4, pool_update_s=2,
quiet=False)

my_model = SuperTrackingModel(batch_s, stack_s, out_path=output_path,
downscale=downscale)
#my_model.save('check')

my_model.train(train_gen.random_g, val_gen.random_g)

```

```

Running stub for training model...
tensor(0.5030, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.1381, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0478, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0288, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0213, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0160, device='cuda:0', grad_fn=<DivBackward0>)
loading clip data (game 3, clip 3) downscaled
loading clip labels (game 3, clip 3)
loading clip data (game 1, clip 5) downscaled
loading clip labels (game 1, clip 5)
loading clip data (game 3, clip 6) downscaled
loading clip labels (game 3, clip 6)
loading clip data (game 2, clip 6) downscaled
loading clip labels (game 2, clip 6)
tensor(0.0132, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0106, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0093, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0085, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0081, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0071, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0065, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0066, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0065, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0061, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0059, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0055, device='cuda:0', grad_fn=<DivBackward0>)
loading clip data (game 2, clip 2) downscaled
loading clip labels (game 2, clip 2)
loading clip data (game 2, clip 1) downscaled
loading clip labels (game 2, clip 1)
loading clip data (game 1, clip 4) downscaled
loading clip labels (game 1, clip 4)
loading clip data (game 3, clip 2) downscaled
loading clip labels (game 3, clip 2)
tensor(0.0059, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0059, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0056, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0055, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0056, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0054, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0057, device='cuda:0', grad_fn=<DivBackward0>)
tensor(0.0056, device='cuda:0', grad_fn=<DivBackward0>)

```

```

-----
-----
KeyboardInterrupt                                Traceback (most recent call
last)
Cell In[47], line 1
----> 1 my_model.train(train_gen.random_g, val_gen.random_g)

```



```

Cell In[45], line 174, in SuperTrackingModel.train(self, param_1,
param_2, param_3, param_4, param_5, param_6)
    169 #model_answers2 = self.model(batch_train2)
    170 #model_answers3 = self.model(batch_train3)
    171
    172 #tensor_list = [model_answers, model_answers2, model_answers3]
    173 new_loss = criterion(model_answers, batch_answers)
--> 174 new_loss.backward()
    175 optimizer.step()
    176 cnt += 1

```

```

File /opt/conda/lib/python3.10/site-packages/torch/_tensor.py:487, in
Tensor.backward(self, gradient, retain_graph, create_graph, inputs)
    477 if has_torch_function_unary(self):
    478     return handle_torch_function(
    479         Tensor.backward,
    480         (self,),
    (...),
    485         inputs=inputs,
    486     )
--> 487 torch.autograd.backward(
    488     self, gradient, retain_graph, create_graph, inputs=inputs
    489 )

```

```

File
/opt/conda/lib/python3.10/site-packages/torch/autograd/__init__.py:200
, in backward(tensors, grad_tensors, retain_graph, create_graph,
grad_variables, inputs)
    195     retain_graph = create_graph
    197 # The reason we repeat same the comment below is that
    198 # some Python versions print out the first line of a multi-
line function
    199 # calls in the traceback and some print out the last line
--> 200 Variable._execution_engine.run_backward( # Calls into the C++
engine to run the backward pass
    201     tensors, grad_tensors_, retain_graph, create_graph,
inputs,
    202     allow_unreachable=True, accumulate_grad=True)

```

KeyboardInterrupt:

```
my_model.save()
```

```
from IPython.display import FileLink
FileLink(r'checkpoint.pth')
```

```
/kaggle/working/checkpoint.pth
```

```

with open('checkpoint.pth', "rb") as fp:
    line = fp.readline()
    print(line)

with open('/kaggle/working/checkpoint3.pth', "wb") as fp:
    torch.save(model.state_dict(), fp)

```

Пример пайплайна для обучения модели:

```

batch_s = 4
stack_s = 3
downscale = True
output_path = prepare_experiment(Path('/kaggle/working/'))
train_gen =
DataGenerator(Path('../input/tennistackingassignment/train/'), [1, 2,
3], stack_s=stack_s, downscale=True, pool_s=10, pool_update_s=4,
quiet=False)
val_gen =
DataGenerator(Path('../input/tennistackingassignment/train/'), [4],
stack_s=stack_s, downscale=True, pool_s=4, pool_update_s=2,
quiet=False)

loading clip data (game 1, clip 4) downscaled
loading clip labels (game 1, clip 4)
loading clip data (game 3, clip 4) downscaled
loading clip labels (game 3, clip 4)
loading clip data (game 2, clip 7) downscaled
loading clip labels (game 2, clip 7)
loading clip data (game 1, clip 11) downscaled
loading clip labels (game 1, clip 11)
loading clip data (game 1, clip 2) downscaled
loading clip labels (game 1, clip 2)
loading clip data (game 3, clip 7) downscaled
loading clip labels (game 3, clip 7)
loading clip data (game 1, clip 7) downscaled
loading clip labels (game 1, clip 7)
loading clip data (game 2, clip 1) downscaled
loading clip labels (game 2, clip 1)
loading clip data (game 3, clip 2) downscaled
loading clip labels (game 3, clip 2)
loading clip data (game 1, clip 10) downscaled
loading clip labels (game 1, clip 10)
loading clip data (game 4, clip 5) downscaled
loading clip labels (game 4, clip 5)
loading clip data (game 4, clip 9) downscaled
loading clip labels (game 4, clip 9)
loading clip data (game 4, clip 11) downscaled
loading clip labels (game 4, clip 11)

```

```

loading clip data (game 4, clip 13) downscaled
loading clip labels (game 4, clip 13)

for a, b in train_gen.random_g(1):
    print(a.shape)
    print(b.shape)
    break

plt.imshow(a[:, :, 0: 3])

batch_s = 4
stack_s = 3
downscale = True

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

model = SuperTrackingModel(batch_s, stack_s, out_path=output_path,
downscale=downscale)

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

model.train(train_gen.random_g, val_gen.random_g)

```

Пример пайплайна для тестирования обученной модели:

```

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

Downloading...
From (uriginal): https://drive.google.com/uc?
id=1_R4wSql8UpXICK_nyyXdx0mxhMM6CuDQ
From (redirected): https://drive.google.com/uc?
id=1_R4wSql8UpXICK_nyyXdx0mxhMM6CuDQ&confirm=t&uuid=21eea622-8fd1-
433d-be34-a925359cad15
To: /kaggle/working/checkpoint2.pth
100%|██████████| 123M/123M [00:00<00:00, 177MB/s]

```

```
loading clip data (game 1, clip 1) downscaled
loading clip data (game 1, clip 1) downscaled
loading clip labels (game 1, clip 1)
doing predictions
predictions are made
performing clip visualization
loading clip data (game 1, clip 2) downscaled
loading clip data (game 1, clip 2) downscaled
loading clip labels (game 1, clip 2)
doing predictions
predictions are made
performing clip visualization
loading clip data (game 1, clip 3) downscaled
loading clip data (game 1, clip 3) downscaled
loading clip labels (game 1, clip 3)
doing predictions
predictions are made
performing clip visualization
loading clip data (game 1, clip 4) downscaled
loading clip data (game 1, clip 4) downscaled
loading clip labels (game 1, clip 4)
doing predictions
predictions are made
performing clip visualization
loading clip data (game 1, clip 5) downscaled
loading clip data (game 1, clip 5) downscaled
loading clip labels (game 1, clip 5)
doing predictions
predictions are made
performing clip visualization
loading clip data (game 1, clip 6) downscaled
loading clip data (game 1, clip 6) downscaled
loading clip labels (game 1, clip 6)
doing predictions
predictions are made
performing clip visualization
loading clip data (game 1, clip 7) downscaled
loading clip data (game 1, clip 7) downscaled
loading clip labels (game 1, clip 7)
doing predictions
predictions are made
performing clip visualization
loading clip data (game 1, clip 8) downscaled
loading clip data (game 1, clip 8) downscaled
loading clip labels (game 1, clip 8)
doing predictions
predictions are made
performing clip visualization
SiBaTrAcc final value: 0.8530131086620147
```

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_1.zip" "exp_1"  
from IPython.display import FileLink  
FileLink(r'exp_1.zip')
```

удалить лишние директории или файлы в `output` тоже легко:

```
!rm -r /kaggle/working/exp_1
```

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`

```
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)
```