1 Задание «Предсказание карт внимания»

1.1 Описание

Предлагается реализовать нейросетевой алгоритм построения карт внимания

Требования: * Модель должна быть реализована на фреймворке PyTorch * Для обучения разрешается использовать только обучащую выборку, внешние данные использовать нелья. Однако не запрещено (и, наоборот, приветствуется) использовать аугментации (размножение датасета) и transfer-learning (использование предобученной модели) с дообучением * Обработка одного кадра должна быть возможна на видеокарте с объемом памяти 24Gb * Соблюдать кодекс чести. Виновные будут найдены и наказаны

1.1.1 Оценивание

1. Каждый участник может представить не более 1 алгоритма для финального тестирования
2. Тестирование будет проводиться на закрытой тестовой выборке, содержащей N (N < 20) тестовых видео. В каждом > 200 кадров.
3. В качестве метрик будут использованы:

   - Normalized Scanpath Saliency (NSS)
   - Similarity score (SIM)
   - Pearson's Correlation Coefficient (CC)

   Подробнее про метрики

4. По итогам тестирования будет составлена общая таблица результатов по каждой из метрик
5. Место алгоритма определяется по формуле:

$$\text{Place}_{\text{algo}} = \frac{\text{Place}_{\text{NSS}} + \text{Place}_{\text{SIM}} + \text{Place}_{\text{CC}}}{3}$$

6. Баллы участника зависят от места его алгоритма:

$$\text{Score}_{\text{stud}} = \text{score}(\text{Place}_{\text{algo}})$$

```
GPU 0: NVIDIA A100-SXM4-80GB (UUID: GPU-10ae9fae-da49-67e4-30ce-5e8c30720aa9)
```

1.2 1. Подготовка данных

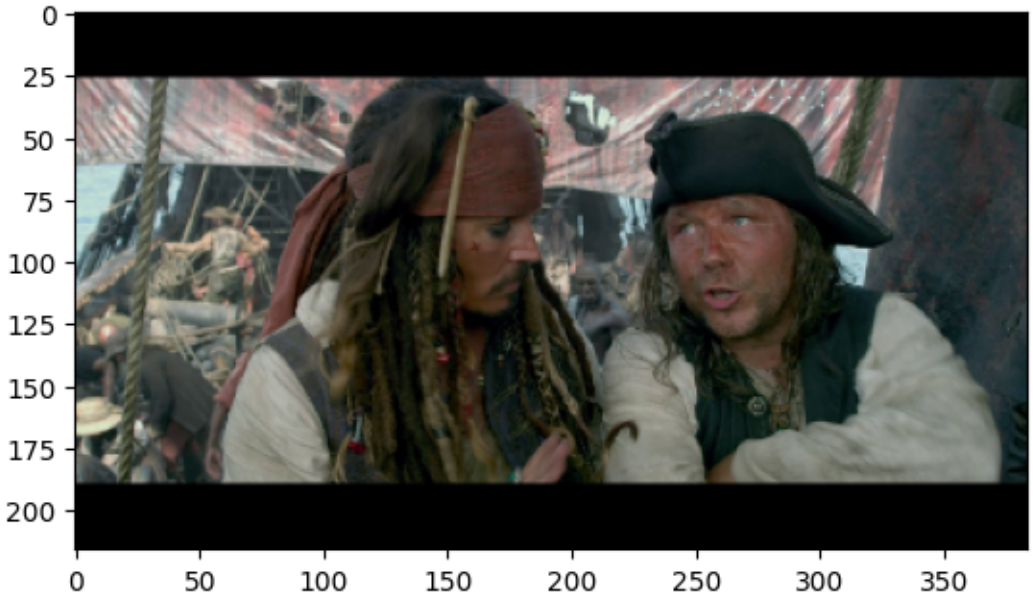
Дублирование датасета с страницы задания на Google Диск для пользователей Colab [5.1 GB]

1.2.1 Датасеты и даталоадеры

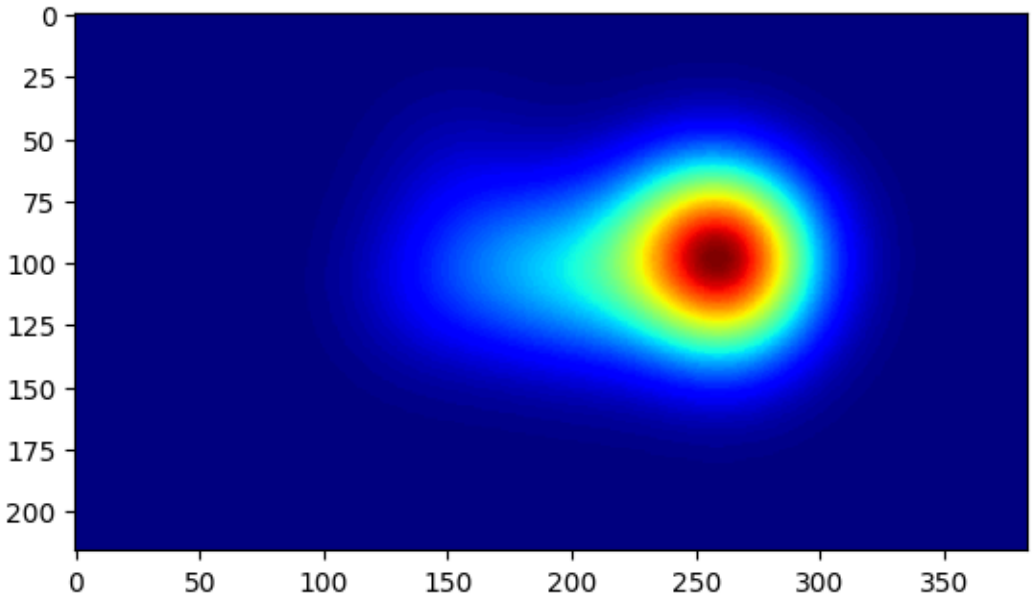
Вы можете полностью менять эту реализацию, использовать более одного кадра, добавлять transforms и т.д.!

Посмотрим на случайный пример из датасета

```
<matplotlib.image.AxesImage at 0x7f93f1a99bd0>
```




```
<matplotlib.image.AxesImage at 0x7f955cad4290>
```



1.3 2. Создадим класс нашей модели

Так как данных мало, предлагается использовать технику **transfer learning**, используя нейросеть, предобученную на сегментацию изображений, например, **Deeplabv3**.

Модель построим из трех частей: * Сверточный Encoder, например, ResNet50 * Пирамидальный пулинг ASPP состоящий из нескольких параллельных сверток с разным рецептивным полем благодаря dilation convolution * Сверточный Decoder, получающий финальное предсказание

Архитектура Deeplab:

Dilation convolution:

В моделе, доступной в библиотеке `torchvision.models`, Decoder получает изображение в низком разрешении и имеет 21 класс (количество выходных каналов):

```
Conv2d(256, 21, kernel_size=(1, 1), stride=(1, 1))
```

В нашей же задаче требуется предсказания одноканального изображения с разрешением, как у исходного. Поэтому сделаем свой декодер, постепенно повышая разрешение в 2 раза. На последнем слое применим нормализацию в 0-1.

Документация PyTorch: https://pytorch.org/docs/stable/index.html

Ваша модель должна быть описана в файле `saliency.py`, этот файл **СДАЕТСЯ** в проверяющую систему!

Файл `saliency.py` должен быть самодостаточным, т.е. содержать все необходимые import-ы и функции для создания и запуска модели.

```
==========================================================================================
Layer (type:depth-idx)                                  Output Shape              Param #
==========================================================================================
SaliencyModel                                           [8, 1, 216, 384]          --
├─IntermediateLayerGetter: 1-1                          [8, 2048, 27, 48]         --
│    └─Conv2d: 2-1                                      [8, 64, 108, 192]         9,408
│    └─BatchNorm2d: 2-2                                 [8, 64, 108, 192]         128
│    └─ReLU: 2-3                                        [8, 64, 108, 192]         --
│    └─MaxPool2d: 2-4                                   [8, 64, 54, 96]           --
│    └─Sequential: 2-5                                  [8, 256, 54, 96]          --
│    │    └─Bottleneck: 3-1                             [8, 256, 54, 96]          75,008
│    │    └─Bottleneck: 3-2                             [8, 256, 54, 96]          70,400
│    │    └─Bottleneck: 3-3                             [8, 256, 54, 96]          70,400
│    └─Sequential: 2-6                                  [8, 512, 27, 48]          --
│    │    └─Bottleneck: 3-4                             [8, 512, 27, 48]          379,392
│    │    └─Bottleneck: 3-5                             [8, 512, 27, 48]          280,064
│    │    └─Bottleneck: 3-6                             [8, 512, 27, 48]          280,064
│    │    └─Bottleneck: 3-7                             [8, 512, 27, 48]          280,064
│    └─Sequential: 2-7                                  [8, 1024, 27, 48]         --
│    │    └─Bottleneck: 3-8                             [8, 1024, 27, 48]         1,512,448
│    │    └─Bottleneck: 3-9                             [8, 1024, 27, 48]         1,117,184
│    │    └─Bottleneck: 3-10                            [8, 1024, 27, 48]         1,117,184
│    │    └─Bottleneck: 3-11                            [8, 1024, 27, 48]         1,117,184
│    │    └─Bottleneck: 3-12                            [8, 1024, 27, 48]         1,117,184
│    │    └─Bottleneck: 3-13                            [8, 1024, 27, 48]         1,117,184
│    └─Sequential: 2-8                                  [8, 2048, 27, 48]         --
│    │    └─Bottleneck: 3-14                            [8, 2048, 27, 48]         6,039,552
│    │    └─Bottleneck: 3-15                            [8, 2048, 27, 48]         4,462,592
│    │    └─Bottleneck: 3-16                            [8, 2048, 27, 48]         4,462,592
├─ASPP: 1-2                                             [8, 256, 27, 48]          --
│    └─ModuleList: 2-9                                  --                        --
```

```
|    |    └─Sequential: 3-17                  [8, 256, 27, 48]          524,800
|    |    └─ASPPConv: 3-18                    [8, 256, 27, 48]          4,719,104
|    |    └─ASPPConv: 3-19                    [8, 256, 27, 48]          4,719,104
|    |    └─ASPPConv: 3-20                    [8, 256, 27, 48]          4,719,104
|    |    └─ASPPPooling: 3-21                 [8, 256, 27, 48]          524,800
|    └─Sequential: 2-10                       [8, 256, 27, 48]          --
|    |    └─Conv2d: 3-22                      [8, 256, 27, 48]          327,680
|    |    └─BatchNorm2d: 3-23                 [8, 256, 27, 48]          512
|    |    └─ReLU: 3-24                        [8, 256, 27, 48]          --
|    |    └─Identity: 3-25                    [8, 256, 27, 48]          --
├─Sequential: 1-3                            [8, 128, 27, 48]          --
|    └─Conv2d: 2-11                          [8, 128, 27, 48]          294,912
|    └─BatchNorm2d: 2-12                     [8, 128, 27, 48]          256
|    └─ReLU: 2-13                            [8, 128, 27, 48]          --
├─Sequential: 1-4                            [8, 64, 54, 96]           --
|    └─Conv2d: 2-14                          [8, 64, 54, 96]           73,728
|    └─BatchNorm2d: 2-15                     [8, 64, 54, 96]           128
|    └─ReLU: 2-16                            [8, 64, 54, 96]           --
├─Sequential: 1-5                            [8, 32, 108, 192]         --
|    └─Conv2d: 2-17                          [8, 32, 108, 192]         18,432
|    └─BatchNorm2d: 2-18                     [8, 32, 108, 192]         64
|    └─ReLU: 2-19                            [8, 32, 108, 192]         --
├─Conv2d: 1-6                                [8, 1, 216, 384]          289
==========================================================================================
Total params: 39,430,945
Trainable params: 39,430,945
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 417.26
==========================================================================================
Input size (MB): 7.96
Forward/backward pass size (MB): 5695.96
Params size (MB): 157.72
Estimated Total Size (MB): 5861.65
==========================================================================================
```

1.4 3. Зададим функцию потерь

Будем использовать дивергенцию Кульбака-Лейблера, сравнивая предсказанное распределение с эталонной картой внимания. Вы так же можете использовать другие функции потерь при обучении.

1.5 4. Обучим модель

Пайплайн обучения: * Определить **таргет**. В нашей задаче это эталонная карта внимания * Определить **функцию потерь (loss)**. Используем KLD из предыдущего этапа * Выбрать **оптимизатор**. Чтобы всё быстро заработало, возьмем `Adam/AdamW` с lr=3e-4

```
100%|██████████| 50/50 [00:03<00:00, 13.41it/s, loss=0.478]
100%|██████████| 3/3 [00:00<00:00,  9.59it/s]

| Epoch:  0 | Val Loss:  0.4420415957768758 | Train Loss:  0.6352256137132645



100%|██████████| 50/50 [00:03<00:00, 15.00it/s, loss=0.545]
100%|██████████| 3/3 [00:00<00:00, 22.31it/s]
```

```
| Epoch:  1 | Val Loss:  0.41024504601955414 | Train Loss:  0.4513975021243095


100%|██████████| 50/50 [00:03<00:00, 15.09it/s, loss=0.472]
100%|██████████| 3/3 [00:00<00:00, 18.52it/s]

| Epoch:  2 | Val Loss:  0.40334685643513996 | Train Loss:  0.39695730298757553


100%|██████████| 50/50 [00:03<00:00, 15.02it/s, loss=0.327]
100%|██████████| 3/3 [00:00<00:00, 12.21it/s]

| Epoch:  3 | Val Loss:  0.3442202905813853 | Train Loss:  0.38187201112508773


100%|██████████| 50/50 [00:03<00:00, 14.99it/s, loss=0.33]
100%|██████████| 3/3 [00:00<00:00, 22.48it/s]

| Epoch:  4 | Val Loss:  0.4235289345184962 | Train Loss:  0.40758650422096254


100%|██████████| 50/50 [00:03<00:00, 15.04it/s, loss=0.284]
100%|██████████| 3/3 [00:00<00:00, 12.01it/s]

| Epoch:  5 | Val Loss:  0.3127121130625407 | Train Loss:  0.3616190379858017


100%|██████████| 50/50 [00:03<00:00, 15.02it/s, loss=0.382]
100%|██████████| 3/3 [00:00<00:00, 13.17it/s]

| Epoch:  6 | Val Loss:  0.3085400362809499 | Train Loss:  0.3455543836951256


100%|██████████| 50/50 [00:03<00:00, 15.22it/s, loss=0.217]
100%|██████████| 3/3 [00:00<00:00, 15.91it/s]

| Epoch:  7 | Val Loss:  0.2663857738176982 | Train Loss:  0.3469843310117722


100%|██████████| 50/50 [00:03<00:00, 15.07it/s, loss=0.308]
100%|██████████| 3/3 [00:00<00:00, 18.21it/s]

| Epoch:  8 | Val Loss:  0.27001820504665375 | Train Loss:  0.31811190843582154


100%|██████████| 50/50 [00:03<00:00, 15.06it/s, loss=0.31]
100%|██████████| 3/3 [00:00<00:00, 12.00it/s]

| Epoch:  9 | Val Loss:  0.30045921603838605 | Train Loss:  0.3230880516767502
```

1.6 5. Протестируем модель

Сохраняем веса модели (этот файл **СДАЕТСЯ** в проверяющую систему)

В качестве метрик будут использованы: * Normalized Scanpath Saliency (NSS) * Similarity score (SIM) * Pearson's Correlation Coefficient (CC)

Подробнее про метрики

Помимо описания класса модели в **СДАВАЕМОМ** файле `saliency.py` необходимо описать класс `SaliencyEvaluator`, выполняющий логику загрузки модели и инференса на заданном видео с сохранением предсказаний.

Функция принимает путь до входных последовательностей кадров и путь до выходной папки предсказаний.

В этой функции нужно описать процесс загрузки весов модели получение карт внимания для всех входных видеопоследовательностей. Если ваша модель использует более одного кадра или требует дополнительных преобразований входа, реализуйте их внутри этой функции.

```
100%|██████████| 450/450 [00:33<00:00, 13.60it/s]
100%|██████████| 450/450 [00:31<00:00, 14.33it/s]
100%|██████████| 450/450 [00:29<00:00, 15.01it/s]


100%|██████████| 3/3 [00:00<00:00, 34100.03it/s]
100%|██████████| 450/450 [00:05<00:00, 75.44it/s]
100%|██████████| 450/450 [00:06<00:00, 73.30it/s]
100%|██████████| 450/450 [00:06<00:00, 72.02it/s]


{'cc': 0.5673946224045094,
 'sim': 0.5195382390985229,
 'nss': 1.9877638927080243}

100%|██████████| 90/90 [00:01<00:00, 56.75it/s]
```

1.7 6. Дальнейшие шаги

Для улучшения качества вашей модели можно попробовать следующие этапы: * Сделать более информативную валидацию, например, добавив подсчет тестовых метрик прямо в validation loop * Попробовать обучать большее число эпох, использовать регуляризацию, например, через Dropout * Добавить skip-connection в Encoder-Decoder * Сделать аугментации (albumentations). Будьте аккуранты с преобразованиями, которые могут потенциально изменить эталонные карты внимания. Начать можно с горизонтальных отражений * Попробовать другие архитектуры, функции потерь и стратегии обучения * Использовать информацию из более чем одного кадра (3D Conv/LSTM/GRU/любой другой способ агрегации). Обратите внимание: даже если ваш метод требует окно из кадров, тестирование всё равно будет учитывать все кадры видео. Во время тестирования вы можете искуственно дублировать первый кадр для накопления нужной ширины окна для предсказания. * Поискать методы, решающие похожие задачи * Пофантазировать и вдохновиться