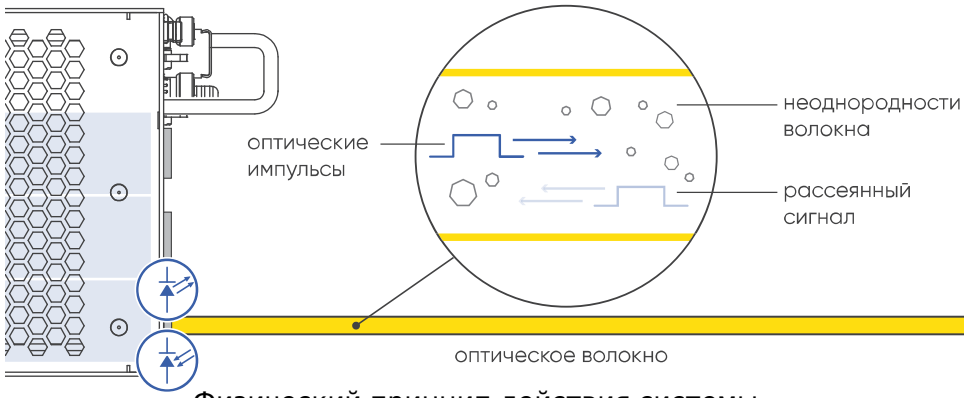
Модель сегментации сигналов целевых воздействий на данных с распределенного акустического сенсора

Черненко А.Е.

Распределенный акустический сенсор

- Распределенный акустический сенсор (Distributed Acoustic Sensor, DAS) позволяет обнаруживать вибрацию грунта (акустические колебания) на расстоянии до нескольких десятков километров вдоль оптического кабеля.
- В основе работы системы лежит принцип когерентной рефлектометрии. В волокно периодически вводятся оптические импульсы, часть света рассеивается на неоднородностях волокна и распространяется в обратном направлении. При микродеформациях волокна, вызванных виброакустическими воздействиями, параметры рассеяного (отраженного) сигнала изменяются. Анализируя изменения в интерференционной картине сигнала обратного рассеяния, можно определить место и характер воздействия на волокно.
- Определение характера воздействия возможно за счет применения различных нейросетевых распознавателей.

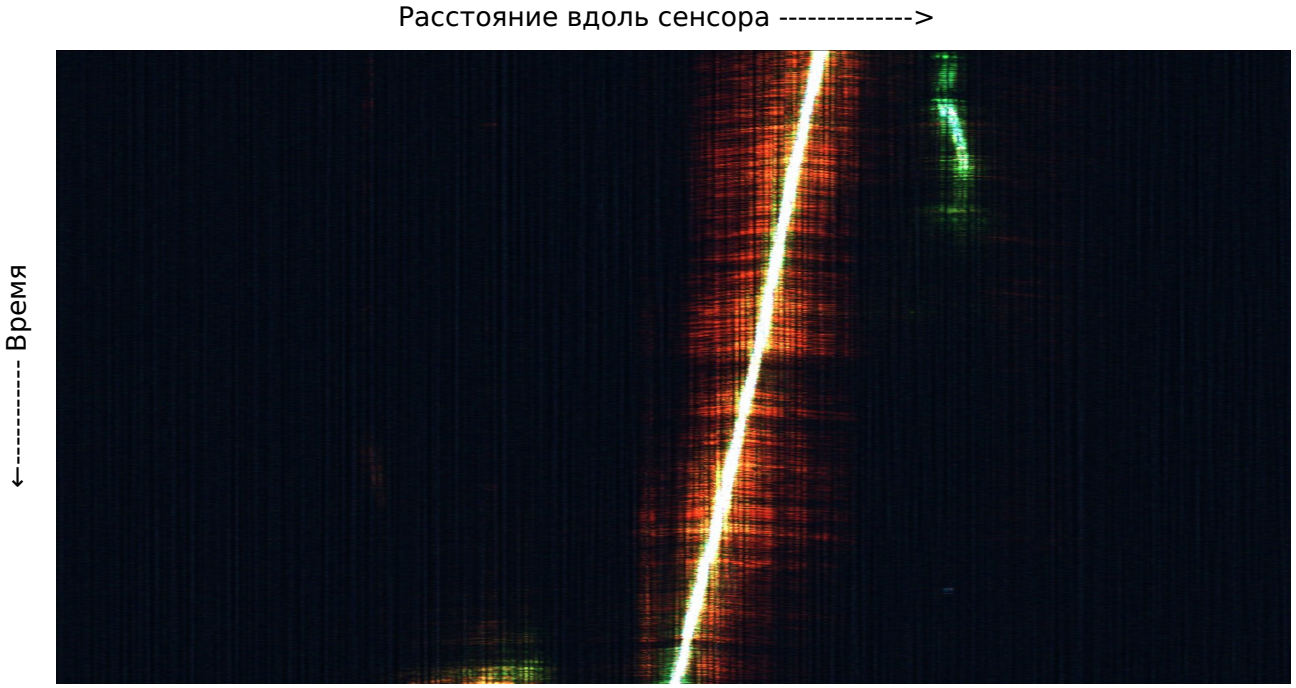


Физический принцип действия системы

Данные распределенного акустического сенсора

- Одним из возможных представлений данных с распределенного акустического сенсора является трехканальное изображение (“водопад”), горизонтальная ось связана с расстоянием вдоль сенсора, вертикальная – со временем.

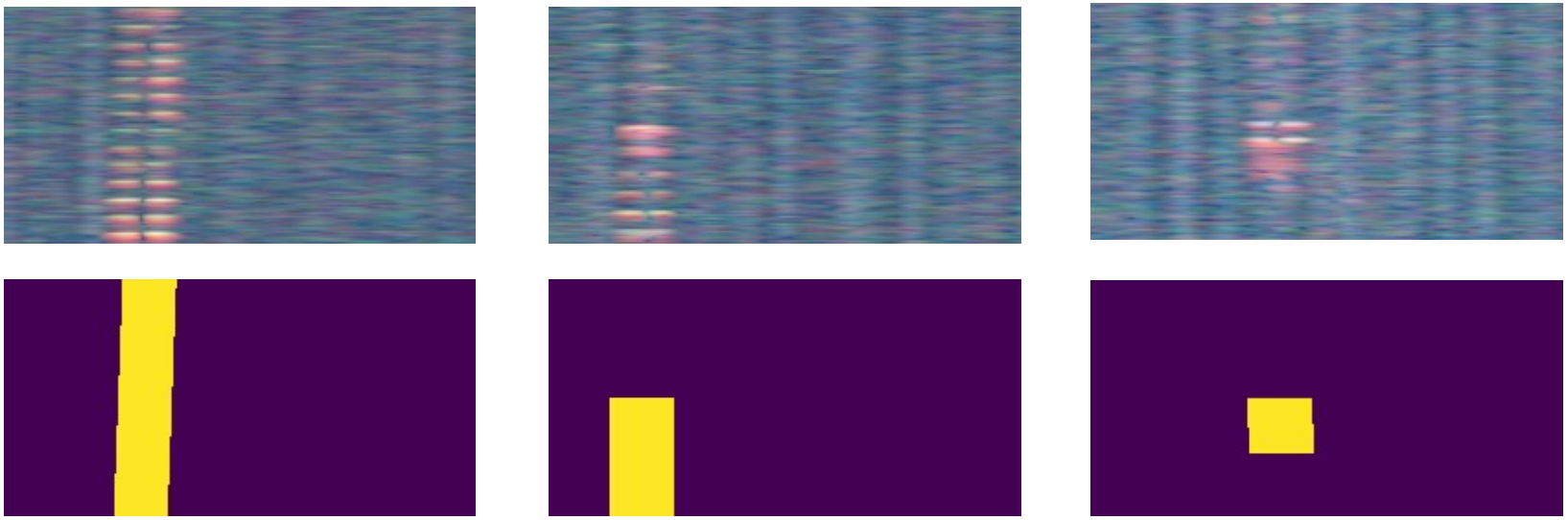
Задачи локализации и определения характера воздействия по данному типу представления данных можно решать методами компьютерного зрения.



Фрагмент водопада с характерным следом сигнала от проезжающего автомобиля

Задача проекта

- **Задача проекта** – создание прототипа фреймворка по подготовке моделей сегментации целевых воздействий
- В данном проекте для примера подготовлена модель сегментации акустических следов сигнала от движения пешехода.



Примеры сэмплов и их масок разметки

Подготовка данных

- Имеется набор подготовленных и размеченных записей водопада. Данные и разметка сохранены в файлах формата HDF5. Записи разбиты на тренировочную, валидационную и тестовую выборки. Пути к записям по выборкам сохранены в файле **bundle/records.py**.
- Скрипт **bundle/lim_seg_bundle.py** подготавливает бандл **data/c_bundle_hs_128_256.h5** с информацией для нарезки сэмплов по заданным правилам (размеры сэмпла, шаг сдвига центра сэмпла).
- Структура бандла **data/c_bundle_hs_128_256.h5**:
  - **Атрибуты**: размеры сэмпла, прореживание сэмпла, список классов разметки, индексы которых соответствуют лэйблу класса.
  - Датасет **records_map** – массив строк, содержащих пути к записям. Индекс элемента массива соответствует лейблу записи.
  - Группы **train**, **val**, **test**. Структура группы:
    - Датасет **coords** – массив координат центра сэмпла в записи.
    - Датасет **labels** – массив лэйблов класса сигнала в сэмпле.
    - Датасет **records** – массив лэйблов записи, с которой нарезается сэмпл.
    - Индексы элементов в датасетах соответствуют номеру сэмпла.

Датасет

- Класс датасета реализован в **utils/dataset.py** на базе torch.utils.data.Dataset.

```
class SegDataset(torch.utils.data.Dataset):

    def __init__(self,
                 bundle_path: str,
                 group_name: str,
                 markup_mapping: Dict,
                 filter_id: int = 0):
```

Аргументы конструктора:

- ***bundle_path*** – путь к бандлу data/c_bundle_hs_128_256.h5;
- ***group_name*** – название выборки (train, val, test);
- ***markup_mapping*** – маппинг классов разметки в целевые классы предсказаний;
- ***filter_id*** – номер фильтра, в данном проекте к каждому сэмплу применяется преобразование log(x+1).

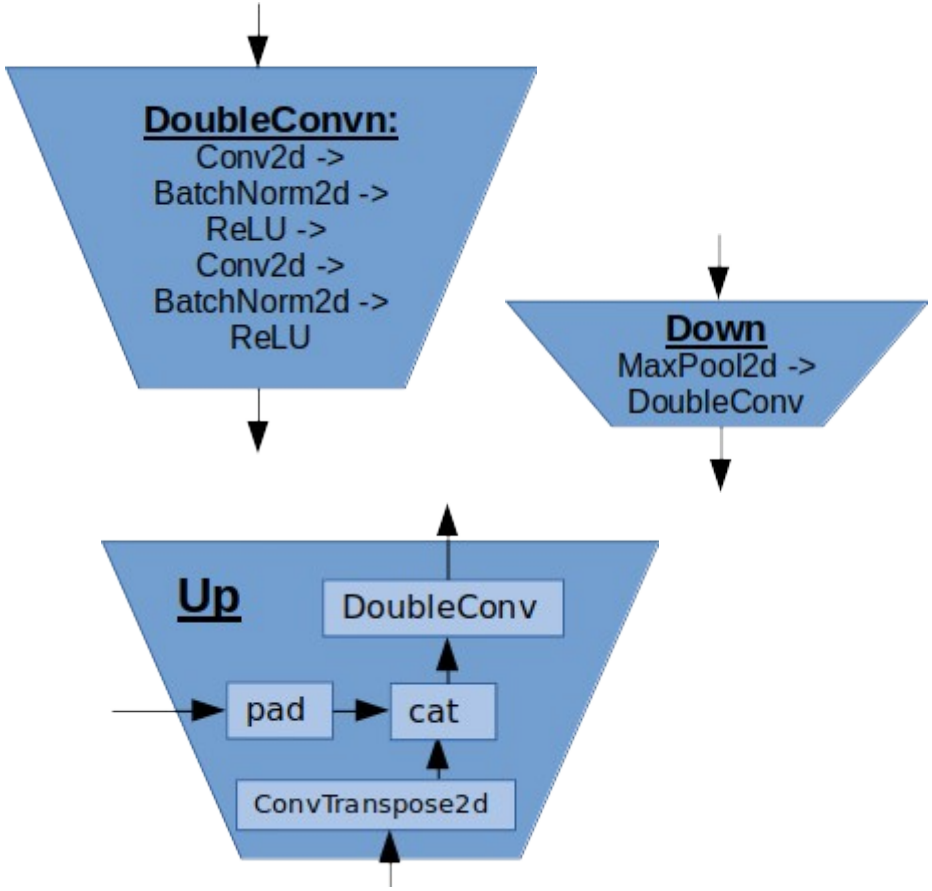
Также в конструкторе датасета рассчитываются веса сэмплов (свойство **weights**).

Метод **SegDataset.__getitem__(index)** считывает из выборки бандла информацию по нарезке сэмпла с номером index. Вырезает из записи по координатам и правилам сэмпл, выполняет перобразование фильтром. Также вырезает из записи разметки соответствующий сэмплу и лейблу класса фрагмент маски. Возвращает словарь с двумя элементами: ‘image’ - torch.Tensor с данными сэмпла, ‘mask’ torch.Tensor – с данными маски разметки.

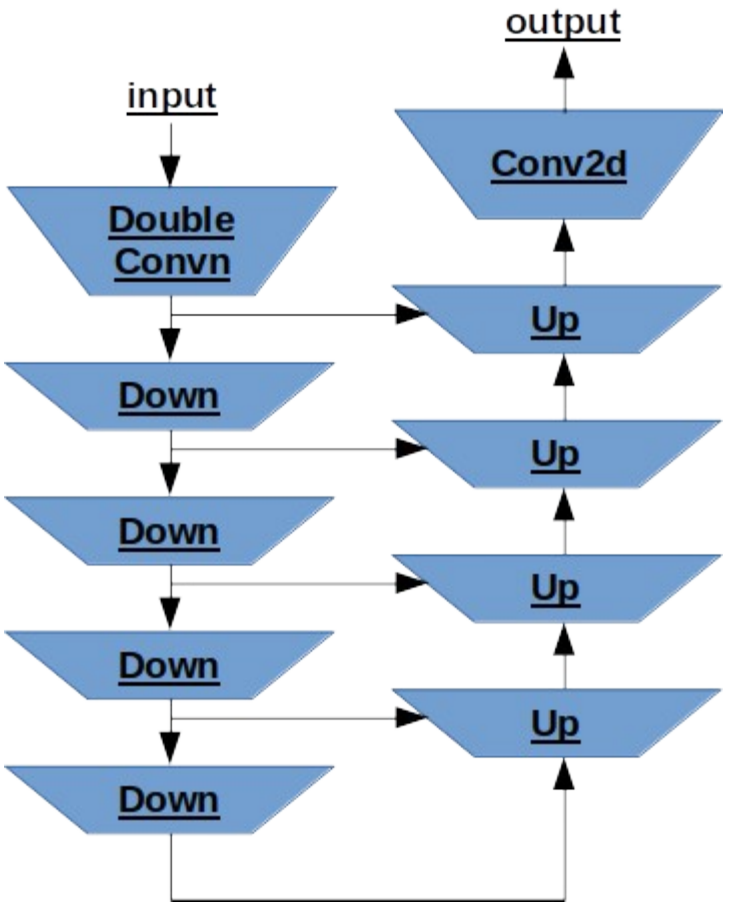
Архитектура модели

Архитектура модели – Unet. *Модель взята из проекта https://github.com/milesial/Pytorch-UNet*

- **unet/blocks.py –** блоки модели



unet/model.py – каркас модели



Тренировка модели

- **В conf.py задаются параметры для обучения**
- **Процедура обучения сети реализована в train.py**
- **Процедура оценки метрики (dice) реализована в evaluate.py, используется в конце каждой эпохи**
- **Данные в бандле избыточны: один и тот же участок сиганла может присутствовать в нескольких сэмплах, но локализован в разных участках сэмпла. Поэтому на этапе обучения DataLoader использует WeightedRandomSampler, который случайно (используя веса сэмплов) сэмплирует из датасета ограниченное число примеров на одну эпоху. Случайность в валидационном сэмплере заморожена, чтобы на всех эпохах использовалсь одна и таже валидационная выборка**
- **Optimizer - RMSprop**
- **Sheluder - ReduceLROnPlateau**
- **Loss – CrossEntropy + Dice**

Тренировка модели

Лог обучения:

```
INFO: Using device cuda
INFO: Network:
        3 input channels
        1 output channels (classes)


INFO: Starting training:
        Epochs:          50
        Batch size:      16
        Learning rate:   0.0001
        Training size:   1024
        Validation size: 1024
        Checkpoints:     True
        Device:          cuda
        Mixed Precision: True


INFO: Validation Dice score: 0.770007312297821
Epoch 1/50: 100%|██████████| 1024/1024 [00:11<00:00, 86.27img/s, loss (batch)=0.246]
INFO: Checkpoint 1 saved!
INFO: Validation Dice score: 0.8117258548736572
Epoch 2/50: 100%|██████████| 1024/1024 [00:11<00:00, 88.48img/s, loss (batch)=0.226]
INFO: Checkpoint 2 saved!
INFO: Validation Dice score: 0.806232213973999
Epoch 3/50: 100%|██████████| 1024/1024 [00:11<00:00, 88.35img/s, loss (batch)=0.178]
INFO: Checkpoint 3 saved!
INFO: Validation Dice score: 0.7977456450462341
Epoch 4/50: 100%|██████████| 1024/1024 [00:11<00:00, 88.31img/s, loss (batch)=0.191]
INFO: Checkpoint 4 saved!
INFO: Validation Dice score: 0.8244051933288574
Epoch 5/50: 100%|██████████| 1024/1024 [00:11<00:00, 88.69img/s, loss (batch)=0.188]
INFO: Checkpoint 5 saved!
INFO: Validation Dice score: 0.7886080741882324
Epoch 6/50: 100%|██████████| 1024/1024 [00:11<00:00, 88.90img/s, loss (batch)=0.198]
INFO: Checkpoint 6 saved!
INFO: Validation Dice score: 0.8325081467628479
Epoch 7/50: 100%|██████████| 1024/1024 [00:11<00:00, 88.65img/s, loss (batch)=0.193]
INFO: Checkpoint 7 saved!
INFO: Validation Dice score: 0.8189709782600403
Epoch 8/50: 100%|██████████| 1024/1024 [00:11<00:00, 89.28img/s, loss (batch)=0.223]
INFO: Checkpoint 8 saved!
INFO: Validation Dice score: 0.8358147740364075
Epoch 9/50: 100%|██████████| 1024/1024 [00:11<00:00, 89.12img/s, loss (batch)=0.162]
INFO: Checkpoint 9 saved!
INFO: Validation Dice score: 0.8286530375480652
Epoch 10/50: 100%|██████████| 1024/1024 [00:11<00:00, 88.89img/s, loss (batch)=0.211]
INFO: Checkpoint 10 saved!
```

Тестирование модели

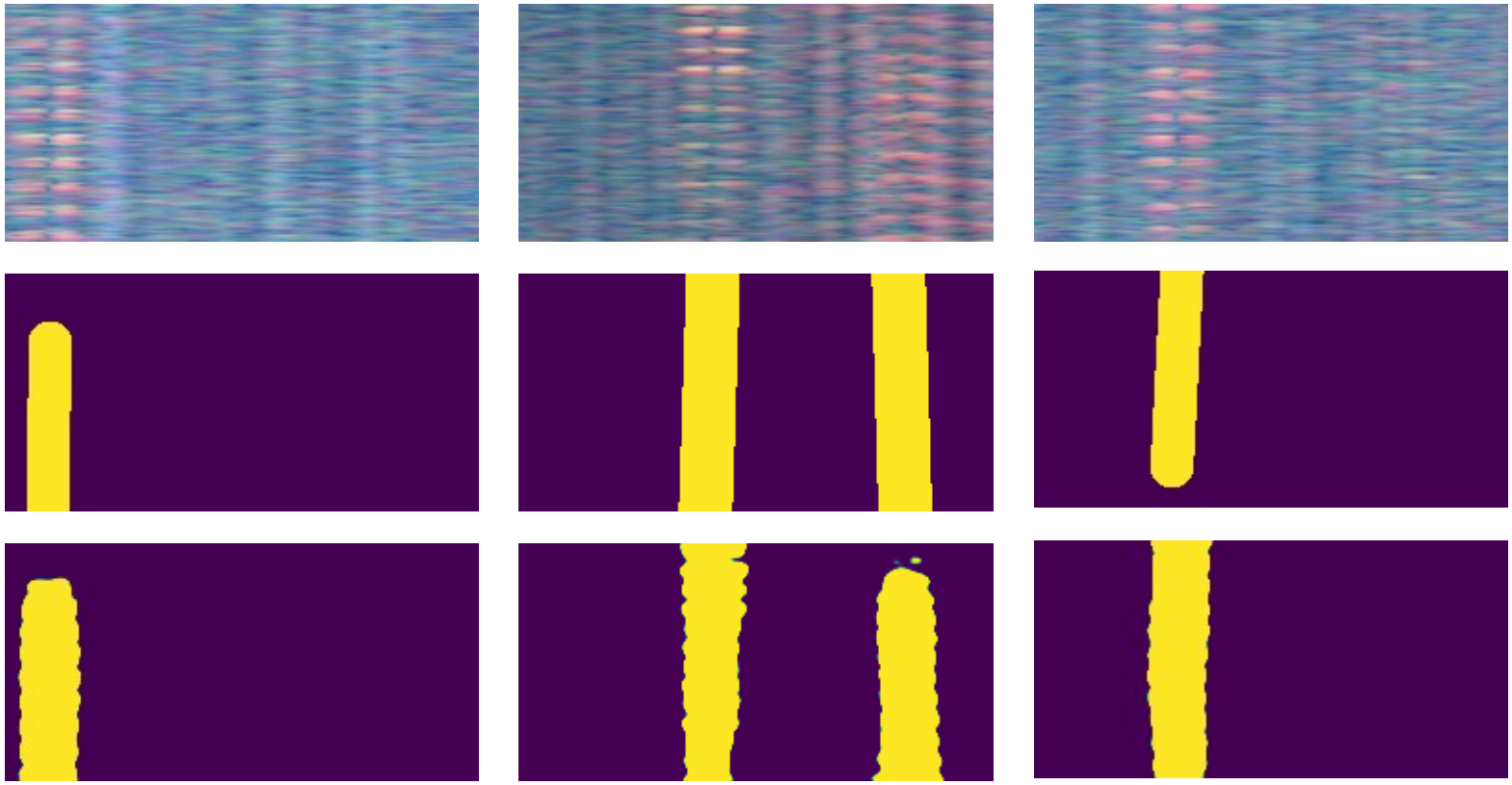
В **testing.py** реализован класс **TestModel**, инструмент для тестирования модели. Для инициализации требуется указать путь к модели и экземпляр датасета.

Метод **predict_by_idx(idx, threshold=None)** возвращает массив с данными сэмпла, массив с маской разметки сэмпла, массив с данными сегментации. Номер сэмпла в датасете задается idx. При указании порог threshold будет применен к данным сегментации, по умолчанию данные сегментации отражают тепловую карту вероятностей предсказаний.

Метод **show_predict_by_idx(idx, threshold=None)** показывает изображения сэмпла, маски разметки и сегментации.

Метод **evaluate(n_samples, batch_size=8, random_state=0)** возвращает среднюю по случайной выборке из n_samples сэмплов оценку метрики dice.

Тестирование модели



Средняя оценка метрики dice на тестовой выборке из 1024 сэмплов: **0.8248**

Что следует улучшить

- **Реализовать многоклассовую сегментацию**
- **Разработать метрики, которые лучше соответствуют бизнес-задаче.**
- **Реализовать процедуру оптимизации гиперпараметров при обучении моделей**
- **Реализовать технику распределеннго обучения на нескольких гпу и серверах**