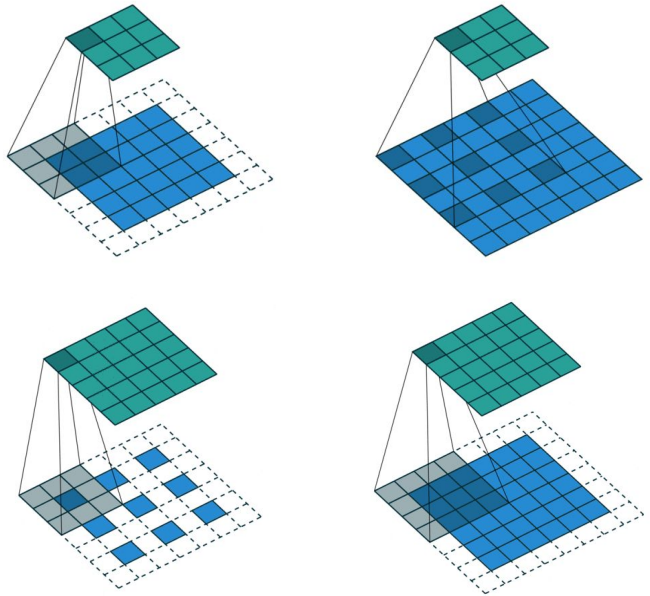
Как писать CNN-модели и не утонуть в свертках

Кузнецов Андрей
ITMO University, Saint-Petersburg
лаборант LISA
2024 г.

Проблема и мотивация

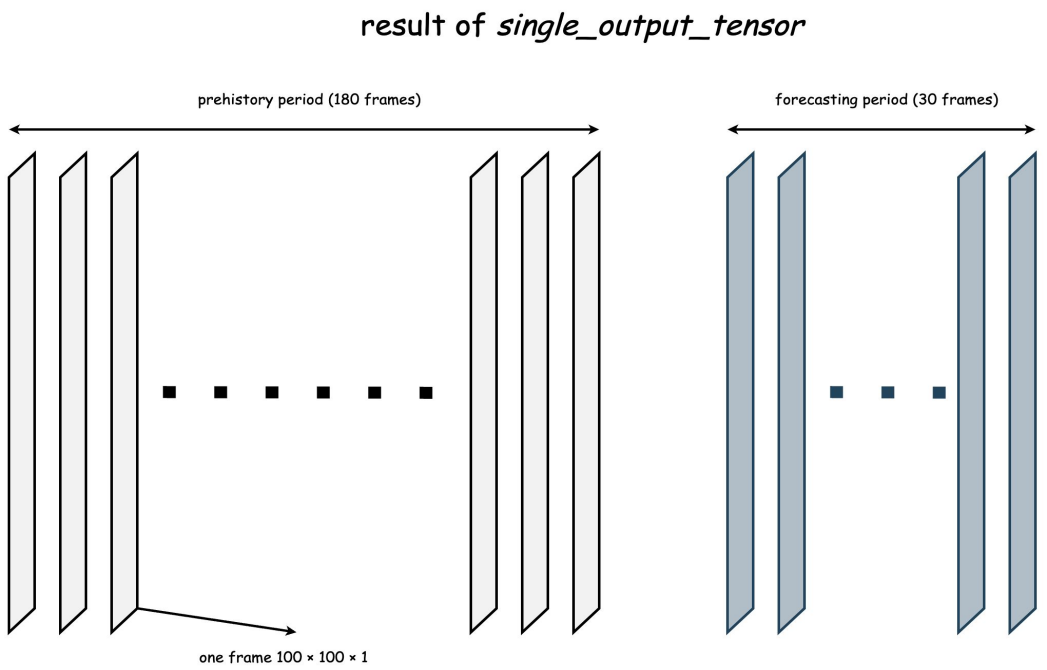
- сложность контролирования размерностей тензора в процессе прохода через сверточные слои
- неудобство корректной подборки гиперпараметров сверточных слоев
- желание проведения большого количества экспериментов с разными слоями и параметрами в разумных временных рамках



Примеры сверточных операций
[https://github.com/vdumoulin/conv_arithmetic]

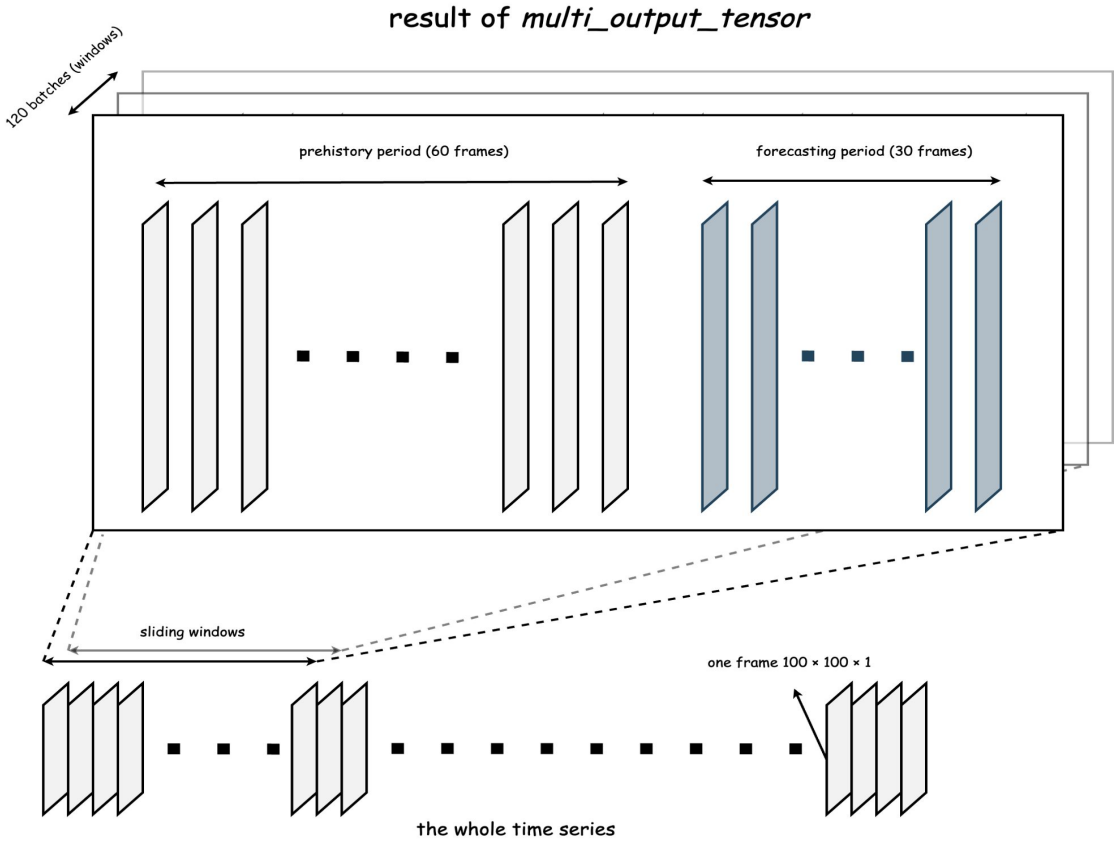
Предобработка временных рядов

Тензоры могут быть любой размерности (в примере они двумерные)



Предобработка временных рядов

Тензоры могут быть любой размерности (в примере они двумерные)



Предобработка временных рядов

```
dataset = multi_output_tensor(data=noise_data,
                              additional_x=[noise_data.copy(), noise_data.copy(), noise_data.copy()],
                              additional_is_array=True,
                              forecast_len=30,
                              pre_history_len=60,
                              threshold=0.5,
                              x_binarize=True)
```

доп.временные ряды

предыстория и дальность прогноза

порог бинаризации

бинаризация X

```
for i, batch in enumerate(dataset):
    print(f'batch number: {i}',
          f'new stacked X shape: {batch[0].shape}\nY shape: {batch[1].shape}',
          f'new stacked X max: {batch[0].max()} | min: {batch[0].min()}\nY max: {batch[1].max()} | min: {batch[1]
          sep='\n',
          end='\n\n')
    if i == 1:
        break
print(f'Dataset len (number of batches/X-windows): {len(dataset)}')
```

```
batch number: 0
new stacked X shape: torch.Size([60, 4, 100, 100])
Y shape: torch.Size([30, 100, 100])
new stacked X max: 1.0 | min: 0.0
Y max: 1.0 | min: 0.0

batch number: 1
new stacked X shape: torch.Size([60, 4, 100, 100])
Y shape: torch.Size([30, 100, 100])
new stacked X max: 1.0 | min: 0.0
Y max: 1.0 | min: 0.0

Dataset len (number of batches/X-windows): 120
```

проверка размерностей данных

Примеры API builder

In [2]:
```
from torchcnnbuilder.builder import conv1d_out, conv2d_out, conv3d_out
```

импорт функций для подсчета результатов после сверток

$$H_{out} = \lfloor \frac{H_{in} + 2 \times padding[0] - dilation[0] \times (kernel[0] - 1) + 1}{stride[0]} \rfloor + 1$$

In [3]:
```
new_size = conv1d_out(input_size=33,
                      kernel_size=5,
                      stride=2)

print(f'Tensor size after nn.Conv1d: {new_size}')
```

```
Tensor size after nn.Conv1d: 15
```

параметры свертки

Аналогично для транспонированных операций

Класс Builder

In [8]:
```
from torchcnnbuilder.builder import Builder
```

Initialization params:

описание атрибутов классов

- **input_size** (Sequence[int]): input size of the input tensor
- **minimum_feature_map_size** (Union[Sequence[int], int]): minimum feature map size. Default: 5
- **max_channels** (int): maximum number of layers after any convolution. Default: 512
- **min_channels** (int): minimum number of layers after any convolution. Default: 32
- **activation_function** (nn.Module): activation function. Default: nn.ReLU(inplace=True)
- **finish_activation_function** (Union[str, Optional[nn.Module]]): last activation function, can be same as activation_function (str `'same'`). Default: None
- **default_convolve_params** (dict[str, Union[int, tuple]]): parameters of convolutional layers (by default same as in torch)
- **default_transpose_params** (dict[str, Union[int, tuple]]): parameters of transpose convolutional layers (by default same as in torch)

Other attributes:

- **conv_channels** (List[int]): list of output channels after each convolutional layer
- **transpose_conv_channels** (List[int]): list of output channels after each transposed convolutional layer
- **conv_layers** (List[tuple]): list of output tensor sizes after each convolutional layer
- **transpose_conv_layers** (List[tuple]): list of output tensor sizes after each transposed convolutional layer

Suppose we create a CNN-model for a tensor of size 100 by 100, let the smallest feature map after the convolutions be of size 3 by 3

In [9]:
```
builder = Builder(input_size=[125, 125],
                  minimum_feature_map_size=3)
```

Подробное описание каждого метода есть в репозитории проекта в директории examples

Класс Builder

описание метода ascending

$$\begin{cases} channel_i = start \times ratio^i, \quad i = 1 \ldots n & \text{if } ascending = \text{False} \\ stop = \lfloor \frac{(input\_size[0] + input\_size[1]) * 0.5}{2} \rfloor + in\_channels & \text{if } ascending = \text{True} \\ step = \frac{stop - in\_channels}{n\_layers} \\ channels = \text{range}(in\_channels, stop, step) \end{cases}$$

Из документации метода build_convolve_sequence

```
builder.build_convolve_sequence(n_layers=3,
                                in_channels=3,
                                ascending=False, # by default
                                ratio=3)
```

```
Sequential(
  (conv 1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv 2): Sequential(
    (0): Conv2d(32, 96, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv 3): Sequential(
    (0): Conv2d(96, 288, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU(inplace=True)
  )
)
```

сверточная последовательность

```
builder.build_convolve_sequence(n_layers=2,
                                in_channels=3,
                                params={'kernel_size': 9},
                                sub_blocks=2,
                                ascending=True)
```

параметры сверток

```
Sequential(
  (conv 1): Sequential(
    (sub-block 1): Sequential(
      (0): Conv2d(3, 34, kernel_size=(9, 9), stride=(1, 1), padding=(4, 4))
      (1): ReLU(inplace=True)
    )
    (sub-block 2): Sequential(
      (0): Conv2d(34, 34, kernel_size=(9, 9), stride=(1, 1), padding=(4, 4))
      (1): ReLU(inplace=True)
    )
  )
  (conv 2): Sequential(
    (sub-block 1): Sequential(
      (0): Conv2d(34, 65, kernel_size=(9, 9), stride=(1, 1), padding=(4, 4))
      (1): ReLU(inplace=True)
    )
    (sub-block 2): Sequential(
      (0): Conv2d(65, 65, kernel_size=(9, 9), stride=(1, 1), padding=(4, 4))
      (1): ReLU(inplace=True)
    )
  )
)
```

сверточная последовательность

Класс Builder

замена финальной функции активации

настройка нормализации

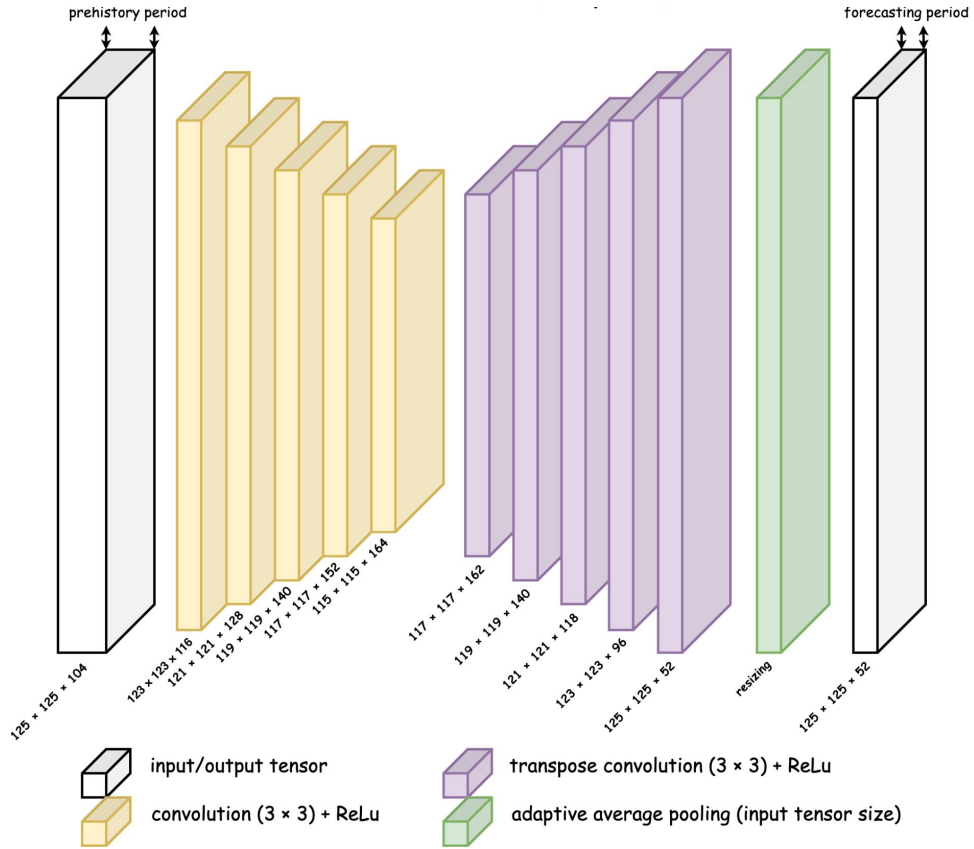
```
builder.finish_activation_function = nn.Softmax()
deconv_layer = builder.build_transpose_convolve_block(in_channels=3,
                                                      out_channels=64,
                                                      normalization='dropout',
                                                      p=0.2,
                                                      sub_blocks=3,
                                                      last_block=True)
deconv_layer
```

```
Sequential(
  (transpose sub-block 1): Sequential(
    (0): ConvTranspose2d(3, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): Dropout2d(p=0.2, inplace=False)
    (2): ReLU(inplace=True)
  )
  (transpose sub-block 2): Sequential(
    (0): ConvTranspose2d(3, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): Dropout2d(p=0.2, inplace=False)
    (2): ReLU(inplace=True)
  )
  (transpose sub-block 3): Sequential(
    (0): ConvTranspose2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): Dropout2d(p=0.2, inplace=False)
    (2): Softmax(dim=None)
  )
)
```

сверточный блок

Можно строить сверточные блоки отдельно (на примере транспонированной свертки)

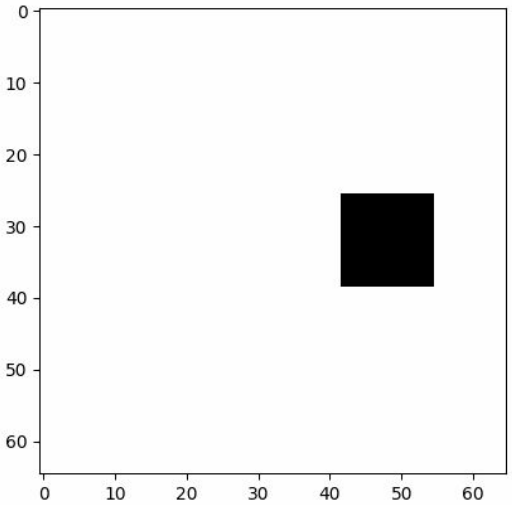
Шаблоны моделей (ForecasterBase)



Модель была предложена для решения задачи прогнозирования концентрации льда в статье “Forecasting of Sea Ice Concentration using CNN, PDE discovery and Bayesian Networks”

[https://www.sciencedirect.com/science/article/pii/S1877050923020094]

Возможности архитектуры



train часть 2-мерного временного ряда
(движение квадрата по кругу)

Возможности архитектуры

In [3]:
```
from torchcnnbuilder.models import ForecasterBase
```

In [4]:
```
import torch.nn as nn

# example of initialization
model = ForecasterBase(input_size=[65, 65],
                       in_channels=120,
                       out_channels=40,
                       n_layers=5,
                       normalization='batchnorm',
                       finish_activation_function=nn.ReLU(inplace=True))
```

входной размер матрицы
предыстория
дальность прогноза
кол-во слоев
добавление нормализации
финальная ф-ция активации

Параметры инициализации

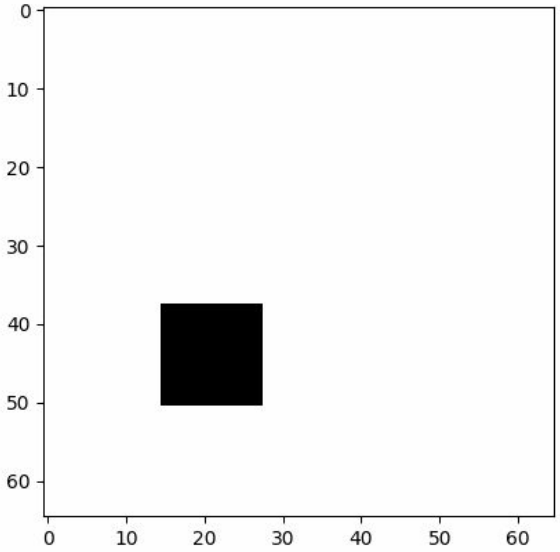
In [11]:
```
train_dataset = multi_output_tensor(data=train,
                                    pre_history_len=120,
                                    forecast_len=40)

test_dataset = single_output_tensor(data=test,
                                    forecast_len=40)
```

предыстория и дальность прогноза

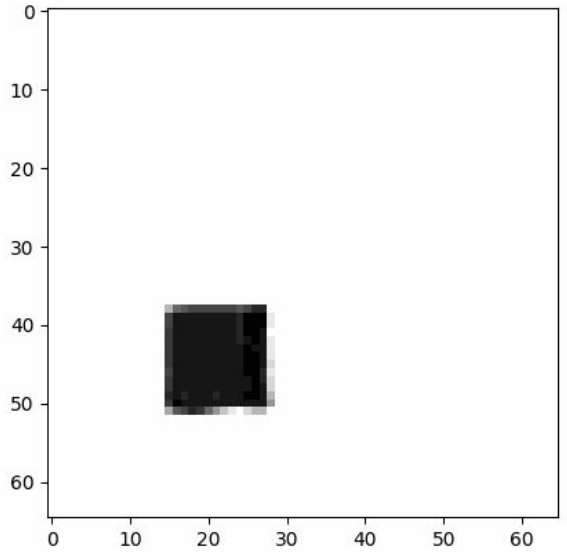
Подготовка датасета

Немного мемов про мой процесс обучения

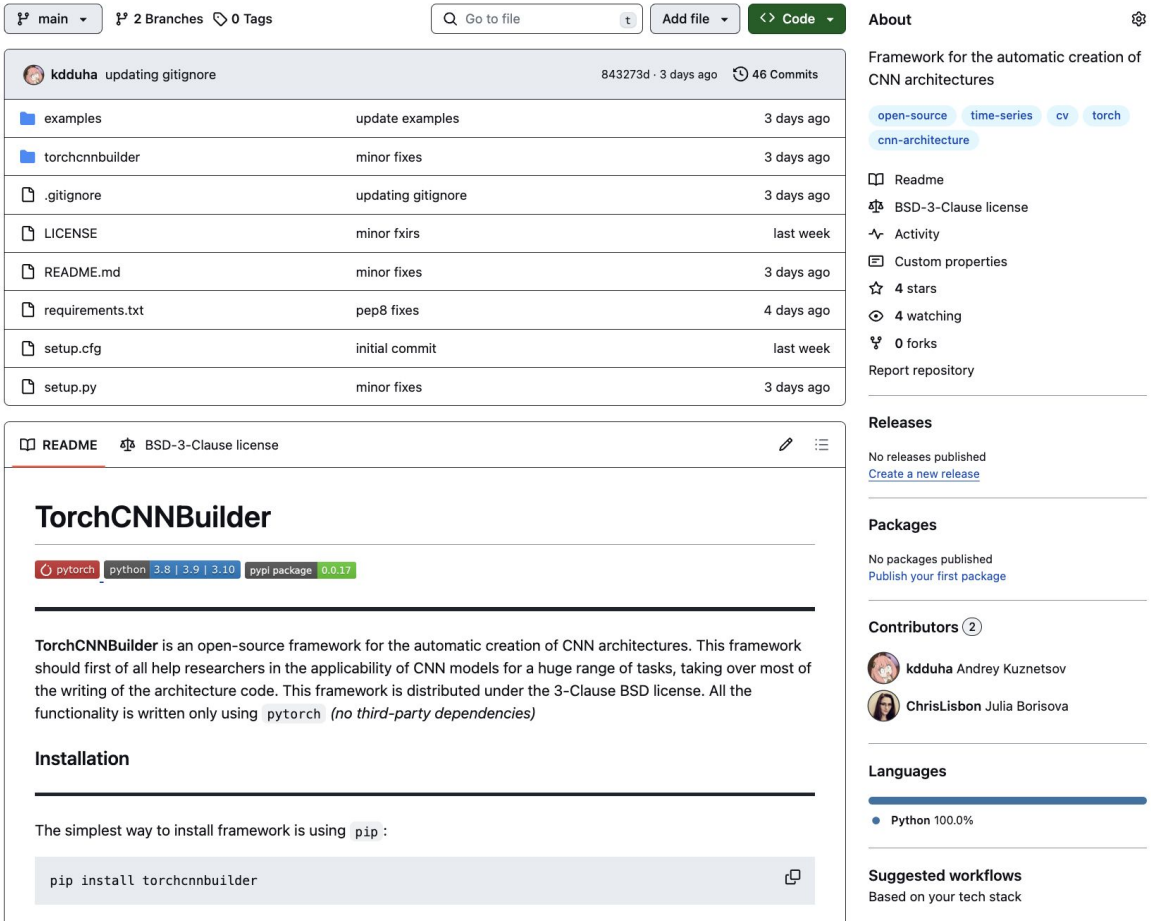
Результат



test



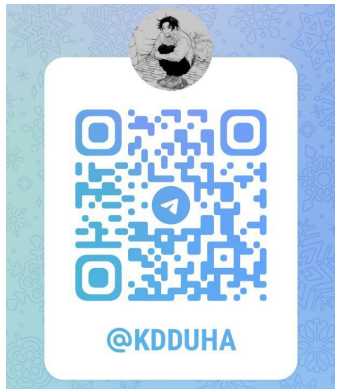
predict



GitHub страница разработанной библиотек

ИТМО

Подробное описание проекта с примерами всей функциональности и документацией



Спасибо за внимание!

ITMO more than a UNIVERSITY