**Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая кибернетика и информационные технологии»

Отчет по лабораторной работе №1

по дисциплине «Системы машинного зрения»

на тему:

«Разработка нейросетевых функций. Операция Convolution 2D»

Выполнила: студент группы БВТ2001

Щавлев К.В.

**Цель работы**

Разработать алгоритм, используя язык python, реализующий работу операции двумерной свертки.

**Задание**

При выполнении лабораторной работы необходимо:

* ознакомиться с описанием операции библиотеки PyTorch (https://pytorch.org/docs/stable/generated/torch.nn.Conv2d);
* используя язык программирования Python написать алгоритм, реализующий алгоритм свертки;
* составить отчет по лабораторной работе;

**Теория**

Свертка (convolution) в двумерном пространстве (например, на изображениях) — это операция, которая применяет ядро (или фильтр) к входным данным для получения выходного тензора. Она является основным строительным блоком в задачах обработки изображений в глубоком обучении.

Пусть I — это входное изображение, а K - ядро свертки. Тогда операция свертки в двумерном пространстве обозначается как I∗K и определяется следующим образом:

,

где суммирование происходит по всем возможным значениям индексов m и n, при которых ядро K перекрывается с пикселями входного изображения I.

Для свертки в глубоком обучении обычно используется операция свертки с ядром размера (m,n), которое имеет веса, определяемые в процессе обучения. Операция также может включать параметр смещения (bias). В этом случае формула обновляется следующим образом:

,

где B — это тензор смещения.

Операция свертки в глубоком обучении выполняется по всем каналам изображения и для всех фильтров (фильтры соответствуют выходным каналам) одновременно.

В процессе обучения параметры ядра K и тензора смещения B обновляются с использованием градиентного спуска или других методов оптимизации для минимизации функции потерь.

Таким образом, операция свертки является мощным инструментом для выделения различных признаков в изображениях и является основой для создания сложных моделей в области компьютерного зрения.

**Выполнение работы**

В качестве собственной реализации свертки convolution2d из библиотеки pytorch, был реализован класс *conv2Self* со следующими функциями:

* \_\_init\_\_ - функция, для инициализации объекта класса. Внутри нее происходит проверка на соответствие входных параметров к необходимым типам.
* conv2d – функция, в которой был реализован алгоритм двумерной свертки.
* torch\_conv2d – функция для вызова библиотечного варианта свертки.
* test – функция для сверки работы моего алгоритма и библиотечного варианта

Исходный код функции \_\_init\_\_ представлен в листинге 1.

Листинг 1. Код функции \_\_init\_\_.

def \_\_init\_\_(

        self,

        input\_data,

        kernel\_size: tuple | int,

        bias: float | None = None,

        stride: int = 1,

        padding: tuple[int, int] | int | str = (0, 0),

        dilation: int = 1,

    ):

        self.input\_data = input\_data[0,0].numpy()

        self.input\_data\_for\_torch = input\_data

        self.bias = bias

        if type(kernel\_size) == tuple:

            self.kernel\_size = kernel\_size

        else:

            self.kernel\_size = (kernel\_size, kernel\_size)

        self.stride = stride

        self.dilation = dilation

        if type(padding) == tuple:

            self.padding = padding[0]

        elif padding == "same":

            if self.stride != 1:

                raise ValueError("padding 'same' work only with stride=1")

            self.padding = self.kernel\_size[0]-1

        elif padding == "valid":

            self.padding = 0

        else:

            self.padding = (padding, padding)

        self.weight\_tensor\_for\_torch = torch.randn(1,1,self.kernel\_size[0], self.kernel\_size[1])

        self.weight\_tensor = self.weight\_tensor\_for\_torch[0,0].numpy()

Исходный код функции conv2d представлен в листинге 2.

Листинг2. Код conv2d.

def conv2d(self):

        image\_height, image\_width = self.input\_data.shape

        weight\_height, weight\_width = self.weight\_tensor.shape

        H\_out = int((image\_height - self.dilation \* (weight\_height - 1) - 1 + 2\* self.padding)/self.stride) + 1

        W\_out = int((image\_width - self.dilation \* (weight\_width - 1) - 1 + 2\* self.padding)/self.stride) + 1

        if self.padding>0:

            self.input\_data = np.pad(self.input\_data, self.padding, mode='constant')

        result = np.zeros((H\_out, W\_out))

        for y in range(H\_out):

            for x in range(W\_out):

                input\_slice = self.input\_data[y \* self.stride:y \* self.stride + weight\_height, x \* self.stride:x \* self.stride + weight\_width]

                result[y, x] = np.sum(input\_slice \* self.weight\_tensor)

        if self.bias:

            result+=self.bias

Исходный код функции test представлен в листинге 3.

Листинг 3. Код test.

    def test(self, print\_flg=False):

        my\_conv2d = self.conv2d()

        torch\_out = np.array(self.torch\_conv2d())

        if print\_flg:

            print(my\_conv2d)

            print(torch\_out[0,0])

        print(np.allclose(my\_conv2d, torch\_out[0, 0]))

Также, в рамках работы было произведено тестирование алгоритма, листинг 4, код тестовых случаев.

Листинг 4. Код тестовых случаев.

image = torch.randn(1,1,5,5)

# Тестовый случай 1: Обычная свертка

c1 = conv2Self(image, kernel\_size=1)

# Тестовый случай 2: Свертка с padding='same'

c2 = conv2Self(image, kernel\_size=1, padding='same')

# Тестовый случай 3: Свертка с padding='valid'

c3 = conv2Self(image, kernel\_size=1, padding='valid')

# Тестовый случай 4: Различные размеры входных данных и ядра

c4 = conv2Self(image, kernel\_size=4, padding='same')

# Тестовый случай 5: Свертка с dilation

c5 = conv2Self(image, kernel\_size=1, dilation=5)

# Тестовый случай 6: Свертка с stride

c6 = conv2Self(image, kernel\_size=1, stride=5)

Результаты тестирования представлены на рисунке 1.

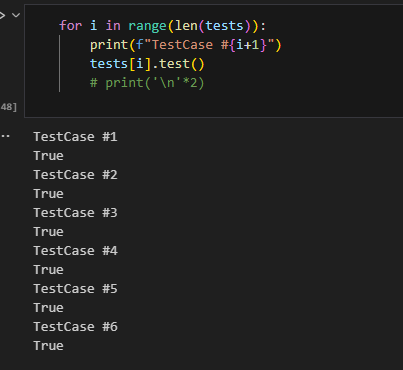


Рисунок 1. Результат тестирования

**Вывод**

В результате выполнения работы я получил собственную имплементацию алгоритма двумерной свертки.

Ссылка на git: