**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 №2

по дисциплине «Системы машинного зрения»

на тему:

«Разработка нейросетевых функций. Операция Convolution 3D»

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;

**Требования к выполнению работы**

1. Должна быть реализована функция Convolution 2D. Входные параметры функции должны соответствовать входным параметрам функции Conv2d библиотеки PyTorch;
2. Должны быть разработаны группы тестов, для проверки реализованного функционала;
3. Составить отчет по лабораторной работе, включающий краткое описание выполненной работы, ссылки на место хранения разработанного алгоритма и существенные замечания, возникшие в ходе выполнения работ;

Выполнение работы подтверждается итоговым отчетом, включающим в себя актуальную информацию из отчетов по каждому из этапов работы.

**Теория**

Операция свертки в трехмерном пространстве (conv3d) широко используется в глубоком обучении для обработки трехмерных данных, таких как видео и медицинские изображения (например, данные компьютерной томографии). Она расширяет концепцию двумерной свертки на третье измерение, что позволяет модели работать с объемными данными.

Пусть I — это трехмерный входной тензор, а K - трехмерное ядро свертки. Операция трехмерной свертки обозначается как I∗K и определяется следующим образом:

,

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

Аналогично двумерной свертке, в глубоком обучении для трехмерной свертки также используются ядра, имеющие веса, определенные в процессе обучения. Также может присутствовать параметр смещения (bias). Обновленная формула будет выглядеть следующим образом:

,

где B — это трехмерный тензор смещения. Операция трехмерной свертки выполняется по всем каналам входного тензора и для всех фильтров (фильтры соответствуют выходным каналам) одновременно. Трехмерная свертка широко используется в моделях, обрабатывающих объемные данные, где важными являются пространственные и временные зависимости между данными.

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 *Conv3DSelf* со следующими функциями:

* \_\_init\_\_ - функция, для инициализации объекта класса. Внутри нее происходит проверка на соответствие входных параметров к необходимым типам.
* conv3d – функция, в которой был реализован алгоритм двумерной свертки.
* torch\_conv3d – 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,

        in\_channels: int,

        out\_channels: int,

        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.numpy()

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

        self.bias = bias

        self.in\_channels, self.out\_channels = in\_channels, out\_channels

        if type(kernel\_size) == tuple:

            self.kernel\_size = kernel\_size

        else:

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

        if type(stride) == tuple:

            self.stride = stride

        else:

            self.stride = (stride, stride, stride)

        if type(dilation) == tuple:

            self.dilation = dilation

        else:

            self.dilation = (dilation, dilation, dilation)

        if type(padding) == tuple:

            self.padding = padding

        elif padding == "same":

            if self.stride[0] != 1 or self.stride[1] != 1 or self.stride[2] != 1:

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

            self.padding = (self.kernel\_size[0]-1,self.kernel\_size[1]-1,self.kernel\_size[2]-1)

        elif padding == "valid":

            self.padding = (0,0,0)

        else:

            self.padding = (padding,padding,padding)

        self.weight\_tensor\_for\_torch = torch.randn(1, 1, self.kernel\_size[0], self.kernel\_size[1], self.kernel\_size[2])

        self.weight\_tensor = self.weight\_tensor\_for\_torch.numpy()

Исходный код функции conv3d представлен в листинге 2.

Листинг2. Код conv2d.

def conv3d(self):

        batches = len(self.input\_data)

        out = []

        for b in range(batches):

            d\_in = self.input\_data[b].shape[1]

            h\_in = self.input\_data[b].shape[2]

            w\_in = self.input\_data[b].shape[3]

            if self.kernel\_size[0] > h\_in or self.kernel\_size[1] > w\_in or self.kernel\_size[2] > d\_in:

                raise ValueError('kernel size can\'t be greater than input size')

            d\_out = int(

                (d\_in + 2 \* self.padding[0] - self.dilation[0] \* (self.kernel\_size[0] - 1) - 1) / (self.stride[0]) + 1)

            h\_out = int(

                (h\_in + 2 \* self.padding[1] - self.dilation[1] \* (self.kernel\_size[1] - 1) - 1) / (self.stride[1]) + 1)

            w\_out = int(

                (w\_in + 2 \* self.padding[2] - self.dilation[2] \* (self.kernel\_size[2] - 1) - 1) / (self.stride[2]) + 1)

            out.append(np.zeros((self.out\_channels, d\_out, h\_out, w\_out)))

            for c\_out in range(self.out\_channels):

                for z\_out in range(d\_out):

                    for y\_out in range(h\_out):

                        for x\_out in range(w\_out):

                            sum = 0

                            for c\_in in range(self.in\_channels):

                                for kernel\_z in range(self.kernel\_size[0]):

                                    for kernel\_y in range(self.kernel\_size[1]):

                                        for kernel\_x in range(self.kernel\_size[2]):

                                            z\_in = z\_out \* self.stride[0] + kernel\_z \* self.dilation[0] - self.padding[0]

                                            y\_in = y\_out \* self.stride[1] + kernel\_y \* self.dilation[1] - self.padding[1]

                                            x\_in = x\_out \* self.stride[2] + kernel\_x \* self.dilation[2] - self.padding[2]

                                            if 0 <= z\_in < d\_in and 0 <= y\_in < h\_in and 0 <= x\_in < w\_in:

                                                sum += self.input\_data[b][c\_in][z\_in][y\_in][x\_in] \* self.weight\_tensor[c\_out][c\_in][kernel\_z][kernel\_y][kernel\_x]

                            out[b][c\_out][z\_out][y\_out][x\_out] = sum + (self.bias if self.bias else 0)

        return np.array(out)

Исходный код функции test представлен в листинге 3.

Листинг 3. Код test.

    def test(self, print\_flg=False):

        my\_conv3d = self.conv3d()

        torch\_out = self.torch\_conv3d().squeeze().detach().numpy()

        if print\_flg:

            print(my\_conv3d[0][0])

            print(torch\_out)

        print(np.allclose(my\_conv3d[0][0], torch\_out))

Также, в рамках работы было произведено тестирование алгоритма, Результаты тестирования представлены на рисунке 1.

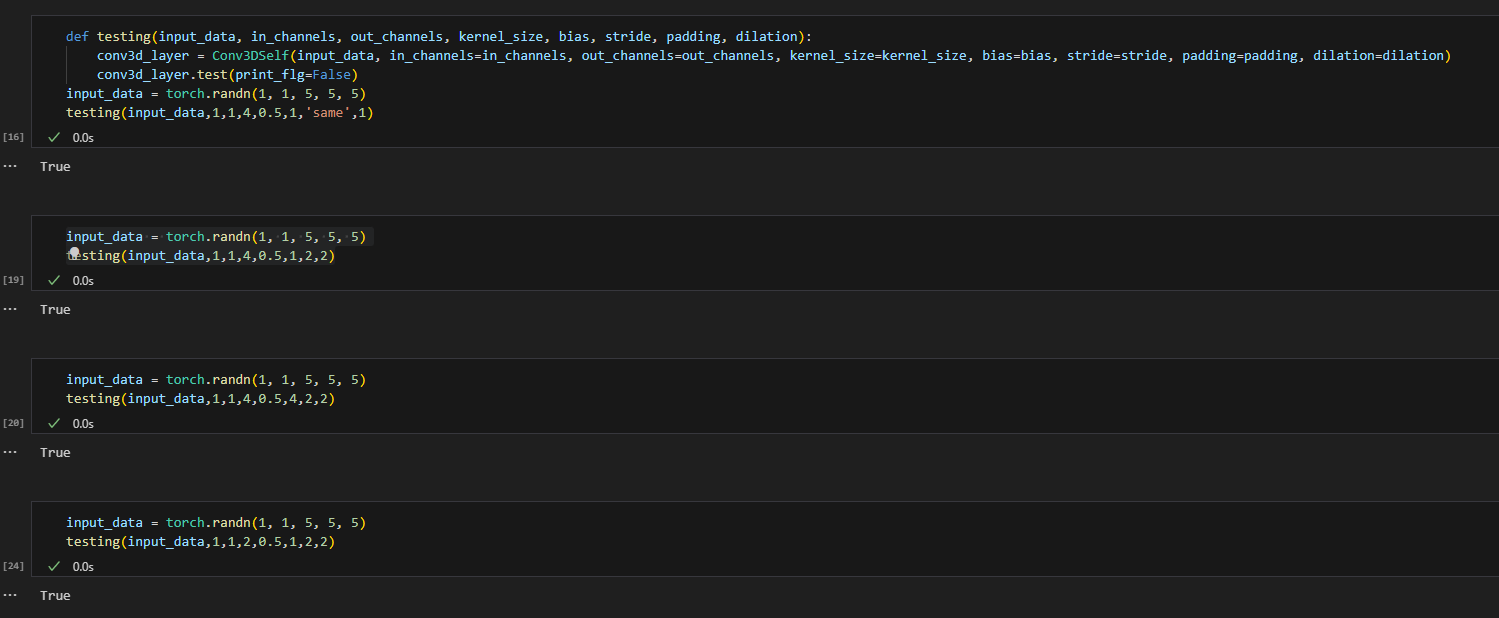


Рисунок 1. Результат тестирования

**Вывод**

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

Ссылка на git: