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»

Выполнил:

студент группы БВТ2002

Митрофанов А.О.

Оглавление

[Цель работы 3](#_Toc153003983)

[Задание 3](#_Toc153003984)

[Требования к выполнению работы 3](#_Toc153003985)

[Ход работы 4](#_Toc153003986)

[Вывод 8](#_Toc153003987)

[Приложение 8](#_Toc153003988)

# Цель работы

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

# Задание

При выполнении лабораторной работы необходимо:

* ознакомиться с описанием операции библиотеки PyTorch (https://pytorch.org/docs/stable/generated/torch.nn.Conv3d);
* используя язык программирования Python написать алгоритм, реализующий алгоритм свертки;
* составить отчет по лабораторной работе;

# Требования к выполнению работы

1. Должна быть реализована функция Convolution 3D. Входные параметры функции должны соответствовать входным параметрам функции Conv3d библиотеки PyTorch;
2. Layout для данной операции принимается ‘NHWDC’.
3. Должны быть разработаны группы тестов, для проверки реализованного функционала;
4. Составить отчет по лабораторной работе, включающий краткое описание выполненной работы, ссылки на место хранения разработанного алгоритма и существенные замечания, возникшие в ходе выполнения работ;

# Ход работы

В ходе лабораторной работы была реализована работа функции для двумерной свёртки. За основу была взята функция свёртки conv3D библиотеки pytorch. Опираясь на документацию библиотеки, была реализована функция convolution3D, принимающая те же аргументы, что и оригинальная функция: input, weights, padding, dilation, stride, groups. Реализация данной функции представлена ниже:

*# функция, выделяющая все скользящие окна из входного тензора*

def sliding\_window\_3d(arr, window\_shape, stride = (1,1,1), dilation=(1,1,1), padding=0):

arr\_padded = np.pad(arr, padding)

print(f'paded: {arr\_padded.shape}')

dilated\_filled\_shape = np.array(window\_shape) + ((np.array(dilation)-1) \* (np.array(window\_shape)-1))

window\_view = np.lib.stride\_tricks.sliding\_window\_view(arr\_padded, dilated\_filled\_shape)

shrink = lambda x: x[::, ::, ::, ::dilation[0], ::dilation[1], ::dilation[2]]

shrinked\_windows = shrink(window\_view)

shrinked\_windows = window\_view[::, ::, ::, ::dilation[0], ::dilation[1], ::dilation[2]]

strided = shrinked\_windows[::stride[0], ::stride[1], ::stride[2]]

print(f'strided: {strided.shape}')

return strided

def dim\_window\_count(input\_size, kernel\_size, stride, padding, dilation, dim):

return floor(

(

input\_size[dim]

+ 2 \* padding

- dilation[dim] \* (kernel\_size[dim] - 1)

- 1

)

/ stride[dim]

+ 1

)

def custom\_conv3d(

input: Tensor,

weights: Tensor,

bias: Tensor | None = None,

stride: tuple | int = 1,

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

dilation: tuple | int = 1,

groups: int = 1,

):

if len(input.shape) < 4 or len(input.shape) > 5:

raise ValueError(

f"Expected 3D (unbatched) or 4D (batched) input to conv2d, but got input of size: {input.shape}"

)

if len(input.shape) == 4:

input = input.unsqueeze(0)

print(f"unsqueezed input: {input.shape}")

if len(weights.shape) != 5:

raise ValueError(

f"Expected 4D weights with shape [out\_channels, in\_channels/groups, kH, kW], but got weights with shape: {weights.shape}"

)

if bias is not None and bias.shape[0] != weights.shape[0]:

raise ValueError(

f"Expected bias shape to be [out\_channels], but got bias with shape: {bias.shape}"

)

batch\_size, input\_channels, input\_d, input\_h, input\_w = input.shape

output\_channels, kernel\_channels, kernel\_d, kernel\_h, kernel\_w = weights.shape

if input\_channels % groups != 0 or output\_channels % groups != 0:

raise ValueError(

f"Groups should be divisible by both input\_channels and output\_channels. Got:\ninput channels: {input\_channels}\noutput channels: {output\_channels}\ngroups: {groups}"

)

if kernel\_channels != input\_channels // groups:

raise ValueError(

f"Expected kernel channels to be [input\_channels/groups], got:\n\

input channels: {input\_channels}\n\

groups: {groups}\n\

output channels: {output\_channels}"

)

if type(dilation) is int:

dilation = (dilation, dilation, dilation)

if type(stride) is int:

stride = (stride, stride, stride)

output\_d = dim\_window\_count((input\_d, input\_h, input\_w), (kernel\_d, kernel\_h, kernel\_w), stride, padding, dilation, 0)

output\_h = dim\_window\_count((input\_d, input\_h, input\_w), (kernel\_d, kernel\_h, kernel\_w), stride, padding, dilation, 1)

output\_w = dim\_window\_count((input\_d, input\_h, input\_w), (kernel\_d, kernel\_h, kernel\_w), stride, padding, dilation, 2)

print(f'calculated output shape: {output\_d}, {output\_h}, {output\_d}')

output = torch.zeros(size=(batch\_size, output\_channels, output\_d, output\_h, output\_w))

windows\_per\_input\_channel = output\_d\*output\_h\*output\_w

for img\_num, \_ in enumerate(input):

for kernel\_num, kernel in enumerate(weights):

group\_num = kernel\_num // (output\_channels // groups)

input\_channels\_start = group\_num \* (input\_channels // groups)

for input\_ch\_num in range(

input\_channels\_start, input\_channels\_start + input\_channels // groups

):

kernel\_ch\_num = input\_ch\_num % kernel\_channels

kernel\_area = kernel\_h \* kernel\_w

windows = sliding\_window\_3d(input[img\_num, input\_ch\_num, :, :, :], (kernel\_d, kernel\_h, kernel\_w), stride=stride, dilation=dilation, padding=padding)

current\_input\_channel\_windows = torch.from\_numpy(windows.reshape(windows\_per\_input\_channel, kernel\_d, kernel\_h, kernel\_w))

product = current\_input\_channel\_windows \* kernel[kernel\_ch\_num]

weighted\_sum = torch.sum(product, dim=(1, 2, 3), keepdim=True)

weighted\_sum = weighted\_sum.reshape(output\_d, output\_h, output\_w)

output[img\_num, kernel\_num] += weighted\_sum

if bias is not None:

for batch in output:

for i, layer in enumerate(batch):

layer += bias[i]

return output

В данной функции были реализованы: padding – параметр, отвечающий за дополнение тензора нулями по краям, bias – нейрон смещения, stride – шаг, с которым ядро будет проходить по входному тензору, groups – число групп, на которые разбивается входной тензор.

После чего с помощью библиотеки pytest были реализованы тесты, отвечающие за проверку корректности выполненной работы.

Далее приведены сами тесты:

%%ipytest

def test\_1():

stride = 1

dilation = 1

padding = 0

groups = 1

bias = torch.Tensor([1,2,3,4])

filters = torch.rand((4, 8, 8, 8, 8)).float()

inputs = torch.rand((2, 8, 16, 16, 16)).float()

conv = F.conv3d(inputs, filters, stride=stride, dilation=dilation, padding=padding, groups=groups, bias=bias)

my\_conv = custom\_conv3d(inputs, filters, stride=stride, dilation=dilation, padding=padding, groups=groups, bias=bias)

assert torch.allclose(conv, my\_conv)

def test\_2():

stride = 2

dilation = 2

padding = 0

groups = 1

bias = torch.Tensor([1,2,3,4])

filters = torch.rand((4, 8, 8, 8, 8)).float()

inputs = torch.rand((2, 8, 16, 16, 16)).float()

conv = F.conv3d(inputs, filters, stride=stride, dilation=dilation, padding=padding, groups=groups, bias=bias)

my\_conv = custom\_conv3d(inputs, filters, stride=stride, dilation=dilation, padding=padding, groups=groups, bias=bias)

assert torch.allclose(conv, my\_conv)

def test\_3():

stride = 2

dilation = 3

padding = 2

groups = 2

inp = 16

kern = 3

bias = torch.Tensor([1,2,3,4])

filters = torch.rand((4, 4, kern, kern, kern)).float()

inputs = torch.rand((2, 8, inp, inp, inp)).float()

conv = F.conv3d(inputs, filters, stride=stride, dilation=dilation, padding=padding, groups=groups, bias=bias)

my\_conv = custom\_conv3d(inputs, filters, stride=stride, dilation=dilation, padding=padding, groups=groups, bias=bias)

assert torch.allclose(conv, my\_conv)

Сравнение результатов изначальной функции и функции библиотеки pytorch было проведено с помощью функционала этой же библиотеки, а именно функции allclose, отвечающей за приблизительное сравнение полученных обоими функциями значений, т.к. полная эквивалентность чисел с плавающей запятой не гарантирована.

Все 4 реализованных теста прошли успешно, что можно увидеть на рисунке 1.

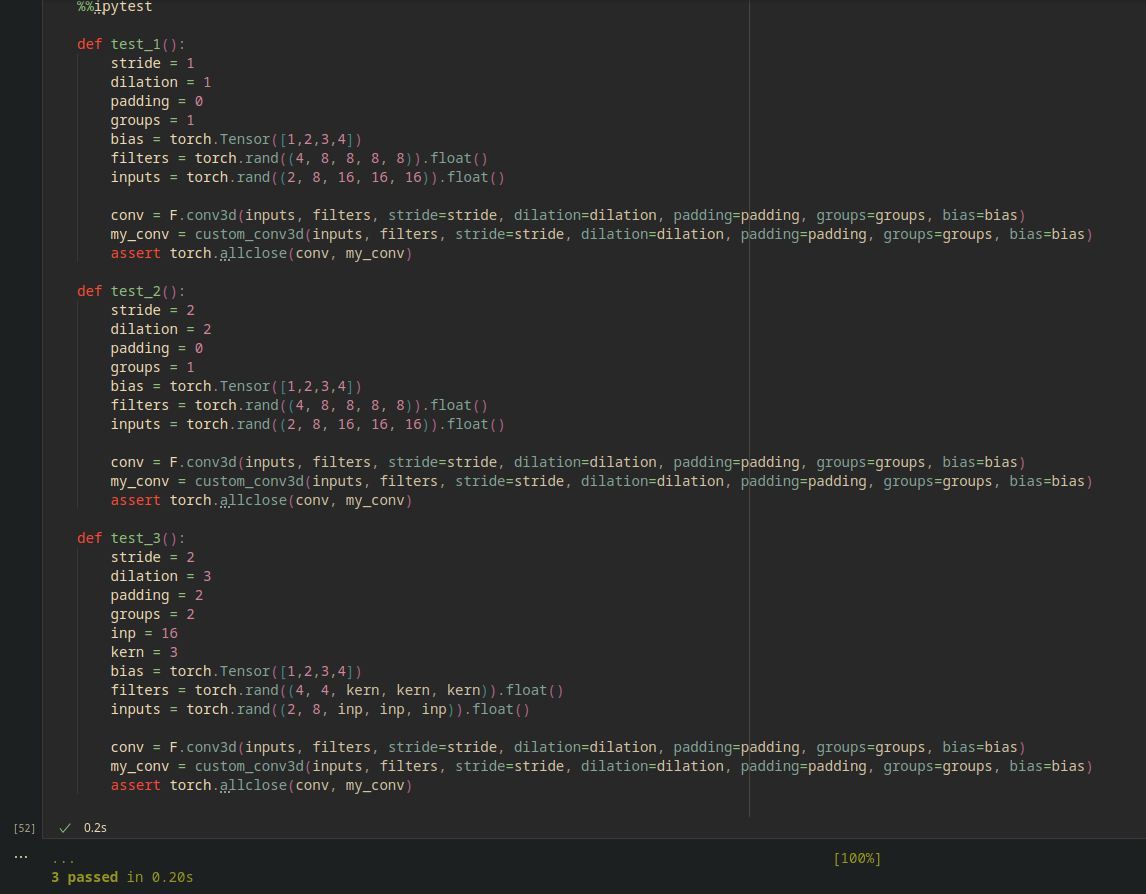


Рисунок 1 – Результат работы тестов.

# Вывод

В ходе работы была реализована функция трёхмерной свёртки, включающая в себя большую часть возможностей оригинала. Кроме того, были проведены тесты с успешным результатом, что означает корректность работы реализованного функционала.

# Приложение

Ссылка на репозиторий гитхаба с реализованным функционалом: <https://github.com/d1agnozzz/CV_labs>