**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 №3

по дисциплине «Системы машинного зрения»

на тему:

«Разработка нейросетевых функций. Операция Convolution Transpose»

Выполнила: студент группы БВТ2001

Щавлев К.В.

**Цель работы**

Разработать алгоритм, используя язык python, реализующий работу операции транспонированной 2d свертки.

**Задание**

При выполнении лабораторной работы необходимо:

1. ознакомиться с описанием операции библиотеки PyTorch (https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d);
2. используя язык программирования Python написать алгоритм, реализующий алгоритм свертки;
3. составить отчет по лабораторной работе;

**Требования к выполнению работы**

1. Должна быть реализована функция Convolution Transpose. Входные параметры функции должны соответствовать входным параметрам функции ConvTranspose 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;
4. Доп. Задание: реализовать алгоритм работы транспонированной свертки, через алгоритм двумерной свертки, реализованный в первой лабораторной. Необходимо перерассчитать входные параметры. Выполнение данного задания засчитывается как выполнение лабораторной № 4.

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

**Теория**

Операция транспонированной свертки (conv\_transposed2d), также известная как деконволюция, транспонированная свертка или транспонированная сверточная операция, представляет собой обратную операцию по отношению к свертке. Она используется для увеличения размерности входных данных.

Пусть I - это входной тензор, K - ядро свертки, B - параметр смещения (bias), а O - выходной тензор. Операция транспонированной свертки обозначается как и определяется следующим образом:

,

где - это ядро свертки, транспонированное и дополненное нулями (s-padding). Важно отметить, что транспонированная свертка увеличивает размерность данных.

Различия между сверткой и транспонированной сверткой:

* Операция транспонированной свертки увеличивает размерность: В то время как обычная свертка уменьшает размерность данных, транспонированная свертка увеличивает ее.
* Использование транспонированного ядра: В транспонированной свертке используется транспонированное ядро (или ядро с "зеркальным" отражением).
* Дополнение нулями (s-padding): Транспонированная свертка часто использует дополнение нулями для учета перекрытия ядра.

В глубоком обучении транспонированная свертка часто используется для задач, таких как апсемплинг, генерация изображений и восстановление данных. Важно правильно настроить параметры транспонированной свертки, такие как размер ядра, шаг и дополнение, чтобы получить желаемый результат.

**Выполнение работы**

В качестве собственной реализации свертки convolution\_transposed2d из библиотеки pytorch, был реализована функция *conv2d\_transposed.* Ее код можно наблюдать в листинге 1.

Листинг 1. Код функции \_\_init\_\_.

def conv2d\_transposed(

    matrix, in\_channels, out\_channels, kernel\_size,

    stride=1, padding=0, output\_padding=0, dilation=1,

    bias=True, padding\_mode='zeros'

):

    #генерация bias

    if bias == True:

      bias\_val = torch.rand(out\_channels)

    else:

      bias\_val = torch.zeros(out\_channels)

    #padding\_mode

    if (padding\_mode != 'zeros'):

      raise ValueError('only "zeros" padding\_mode in ConvTranspose2d')

    #генерация ядра

    if type(kernel\_size) == tuple:

      weights = torch.rand(in\_channels, out\_channels, kernel\_size[0], kernel\_size[1])

    if type(kernel\_size) == int:

      weights = torch.rand(in\_channels, out\_channels, kernel\_size, kernel\_size)

    res\_tensor = []

    for l in range(out\_channels):

      feature\_map = torch.zeros((matrix.shape[1]-1)\*stride + dilation \* (kernel\_size-1)+1, (matrix.shape[2]-1)\*stride  + dilation \* (kernel\_size-1)+1 ) #генерация пустой feature-map

      for c in range (in\_channels):

        for i in range (0, matrix.shape[1]):  #проход по всем пикселям изображения

          for j in range (0, matrix.shape[2]):

            val = matrix[c][i][j]

            proizv = val\*weights[c][l]

            zero\_tensor = torch.zeros((weights.shape[2]-1)\*dilation+1, (weights.shape[3]-1)\*dilation+1)

            for a in range (0, zero\_tensor.shape[0], dilation):

              for b in range (0, zero\_tensor.shape[1], dilation):

                zero\_tensor[a][b] = proizv[a//dilation][b//dilation]

            res = np.add((zero\_tensor), feature\_map[i\*stride:i\*stride+(weights.shape[2]-1)\*dilation+1, j\*stride:j\*stride+(weights.shape[3]-1)\*dilation+1])

            feature\_map[i\*stride:i\*stride+(weights.shape[2]-1)\*dilation+1, j\*stride:j\*stride+(weights.shape[3]-1)\*dilation+1] = res

      res\_tensor.append(np.add(feature\_map, np.full((feature\_map.shape), bias\_val[l])))

    for t in range(len(res\_tensor)):

      if output\_padding > 0:

        pad\_func = torch.nn.ConstantPad1d((0, output\_padding, 0, output\_padding), 0)

        res\_tensor[t] = pad\_func(res\_tensor[t])

      res\_tensor[t] = res\_tensor[t][0+padding:res\_tensor[t].shape[0]-padding, 0+padding:res\_tensor[t].shape[1]-padding]

    return res\_tensor, weights, torch.tensor(bias\_val)

Также была написана функция для сравнения выходных данных, исходный код функции compare представлен в листинге 2.

Листинг2. Код compare.

def compare(tensor, in\_channels, out\_channels, kernel\_size, stride, padding, output\_padding, dilation, bias=True, padding\_mode='zeros'):

    myres, kernel, bias\_val = conv2d\_transposed(

        tensor,

        in\_channels=in\_channels, out\_channels=out\_channels,

        kernel\_size=kernel\_size, stride=stride,

        padding=padding, output\_padding=output\_padding,

        dilation=dilation, bias=bias,

        padding\_mode=padding\_mode,

        )

    torchFunction = torch.nn.ConvTranspose2d(

        in\_channels=in\_channels, out\_channels=out\_channels,

        kernel\_size=kernel\_size, stride=stride,

        padding=padding, output\_padding=output\_padding,

        dilation=dilation, bias=bias,

        padding\_mode=padding\_mode,

        )

    torchFunction.weight.data = kernel

    torchFunction.bias.data = bias\_val

    result = str(np.round([tensor.tolist() for tensor in myres],2))

    torch\_res = str(np.round(torchFunction(tensor).data.numpy(),2))

    result==torch\_res

Также, в рамках работы было произведено тестирование алгоритма, результат тестирования представлен на рисунке 1.

Результаты тестирования представлены на рисунке 1.

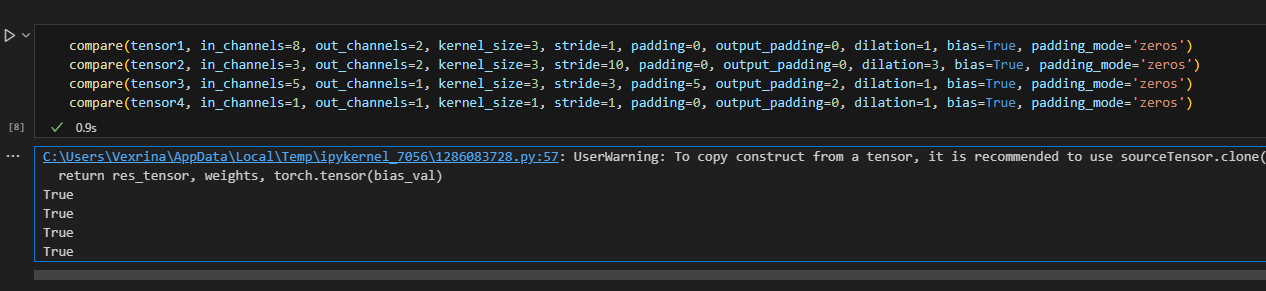


Рисунок 1. Результат тестирования

**Дополнительное задание.**

В качестве дополнительного задания, надо было реализовать conv\_transposed2d используя свертку из первой лабораторной работы. Немного поработав над сверткой из первой ЛР, получил следующий листинг conv\_transposed2d, листинг 3.

Листинг 3. Код cooler\_transpconv2d.

def cooler\_transpconv2d(

  input, in\_channels, out\_channels, kernel\_size,

  transp\_stride=1, padding=0, dilation=1, bias=True,

  padding\_mode='zeros'

):

  stride = 1

  pad = kernel\_size - 1

  result\_input = []

  for matr in input:

    zero\_tensor = np.zeros((((matr.shape[0]-1)\*(transp\_stride)+1), ((matr.shape[1]-1)\*(transp\_stride)+1)))

    for a in range (0, zero\_tensor.shape[0], transp\_stride):

      for b in range (0, zero\_tensor.shape[1], transp\_stride):

        zero\_tensor[a][b] = matr[a//(transp\_stride)][b//(transp\_stride)]

    pad\_matr = np.pad(zero\_tensor, pad\_width=pad, mode='constant')

    result\_input.append(pad\_matr)

  input = torch.tensor(result\_input)

  if bias == True:

    bias\_val = torch.rand(out\_channels)

  else:

    bias\_val = torch.zeros(out\_channels)

  if (padding\_mode == 'zeros'):

    pad = torch.nn.ZeroPad2d(padding)

    input = pad(input)

  if (padding\_mode == 'reflect'):

    pad = torch.nn.ReflectionPad2d(padding)

    input = pad(input)

  if (padding\_mode == 'replicate'):

    pad = torch.nn.ReplicationPad2d(padding)

    input = pad(input)

  if (padding\_mode == 'circular'):

    pad = torch.nn.CircularPad2d(padding)

    input = pad(input)

  weights = np.array(torch.rand(out\_channels, in\_channels, kernel\_size, kernel\_size))

  weights\_for\_transpose = []

  for j in range(out\_channels):

    weights\_in = []

    for i in range(in\_channels):

      weights\_in.append(np.flip(np.array(weights[j][i])))

    weights\_for\_transpose.append(weights\_in)

  weights\_for\_transpose = torch.tensor(weights\_for\_transpose)

  weights\_for\_transpose = weights\_for\_transpose.reshape(in\_channels, out\_channels, kernel\_size, kernel\_size)

  res\_tensor = []

  for l in range(out\_channels):

    feature\_map = np.array([])

    for i in range (0, input.shape[1]-((weights.shape[2]-1)\*dilation+1)+1, stride):

      for j in range (0, input.shape[2]-((weights.shape[3]-1)\*dilation+1)+1, stride):

        summa = 0

        for c in range (in\_channels):

          val = input[c][i:i+(weights.shape[2]-1)\*dilation+1:dilation, j:j+(weights.shape[3]-1)\*dilation+1:dilation]

          mini\_sum = (val\*weights[l][c]).sum()

          summa = summa + mini\_sum

        feature\_map = np.append(feature\_map, float(summa + bias\_val[l]))

    res\_tensor.append(feature\_map.reshape((input.shape[1]-((weights.shape[2]-1)\*dilation+1))//stride+1, (input.shape[2]-((weights.shape[3]-1)\*dilation+1))//stride+1))

  return np.array(res\_tensor), torch.tensor(np.array(weights\_for\_transpose)), torch.tensor(np.array(bias\_val))

Также было произведено тестирование алгоритма, код функции сравнения представлен в листинге 4.

Листинг 4. Код cooler\_compare.

def cooler\_compare(tensor, in\_channels, out\_channels, kernel\_size, stride, transp\_stride, bias=True,):

    myres, kernel, bias\_val = cooler\_transpconv2d(

        tensor,

        in\_channels=in\_channels,

        out\_channels=out\_channels,

        kernel\_size=kernel\_size,

        transp\_stride=transp\_stride,

        bias=bias,

        )

    torchFunction = torch.nn.ConvTranspose2d(

        in\_channels=in\_channels,

        out\_channels=out\_channels,

        kernel\_size=kernel\_size,

        stride=stride,

        bias=bias,

        )

    torchFunction.weight.data = kernel

    torchFunction.bias.data = bias\_val

    result = str(np.round(myres, 2))

    torch\_res = str(np.round(torchFunction(tensor).data.numpy(),2))

    print(result==torch\_res)

Результаты тестирования представлены на рисунке 2.

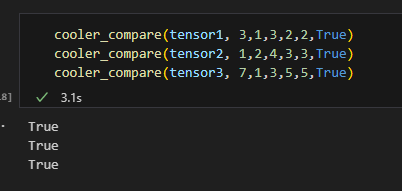


Рисунок 2. Результаты тестирования.

**Вывод**

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

Ссылка на git: