تحویل روز چهارشنبه مورخ ۱۳ تیر ۱۴۰۳ به صورت حضوری

- **تحویل این پروژه به صورت حضوری بوده و در صورت انجام گروهی (هر گروه حداکثر دو نفر) اعضای گروه بایستی به صورت جداگانه ارائه داده و به تمامی بخش‌ها اشراف کامل داشته باشند.**
- **گزارش پروژه به همراه سورس ایجاد شده بایستی در سامانه دروس بارگذاری شود.**
- **همکاری و همفکری با گروه‌های دیگر مجاز می‌باشد اما هر گروه می‌بایست پیاده‌سازی منحصر به فرد خود را داشته باشد.**
- **نمره پروژه بنا بر میزان تسلط و همچنین نحوه عملکرد به اعضا داده خواهد شد و در انتها به گروهی که دارای کد با سرعت بیشتر و یا روش ابتکاری انجام کار باشد امتیاز اضافه تعلق خواهد گرفت.**

چنانچه ابهامی در زمینه تمرینات دارید، می‌توانید اشکالات خود را از طریق پست الکترونیکی زیر با موضوع PDS-2024 مطرح نمایید.

h.malakouty@aut.ac.ir

ملکوتی

موفق و پیروز باشید

تحویل روز چهارشنبه مورخ ۱۳ تیر ۱۴۰۳ به صورت حضوری

لایه کانوولوشن (Convolutional Layer) یکی از اجزای اصلی شبکه‌های عصبی پیچشی (CNN) است که به‌طور گسترده در تحلیل تصاویر و داده‌های چندبعدی استفاده می‌شود. از این لایه برای استخراج ویژگی‌ها از تصویر جهت استفاده در لایه‌های بعدی استفاده می‌شود و یکی از مهمترین روش‌های به‌کارگیری شده در راستای افزایش دقت در پردازش تصاویر با کمک شبکه‌های عصبی مصنوعی است. با توجه به استفاده از تعداد بسیار زیاد این عملیات در شبکه‌های CNN، قسمت زیادی از بار پردازشی را در این نوع شبکه‌ها توسط این لایه ایجاد می‌شود و لذا سرعت‌بخشی به اجرای آن با کمک سخت‌افزار بسیار پرکاربرد می‌باشد.

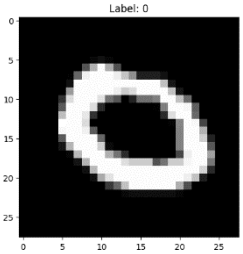
در ادامه، با استفاده از شکل‌ها و توضیحات مربوطه، نحوه کار این لایه را بررسی می‌کنیم.

بخش‌های لایه کانوولوشن

هر لایه کانوولوشن را می‌توان به صورت زیر بررسی نمود:

ورودی (Input):

ورودی به لایه کانوولوشن معمولاً یک تصویر یا یک ماتریس چند بعدی است. به عنوان مثال، یک تصویر خاکستری ۲۸ در ۲۸ را در نظر بگیرید که هر پیکسل آن میتواند از ۰ تا ۲۵۵ میزان روشنایی را داشته باشد:



که معادل ماتریس زیر است :

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   9  17   8   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0  84 178 253 172  80   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   3 103 241 253 253 253 238 197 144  33  33  33  18   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0  17 253 253 253 253 253 253 253 253 253 253 253 253 253 194  24   0   0   0   0   0   0]
[  0   0   0   0   0   0 155 253 253 253 253 253 253 240 204 220 253 253 253 253 194  24   0   0   0   0   0   0]
[  0   0   0   0   0  50 220 253 253 253 253 226 114  85   0  37 114 114 128 253 253 194 109   0   0   0   0   0]
[  0   0   0   0   0  91 253 253 253 253 221  45   0   0   0   0   0   0   3 128 253 253 248  96   0   0   0   0]
[  0   0   0   0   0 191 253 253 233  50   0   0   0   0   0   0   0   0  45 223 253 253 106   0   0   0   0   0]
[  0   0   0   0   0 255 253 253 220   0   0   0   0   0   0   0   0   0   0   6 134 253 249  68   0   0   0   0]
[  0   0   0   0   0 249 253 253 242  93   2   0   0   0   0   0   0   0   0   0 131 253 253 179   0   0   0   0]
[  0   0   0   0   0  91 253 253 253 253 104   0   0   0   0   0   0   0   0   0 131 253 253 208  35   0   0   0]
[  0   0   0   0   0  23 183 253 253 253 208  12   0   0   0   0   0   0   0   0 131 253 253 253  89   0   0   0]
[  0   0   0   0   0   0   9 186 253 253 253 177 116  45   0   0   0   0   0   0  72 239 253 253 163   0   0   0]
[  0   0   0   0   0   0   0  21 181 253 253 253 253 223 205 205 205  83 116 205 205 247 253 253 120   0   0   0]
[  0   0   0   0   0   0   0   0  16 185 253 253 253 253 253 253 253 253 253 253 253 253 253 214  43   0   0   0]
[  0   0   0   0   0   0   0   0   0  16 170 227 253 253 253 253 253 253 253 253 253 240 195  32   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0  59 106 196 253 253 253 253 253 253 176  82   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0  10  15  15  15  15  15  15   8   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
```

تحویل روز چهارشنبه مورخ ۱۳ تیر ۱۴۰۳ به صورت حضوری

به طور کلی این تصویر می‌تواند کوچکتر یا بزرگتر از مثال داده شده بوده و همچنین دارای چندین کانال مجزا برای تصاویر رنگی (هر یک، یک ماتریس مجزا) باشد اما برای سادگی در این پروژه از داده با سایز ۲۸ در ۲۸ (مانند مجموعه داده MNIST[1]) استفاده می‌شود.

برای سادگی بخش کوچکی از این ماتریس را جدا کرده و عملیات کانوولوشن را بر روی آن توضیح می‌دهیم :

[0 84 178 253 172 80]

[103 241 253 253 253 238]

[253 253 253 253 253 253]

[253 253 253 253 253 240]

[253 253 253 226 114 85]

[253 253 221 45 0 0]

فیلتر (Kernel/Filter):

یک فیلتر (یا کرنل) یک ماتریس کوچک است که روی تصویر اصلی جابجا می‌شود تا با انجام کانوولوشن ویژگی‌های خاصی را استخراج کند. یک فیلتر فرضی ۳ در ۳ را در نظر بگیرید:

[1 0 -1]

[1 0 -1]

[1 0 -1]

پس از انتخاب کرنل کانوولوشن به صورت زیر انجام می‌شود :

$$Conv2D(i,j) = \sum_m \sum_n I(m,n)\, K(i-m,\ j-n)$$

که در آن I تصویر ورودی و K کرنل انتخابی است.

برای انجام کانوولوشن یک قسمت ۳ در ۳ از ماتریس انتخابی را جدا کرده و فرمول بالا را بر روی آن انجام می‌دهیم :

[0 84 178]

[103 241 253]

[253 253 253]

[1] - http://yann.lecun.com/exdb/mnist/

(0*1 + 84*0 + 178*(-1)) +

(103*1 + 241*0 + 253*(-1)) +

(253*1 + 253*0 + 253*(-1))

= (0 + 0 - 178) + (103 + 0 - 253) + (253 + 0 - 253) = -178 - 150 + 0 = -328

نتیجه نهایی برای کل ماتریس جدا شده به صورت زیر خواهد بود :

[-328. -181. 6. 188.]

[-150. -12. 0. 28.]

[0. 27. 139. 154.]

[32. 235. 360. 199.]

با توجه به بزرگ بودن تصاویر ورودی و تعداد زیاد کرنل‌های مورد نیاز برای عملیات بر روی آن هدف در این پروژه ایجاد یک شتاب‌دهنده برای اجرای عملیات کانوولوشن است.

برای این امر تصویر و کرنل به صورت دو ماتریس ورودی ۲۸ در ۲۸ و ۳ در ۳ داده شده و نتیجه آن پس از پردازش توسط این واحد شتاب‌دهنده در یک آرایه خروجی ذخیره می‌شود.

به طور کلی انجام کانوولوشن به سادگی با یک سری حلقه تو در تو قابل انجام است و هدف در این پروژه کار با ابزار Vitis + Vivado جهت پیاده‌سازی سخت‌افزاری آن می‌باشد.

نکات انجام پروژه:

برای انجام پروژه کد پایتون زیر را به عنوان نمونه بررسی کنید و پس از تبدیل به صورت C جهت استفاده در ابزار HLS و یا VHDL/Verilog استفاده نمایید.

در صورت استفاده از BRAM و یا حافظه توزیع شده (به‌جای حافظه خارجی)، سرعت اجرا به صورت قابل توجهی افزایش می‌یابد.

با توجه به منابع FPGA می‌توان کل یا بخش اعظم عملیات را به صورت موازی انجام داد.

با استفاده از ضرب‌کننده‌های سخت‌افزاری سرعت عملیات افزایش زیادی خواهد یافت.

تحویل روز چهارشنبه مورخ ۱۳ تیر ۱۴۰۳ به صورت حضوری

```
import numpy as np
# Define the 6x6 matrix
matrix = np.array([
[    0 ,  84 , 178, 253, 172,  80 ],
[ 103 ,241 ,253 ,253 ,253 ,238] ,
[ 253 ,253 ,253 ,253 ,253 ,253] ,
[ 253 ,253 ,253 ,253 ,253 ,240],
[ 253 ,253 ,253 ,226 ,114  ,85 ] ,
[ 253 ,253 ,221 , 45  , 0 ,  0     ]
])
# Define the 3x3 kernel
kernel = np.array([
   [1, 0, -1],
   [1, 0, -1],
   [1, 0, -1]
])
# Create a new matrix to store the result
result = np.zeros((4, 4))
# Perform the convolution
for i in range(4):
   for j in range(4):
      for k in range(3):
         for l in range(3):
            result[i, j] += int(matrix[i + k, j + l] * kernel[k, l])


# Print the result
print(result)
```