# سالی و وازوفسکی

**توجه:** تابع سوم (کانولوشن) فقط ۲۰ درصد از نمره سوال را تشکیل می‌دهد. در صورت عدم تمایل می‌توانید آن را پیاده نکنید.

---

سالیوان و وازوفسکی، پست جدیدی در شرکت هیولاها یک ماموریت جدید دارند. شرکت هیولاها مشغول ساختن رقیب Snapchat بوده و به تخصص چند برنامه‌نویس نیاز دارند. آنها شما را به عنوان نیروی پیشنهادی شرکت انتخاب کرده‌اند تا دستی در تولید این محصول داشته باشید!

در شرکت هیولاها به شما وظیفه نوشتن سه تابع را داده‌اند که قرار است در بخش اعمال فیلتر روی عکس‌ها استفاده شوند! به ترتیب عملیات هرکدام را با هم بررسی می‌کنیم.

## توابع

### تابع اول: گسترش ماتریس

در این تابع، شما یک ماتریس $n \times n$ را به صورت پارامتر دریافت نموده و یک ماتریس $(n+2) \times (n+2)$ برمی‌گردانید که مرکز آن همان ماتریس ورودی است و سایر درایه‌هایش صفر است. برای مثال اگر ماتریس اولیه به صورت زیر باشد:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

ماتریسی که باید برگردانید به صورت زیر است:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

تابع شما باید `pad_matrix` نام داشته باشد. در واقع خط اول کد شما برای این تابع به صورت زیر خواهد بود:

```python
def pad_matrix(matrix: list[list[int]]):
# کد شما
```

## تابع دوم: ضرب داخلی دو ماتریس $3 \times 3$

در این تابع باید دو ماتریس $3 \times 3$ را به عنوان پارامتر گرفته و ضرب داخلی آن‌ها را خروجی دهید. مثلا اگر:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad , \quad B = \begin{bmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{bmatrix}$$

شما باید حاصل عبارت زیر را خروجی دهید:

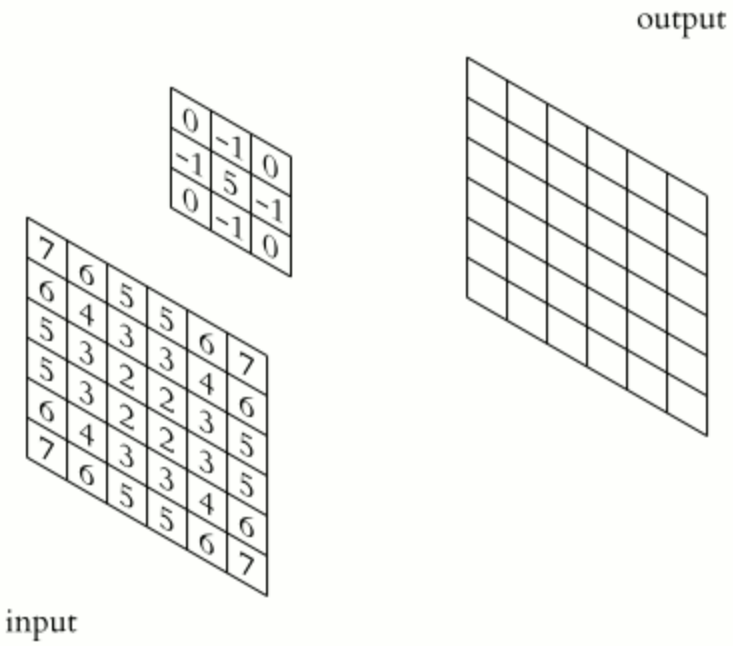
$$< A, B >= (1 \times 11) + (2 \times 12) + (3 \times 13) + (4 \times 14) + (5 \times 15) + (6 \times 16) + (7 \times 1$$

تابع شما باید `inner_product_3x3` نام داشته باشد، یعنی خط اول کد شما برای این تابع به صورت زیر خواهد بود:

```
def inner_product_3x3(matrix1: list[list[int]], matrix2: list[list[int]
# کد شما
```

## تابع سوم: کانولوشن

در این تابع شما باید کانولوشن (برای مطالعه دقیقتر این لینک را ببینید) روی یک ماتریس $n \times n$ و یک هسته (یک ماتریس دیگر) که ابعاد آن $3 \times 3$ است پیاده کنید. کانولوشن برای مثلاً ماتریس $6 \times 6$ عملیات زیر است:




که یعنی شما ابتدا باید با تابع اول (گسترش ماتریس) ماتریس ورودی را گسترش دهید، سپس ماتریس‌های $3 \times 3$ را از ماتریس گسترده جدیدتان جدا کرده و در هسته، با استفاده از تابع دوم (ضرب داخلی) ضرب کنید. در واقع اگر قسمتی که از گسترده‌ی ماتریس اصلی (فرض کنید $A$ به ابعاد $n \times n$) به ابعاد $3 \times 3$ جدا نموده‌اید مرکزش در ردیف $i$ و ستون $j$ ماتریس اصلی ($A$) باشد، در ماتریس خروجی باید حاصل ضرب داخلی را در همان مکان (ردیف $i$ام و ستون $j$ام) قرار دهید.

تابع شما باید `convolve_3x3_kernel` نام داشته باشد و چون ابتدا آن را گسترش می‌دهید، خطوط ابتدایی کد شما برای این تابع به صورت زیر خواهد بود:

```
def convolve_3x3_kernel(matrix: list[list[int]], kernel: list[list
    padded_matrix = pad_matrix(matrix)
```

## نکات سوال

۱. به منظور راحتی کار شما، تضمین می‌شود تمام ماتریس‌های سوال $n \times n$ و همچنین تمامی کرنل‌ها $3 \times 3$ باشند.

۲. تمامی ماتریس‌ها به صورت لیست‌های دوبعدی (لیست درون لیست) به شما داده خواهند شد، برای مثال ماتریس زیر را

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

به صورت زیر به شما می‌دهیم:

```
A = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]
```

که این معادل نمایش زیر است:

```
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

۳. به گسترش متفاوت ماتریس در انیمیشن GIF قرار داده شده توجه نکنید و گسترش را با همان افزودن صفر در نظر بگیرید.

## ورودی و خروجی

این سوال ورودی و خروجی نداشته و پاسخ شما صرفاً از روی توابعی که نوشته‌اید توسط داور کوئرا بررسی خواهند شد.

## راهنمایی‌ها

▼ راهنمایی ۱

می‌توانید در هر مرحله کانولوشن زیرماتریسی به ابعاد $3 \times 3$ ذخیره کرده و سپس با تابع `inner_product_3x3` که نوشتید مقدار مورد نیاز را برای عنصری که نیاز دارید بدست آورید.

▼ راهنمایی ۲

شما در واقع در تابع کانولوشن خود به سه حلقه تو در توی `for` نیاز دارید.

▼ راهنمایی ۳

کدی که ما برای حل این سوال زدیم (با استفاده از آموخته‌های شما) جمعا ۲۷ خط شد. اگر کد شما دارد خیلی طولانی‌تر می‌شود احتمالاً دارید مساله را برای خود پیچیده می‌کنید.

## بیشتر بدانید

▼ مطالعه آزاد: RGB

احتمالاً می‌دانید که سه رنگ اصلی نور (یعنی در حالت additive یا افزایشی، زمانی که زمینه تیره است و رنگ زدن باعث افزایش روشنایی می‌شود) قرمز و سبز و آبی هستند. این یعنی با ترکیب شدت‌دار این سه نور می‌توان هر رنگی را ساخت. به صورت جبری گزاره‌ی گفته شده را نمایش می‌دهیم:

$$w_{\text{red}} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + w_{\text{green}} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + w_{\text{blue}} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} w_{\text{red}} \\ w_{\text{green}} \\ w_{\text{blue}} \end{bmatrix}$$

که یعنی هر رنگ را می‌توان به صورت یک بردار با سه مولفه نشان داد به طوری که مولفه اول نشان‌دهنده‌ی شدت نور قرمز، مولفه‌ی دوم نشان‌دهنده‌ی شدت نور سبز و مولفه‌ی سوم نشان‌دهنده‌ی شدت نور آبی باشد. شدت نورها را به صورت قراردادی (برحسب توان معمول نمایشگرها) بین ۰ تا ۲۵۵ در نظر می‌گیریم به

طوری که ۰ یعنی عدم تابش نور به آن رنگ و ۲۵۵ یعنی تابش با شدت حداکثری نور با آن رنگ. به این سیستم نمایش، RGB می‌گویند. چند نمونه رنگ با بردار RGBشان را بررسی می‌کنیم:

- قرمز: $[255, 0, 0]$
- سبز: $[0, 255, 0]$
- آبی: $[0, 0, 255]$
- زرد: $[255, 255, 0]$ (چرا؟)
- بنفش (magenta): $[255, 0, 255]$

▼ مطالعه آزاد: تبدیل عکس رنگی به سیاه و سفید

حتماً از روی بردارهای RGB متوجه شده‌اید که هرچه اختلافات مولفه‌های بردار RGB یک رنگ کمتر باشند، saturation یا همان غلظت آن رنگ کمتر است و در نتیجه بیشتر روی طیف سفید-سیاه قرار می‌گیرد. در نتیجه می‌توان گفت کمترین غلطت رنگ، در رنگ‌هاییست که مولفه‌های RGBشان با یکدیگر برابرند.

در نمایش یک عکس در کامپیوتر، ما عکس را به صورت یک ماتریس دوبعدی می‌بینیم که هر مولفه از آن (که به آن پیکسل یا pixel می‌گوییم) یک بردار با سه مولفه است که نشان‌دهنده‌ی رنگ آن پیکسل است (جالب است بدانید به ابعاد آن ماتریس دو بعدی، resolution یا وضوح گفته می‌شود). حال برای سفید و سیاه کردن یک تصویر رنگی باید مولفه‌های RGB هر پیکسلش را با هم برابر کنیم. یک روش برای این کار، این است که به جای مولفه‌های رنگی هر پیکسل، میانگین آن مولفه‌ها را بگذاریم. یعنی برای مثال، اگر رنگ یک پیکسل به صورت زیر باشد:

$$[R, G, B]$$

ما رنگش را به صورت زیر تغییر دهیم:

$$[\frac{R+G+B}{3}, \frac{R+G+B}{3}, \frac{R+G+B}{3}]$$

پیشنهاد می‌شود کتابخانه‌ی PIL را نصب نموده و با استفاده از آن، یک عکس رنگی را سیاه و سفید کنید!

▼ مطالعه آزاد: فیلتر تصاویر

احتمالاً از ابتدای این مساله، این سوال برایتان پیش آمده که کانولوشن روی ماتریس چه ربطی به فیلتر اسنپ‌چت دارد!

ابتدا تصویری که در بخش قبل سیاه و سفید کردید را وارد کنید (ترجیحاً تصویر مربعی باشد که با کد فعلیتان کار کند). سپس (از آنجا که تمام مولفه‌های RGBاش برابرند و حالا می‌توان به صورت یک عدد نشانشان داد) پیکسل‌ها را از بردار به عدد اسکالر تبدیل کنید. سپس در تابع کانولوشنی که زدید، ماتریس عکس را به عنوان ماتریس ورودی دهید. همچنین کرنل زیر را وارد کنید:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

اگر همه‌چیز خوب پیش رفته باشد و فرضا عکس زیر را به عنوان ورودی کل (قبل از سیاه و سفید کردن) داده باشید:

باید تصویر زیر را بعد از اجرای برنامه خروجی بگیرید:

اگر تمایل به مطالعه‌ی بیشتر در این زمینه دارید، این لینک را بخوانید.

دو سوال:

- بنظر شما چگونه بدون سیاه و سفید کردن عکس، فیلترها را روی یک عکس رنگی اعمال کنیم؟
- آیا به ازای کرنل‌های مختلف، قرار است خروجی‌های متفاوتی بگیریم؟

▼ مطالعه آزاد: یادگیری ماشین (Machine Learning)

کانولوشن فقط در پردازش تصویر کاربرد ندارد! این مفهوم، که از ریاضیات و پردازش سیگنال سرچشمه گرفته، یکی از ابزارهای کلیدی در یادگیری ماشین، به‌ویژه در شبکه‌های عصبی کانولوشنی (CNN)، است. در ادامه با چندتا از کاربردهای آن بیشتر آشنا می‌شوید:

- پردازش تصویر: کانولوشن برای استخراج ویژگی‌های مهم (مانند لبه‌ها، بافت‌ها و الگوها) از تصاویر استفاده می‌شود. این عملیات در لایه‌های CNN منجر به شناسایی اجزای مختلف در تصاویر می‌شود.
- پردازش صوت و سیگنال: کانولوشن برای تحلیل سیگنال‌ها مانند صدا، گفتار و داده‌های زمانی به کار می‌رود. در این کاربردها، فیلترهای کانولوشنی به استخراج ویژگی‌های زمانی و فرکانسی کمک می‌کنند.
- علوم اعصاب و پزشکی: در تحلیل داده‌های EEG، MRI یا CT Scan، کانولوشن به شناسایی الگوهای پنهان کمک می‌کند.

کانولوشن به یک مدل یادگیری ماشین اجازه می‌دهد که ویژگی‌های محلی را در داده‌ها تشخیص داده و آن‌ها را در سطوح مختلف ترکیب کنند. این قابلیت منجر به کاهش پیچیدگی محاسباتی و بهبود دقت مدل می‌شود.

برای مطالعه‌ی بیشتر درباره کانولوشن و کاربردهای آن در یادگیری ماشین، پیشنهاد می‌کنیم این مقاله و این ویدیو را ببینید.

▼ مطالعه آزاد: پردازنده‌های گرافیکی (GPU)

احتمالاً متوجه شده‌اید که کارهایی مانند پردازش تصویر و یادگیری ماشین به توان پردازشی بالایی نیاز دارند و هرچه تصاویر و نمونه‌ها بیشتر می‌شوند، ما باید زمان بیشتری را صرف کنیم تا به نتیجه برسیم.

فرض کنید شما ۱۰ تا یخچال و ۸۰۰۰ دانه برنج دارید. همچنین دو تیم دارید که:

- تیم اول متشکل از ۸ قهرمان بدنسازی است!
- تیم دوم متشکل از ۲۰۰۰ تا مینیون است!

فرض کنید به شما گفته می‌شود برای بالا بردن ۱۰ یخچال از ۲۹ طبقه باید یک گروه را انتخاب کنید. هر گروهی را انتخاب کنید گروه دیگر بی‌کار می‌ماند و همچنین هیچ یک از اعضای یک گروه نمی‌توانند بیشتر از یک شی در دست خود داشته باشند. شما کدام گروه را انتخاب می‌کنید؟ برای بالا بردن ۸۰۰۰ دانه برنج چطور؟

مثالی که آورده شد، دقیقا تفاوت CPU و GPU را نشان می‌دهد. در واقع CPU برای انجام کارهاییست که نیاز به قدرت زیاد دارند ولی سری (متوالی) هستند و GPU برای کارهایی طراحی شده است که شامل تعداد زیادی عملیات کوچک ولی موازی هستند. در مثال بالا، گروه مینیون‌ها شبیه GPU عمل می‌کنند: تعداد زیاد آن‌ها و توانایی انجام کارهای کوچک به صورت هم‌زمان باعث می‌شود که برای بردن دانه‌های برنج بهترین انتخاب باشند. از طرف دیگر، گروه قهرمانان بدنسازی شبیه CPU عمل می‌کنند: قدرت زیادی برای انجام کارهای بزرگ و سنگین دارند، اما تعدادشان محدود است و برای کارهای بسیار کوچک و متعدد، کارآمد نیستند.

در دنیای واقعی، CPUها معمولا بین ۴ تا ۱۶ هسته دارند و برای اجرای تعداد کمی وظایف پیچیده به‌صورت متوالی طراحی شده‌اند. این پردازنده‌ها، مناسبند برای کارهایی مثل اجرای سیستم‌عامل، برنامه‌های روزمره و وظایفی که به تصمیم‌گیری منطقی سریع نیاز دارند.

در آن سو GPUها هزاران هسته کوچک دارند و برای اجرای تعداد زیادی وظایف کوچک به‌صورت موازی طراحی شده‌اند. مناسب برای پردازش تصویر، یادگیری عمیق، رندرینگ گرافیکی و تحلیل داده‌های حجیم هستند.

یادگیری ماشین، به‌ویژه شبکه‌های عصبی عمیق، به ماتریس‌های بزرگ و عملیات ریاضی (مانند ضرب ماتریس‌ها) نیاز دارد که تعداد زیادی محاسبات کوچک اما موازی هستند. GPUها به‌دلیل تعداد هسته‌های زیاد و معماری موازی‌شان، این عملیات را بسیار سریع‌تر از CPU انجام می‌دهند.

یک مثال خوب برای درک تفاوت CPU و GPU را در این ویدیو می‌توانید ببینید.

▼ مطالعه آزاد: پردازش تصاویر سه‌بعدی (3D Rendering)

ماتریس‌ها در دنیای گرافیک، فقط برای نمایش رنگ کاربرد ندارند! آن‌ها ابزار کلیدی در تبدیل و جابجایی اشیاء در فضای سه‌بعدی هستند. اولین گام‌های رندر سه‌بعدی در بازی‌های ویدیویی به دوران بازی‌هایی مانند Wolfenstein و DOOM بازمی‌گردد که از تکنیک **Raycasting** برای شبیه‌سازی محیط‌های سه‌بعدی استفاده می‌کردند. در این روش، دیوارها و محیط‌ها با استفاده از محاسبات ساده و نقشه‌های دوبعدی به شکل سه‌بعدی نمایش داده می‌شدند. هرچند این تکنیک محدودیت‌هایی داشت، اما راه را برای پیشرفت‌های بعدی هموار کرد. با عرضه **Quake** در سال 1996، مفهوم رندر سه‌بعدی واقعی وارد بازی‌های ویدیویی شد. این بازی از مدل‌های چندضلعی و سیستم **Z-buffering** برای مدیریت عمق استفاده کرد و جهشی بزرگ در گرافیک بازی‌ها ایجاد کرد. در سال‌های بعد، تکنیک‌های نورپردازی پیشرفته مانند **Phong Shading** و استفاده از

Shaderها باعث شد بازی‌ها واقع‌گرایانه‌تر به نظر برسند و موتورهای گرافیکی قدرتمندتری مانند Unreal Engine و Source به توسعه‌دهندگان این امکان را دادند تا مرزهای جدیدی را در دنیای سه‌بعدی کشف کنند.

اگر علاقه‌مندید، می‌توانید از این لینک در مورد raycasting بیشتر مطالعه کنید. می‌توانید حتی یک موتور raycasting ساخته و آن را در GitHub خود به اشتراک بگذارید!

▼ مطالعه آزاد: و اینک، RTX!

ظهور تکنولوژی RTX نقطه عطفی در دنیای گرافیک رایانه‌ای و بازی‌های ویدیویی است. RTX که توسط شرکت NVIDIA معرفی شد، نسل جدیدی از پردازش گرافیکی را با استفاده از تکنیک Ray Tracing بی‌درنگ ممکن شد. برخلاف روش‌های قدیمی که از ترفندهایی برای *شبیه‌سازی* نور استفاده می‌کردند، RTX با شبیه‌سازی دقیق مسیر **پرتوهای نور** در محیط، سایه‌ها، انعکاس‌ها و انکسارهایی کاملاً واقع‌گرایانه ارائه می‌دهد. این تکنیک ابتدا در انیمیشن و جلوه‌های ویژه سینمایی استفاده می‌شد، اما اکنون با پیشرفت سخت‌افزارهای گرافیکی، امکان استفاده از آن در بازی‌ها نیز فراهم شده است.

فناوری RTX علاوه بر افزایش کیفیت بصری، تجربه‌ای بسیار نزدیک‌تر به واقعیت را به بازیکنان ارائه می‌کند. در بازی‌هایی مانند **Cyberpunk 2077** و **The Witcher 3**، انعکاس نور روی سطوح براق یا سایه‌هایی که دقیقاً مطابق با منبع نور شکل می‌گیرند، بازیکن می‌تواند دنیای محیطش را تقریباً واقع‌گرایانه تجربه کند. البته این پیشرفت هزینه‌بر است، چرا که برای بهره‌گیری کامل از قابلیت‌های RTX به سخت‌افزارهای قدرتمند و پیشرفته نیاز است.

در اینجا می‌توانید بازی Cyberpunk 2077 را در دو حالت که RTX روی آن فعال و غیرفعال بوده ببینید!