



به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

جبر خطی

پروژه نهایی

استاد سرافراز

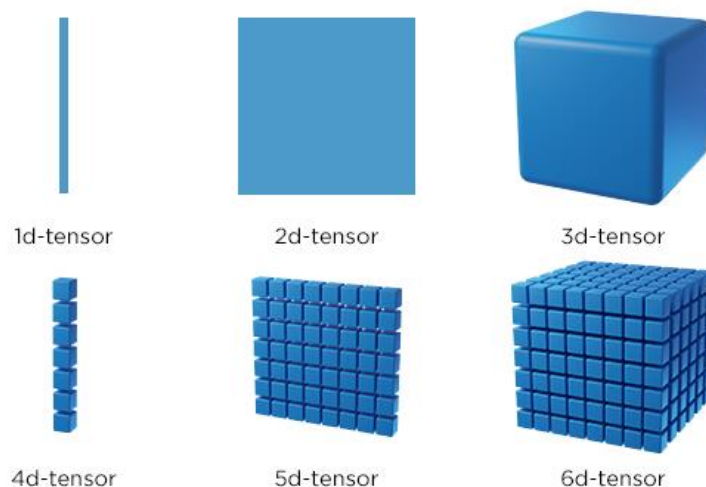
سیده دیبا روانشید شیرازی

۸۱۰۱۹۹۴۳۱

جداسازی پس زمینه با RPCA:

سوالات مفهومی:

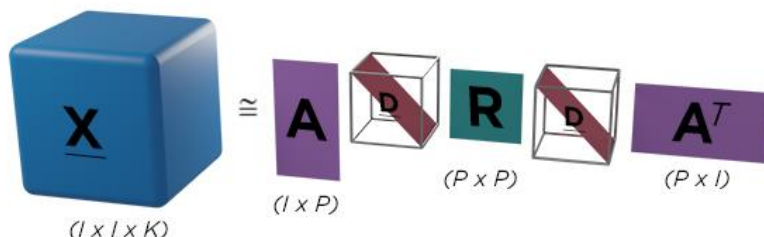
۱. ساختار داده تنسور و کاربرد آن در پردازش تصویر : همانطور که در صورت پروژه نیز اشاره شد این یک ساختار ریاضی است برای نشان دادن رابطه های چند خطی میان مجموعه ای از داده ها. یعنی این ساختار داده میتواند اطلاعات در ابعاد بالا را ذخیره کند. به این صورت که وقتی اطلاعات یک بعدی هستند بردار میشوند. دو بعدی همان ماتریس است و ابعاد بالاتر نیز به شکل زیر میشود :



وقتی داده ها را به این صورت نشان می‌دهیم میتوانیم از خواص جبری برای جمع و ضرب آن ها برخوردار شویم.

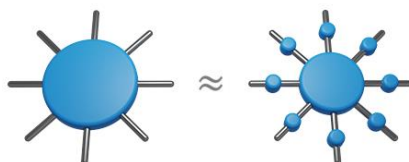
برای نشان دادن عکس ها به طور خاص چون هر پیکسل عکس دارای سه مقدار red green blue میباشد میتوانیم برای نشان دادن هر پیکسل از یک تنسور سه بعدی استفاده کنیم. به طور کلی یک عکس کامل شامل 5 بعد میشود که دو بعد دیگر همان موقعیت عمودی و افقی پیکسل ها هستند. یا مثلا میتوان به عکس ها لیبل زد و یک بعد را به این لیبل ها اختصاص داد. اما مشکلی که به وجود می آید این است که سایز تنسور ها که زیاد میشود محاسبات به طور نمایی با مقدار بعد بالا می‌رود. که اگر از یک حدی بالاتر برود دچار مشکل به نام "the curse of dimensionality" میشویم.

اما میتوان از این اتفاق جلوگیری کرد. به این صورت که ما مانند تجزیه های ماتریس ها که در درس یاد گرفتیم میتوانیم تنسور ها را نیز به قسمت های کوچک تر تقسیم یا تجزیه کنیم. پس به جای اینکه یک تنسور بسیار بزرگ داشته باشیم آن را تبدیل به چند قسمت کوچکتر میکنیم.

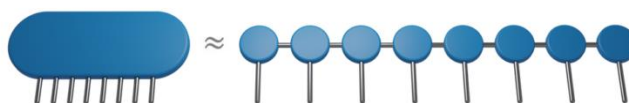


روش های مختلف با کاربرد های مختلفی برای این کار وجود دارد. مثلا روش Tucker decomposition برای کمپرس کردن تنسور ها روشی است که تنسور را به یک بخش core و چندین ماتریس فاکتور گرفته شده تبدیل میکند.

این همان روش higher order SVD است که در ادامه نیز به آن اشاره میشود
مثلا یک تنسور 8 بعدی تبدیل به یک core و 8 ماتریس کوچک شده است.



کاربرد این روش در face recognition میباشد که مانند مفهوم eigenfaces در اینجا نیز قابل انجام شدن است.
ولی روش اصلی آن Tensorfaces است که در آن میتوان احساسات و زاویه عکس و همه ی این اطلاعات را نیز وارد تنسور کرد و دیگر مانند eigenfaces فقط محدود به عوض کردن یک پارامتر نیستیم.
یک کاربرد دیگر میتواند حذف کردن یا کم کردن اثر اطلاعات بی ارزش مانند سایه ها باشد.
در روش های دیگر تجزیه مانند Tensor Train برای کم کردن حجم داده ها استفاده میشود. و حتی در شبکه ها میتواند محاسبات را به شدت کاهش دهد.



کاربرد این بخش برای کم کردن نویز روی عکس ها میباشد. (به دلیل وصل بودن تنسور ها به هم ارتباطات بین پیکسل ها مشخص میشود و میتوانیم از آن برای حذف نویز استفاده کنیم)

۲. **RPCA**: اگر بخواهیم توضیح مختصری در مورد PCA بدهیم این یک روش برای کاهش بعد داده ها میباشد. به این صورت که ما دنبال یک نگاشتی هستیم که بیشترین پراکندگی را به ما بدهد. که برای این کار با داشتن یک ماتریس با بعد زیاد ابتدا مقادیر و بردار های ویژه متناظر آن را بدست می آوریم و از بزرگ به کوچک مرتب میکنیم. حال میدانیم که بیشترین پراکندگی در جهت بردار های ویژه متناظر با مقادیر ویژه بزرگ است.

اما در روش RPCA ما دنبال کاهش بعد هستیم زیرا میخواهیم دو بخش **sparse** و **low rank** را از روی تنسور بزرگتر بسازیم. که **sparse** به ما اجسامی که حرکت میکنند و **low rank** به ما پس زمینه را که ثابت است میدهد. در این روش میخواهیم:

$$\text{minimize } \|L\|_* + \lambda \|S\|_1$$

$$\text{subject to } L+S=M$$

دلیل : با توجه به اینکه رابطه ی nuclear norm به صورت زیر است :

$$\|A\|_* = \sum_{i=1}^r \sigma_i$$

که سیگما همان مقادیر ویژه تکین ماتریس است. که هر چه این مقدار کمتر باشد یعنی ماتریس مقادیر صفر بیشتری داشته است و رنک آن ماتریس پایین تر میباشد.

و برای نرم اول میدانیم که هر چه کمتر باشد یعنی دیتا **sparse** تر یا دارای مقادیر صفر بیشتری میباشد.

الگوریتم :

با نوشتن رابطه ی لاگرانژین این بخش و استفاده از روش PCP داریم :

$$\text{argmin}_{L,S} = \|L\|_* + \lambda \|S\|_1 + \langle Y, M-L-S \rangle + (\mu/2) \|M-L-S\|^2_F.$$

الگوریتم این بخش ADMM نام دارد و از Y برای کمک استفاده میشود و در هر بخش L و S و همچنین Y آپدیت میشوند.

1. ابتدا L و S و Y را مساوی صفر میگذاریم و μ و λ را نیز مقدار میدهیم.

2. مراحل زیر را تا وقتی که $\|M-L-S\|_F > \delta \|M\|_F$ این رابطه برقرار است ادامه میدهیم.

$$U, \Sigma, V^T = \text{SVD}(M - S_k + \mu^{-1} Y_k).$$

$$L_{k+1} = U \Sigma \mu^{-1} V^T.$$

$$S_{k+1} = S_k \mu^{-1} (M - L_{k+1} + \mu^{-1} Y_k).$$

$$Y_{k+1} = Y_k + \mu (M - L_{k+1} - S_{k+1}).$$

3. L و S را برمیگردانیم.

۳. **SVD**: میدانیم که این یک روش برای کاهش ابعاد ماتریس است که برای داده های **sparse** بهترین عملکرد را دارد. در این روش ماتریس A را به سه قسمت تقسیم میکنیم: $A = USV^T$.

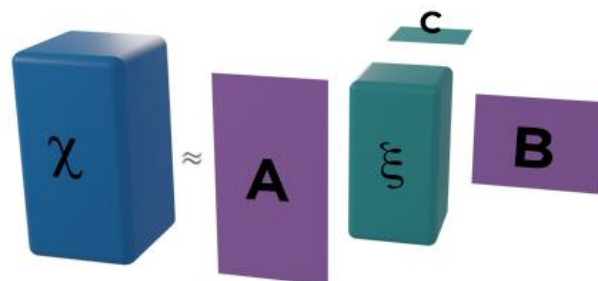
که S قطری است و U و V متعامد هستند. یعنی: $U^T U = I$.

Randomized SVD: اگر ابعاد ماتریس اولیه ما بزرگ باشد روش **SVD** زمانبر و حافظه بر میشود. به همین خاطر این روش جدید بوجود آمده است که در آن از روش های احتمالی و تخمین زدن استفاده میشود.

میخواهیم ماتریس A را به صورت زیر بنویسیم: $A \approx QQ^*A$ ابعاد Q کوچکتر از A است و متعامد است. پس نیاز داریم که Q را تخمین بزنیم که از روش **Randomized range finder** انجام میشود که در آن با کم و زیاد کردن یک پارامتر میتوان دقت را افزایش داد اما ابعاد Q را افزایش میدهد. سپس مینویسیم که $B = Q^*A$. حال برای ماتریس B الگوریتم **SVD** را انجام میدهیم. به صورت $B = U^*SV^*$ سپس $U = QU^*$ و در آخر U, S, V^* را برمیگردانیم. در اینجا چون B ابعاد کوچکتری نسبت به A دارد محاسبات را کاهش میدهد.

Truncated SVD: این روش در همانطور که از اسمش مشخص است ما داده را طبق اهمیت آن قطع میکنیم تا کوچک تر شود. مانند وقتی که اعداد را گرد میکنیم. این روش مانند **PCA** است اما فرق آن این است که قبل از اینکه یک سری از داده ها را حذف کنیم دیتا را مرتب نمیکنیم. که باعث میشود این روش برای داده های **sparse** به خوبی جواب بدهد. در این روش ما فقط یک تعدادی از مقادیر منفرد را بدست می آوریم.

Higher-Order SVD: در زمانی که ما از تنسور ها استفاده میکنیم دیگر روش های **svd** قابل اجرا نمیشوند پس نیاز به روش جدیدی داریم که بتواند روی تنسور ها انجام شود. در این روش از **tucker decomposition** که در قبل نیز به آن اشاره شد استفاده میشود. که در این روش ما تنسور را به صورت زیر تجزیه میکند:



4. رابطه ی nuclear norm :

$$\|A\|_* = \sum_{i=1}^{\min(m,n)} \sigma_i$$

که سیگما همان مقادیر ویژه تکین ماتریس $A_{m \times n}$ است.

پیاده سازی:

به دلیل مشکل وصل نشدن گوگل کولب به GPU در برخی مواقع این بخش را در متلب زدیم.
در ابتدا 10 ثانیه از فیلم را جدا میکنیم و فیلم را سیاه سفید میکنیم. که آخرین فریم آن به صورت زیر در می آید.



وقتی نویز را به عکس اضافه میکنیم کیفیت جداسازی کاهش پیدا میکند. بنابراین نویز را اضافه نکردیم.

همینطور میدانیم که اگر فیلم را به تنسور تبدیل کنیم سه بعدی میشود که دو بعد آن ابعاد عکس است و بعد سوم تعداد فریم های فیلم است که در اینجا 200 تا میباشد. اگر فیلم را سیاه سفید نمیکردیم این تنسور 5 بعدی میشد زیرا که هر یک از ابعاد عکس خودشان سه مقدار مختلف RGB داشتند.

با انجام دادن الگوریتم robust SVD با مقادیر اولیه دلخواه که جز هایپر پارامتر ها محسوب میشدند. در انتها به ارور زیر میرسیم که نشان دهنده ی این است که جمع L و S ای که ساخته شد دقیقاً برابر با ماتریس اولیه نمیشود و از اول نیز میخواستیم این ارور را کم میکنیم و میدانیم که به صفر نمیرسد اما بسیار به صفر نزدیک است.

error

error = 4.6243e-06

نشان دادن فریمی دلخواه از دو ماتریس L و S :

یک فریم از L که مربوط به پس زمینه بی تحرک میباشد :



یک فریم از S که مربوط به بخش متحرک میباشد که میبینیم که شامل ماشین ها و موتور ها و آدم ها میباشد:



اگر فقط یک فریم از این فیلم را داشتیم نمیتوانستیم بخش متحرک و یا بخش بدون تحرک یا همان پس زمینه را پیدا کنیم. میتوانستیم کل ویدیو را پردازش کنیم اما به دلیل حجم زیاد فقط 10 ثانیه را جدا کردیم . هر چه تعداد فریم ها بیشتر باشد کیفیت جداسازی نیز بالاتر میرود.

ویدیو ها نیز سیو شدند.

نشانه گذاری دیجیتال:

سوالات مفهومی:

۵. ساختار کلی **Watermarking** و استخراج تصویر نهان شده : این یک تکنیک است که برای افزودن اطلاعات اضافی به مثلاً یک ویدیو استفاده میشود. این اطلاعات اضافی به عنوان امضای دیجیتالی هستند برای این که از عوض شدن داده جلوگیری شود. ابتدا واترمارک خود را انتخاب میکنیم که میتواند یک تصویر باشد. حال باید آن را وارد داده ی خود کنیم. این کار باید به گونه ای انجام شود که داده اصلی ما کیفیت خود را از دست ندهد. حال باید برای سنجیدن صحت داده واترمارک آن را استخراج کنیم تا مطمئن شویم که تغییری نکرده است.

روش های مختلفی برای این بخش وجود دارد که دقیقاً برعکس وارد کردن واترمارک به داده عمل میکند. ممکن است داده را با داده های مشابهی مقایسه کنیم و وجود واترمارک را شناسایی کنیم (retrieval algorithms). به هر حال باید واترمارک انتخاب شده مناسب این کار باشد و مقاومت خوبی داشته باشد.

۶. SVD based Watermarking:

الگوریتم نهان سازی :

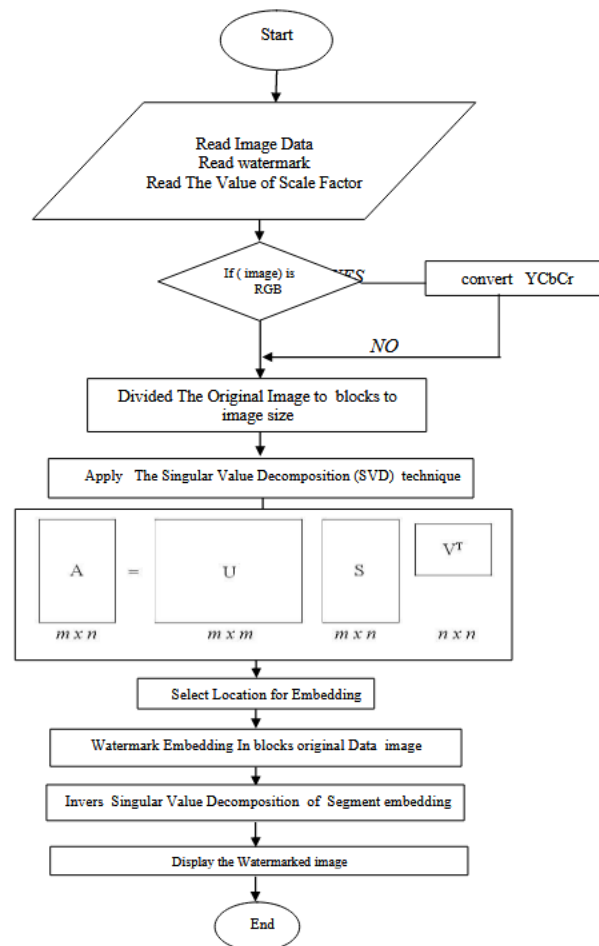
این روش به این صورت است که ما برای هر فریم از ویدیو **svd decomposition** را انجام میدهیم و در عین حال روی عکس **watermark** نیز این کار را میکنیم. حال دو ماتریس **S** که شامل مقادیر ویژه ی تکین این دو ماتریس میباشد را داریم. در اینجا با یک ضرب آلفا این دو را با هم جمع میکنیم. حال یک ماتریس **S** جدید داریم که میتوانیم با **U** و **V** که از فریم ویدیو داشتیم ، فریم واترمارک دار شده را تولید کنیم و با انجام این روش روی کل داده میتوانیم کل فیلم را واترمارک دار کنیم.

الگوریتم آشکار سازی:

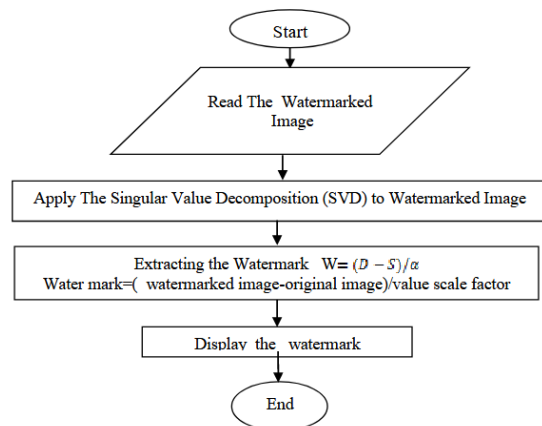
در این جا با داشتن ویدیو بدون واترمارک و ویدیو واترمارک دار میتوانیم با گرفتن انجام **SVD** روی هر دو و کم کردن ماتریس های **S** از یکدیگر به ماتریس **S** واترمارک دست پیدا کنیم که با کمک آن میتوان خود واترمارک را بازسازی کرد.

روش دوم [4]:

الگوریتم نهان سازی :



الگوریتم آشکار سازی:



روش سوم [5]:

الگوریتم نهان سازی:



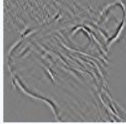

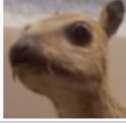
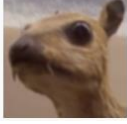
- Step 1. Partition the host image into blocks.
- Step 2. Perform SVD transformation.
- Step 3. Extract the largest coefficient $D(1, 1)$ from each D component and quantize it by using a predefined quantization coefficient Q . Let $Z = D(1, 1) \bmod Q$.
- Step 4. For an embedded watermark bit valued of 0, if $Z < 3Q/4$, $D(1, 1)$ modify to $D'(1, 1) = D(1, 1) + Q/4 - Z$. Otherwise, $D'(1, 1) = D(1, 1) + 5Q/4 - Z$.
- Step 5. For an embedded watermark bit valued of 1, if $Z < Q/4$, $D(1, 1)$ modify to $D'(1, 1) = D(1, 1) - Q/4 + Z$. Otherwise, $D'(1, 1) = D(1, 1) + 3Q/4 - Z$.
- Step 6. Perform the inverse of the SVD transformation to reconstruct the watermarked image.

الگوریتم آشکار سازی:

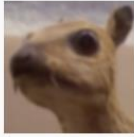
- Step 1. Partition the watermarked image into blocks.
- Step 2. Perform SVD transformation.
- Step 3. Extract the largest coefficient $D'(1, 1)$ from each D component and quantize it by using the predefined quantization coefficient Q . Let $Z = D'(1, 1) \bmod Q$.
- Step 4. If $Z < Q/2$, the extracted watermark has a bit value of 0. Otherwise, the extracted watermark has a bit value of 1.

که در اینجا از روش اول استفاده شده است.

۷. **Blur**: استفاده از کرنل ها که به صورت مربعی هستند که روی عکس حرکت میکنند باعث میشود که با عوض کردن ضرایب مقادیر پیکسل ها و جایگزین کردن آن ها با مقادیر جدید عکس را تار یا بلر کند. مثلا مدل های مختلف تاثیر کرنل ها را در شکل زیر مشاهده میکنید. هر چه سایز کرنل بزرگ تر باشد عکس تار تر میشود. [6]

Operation	Kernel ω	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

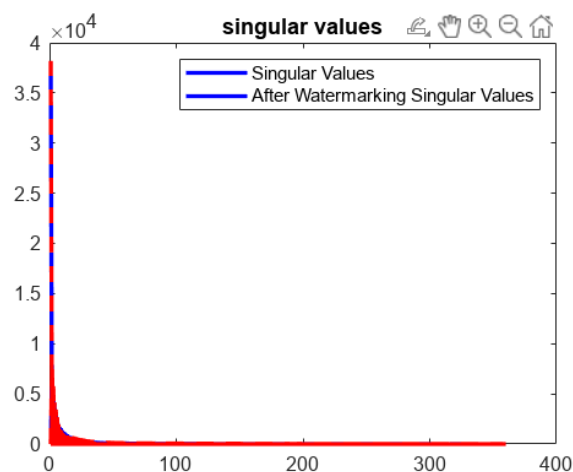
که ما در پروژه از کرنل 5 در 5 گوسی استفاده کردیم که به شکل زیر میباشد :

Gaussian blur 5 × 5 (approximation)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
---	--	---

پیاده سازی:

با توجه به شماره دانشجویی عدد 1 را انتخاب میکنیم و باید ابتدا آن را سیاه و سفید کرده و به سایز ویدیو در بیاوریم.

نمودار مقادیر ویژه تکین را رسم میکنیم :



مشاهده میکنیم که این مقادیر تغییری نکرده اند و این به دلیل است که اینکه یک عکس دارای واترمارک میباشد یا خیر به راحتی قابل تشخیص نباشد.

تصویر یک فریم از ویدیو قبل و بعد از واتر مارک : (میبینیم که کمی از کیفیت فیلم کاهش یافته است)

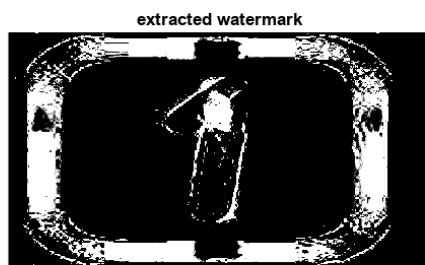
unwatermarked video



watermarked video



آشکار سازی قبل و بعد تار کردن ویدیو :



کیفیت آشکار سازی کمی پایین آمده است. و همینکه تغییری روی واترمارک میبینیم متوجه میشویم که ویدیو اصلی دستکاری شده است.

منطق کرنل با دوباره توضیح میدهیم : در اینجا این کرنل روی عکس به ترتیب میچرخد و در هر جایی که قرار میگیردم مقدار پیکسل وسط را ضربدر این ماتریس میکند و در جای مقادیر اصلی پیکسل ها قرار میدهد. به این ترتیب ما از مدل گوسی استفاده کردیم که مدلی معروف میباشد و در خود متلب تعریف شده است و مقادیر زیر را دارد.

Gaussian blur 5 × 5 (approximation)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

ویدیو رمزگذاری شده تار شده نیز سیو شده است.

با تشکر

1. <https://visagetechologies.com/tensors-in-computer-vision/>
2. <https://medium.com/@aneetkumard8/understanding-robust-principal-component-analysis-rpca-d722aab80202>
3. <https://www.geeksforgeeks.org/add-a-salt-and-pepper-noise-to-an-image-with-python/>
4. https://www.researchgate.net/publication/339224683_Digital_Image_Watermarking_using_Singular_Value_Decomposition
5. <https://www.sciencedirect.com/science/article/abs/pii/S0167865505000140>
6. [https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))