



به نام خدا



University of Tehran  
College of Electrical and Computer Engineering

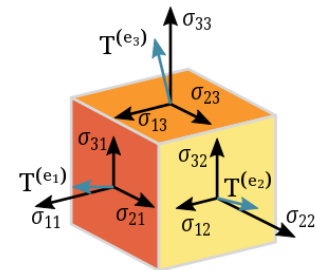
## Linear Algebra

Final Project

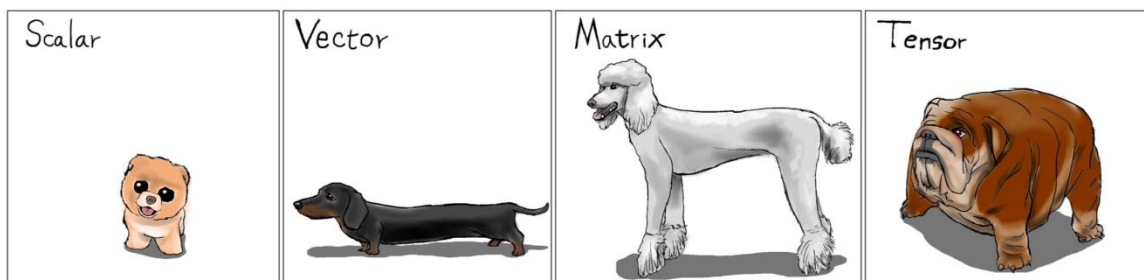
Mohammad Mashregi	name
810199492	SD number
Winter 2023	Time

## What is tensor and what application of it in image processing

In mathematics, tensors are algebraic objects that describe multilinear relationships between sets of algebraic objects related to a vector space. They can map between different objects such as vectors, scalars, and other tensors. Tensors are defined independently of any basis and are often represented by components in a specific coordinate system. Tensors play a crucial role in physics, providing a concise mathematical framework for problem-solving in mechanics, electrodynamics, general relativity, and other areas. In applications, tensor fields are common, where different tensors can occur at each point of an object.



In data science, tensors are data structures frequently used in numerical processing, especially in frameworks like PyTorch and TensorFlow. The rank of a tensor is its number of dimensions, and its shape indicates the size of each dimension. Elements are the values recorded at each location in a tensor. Tensors are extensively used in image processing, clustering, pattern recognition, feature extracture, where they represent images in neural networks and deep learning algorithms. For instance, a 256x256x3 colour image is represented by a three-dimensional tensor. Tensors play a crucial role in capturing intricate features like faces, objects, and patterns in images.



Scalar    Vector    Matrix    Tensor

1

$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

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

## RPCA and different from PCA

While both Principal Component Analysis (PCA) and Robust Principal Component Analysis (RPCA) are methods used in the data analysis and dimensionality reduction domains, their methods and goals differ.

First, let's talk about PCA and then talk about RPCA:

### Principal Component Analysis (PCA):

Data transformation into a new coordinate system where the variance of the data along the new axes is maximized is the goal of PCA, a linear dimensionality reduction technique. The orthogonal directions that exhibit the greatest variation in the data are known as the principal components. The greatest amount of variation is captured by the first principal component, the second largest by the second principal component (which is orthogonal to the first), and so forth.

---

**Algorithm 1 (Principal Component Pursuit by Alternating Directions [32, 51])**

---

```
1: initialize:  $S_0 = Y_0 = 0, \mu > 0$ .
2: while not converged do
3:   compute  $L_{k+1} = \mathcal{D}_\mu(M - S_k - \mu^{-1}Y_k)$ ;
4:   compute  $S_{k+1} = \mathcal{S}_{\lambda\mu}(M - L_{k+1} + \mu^{-1}Y_k)$ ;
5:   compute  $Y_{k+1} = Y_k + \mu(M - L_{k+1} - S_{k+1})$ ;
6: end while
7: output:  $L, S$ .
```

---

Because PCA depends on the covariance matrix, which can be significantly impacted by extreme values, it is susceptible to outliers. The underlying structure may not be correctly represented by PCA if the data contains outliers.

### Robust Principal Component Analysis (RPCA):

An expansion of PCA called RPCA is intended to be more resilient in the face of anomalies or tainted data. Decomposing the input data matrix into two parts—a sparse matrix that represents the noise or outliers and a low-rank matrix that represents the underlying structure (signal)—is the fundamental notion behind RPCA.

Mathematically, given a matrix  $M$ , RPCA decomposes it into the sum of a low-rank matrix  $L$  and a sparse matrix  $S$ :

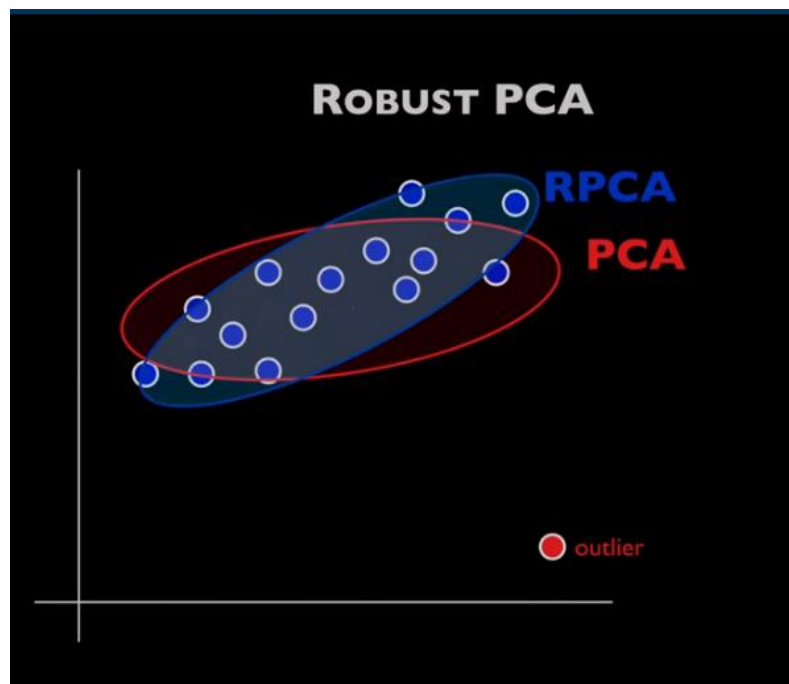
$$M = L + S$$

Where:

- $L$  is a low-rank matrix representing the principal components.
- $S$  is a sparse matrix representing the noise or outliers.

The decomposition is typically achieved by solving an optimization problem that minimizes the rank of  $L$  and the sparsity of  $S$ .

Also a [YouTube](#)



Principal Disparities:

1. Robustness to Outliers: This is where they differ most from one another. While RPCA is made to manage data with outliers by explicitly modelling a sparse component, PCA may be susceptible to outliers.

2. Decomposition Approach: RPCA breaks down the data matrix into low-rank and sparse components as I mentioned above, specifically addressing the existence of outliers, whereas PCA looks to optimize variance and identify orthogonal components.

3. Optimization issue: While RPCA usually entails solving an optimization issue that strikes a balance between the rank and sparsity of the decomposed matrices, PCA frequently includes solving an eigenvalue problem.

$$\begin{array}{ll} \text{minimize} & \|M - L\| \\ \text{subject to} & \text{rank}(L) \leq k. \end{array}$$

### Optimization problem

In conclusion, although PCA is an effective method for reducing dimensionality, it could not work well when there are anomalies present. However, RPCA is made expressly to deal with outliers and offer a more reliable decomposition.

---

#### Algorithm 5 (RPCA via the Inexact ALM Method)

---

**Input:** Observation matrix  $D \in \mathbb{R}^{m \times n}$ ,  $\lambda$ .  
1:  $Y_0 = D/J(D)$ ;  $E_0 = 0$ ;  $\mu_0 > 0$ ;  $\rho > 1$ ;  $k = 0$ .  
2: **while** not converged **do**  
3:   // Lines 4-5 solve  $A_{k+1} = \arg \min_A L(A, E_k, Y_k, \mu_k)$ .  
4:    $(U, S, V) = \text{svd}(D - E_k + \mu_k^{-1} Y_k)$ ;  
5:    $A_{k+1} = U S_{\mu_k}^{-1} [S] V^T$ .  
6:   // Line 7 solves  $E_{k+1} = \arg \min_E L(A_{k+1}, E, Y_k, \mu_k)$ .  
7:    $E_{k+1} = S_{\lambda \mu_k}^{-1} [D - A_{k+1} + \mu_k^{-1} Y_k]$ .  
8:    $Y_{k+1} = Y_k + \mu_k (D - A_{k+1} - E_{k+1})$ .  
9:   Update  $\mu_k$  to  $\mu_{k+1}$ .  
10:    $k \leftarrow k + 1$ .  
11: **end while**  
**Output:**  $(A_k, E_k)$ .

---

The ALM method operates on the *augmented Lagrangian*

$$l(L, S, Y) = \|L\|_* + \lambda \|S\|_1 + \langle Y, M - L - S \rangle + \frac{\mu}{2} \|M - L - S\|_F^2. \quad (5.1)$$

A generic Lagrange multiplier algorithm [5] would solve PCP by repeatedly setting  $(L_k, S_k) = \arg \min_{L, S} l(L, S, Y_k)$ , and then updating the Lagrange multiplier matrix via  $Y_{k+1} = Y_k + \mu(M - L_k - S_k)$ .

For our low-rank and sparse decomposition problem, we can avoid having to solve a sequence of convex programs by recognizing that  $\min_L l(L, S, Y)$  and  $\min_S l(L, S, Y)$  both have very simple and efficient solutions. Let  $\mathcal{S}_\tau : \mathbb{R} \rightarrow \mathbb{R}$  denote the shrinkage operator  $\mathcal{S}_\tau[x] = \text{sgn}(x) \max(|x| - \tau, 0)$ , and extend it to matrices by applying it to each element. It is easy to show that

$$\arg \min_S l(L, S, Y) = \mathcal{S}_{\lambda\mu}(M - L + \mu^{-1}Y). \quad (5.2)$$

Similarly, for matrices  $X$ , let  $\mathcal{D}_\tau(X)$  denote the singular value thresholding operator given by  $\mathcal{D}_\tau(X) = U\mathcal{S}_\tau(\Sigma)V^*$ , where  $X = U\Sigma V^*$  is any singular value decomposition. It is not difficult to show that

$$\arg \min_L l(L, S, Y) = \mathcal{D}_\mu(M - S - \mu^{-1}Y). \quad (5.3)$$

Thus, a more practical strategy is to first minimize  $l$  with respect to  $L$  (fixing  $S$ ), then minimize  $l$  with respect to  $S$  (fixing  $L$ ), and then finally update the Lagrange multiplier matrix  $Y$  based on the residual  $M - L - S$ , a strategy that is summarized as Algorithm 1 below.

---

**Algorithm 1 (Principal Component Pursuit by Alternating Directions [32, 51])**

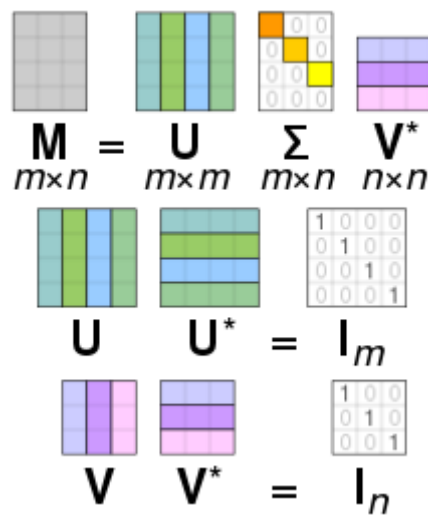
---

- 1: **initialize:**  $S_0 = Y_0 = 0, \mu > 0$ .
  - 2: **while** not converged **do**
  - 3:   compute  $L_{k+1} = \mathcal{D}_\mu(M - S_k - \mu^{-1}Y_k)$ ;
  - 4:   compute  $S_{k+1} = \mathcal{S}_{\lambda\mu}(M - L_{k+1} + \mu^{-1}Y_k)$ ;
  - 5:   compute  $Y_{k+1} = Y_k + \mu(M - L_{k+1} - S_{k+1})$ ;
  - 6: **end while**
  - 7: **output:**  $L, S$ .
-

## Randomized SVD, Truncated SVD, Higher-order SVD

Let's begin with SVD, What is SVD?

A mathematical method called singular value decomposition (SVD) is used in linear algebra to split a matrix into three smaller matrices. The singular values, left singular vectors, and right singular vectors of the original matrix are represented by these matrices. Applications for SVD can be found in machine learning, signal processing, and data analysis.



The diagram illustrates the SVD decomposition of a matrix  $M$  into three matrices:  $U$ ,  $\Sigma$ , and  $V^*$ . The dimensions of these matrices are given as  $m \times n$  for  $M$ ,  $m \times m$  for  $U$ ,  $m \times n$  for  $\Sigma$ , and  $n \times n$  for  $V^*$ .

Below the main equation, two identities are shown:

$$U U^* = I_m$$

$$V V^* = I_n$$

where  $I_m$  and  $I_n$  are identity matrices of size  $m$  and  $n$  respectively.

### Higher-order SVD

An extension of SVD to higher-order tensors is called higher-order SVD (HOSVD). HOSVD is used for decomposing tensors, which are multi-dimensional arrays, whereas SVD is usually applied to matrices. The idea of matrices is extended to more than two dimensions by tensors.

#### 1. Measurements:

- SVD: Mostly applied to two-dimensional matrices.
- HOSVD: Made for tensors, which are multidimensional objects.

#### 2. Disintegration:

-SVD: Divides a matrix into three matrices that stand for vectors and singular values, respectively.

- HOSVD: Splits a tensor into factor matrices along each mode (dimension) and a core tensor.

۳. The number of singular vectors or values

-SVD: In SVD, the minimum of the matrix's rows and columns is usually equal to the number of singular values.

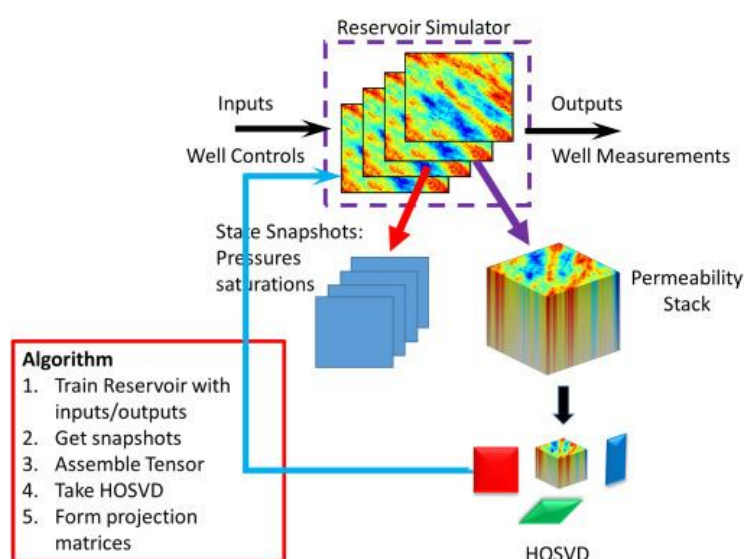
- HOSVD: The quantity of singular values (or rank) along every mode in HOSVD can be selected independently.

۴. Uses:

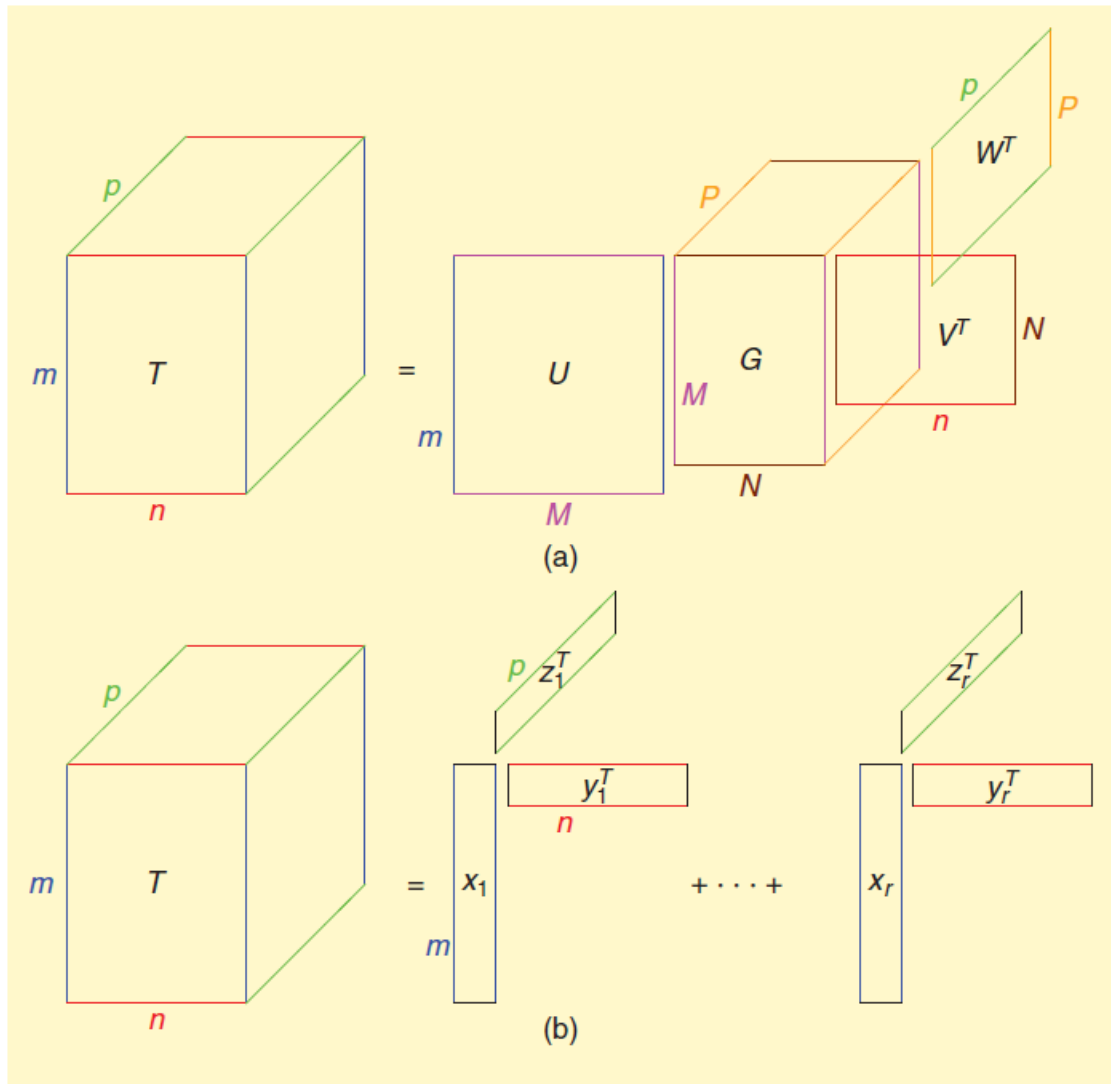
- SVD: Often utilized in cooperative filtering, data analysis, and picture compression applications.

- HOSVD: Used in tensor decomposition, tensor completion, and multidimensional data processing.

In conclusion, HOSVD expands the idea of SVD to tensors, enabling the decomposition of multi-dimensional data structures, whereas SVD is intended for matrices. The type and organization of the data being examined determine how SVD and HOSVD should be applied.







For more info PLS check the reference folder.

## Truncated SVD

A variation of singular value decomposition (SVD) called truncated singular value decomposition (truncated SVD) only requires computing a predetermined number of singular values and the singular vectors that correspond to those values. The truncated SVD only takes into account the most significant singular values, whereas the full SVD computes all singular values and vectors. This results in a lower-rank approximation of the original matrix.

### 1- Efficiency of Computation:

SVD: The singular value and vector computation process (SVD) can be computationally costly, particularly for large matrices.

Truncated SVD: Reduces calculation time and memory requirements by computing only a predetermined number of singular values and vectors with truncated SVD.

### 2- Reduction in Rank:

SVD: Full decomposition including all singular values and vectors is provided by SVD, maintaining the matrix's original rank.

Truncated SVD: By taking into account only the most significant singular values, truncated SVD reduces rank and produces a lower rank approximation of the original matrix.

### 3- Use in the Reduction of Dimensions:

SVD: Used to reduce dimensionality; however, if the data has a reduced intrinsic dimensionality, the full decomposition might not be required.

Truncated SVD: Specifically created for dimensionality reduction, where only the most crucial components are kept, is truncated singular value decomposition (SVD).

### 4- Reduced Noise:

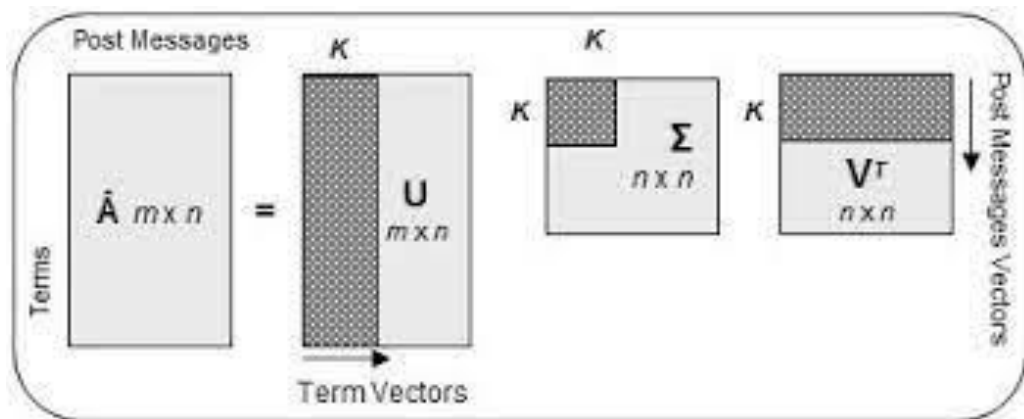
SVD: Records everything, even elements that can be loud.

Truncated SVD: By concentrating on the most important components and disregarding the less important ones, truncated SVD can assist lessen the effect of noise in the data.

5- Approximation of a matrix:

SVD: The original matrix's precise disintegration.

Truncated SVD: Reduced rank approximation of the original matrix is provided by truncated SVD.



## Randomized SVD

A low-rank approximation of a matrix can be computed using the approximate and randomized Randomized Singular Value Decomposition (Randomized SVD) algorithm, which is comparable to the full Singular Value Decomposition (SVD). The primary distinction between Randomized SVD and regular SVD is its scalability and computational efficiency.

### 1. Efficiency of Computation:

**SVD:** Performs the precise Singular Value Decomposition, which may incur significant processing costs, particularly when dealing with huge matrices.

**Randomized SVD:** This method is appropriate for large-scale problems since it makes use of randomized procedures to more effectively obtain a low-rank approximation.

### 2. Disturbance:

**SVD:** Utilizes deterministic techniques that require the computation of all vectors and singular values.

The Randomized Stochastic Variance Decomposition (SVD) algorithm inserts randomness into the computation, resulting in faster approximations without sacrificing accuracy.

### 3. Rank Approximation

**SVD:** Maintains the matrix's initial rank while providing an accurate decomposition with all singular values and vectors.

**Randomized SVD:** Generates an estimated low-rank decomposition; the amount of random samples utilized in the method determines the accuracy of the approximation.

### 4. Utilizing Large-Scale Data Applications

**SVD:** With extremely big matrices or datasets, scalability problems may arise.

Randomized SVD: This type of statistical technique is more scalable by sampling a subset of the data using randomized techniques, which makes it ideal for handling vast amounts of data.

## 5. Algorithmic Methodology:

SVD: Computes singular values and vectors using deterministic techniques, such as the Lanczos algorithm or the QR decomposition.

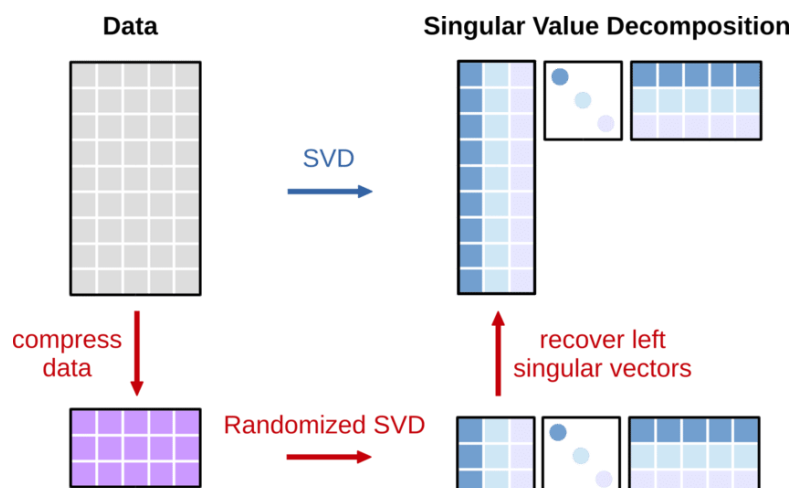
Randomized SVD: Uses random methods to approximate something close to a low rank, like random projections or random matrix factorization.

## 6. Memory Requirements:

SVD: Depending on the size of the matrix, storing all singular values and vectors may demand a significant amount of memory.

Randomized SVD: Because of its randomized sampling technique, it typically requires less memory.

When scalability and computing efficiency are critical, as they are in large-scale machine learning and data processing activities, randomized support vector distribution (SVD) is very helpful. Even though it offers a rough answer, the trade-off between computing efficiency and precision makes it a useful tool for managing huge data issues. The exact needs of the application and the available computing power determine which SVD and Randomized SVD to use.



## Nuclear Norm

The nuclear norm, also known the trace norm or the Ky Fan norm, is a matrix norm defined as the sum of the singular values of a matrix. For a given matrix  $X$ , its nuclear norm is denoted as  $\|X\|_*$  and is defined as follows:

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

Here:

- $m$  is the number of rows in the matrix
- $n$  is the number of columns in the matrix
- $\sigma_i$  represents the singular values of the matrix

In essence, the nuclear norm is equal to the sum of the matrix's singular values, or the total of the unique values' absolute values. Since the nuclear norm of a matrix equals its rank when the matrix is diagonal and has non-negative diagonal elements, it is a convex relaxation of the rank function. This makes it helpful in rank-minimization-related convex optimization problems.

Applications such as low-rank matrix recovery, compressed sensing, and matrix completion frequently use the nuclear norm. As a convex surrogate for a matrix's rank, one might minimize the nuclear norm in optimization problems under specific restrictions.

The relationship between the nuclear norm and matrix rank in this optimization problem favours low-rank solutions. In low-rank matrix recovery situations, the nuclear norm has shown to be a valuable tool, even in cases where the observed entries represent a small portion of the total matrix entries.

- $\|\cdot\|_1$  is the **L1 norm**. Minimizing the **L1 norm** results in sparse values. For a matrix, the L1 norm is equal to the **maximum absolute column norm**.
- $\|\cdot\|_*$  is the **nuclear norm**, which is the L1 norm of the singular values. Trying to minimize this results in sparse singular values --> low rank.



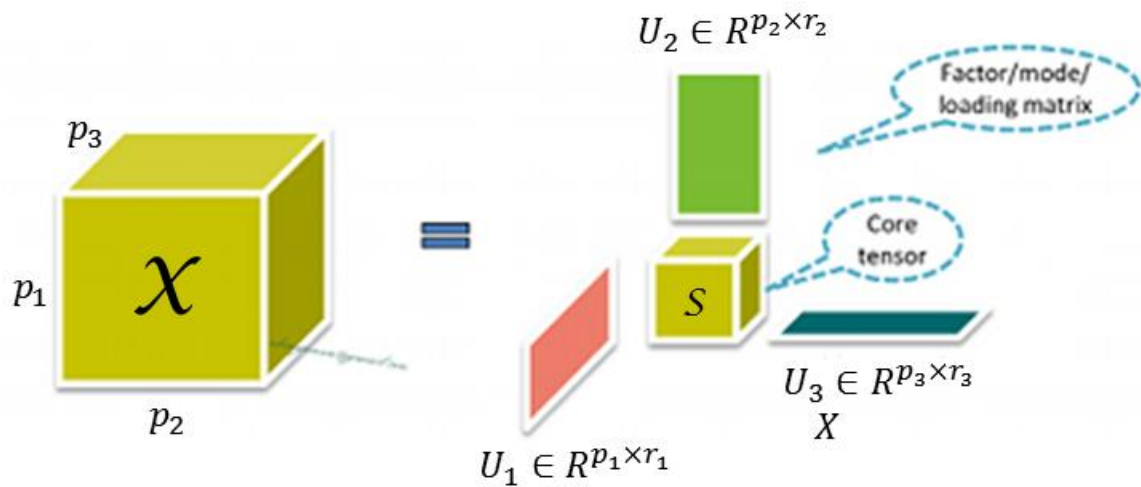
Resolution of Frame 100: 640x360

```
Tensor Shape (dimensions): torch.Size([3, 360, 640])
Channels: 3
Height: 360
Width: 640
```

Dimension 0 (frames): each one is the frame of the video during time.

Dimension 1 (Height): This dimension represents the height of the image.

Dimension 2 (Width): This dimension represents the width of the image.





And now we convert it to black and white:

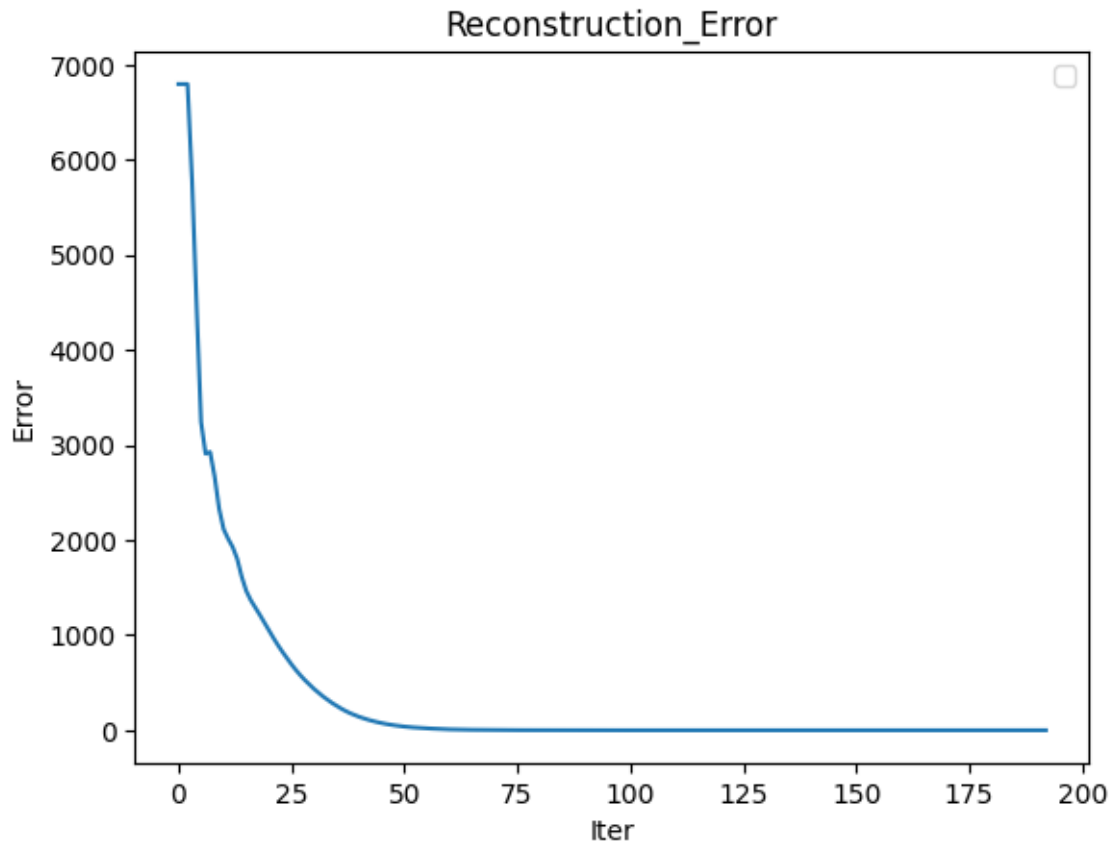


Here's added salt and pepper noise:



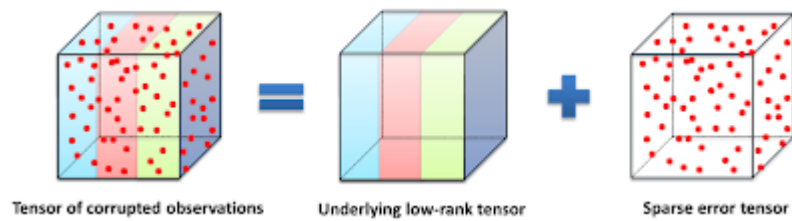


Result:



Reconstruction Error:

The difference between the main video and the sum of low-rank and sparse tensor is Reconstruction Error. In other words, The reconstruction error in this scenario is the difference between the input data and the data reconstructed by the network.





foreground



background

In low rank, we try to save objects or colours of the constant video and don't move(move very very slow) and all about the background

But in sparse we try to find and save object which moves in the video.

Also, we can't use this method for only one frame, because we can't detect which object moves or is steady so we don't know where to put it(spars or low rank).

## 6- watermarking

Digital watermarks are covert identifiers inserted into images, videos, or audio streams to withstand noise and ascertain copyright ownership. Employed through the technique of watermarking, digital data is concealed within a carrier signal, sometimes unrelated to it. These watermarks serve various purposes, including confirming signal integrity, validating ownership, and aiding in banknote authentication or tracking copyright violations.

Throughout the dissemination process, a watermark is integrated into the digital signal, allowing for later extraction from copies to reveal the distribution source. This method has proven effective in identifying the source of illegally duplicated films.

The watermarking process typically involves three stages: embedding, transmission or storage, and potential attacks. In the embedding phase, an algorithm integrates data into a host signal, creating the watermarked signal. After transmission or storage, potential attacks may occur, involving modifications like compression or intentional noise addition. Detection, or extraction, is the final step, where an algorithm retrieves the watermark from the attacked signal. Robust watermarking ensures successful extraction despite significant modifications, while fragile watermarking results in extraction failure upon any alteration, often applied in copyright protection scenarios.

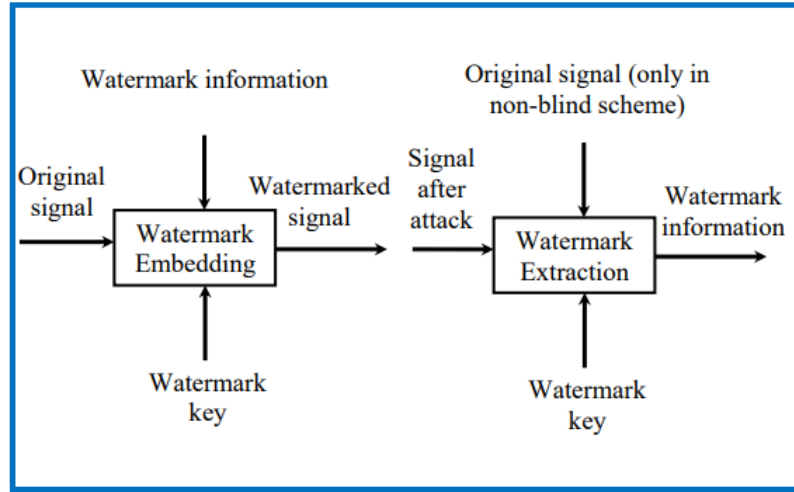
Types of watermarks:

Visible:

- **Overlay watermarking** : Put a logo or text or special pattern on a corner of the video or picture or someother part of it. And sometimes make it transparent or semi-transparent that main video can be seen.

Invisible:

- **Digital Watermarks**: These are embedded directly into digital content such as images, audio, or video files. They can be visible or invisible and are used to identify ownership or authenticate the content.



## Method 1:

If we consider

$A$  represents the original host image.

$W$  is the watermark image.

$U, V, \Sigma$  are the SVD components of the host image.

$\alpha$  is a scaling factor controlling the strength of the watermark.

$U', V', \Sigma'$  are the SVD components of the watermarked image.

$SubI_{\text{watermarked}}$  is the watermarked image.

$W_{\text{extracted}}$  is the extracted watermark.

For **embedding**(hiding) we have:

$$A = U \Sigma V^T$$

$$\Sigma + \alpha W = U_w \Sigma_w V_w^T$$

$$A_{\text{watermarked}} = U (\Sigma_w) V^T$$

For **detection**(extraction) we have:

Assume we have  $U_w$  and  $V_w^T$  from last part so now we have :

$$A_{\text{watermarked}} = U' \Sigma' V'^T$$

We make new matrix:

$$D = U_w \Sigma' V_w^T$$

So only things we need to do:

$$W_{\text{extracted}} = \frac{D - \Sigma}{a}$$

## Method 2:

This method is much easier, but we may have lost more than Method 1

If we consider

$A$  represents the original host image.

$W$  is the watermark image.

$U, V, \Sigma$  are the SVD components of the host image.

$\alpha$  is a scaling factor controlling the strength of the watermark.

$U', V', \Sigma'$  are the SVD components of the watermarked image.

$SubI_{\text{watermarked}}$  is the watermarked image.

$W_{\text{extracted}}$  is the extracted watermark.

For **embedding**(hiding) we have:

$$A = U \Sigma V^T$$

$$w = U_w \Sigma_w V_w^T$$

$$A_{\text{watermarked}} = U (\Sigma + \alpha \Sigma_w) V^T$$

For **detection**(extraction) we have:

Assume:

$$A_{\text{watermarked}} = U' \Sigma' V'^T$$

So only things we need to do:

$$W_{\text{extracted}} = U_w (\Sigma' - \alpha \Sigma_w) V_w^T$$

### The blurring operation:

- The blurring operation simplify important tasks that use convolution to smooth data and decrease noise over the picture or frame of a video.

### Kernel Definition:

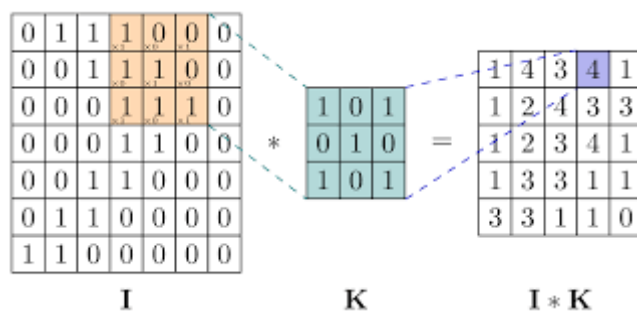
- The kernel is a small matrix, usually square, that contains specific values.
- For a basic blur, a common kernel is the averaging kernel. It has equal weights, and each element is set to  $\frac{1}{N}$  divided by the total number of elements in the kernel.

### Matrix Representation:

- The image is represented as a matrix, where each element corresponds to a pixel's intensity value.

### Convolution Operation:

- The convolution operation involves placing the kernel on top of a pixel in the image matrix and computing the weighted sum of the pixel values.
- This process is repeated for every pixel in the image, resulting in a new matrix representing the blurred image.



### Example with Averaging Kernel:

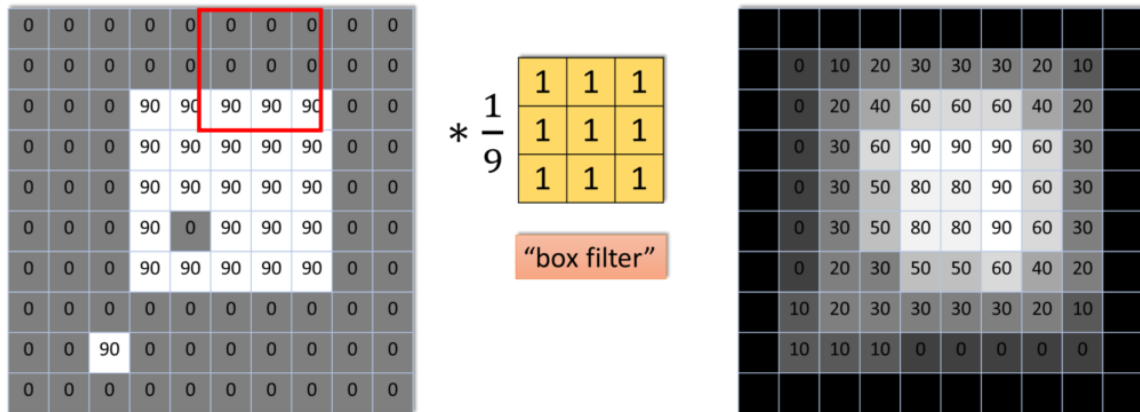
- Suppose we have a simple 3x3 averaging kernel:

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- The convolution operation involves taking a 3x3 section of the image, multiplying each pixel value by the corresponding value in the kernel, and then summing up these products which called simple filter.

## Averaging filter

$$F(x, y) * H(u, v) = G(x, y)$$



$$G = F * H$$

Gaussian filter: The Gaussian Smoothing Operator performs a weighted average of surrounding pixels based on the Gaussian distribution. It is used to remove Gaussian noise and is a realistic model of defocused lens. Sigma defines the amount of blurring.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1





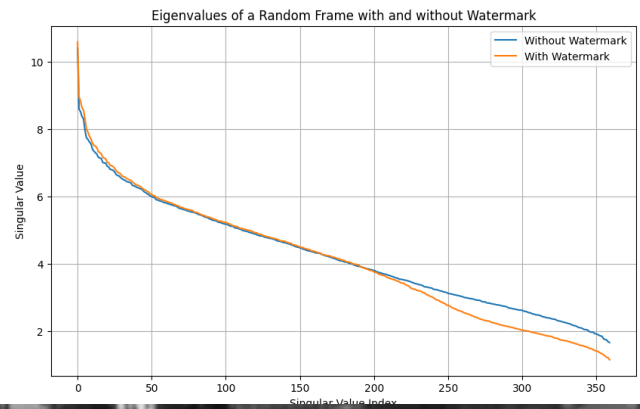
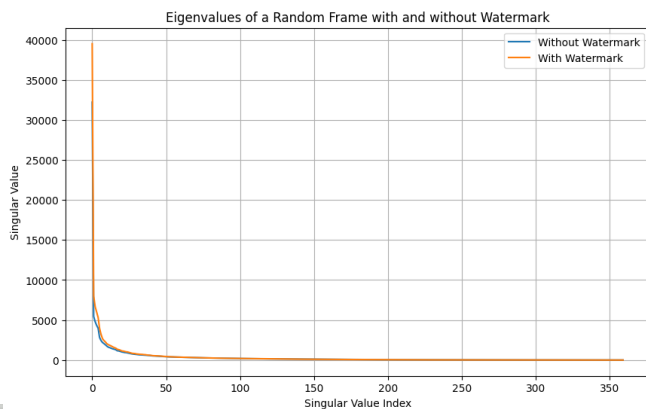
Result:

Method 2:



Watermark image extracted

As you can see the image is a little bad, that because some data because in reconstruction we must check that data of picture must be between 0 to 255 also the video effect on watermark that become like this.



With watermark



Without watermark

Now after adding gaussian filter with filter size 7 we extracted the water mark image we have:

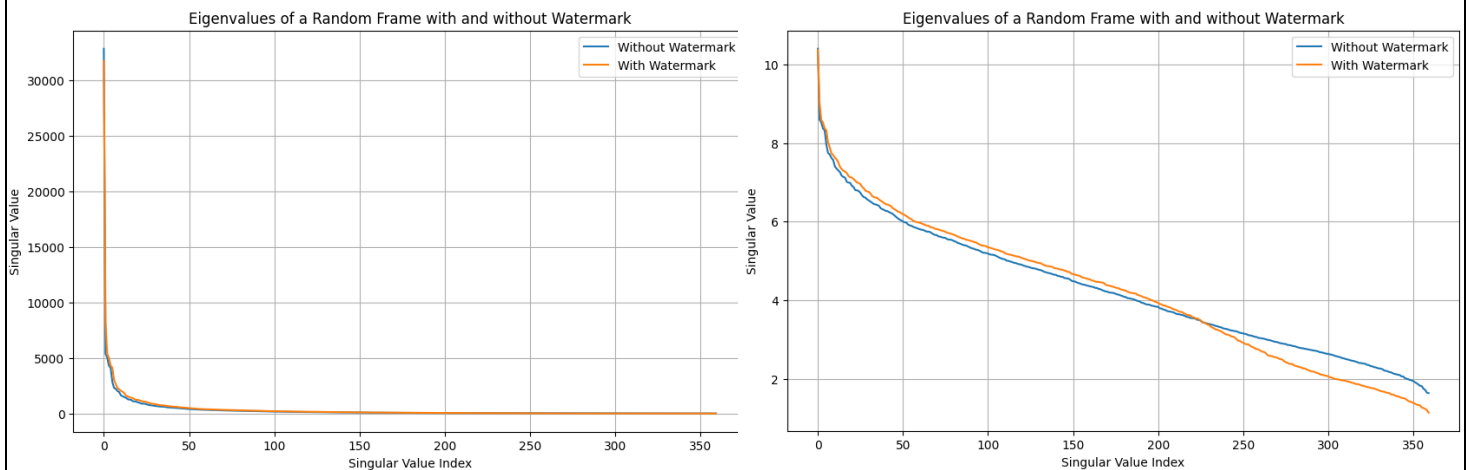


Which a bit lighter before.

Method 1:



The extracted watermark



As can be seen, when the eigen value of main video is big the effect of watermark is very small and we can't see difference a lot, but when the eigen value of main video is small, we can see that watermark effect on it, the right figure show difference in logarithm base which we can see the effect better.

Also it depends on  $\alpha$  we assume  $\alpha = 0.1$  if we choose a bigger number it would affect more on it.



**With watermark**



**Without watermark**

Here's the Gaussian blurred video with a  $\sigma = 0.6$  and kernel size 7\*7



And extracted watermark:

