DDA3005
Numerical Methods

THE CHINESE UNIVERSITY OF HONG KONG, SHENZHEN

Numerical Methods (SVD) for Image and Video Processing

Group

Author:	*Student Number:*
Alfonsus Rodriques Rendy	121040014
Alden Hegel	120040012
Benedict Dharma Sutiarto	121040009
Nicole Catherine Lee Tandaju	121040002

December 27, 2023

1 Singular Value Decomposition

1.1 Background

This project is to implement the two phases to compute the singular value decomposition (SVD), using the Golub-Kahan Bidiagonalization in phase I to simplify any matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ to be a bidiagonal matrix $\boldsymbol{B} \in \mathbb{R}^{m \times n}$ and implement the QR Iteration in phase to obtain the SVD of matrix $\boldsymbol{B}$. The SVD can further be used for computing the matrix representation using the singular value, or it can be used to compute the matrix's pseudoinverse.

We conducted two types of QR iteration for computing the SVD. First is QR iteration using Wilkinson shift. For the ordinary QR iteration, we need to conduct the QR iteration to the matrix $\boldsymbol{B}^T\boldsymbol{B}$ to find the singular value and the right eigenvector of the bidiagonal representation of $\boldsymbol{A}$.

The second method utilizes the Cholesky decomposition. Using the fact that the matrix is positive definite, we simplify the QR iteration so that we can directly use $\boldsymbol{B}$ to find its eigenvalue. Note that the second method is equivalent to pure QR iteration with no shift, which we will prove later on.

1.2 Implementation

The SVD is implemented in two phase: (1) Bidiagonalization (2) QR iteration. Our objective is to achive the decomposition:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal and $\boldsymbol{\Sigma}$ is the eigenvalue of $\boldsymbol{A}$. The two phases results in different orthogonal matrix decomposition of $\boldsymbol{A}$ that we can use directly to construct the singular value decomposition as the matrix multiplied on both sides are orthogonal.

1.2.1 Golub-Kahan Bidiagonalization

The objective of the first stage is to bidiagonalize any input matrix so that it can converge faster to a triangular (diagonal) matrix in the QR iteration. The bidiagonalization is conducted by multiplying the left and right sides of matrix $\boldsymbol{A}$ using householder transformation, which is orthogonal.

Suppose we have $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. The bidiagonalization is implemented in two steps until any off-diagonal value (except the super diagonal) is 0. The two steps are as follows:

$$\boldsymbol{A} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \xrightarrow{\boldsymbol{H}_{l1}\boldsymbol{A}} \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \xrightarrow{\boldsymbol{H}_{l1}\boldsymbol{A}\boldsymbol{H}_{r1}} \begin{bmatrix} * & * & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{bmatrix} \Rightarrow \begin{bmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{bmatrix}$$

For each iteration we apply orthogonalization transformation to both row and column of the matrix alternatively. We use Householder transformation for the orthogonalization, that is finding vector $\boldsymbol{v}$ to be a reflector of the column (or row) vector $\boldsymbol{x}$ such that is reflects to the unit basis. The vector $\boldsymbol{v}$ and $\boldsymbol{H}$ is given as

$$\boldsymbol{v} = \boldsymbol{x} + \text{sign}(x_0) \begin{bmatrix} \|\boldsymbol{x}\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \boldsymbol{H}_l = \boldsymbol{I} - \frac{2\boldsymbol{v}\boldsymbol{v}^T\boldsymbol{A}}{\|\boldsymbol{v}\|^2} \quad \boldsymbol{H}_r = \boldsymbol{I} - \frac{2\boldsymbol{A}\boldsymbol{v}\boldsymbol{v}^T}{\|\boldsymbol{v}\|^2}$$

Let $\boldsymbol{b} = [b_1\, b_2\, b_3\, \ldots\, b_n]$ be the i-th row of the matrix. For the right-side householder transformation, note that we only apply the householder from $\boldsymbol{b}_{i+1}$ so that we can achieve the bidiagonal matrix at the end of the operation.

After the bidiagonaization, we have the decomposition

$$\boldsymbol{Q}_l\boldsymbol{A}\boldsymbol{Q}_r = \boldsymbol{B} \Rightarrow \boldsymbol{A} = \boldsymbol{Q}_l^T\boldsymbol{B}\boldsymbol{Q}_r^T$$

where $\boldsymbol{Q}_l \in \mathbb{R}^{m\times m}$ and $\boldsymbol{Q}_r \in \mathbb{R}^{n\times n}$ The next step is to decompose $\boldsymbol{B}$ into a diagonal matrix using QR iteration.

1.2.2 QR with Wilkinson Shift

The objective of QR iteration is to obtain a triangular matrix similar the input matrix $\boldsymbol{A}$ — or , a diagonal matrix in a real symmetric matrix. The first method utilizes QR iteration in addition with Wilkinson shift. Wilkinson shift is calculated by getting the closest eigenvalue of the rightmost 2x2 matrix to the rightmost value. For instance suppose $\boldsymbol{X}_k$ is the rightmost 2x2 matrix at deflated step k, then the Wilkinson shift is the eigenvalue of $\boldsymbol{X}_k$ closest to $x_{k,k}$

$$\boldsymbol{X}_k = \begin{bmatrix} x_{k-1,k-1} & x_{k-1,k} \\ x_{k,k-1} & x_{k,k} \end{bmatrix}$$

The Wilkinson shift is given by

$$d = \frac{x_{k-1,k-1} - x_{k,k}}{2} \qquad \sigma = x_{nn} + d - \text{sign}(d)\sqrt{d^2 + x_{k,k-1}^2}$$

Since we are interested in the SVD of $\boldsymbol{B}$ and $\boldsymbol{B} \in \mathbb{R}^{m\times n}$ might be a non-square matrix, we conduct the QR iteration to $\boldsymbol{B}^T\boldsymbol{B}$ which is a square, real symmetric matrix which guarantees the QR convergence to a diagonal matrix. Note that even though our implementation of Wilkinson QR is not robust enough to deal with matrix with complex eigenvalue — as it will only converge to a diagonal matrix with order 2 — in this problem, the input matrix of the QR iteration is quaranteed to have n distinct real eigenvalues (e.g., non-defective with all real eigenvalues) and thus we should always converge to a diagonal matrix. At last, we will obtain the decomposition $\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ which can be used for the decomposition of the original matrix $\boldsymbol{A}$.

1.2.3 QR with Cholesky

Notice that we don't have to implement the QR iteration to the matrix $\boldsymbol{B}^T\boldsymbol{B}$. We can utilize Cholesky decomposition to directly transform $\boldsymbol{B}$ to a diagonal matrix. First we can decompose $\boldsymbol{X}^k$ to be $\boldsymbol{Q}_k\boldsymbol{R}_k = (\boldsymbol{X}^k)^T$ where equivalently we have $\boldsymbol{R}_k = \boldsymbol{Q}_k^T(\boldsymbol{X}^k)^T$. Then we obtain

$$\boldsymbol{L}_k\boldsymbol{L}_k^T = \boldsymbol{R}_k\boldsymbol{R}_k^T = \boldsymbol{Q}_k^T(\boldsymbol{X}^k)^T\boldsymbol{X}^k\boldsymbol{Q}_k$$

Notice that the $\boldsymbol{Q}_k^T(\boldsymbol{X}^k)^T\boldsymbol{X}^k\boldsymbol{Q}_k = \boldsymbol{R}_k\boldsymbol{Q}_k$ when we conduct pure QR iteration (with no shift) to $(\boldsymbol{X}^0)^T\boldsymbol{X}^0 = \boldsymbol{B}^T\boldsymbol{B}$ where $\boldsymbol{Q}^k\boldsymbol{R}^k = (\boldsymbol{X}^k)^T\boldsymbol{X}_k$. With the Cholesky decomposition, we have

$$\boldsymbol{L}_k\boldsymbol{L}_k^T = (\boldsymbol{X}^{k+1})^T\boldsymbol{X}^{k+1} \Rightarrow \boldsymbol{X}^{k+1} = \boldsymbol{L}_k^T$$

The convergence of the iteration can be evaluated from the super-diagonal value $\boldsymbol{X}_{i,i+1}$. We implement two type of QR with Cholesky decomposition. First, we only evaluate the convergence from the first and last superdiagonal, that is suppose we are at k-th deflation state which deals with the submatrix of from index i to j. Then the convergence happens when $\boldsymbol{X}^k_{i,i+1} < \epsilon$ or $\boldsymbol{X}^k_{j-1,j} < \epsilon$. The second method uses an improved convergence by checking if any super-diagonal element of $\boldsymbol{X}^k_{l,l+1} < \epsilon$. If a convergence is detected, it will split the matrix into two bidiagonal matrices and iterates on both smaller matrices until everything converges. This convergence method utilizes the divide and conquer strategy, which improves the time of convergence of the iteration. We use a relatively big $\epsilon = 10^4$ for the sake of performance time. However, as we can see later from the result, the obtained SVD is still accurate compared to other SVD method.

1.2.4 Combining the SVD

After the QR iterations we obtained the eigen-decomposition of $\boldsymbol{B}^T\boldsymbol{B}$. The eigenvector that we achieved after the QR convergence is the eigenvector of $\boldsymbol{B}^T\boldsymbol{B}$ where we have $\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$. Suppose we have the singular value decomposition $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, then we have

$$\boldsymbol{B}^T\boldsymbol{B} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^T$$

Meaning that we already obtain $\boldsymbol{\Sigma}, \boldsymbol{V}$ from the QR iteration, where $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}^{\frac{1}{2}}$, $\boldsymbol{V} = \boldsymbol{Q}$ For the QR iteration using Cholesky decomposition, we directly obtain $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}$ as we directly obtain the decomposition of matrix $\boldsymbol{B}$ instead of $\boldsymbol{B}^T\boldsymbol{B}$

To obtain $\boldsymbol{U}$, we use $\boldsymbol{U} = \boldsymbol{B}\boldsymbol{V}\boldsymbol{\Sigma}^-1$. The inversion of Σ is done carefully to avoid any inversion of zero value on the diagonal as a result of some numerical errors. Those zero values will remain as 0 in the inversion. We also extend the bases of U by finding its orthogonal complement to construct the full matrix $U \in \mathbb{R}^{m\times m}$ using random bases projected onto the orthogonal complement space.

Finally as we have two decomposition: $\boldsymbol{A} = \boldsymbol{Q}_l^T\boldsymbol{Q}_r^T$, $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, we can obtain the full SVD of $\boldsymbol{A}$ as:

$$\boldsymbol{A} = \boldsymbol{Q}_l^T\boldsymbol{B}\boldsymbol{Q}_r^T = \boldsymbol{A} = \boldsymbol{Q}_l^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{Q}_r^T = \boldsymbol{U}_A\boldsymbol{\Sigma}\boldsymbol{V}_A^T$$

where $\boldsymbol{U}_A = \boldsymbol{Q}_l^T\boldsymbol{U}$ and $\boldsymbol{V}_A = \boldsymbol{Q}_r\boldsymbol{V}$

1.3 Results and Observations

To test the performance of the three methods, we use random $n \times n$ matrices ranging from $n = 5$ to $n = 500$ and compares it with the built-in library.

Random matrix $n \times n$		**QR Wilkinson**	**QR Cholesky**	**Accelerated QR Cholesky**	**SVD (scipy)**
n=5	Time	5.63×10^{-3} s	3.46×10^{-3} s	1.91×10^{-3} s	2.77×10^{-4} s
	Error	3.89×10^{-15}	2.29×10^{-15}	$2.22\ \times10^{-15}$	1.32×10^{-15}
n=10	Time	2.52×10^{-3} s	5.83×10^{-3} s	7.75×10^{-3} s	2.47×10^{-4} s
	Error	4.11×10^{-15}	4.76×10^{-14}	5.28×10^{-15}	3.82×10^{-14}
n=25	Time	6.20×10^{-3} s	2.11×10^{-2} s	1.10×10^{-2} s	3.61×10^{-4} s
	Error	1.90×10^{-14}	4.76×10^{-14}	2.34×10^{-14}	1.91×10^{-14}
n=50	Time	1.65×10^{-2} s	3.38×10^{-2} s	7.04×10^{-2} s	1.11×10^{-3} s
	Error	5.20×10^{-14}	1.26×10^{-13}	8.08×10^{-14}	4.44×10^{-14}
n=100	Time	1.89×10^{-1} s	6.59×10^{-1} s	4.39×10^{-1} s	3.83×10^{-3} s
	Error	1.16×10^{-13}	$1.17\times10-12$	3.82×10^{-13}	7.89×10^{-14}
n=200	Time	6.39×10^{-1} s	7.62 s	2.87 s	2.98×10^{-2} s
	Error	4.06×10^{-13}	5.77×10^{-5}	3.21×10^{-12}	1.97×10^{-13}
n=500	Time	17.2 s	277.70 s	59.45 s	2.99×10^{-1} s
	Error	1.61×10^{-12}	$3.72\ \times10^{-11}$	1.09×10^{-11}	5.58×10^{-13}

Table 1: Performance comparison of different SVD implementation (with QR iteration vs scipy library) in terms of computation time and error accuracy for random matrices of varying sizes. The error is evaluated as $\|\boldsymbol{USV}^T - \boldsymbol{A}\|_F$

Faster Iteration with Shift As we already proved before, the QR iteration with Cholesky decomposition is equivalent to QR iteration without shift. Therefore, we expect a slower convergence rate than the QR iteration using shift. The faster convergence of QR with Wilkinson shift can be observed from the numerical experiment, as it converges faster on matrices of order larger than 5. The QR iteration with Cholesky only performs better when the matrix size is n.

Effect of Improved Convergence in QR with Cholesky We also observed a quite dramatic acceleration in QR with Cholesky decomposition using an improved convergence. By using the divide and conquer method, the QR iterations converge more than twice as fast as the iteration conducted only on one bidiagonal submatrix. The impact is even larger as the size of the matrix grows, implying the better practicality of using the accelerated convergence conditions.

Conclusion SVD using QR iteration with Wilkinson shift is still preferable due to its faster convergence. However, the algorithm can be improved as we can observe a faster performance of the algorithm in the Scipy library. Some room for improvement includes a faster bidiagonalization method as we spend quite a significant time in phase I, especially as the matrix gets larger.

2 Deblurring Image

2.1 Background

In this project, deblurring is approached through the application of Singular Value Decomposition (SVD), which breaks down a matrix into its constituent components—singular values and singular vectors. These components are instrumental in understanding and manipulating the intrinsic properties of the image matrix. The deblurring problem is modeled as a product of matrices, with a blurry image $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ being the result of an original image $\boldsymbol{X} \in \mathbb{R}^{n\times n}$ convolved with blurring kernels $\boldsymbol{A}_\ell, \boldsymbol{A}_r \in \mathbb{R}^{n\times n}$.

The objective is to recover the sharp image $\boldsymbol{X}$ by solving the equation $\boldsymbol{A}_\ell \boldsymbol{X} \boldsymbol{A}_r = \boldsymbol{B}$. When the kernels $\boldsymbol{A}_\ell$ and $\boldsymbol{A}_r$ are invertible, a direct solution can be pursued, setting $\boldsymbol{X} = \boldsymbol{A}_\ell^{-1} \boldsymbol{B} \boldsymbol{A}_r^{-1}$. However, this direct inversion can be impractical or unstable, particularly when $\boldsymbol{A}_\ell$ and $\boldsymbol{A}_r$ are ill-conditioned or close to singular. To circumvent these challenges, we employ SVD-based truncation techniques, allowing for a pseudo-inverse calculation that is robust to noise and computational errors.

The project explores two distinct computational paths to achieve the SVD. The first path involves the QR iteration with the Wilkinson shift applied to $\boldsymbol{B}^T\boldsymbol{B}$, from which the singular values and vectors are extracted. The second path leverages the positive-definite nature of the matrices, applying Cholesky decomposition to streamline the QR iteration directly on matrix $\boldsymbol{B}$. This method's equivalence to the shift-less QR iteration is established, highlighting the interplay between theoretical linear algebra and practical computational considerations in the pursuit of deblurring.

Through these methodologies, we delve into the computational intricacies of the deblurring process, comparing the performance and efficacy of Wilkinson's shift-based QR iteration and Cholesky-based simplifications. The project thereby contributes to the broader understanding of numerical methods for image restoration.

2.2 Implementation

The deblurring algorithm developed in this project is a sequence that comprises the construction of blurring kernels, the application of SVD-based truncated inversion, and the eventual restoration of the blurred image. As an additional note, when dealing with colored images, we need to treat each channel separately. We are ignoring the Alpha (transparency) channel, so we are dealing with the first 3 RGB channels. We treat each R, G, and B for both blurring and deblurring separately, then recombining all of them at the end. As for PSNR calculation, the MSE is averaged over all channels, and then we calculate PSNR using the averaged MSE.

2.2.1 Constructing the Blurring Kernel

The creation of the blurring kernel is an intricate process designed to emulate the effect of blurring on an image. Specifically, the kernel is constructed as a matrix $\boldsymbol{A}$ that models linear motion blur.

The BlurringKernel function generates this matrix according to the dimensions of the image n and the parameters j and k, which determine the spread and intensity of the blur.

The weights within the blurring matrix $\boldsymbol{A}$ are assigned based on their relative position to the center of motion. The kernel is populated in a manner where the main diagonal and its parallels are assigned weights that linearly decrease from the central axis of motion, capturing the gradation of the blur effect. Mathematically, this is represented as:

$$\boldsymbol{A}_{i,j} = \begin{cases} \frac{2}{k(k+1)}(i+1) & \text{if } i+j = n+1 \text{ to } n+k \\ 0 & \text{otherwise} \end{cases}$$

The resulting matrix $\boldsymbol{A}$ is thus an embodiment of the motion blur, with each entry denoting the blurring intensity applied to the corresponding pixel in the image.

2.2.2 Truncated Inverse and Deblurring

The core of the deblurring mechanism lies in the application of a truncated inverse to the blurring kernel. The TruncatedInverse function is implemented to execute this task by leveraging SVD. The SVD breaks down the matrix $\boldsymbol{A}$ into its singular components as $\boldsymbol{A} = \boldsymbol{U\Sigma V}^T$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices containing the left and right singular vectors, respectively, and $\boldsymbol{\Sigma}$ is a diagonal matrix comprising the singular values.

To mitigate the blur, we employ a truncation strategy where only the prominent singular values—those that are significantly higher than the noise threshold—are inverted. This truncation is governed by a threshold l, which dictates the count of singular values to be utilized in the inversion. The singular values are sorted in descending order, and their corresponding indices are ascertained. The pseudo-inverse of the matrix $\boldsymbol{A}$ is then computed as:

$$\boldsymbol{A}^+ = \sum_{i=1}^{l} \frac{\boldsymbol{v}_i \boldsymbol{u}_i^T}{\boldsymbol{\sigma}_i}$$

where $\boldsymbol{v}_i$ and $\boldsymbol{u}_i$ are the i^{th} columns of $\boldsymbol{V}$ and $\boldsymbol{U}$, and $\boldsymbol{\sigma}_i$ is the i^{th} singular value. This summation effectively recomposes the image by emphasizing the most significant components that characterize the original content while suppressing the elements that contribute to the blur.

The TruncatedInverse function, therefore, generates an approximate inverse of the blurring kernel, which, when applied to the blurred image $\boldsymbol{B}$, reconstructs the original, unblurred image $\boldsymbol{X}$ by the operation $\boldsymbol{X}_{trunc} = \boldsymbol{A}^+_{\ell,trunc}\boldsymbol{B}\boldsymbol{A}^+_{r,trunc}$, hence achieving the deblurring.

2.3 Results and Observations

We conduct our blurring kernel using the configuration: $j_l = j_r = 2$, $k_l = 96, k_r = 128$. For the truncated parameter l, for an image with size $n \times n$, we choose $l_l = l_r = \frac{n}{2}$ The results from the application of our deblurring algorithms are as follows

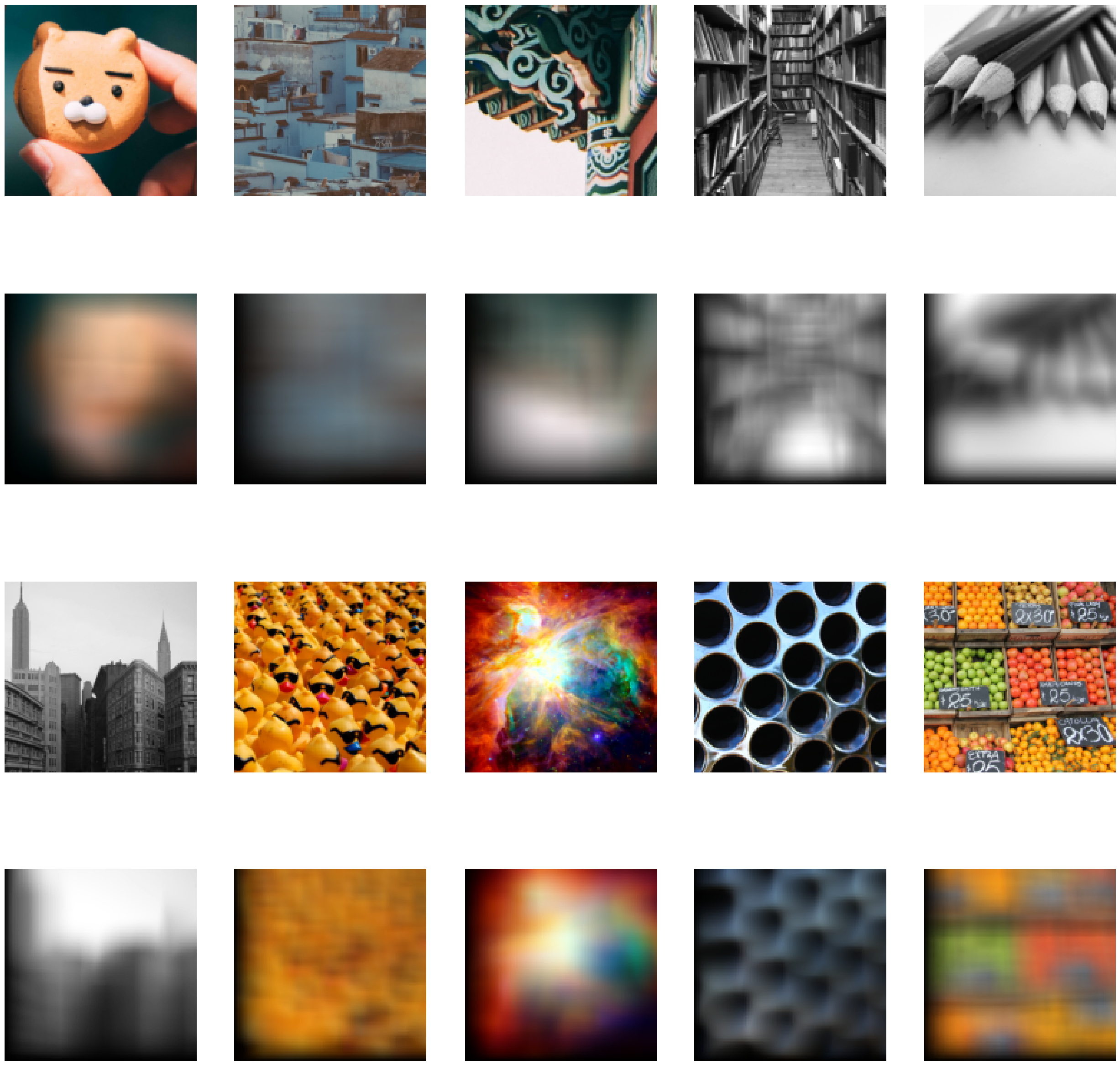

Figure 1: Original and Blurred Images.

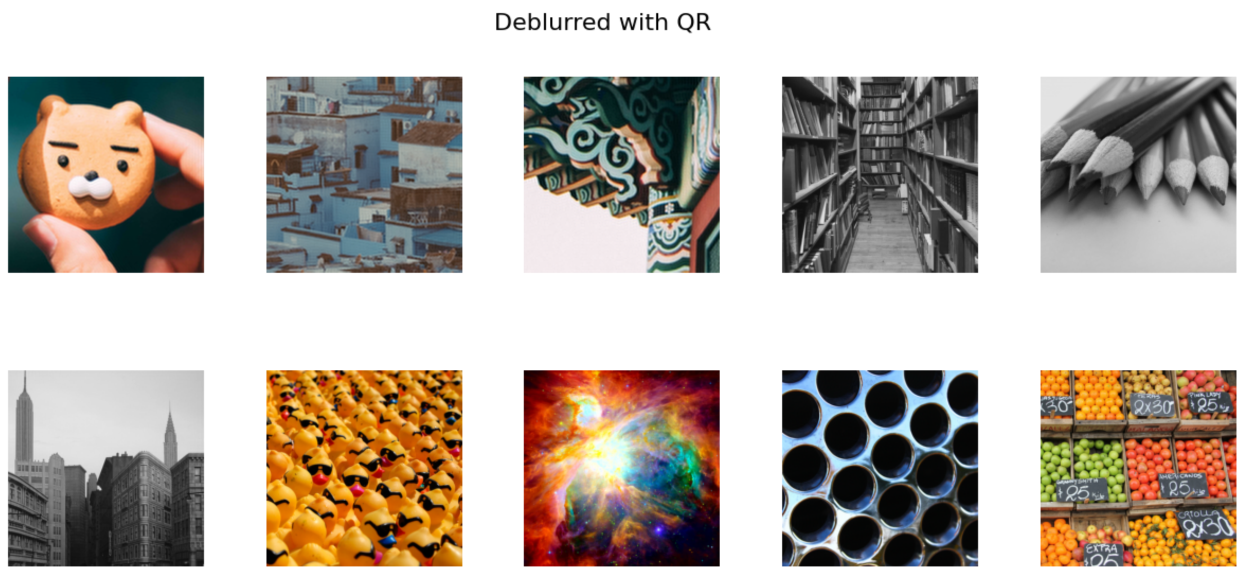

Figure 2: Deblurred Images using QR iteration with Wilkinson shift.

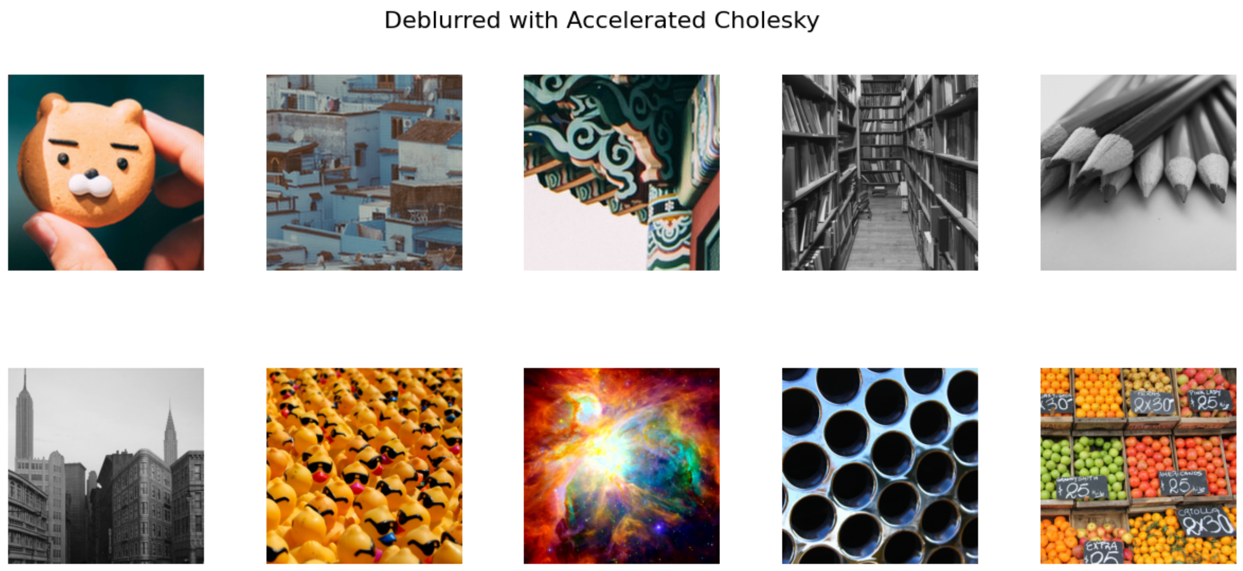

Figure 3: Deblurred Images using Cholesky decomposition.

Size	Kernel	QR (seconds)	Cholesky (seconds)
256	Al	0.4515	1.6557
256	Ar	0.4603	1.6964
512	Al	4.1027	14.2129
512	Ar	3.9189	12.3914

Table 2: Inversion times for blurring kernels using QR and Cholesky factorization methods.

Size	Name	QR PSNR	Cholesky PSNR
256×256	hand	36.5698	36.5678
	buildings	30.2028	30.2023
	casino	30.0476	30.0454
512×512	books	28.0763	28.0766
	pens	37.3812	37.3809
	town_01	33.7947	33.7960
	ducks	35.4106	35.4094
	stars_01	35.4937	35.4932
	circles	33.3033	33.3047
	fruits	28.8321	28.8309

Table 3: Comparison of PSNR values between QR and Cholesky factorization methods for deblurring.

Based on the provided tables of runtime and PSNR values for different deblurring scenarios and images, we can analyze the performance and efficiency of the QR iteration with Wilkinson shift (Phase II–A) and the accelerated Cholesky decomposition (Phase II–B).

Performance Analysis: The PSNR (Peak Signal-to-Noise Ratio) values are indicative of the quality of image reconstruction—the higher the PSNR, the better the quality. Across various images, both QR and Cholesky methods deliver comparable PSNR values, indicating similar accuracy in reconstructing the singular values required for deblurring. For example, the 'hand' image at a resolution of 256x256 pixels shows an almost identical PSNR of 36.5698 for QR and 36.5678 for Cholesky, suggesting that the two methods have equivalent accuracy in this scenario.

Runtime Comparison: The runtime performance shows a significant difference between the two methods. The QR method consistently exhibits a faster convergence time compared to the Cholesky method. For instance, inverting the blurring kernel of size 256, the QR method completes in approximately 0.45 seconds, while the Cholesky takes about 1.66 seconds. This trend is observed in all listed cases, indicating that the QR method is more efficient computationally.

Conclusion: Considering the PSNR values, both Phase II–A (QR with Wilkinson shift) and Phase II–B (accelerated Cholesky) approaches yield comparably accurate results. However, when it comes to computational speed, the QR method outperforms the Cholesky method, generally converging faster. This could be particularly beneficial in time-critical applications or when processing a large number of images, where efficiency gains would accumulate significantly.

3 Video Background Extraction

3.1 Background

This project aims to extract the common background image within the video utilizing the SVD and power iterations to obtain the prevalent features of the video across the time frames. The prominent feature of the video is extracted by obtaining the largest singular value representation of the image matrix, representing the feature (e.g., background) that appears on the video across time.

3.2 Implementation

3.2.1 Matrix Representation of Video

To be able to manipulate the data of the video using its representation as a tensor (matrix in a higher dimension), we extract the video data with the help of the utility library *moviepy* by representing the video object as a VideoFileClip. We conduct the experiment on two scenario: colored and monotone video. For video with higher resolution, we convert the three channel of the video to a one channel representation of the video using rgb2gray function from the *skimage* library.

A video is represented as a 4-dimensional matrix $\boldsymbol{V} \in \mathbb{R}^{m \times n \times 3 \times s}$ where s be the number of frames (duration) of the video and $m \times n$ be the video resolution. To represent the video with a two dimensional matrix, we flatten the first three dimension (two for a black-and-white video) to obtain a matrix representation:

$$\boldsymbol{A} = [\boldsymbol{v}_1 \quad \boldsymbol{v}_2 \quad \boldsymbol{v}_3 \quad \dots \quad \boldsymbol{v}_s]$$

where $\boldsymbol{v}_i \in \mathbb{R}^{mnc}$ be the flattened representation of one frame of the video ($c = 1$ if we are using a monotone video)

3.2.2 Dominant Singular Value Decomposition

The background extraction use the singular value decomposition of $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$. However, considering the scale of the application, we only attempt to obtain enough information for us to construct the background image.

The background image is represented by the largest singular value representation of matrix $\boldsymbol{A}$, that is $\sigma_1, \boldsymbol{u}_1, \boldsymbol{v}_1$. $\boldsymbol{B}$ is obtained by

$$\text{vec}(\boldsymbol{B}) = \sigma_1(\boldsymbol{v}_1^T \boldsymbol{e}_1) \cdot \boldsymbol{u}_1$$

where $\boldsymbol{B}$ is the first column of the matrix $\sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T$

To obtain the singular value, we consider two approaches: power iteration and QR iteration. Since we are only concerned with the dominant singular value, then the most effective algorithm to use is the normalized power iteration.

To obtain $\mathbf{\Sigma}$, we can utilize either $\boldsymbol{A}^T\boldsymbol{A}$ or $\boldsymbol{A}\boldsymbol{A}^T$ to obtain λ_1 as the dominant eigenvalue of the matrix that we can further process to obtain the singular value of $\boldsymbol{A}$ via $\sigma_1 = \sqrt{\lambda_1}$. Considering the size of $\boldsymbol{A} \in \mathbb{R}^{mnc \times s}$ where $mnc \gg s$, we use $\boldsymbol{A}^T\boldsymbol{A}$ for the power iteration.

The power iteration procedure is relatively simple. Suppose we have $x_0 = e_1$, in each iteration we have

$$\tilde{\boldsymbol{x}}^k = \boldsymbol{Y}^k \qquad \boldsymbol{x}^k = \frac{\tilde{\boldsymbol{x}}^k}{\|\tilde{\boldsymbol{x}}^k\|} \qquad \boldsymbol{Y}^k = \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}^k \qquad \lambda_k = (\boldsymbol{x}^k)^T\boldsymbol{Y}^k$$

where the iteration is done until the convergence condition holds when $|\lambda_k - \lambda_{k-1}| < \epsilon$.

After getting $\sigma = \sqrt{\lambda}, \boldsymbol{v}_1 = x^k$, we obtain $\boldsymbol{u}$ by $\boldsymbol{u}_1 = \boldsymbol{A}\boldsymbol{v}_1\sigma^{-1}$. All the information is used to obtain vec($\boldsymbol{B}$). Finally, $\boldsymbol{B}$ is transformed back into its original dimension as a representation of the background image.

3.3 Results and Observations

We test our implementation using all the provided testcase. However, we only use the original colored video for the 640×360 video due to limited memory capacity of the computing device.

Size	Video	Frames	Power Iter SVD	Scipy SVD
640 × 360	walking	1332	15.07 s	13.61 s
	rooster01	345	4.15 s	4.33 s
	street	351	4.13 s	4.36 s
1280 × 720	tiger	883	11.79 s	12.96 s
	squirrel	971	11.68 s	13.61 s
	people01	341	6.06 s	5.76 s
	shoes	486	5.16 s	7.10 s
	city01	1296	18.05 s	17.90 s
	city02	919	11.29 s	13.49 s
	pigeons	971	11.80 s	14.69 s
	bangkok	651	6.98 s	10.53 s
	skateboarder	1060	14.48 s	14.84 s
	road	340	5.49 s	5.90 s
	pedestrians	393	5.42 s	6.31 s
	rooster02	345	5.44 s	5.86 s
1920×1080	sanfrancisco02	351	12.08 s	13.01 s
	waterhweel	502	13.93 s	18.47 s
	sunset	399	11.50 s	16.03 s
2560×1440	street	416	15.43 s	28.28 s
	people02	241	7.58 s	22.94 s

Table 4: Comparison of time between the power iteration SVD and SVD using scipy library

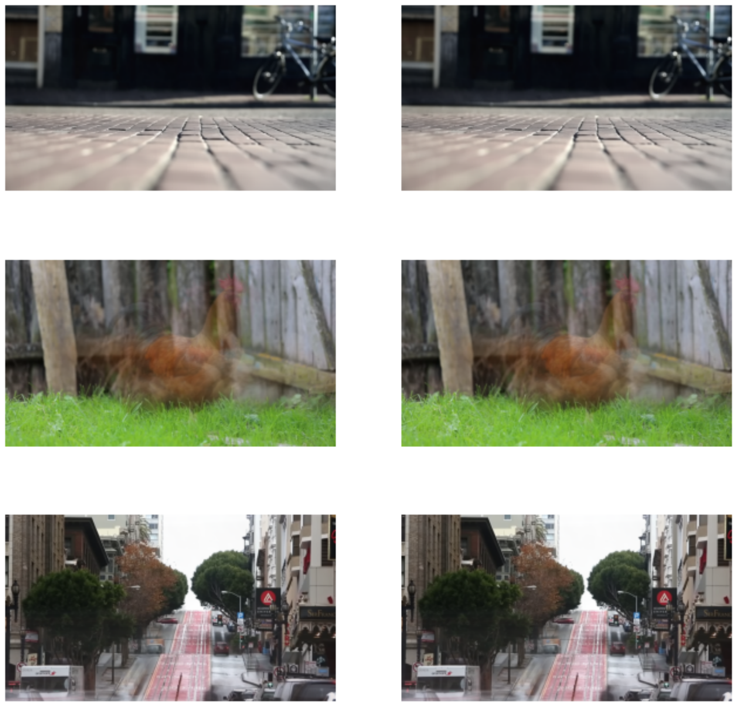

Figure 4: Background extraction of colored video 640 x 360, left using power iteration svd and right using scipy svd

Evaluating Performance Based on the results, both SVD has identical output visible to eyes. However, the difference is located at the time consumption between both SVD. Generally, the Power Iteration SVD achieves faster performance compared to that of scipy library with only a slight difference.

Another observation is the relation of the time consumed with the number of frames of the video. As we can expect, the number of frames correlates higher with the time as we conduct our iteration to $\boldsymbol{A}^T\boldsymbol{A}$, whose size is determined by s, the number of frames. The resolutions don't affect much except the performance when we do multiplication for retrieving $\boldsymbol{u}$ and transforming the vector representation of $\boldsymbol{B}$ back to its original dimension representation. The total computation complexity is $O(n^2)$ for each iteration for multiplying matrix-vector and calculating the norm. Hence it is faster compared to QR iteration with $O(n^3)$ complexity for every iteration.

Conclusion Considering the time comparison between both SVDs, the Power Iteration SVD is generally faster than the Scipy SVD. The difference of time consumption lies within the recovery time of the singular value as power iteration only deals with the dominant eigenvalue and hence will converge faster than recovering the whole SVD of the matrix video. The main takeaway is to adjust the computation based on the problem's neccesity. In this case, recovering only the dominant singular value instead of all— such as when using QR iteration — can improve the algorithm significantly.

Appendix A: Background Extraction Result

Note The left image corresponds to our implementation of power iteration, and the right images are the scipy's SVD

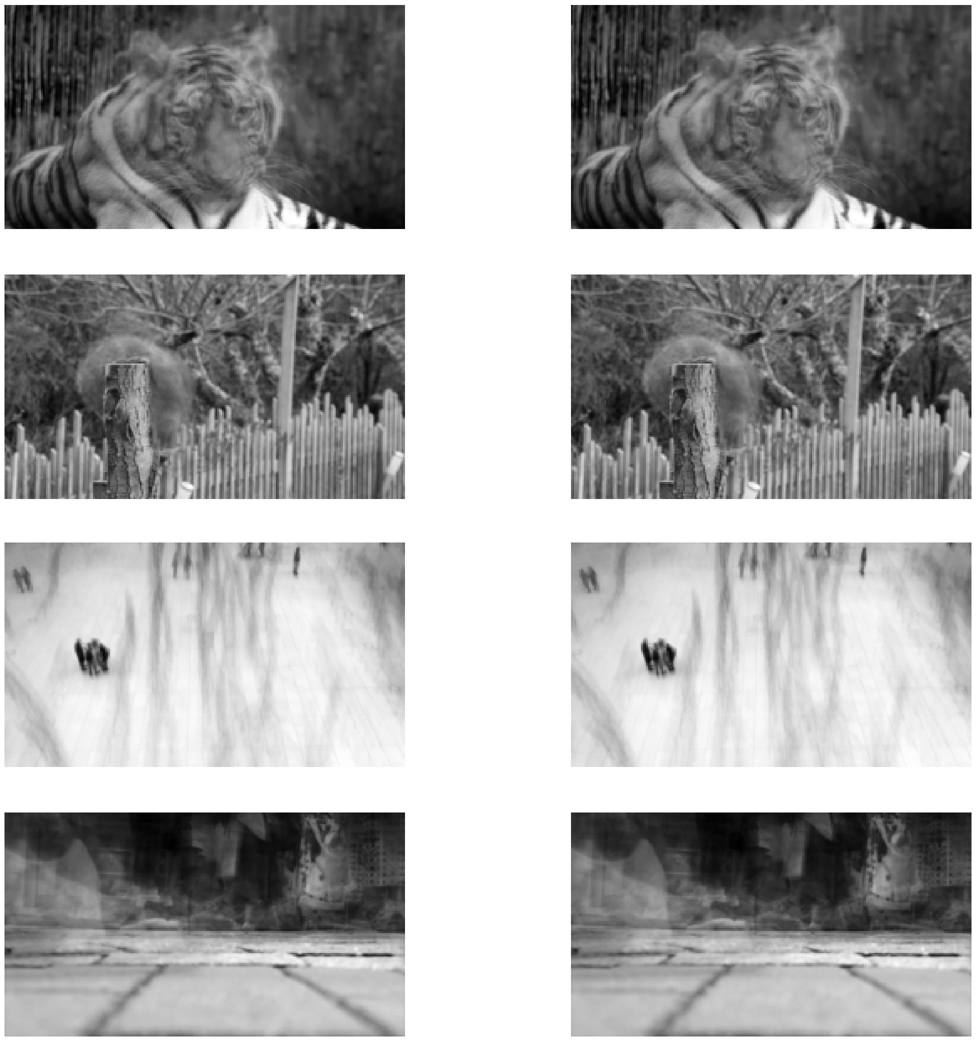

Figure 5: Background extraction of 1280 x 720 videos (tiger, squirrel, people01, shoes)

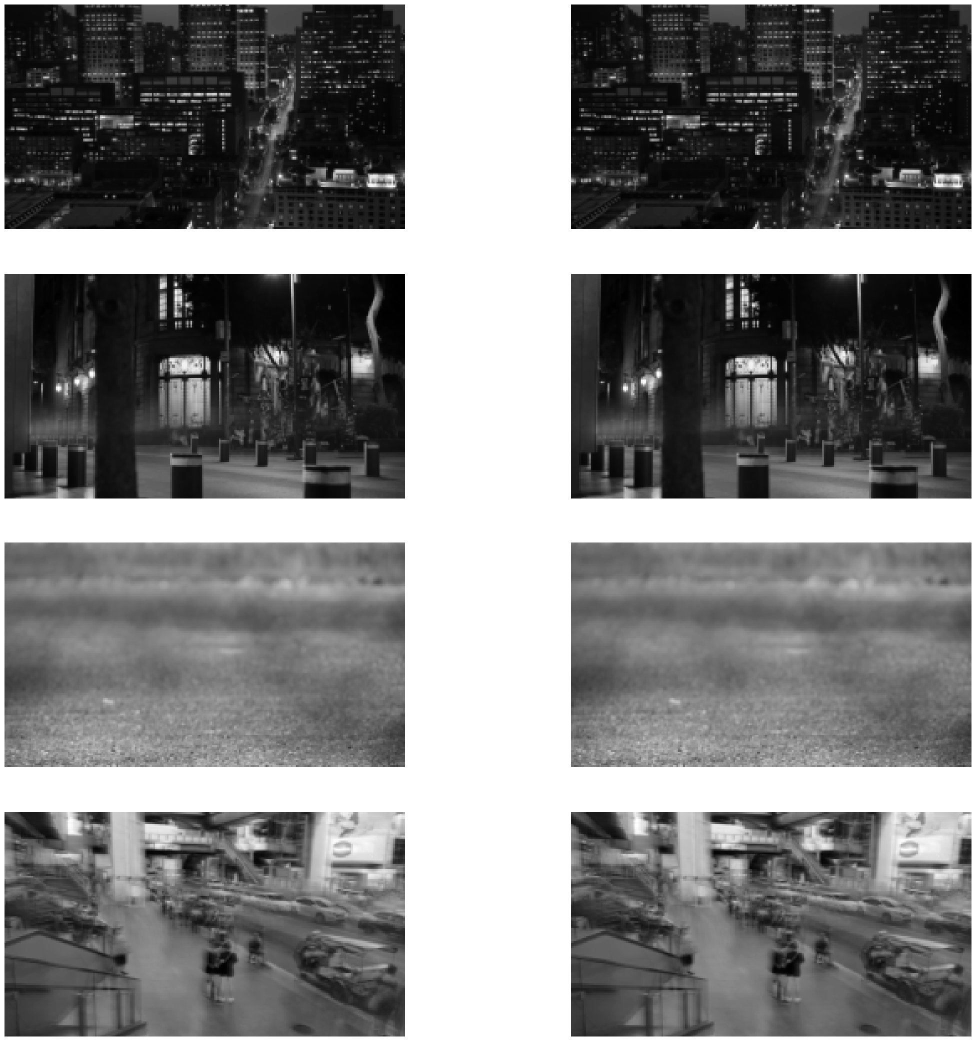

Figure 6: Background extraction of 1280 x 720 videos (city01, city02, pigeons, bangkok)

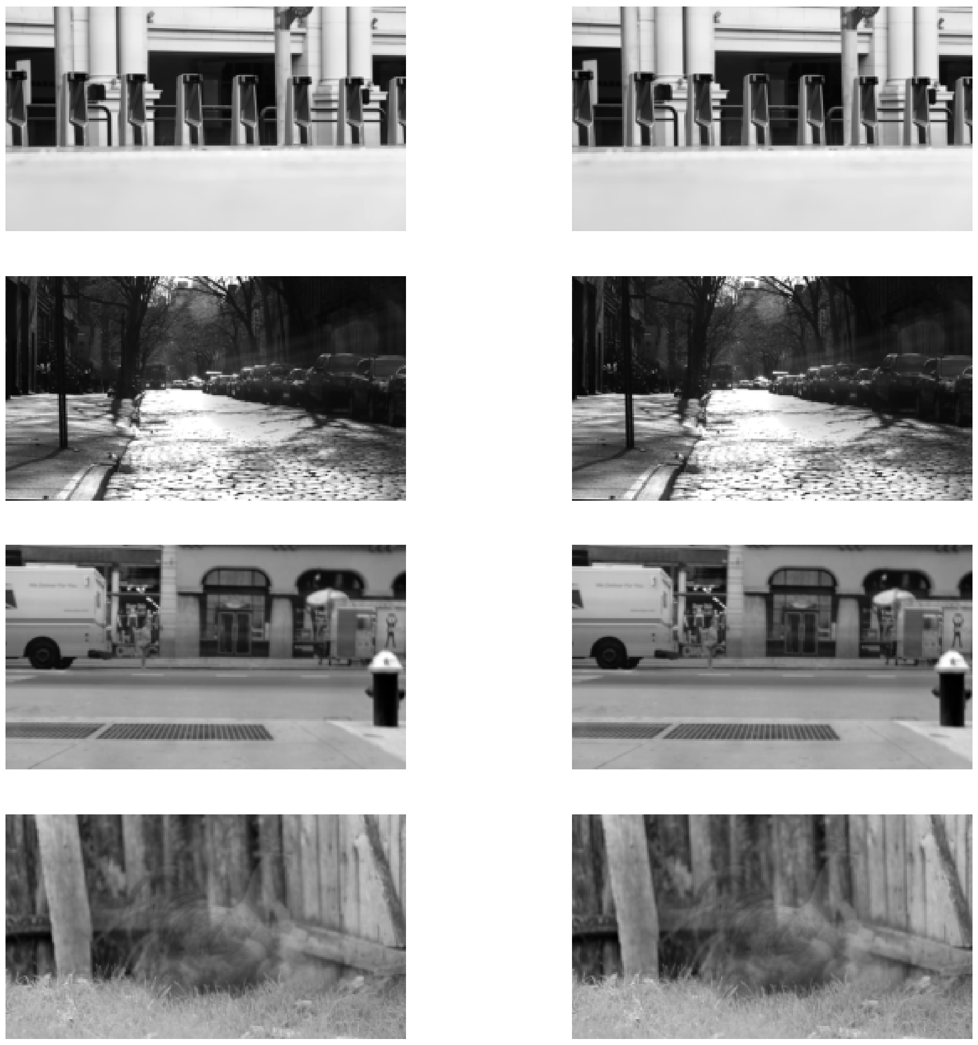

Figure 7: Background extraction of 1280 x 720 videos (skateboarder, road, pedestrians, rooster02)

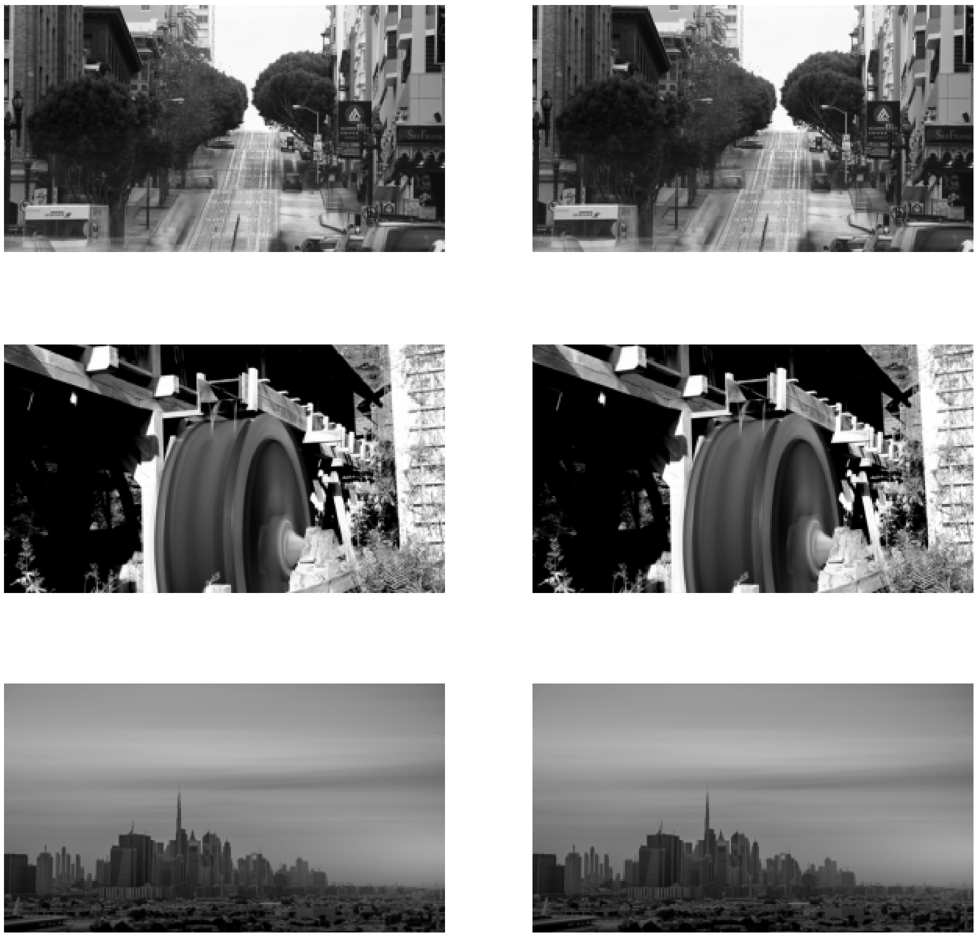

Figure 8: Background extraction of 19200 x 1080 videos

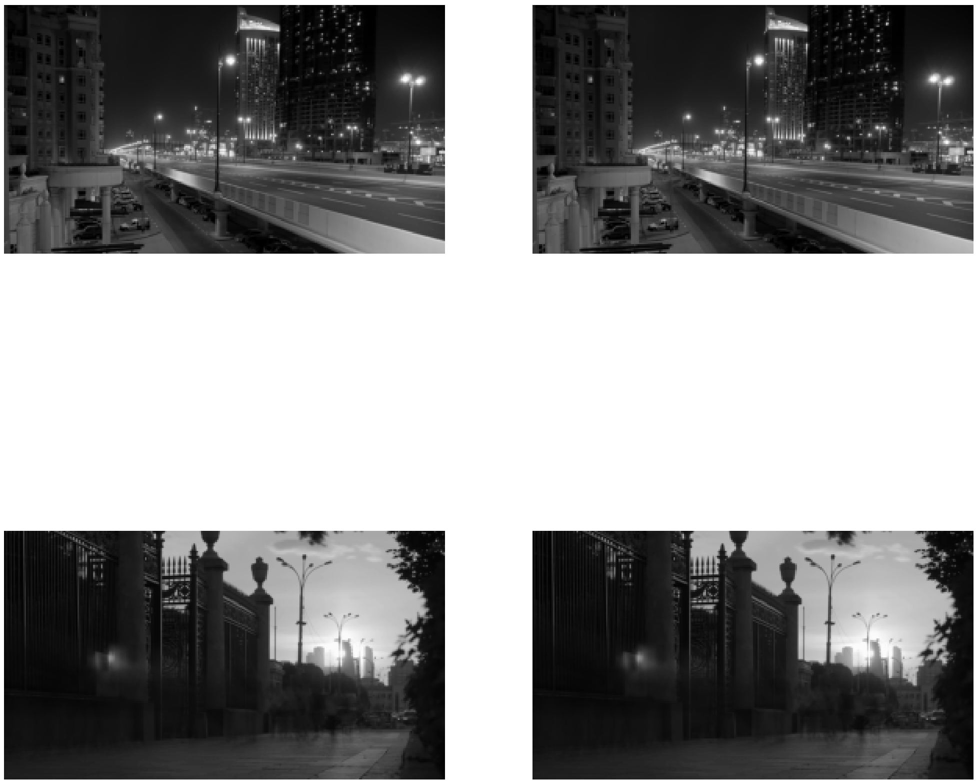

Figure 9: Background extraction of 2560 x 1440 videos