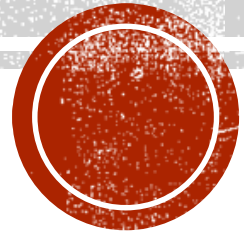
# TENSOR COMPLETION

## FOR ESTIMATING MISSING VALUES IN VISUAL DATA



# WHAT IS A TENSOR?

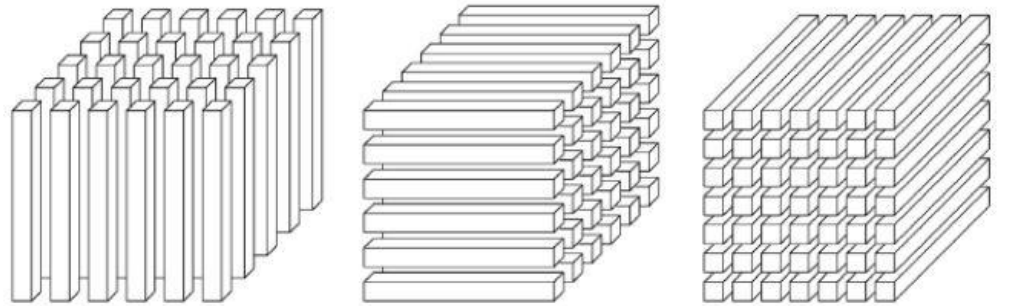
- N-dimensional array
- Can be thought of as a stack of N m×n matrices
- A color image is a 3-D array
- Three matrices for R G & B
- A color image is a mode 3 Tensor
- A grayscale image is a mode 2 Tensor
- A vector is a mode 1 Tensor
- A scalar is a mode 0 Tensor

# TENSOR DECOMPOSITION

- Most common approach to tensor computations consists of three steps:
  - Turn the tensor into a matrix by *unfolding* it
  - Perform computations on the matrix to gather information about it
  - Infer information about the tensor from the information gathered on the matrix
- Notation:
  - $\mathcal{X}$ → 3rd mode tensor,
  - $\mathcal{X}_{(i)}$ → *unfold* tensor along mode *i*
  - $X$ → a matrix
- $\mathcal{X} \in \mathbb{R}^{n_1 \times \cdots \times n_i} \Rightarrow \mathcal{X} \in \mathbb{R}^{n \times m \times 3}$ For a colored image
- $\mathcal{X}_{ijk}$ refers to the number in row i, column j, matrix k
- Mode-3 tensors have three *fibers*
  - Column, Row, and Tube
- Mode-3 tensors have three *slices*
  - Horizontal, Frontal, and Lateral

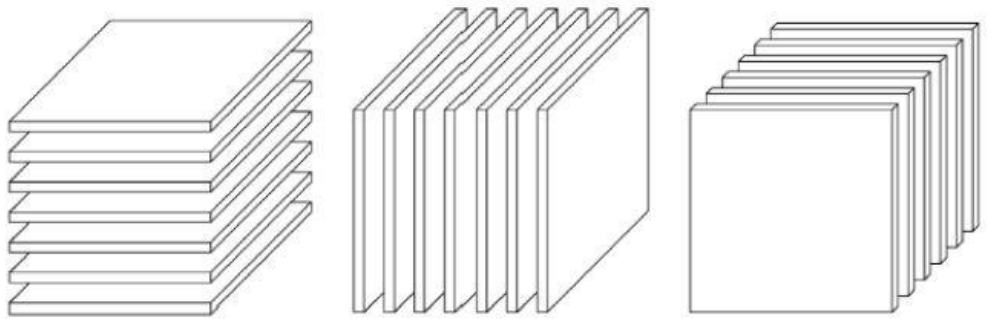
# TENSOR DECOMPOSITION

TAMARA G. KOLDA AND BRETT W. BADER



(a) Mode-1 (column) fibers: $\mathbf{x}_{:jk}$ (b) Mode-2 (row) fibers: $\mathbf{x}_{i:k}$ (c) Mode-3 (tube) fibers: $\mathbf{x}_{ij:}$

*Fibers of a 3rd-order tensor.*



(a) Horizontal slices: $\mathbf{X}_{i::}$ (b) Lateral slices: $\mathbf{X}_{:j:}$ (c) Frontal slices: $\mathbf{X}_{::k}$ (or $\mathbf{X}_k$)

*Slices of a 3rd-order tensor.*

# TENSOR DECOMPOSITION

- Modal unfolding transforms a mode-3 tensor into a matrix
  - $\mathcal{X}_{(1)} = X \in \mathbb{R}^{n \times (m \cdot p)}$
  - $\mathcal{X}_{(2)} = X \in \mathbb{R}^{m \times (n \cdot p)}$
  - $\mathcal{X}_{(3)} = X \in \mathbb{R}^{p \times (m \cdot n)}$
- Can be implemented using the `reshape` function in Python

```
def m_unfold(tensor, m):
    matrix = []
    if m == 1:
        matrix = tensor.reshape((tensor.shape[0], tensor.shape[1]*3))
    if m == 2:
        matrix = tensor.reshape((tensor.shape[1], tensor.shape[0]*3))
    if m == 3:
        matrix = tensor.reshape((3, tensor.shape[0]*tensor.shape[1]))
    return matrix
```

# TENSOR DECOMPOSITION

- Use the `fold` function to fold a matrix back into a tensor
  - ```
    def fold(matrix, original_shape):
        tensor = matrix.reshape(original_shape)
        return tensor
    ```
- Can also use the TensorLy library's `fold` & `unfold` functions
- Three tensor completion algorithms discussed in this paper:
  - SiLRTC – Simple Low Rank Tensor Completion
  - FaLRTC – Fast Low Rank Tensor Completion
  - HaLRTC – High Accuracy Low Rank Tensor Completion
- All three algorithms use the `fold` and `unfold` operations as well as the *shrinkage* and *truncate* operators as their basic underlying machinery


    ```

# TENSOR DECOMPOSITION

- The reason we *unfold* a tensor is so that we may perform Singular Value Decomposition (SVD) on the resulting matrices
  - $X = U\Sigma V^T$ where $\Sigma = diag(\sigma_i)$ and $\sigma_i$ are the singular values of $X$
    - `U, sig, V = numpy.linalg.svd(`$\mathcal{X}_{(i)}$`, full_matrices=False)`
  - Shrinkage operator:
    - $\mathbf{D}_\tau(X) = U\Sigma_\tau V^T$ where $\Sigma_\tau = diag(\max(\sigma_i - \tau, 0))$
    - ```
      def shrinkage_op(u,v,sig,tau):
          sig_diag = sig - tau
          np.maximum(sig_diag,np.zeros(len(sig)),sig_diag)
          shrink = np.dot(u[:,:], np.dot(np.diag(sig_diag[:]), v[:,:]))
          return shrink
      ```
  - Truncate operator:
    - $\mathbf{T}_\tau(X) = U\Sigma_{\bar{\tau}} V^T$ where $\Sigma_{\bar{\tau}} = diag(\min(\sigma_i, \tau))$
    - ```
      def truncate_op(u,v,sig,tau):
          t = np.zeros(len(sig))+tau
          np.minimum(sig,t,sig)
          trunk = np.dot(u[:,:], np.dot(np.diag(sig[:]), v[:,:]))
          return trunk
      ```

# SILRTC ALGORITHM

- **SiLRTC :**
  - Input: $\boldsymbol{\mathcal{X}}$ with $\boldsymbol{\mathcal{X}}_{\Omega} = \boldsymbol{\mathcal{T}}_{\Omega}$, $\boldsymbol{\beta_i}'$s, and K
  - Output: $\boldsymbol{\mathcal{X}}$

```
for k = 1 to K do:
    for i = 1 to n do:
        M_i = D_{α_i/β_i}(X_(i))
    end for
    update X by Equation 17
end for
```

- **Equation 17 :**
  - $$\mathcal{X}_{i_1,\ldots,i_n} = \begin{cases} \left(\frac{\sum_i \beta_i \mathrm{fold}_i(M_i)}{\sum_i \beta_i}\right)_{i_1,\ldots,i_n} & \text{for } (i_1,\ldots,i_n) \notin \Omega \\ \mathcal{T}_{i_1,\ldots,i_n} & \text{for } (i_1,\ldots,i_n) \in \Omega \end{cases}$$

# SILRTC ALGORITHM

- Implementation:
  - ```
    def SiLRTC(X, b, K, Omega):
        a = np.random.dirichlet(np.ones(3),size=1)
        a = a[0,:]
        M = [0, 0, 0]
        for i in range(K):
            X1 = tl.unfold(X, 0)
            X2 = tl.unfold(X, 1)
            X3 = tl.unfold(X, 2)
            u1, s1, v1 = LA.svd(X1, full_matrices=False)
            u2, s2, v2 = LA.svd(X2, full_matrices=False)
            u3, s3, v3 = LA.svd(X3, full_matrices=False)
            U = [u1,u2,u3]
            V = [v1,v2,v3]
            S = [s1,s2,s3]
            for j in range (3):
                M[j] = shrinkage_op(U[j],V[j],S[j],a[j]/b[j],X.shape)
            X = eq_svtn(X,b,M,Omega)
        return X
    ```


    ```

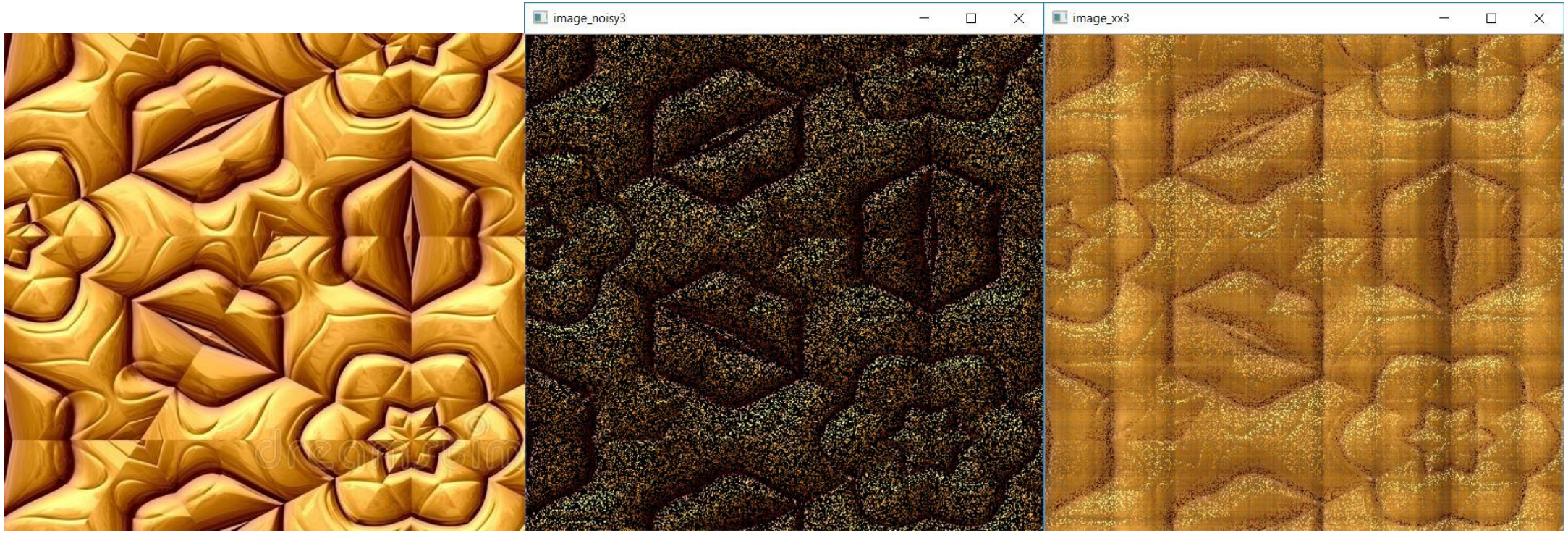
# SILRTC ALGORITHM

- Implementation:
  - ```
    def eq_svtn(T, b, M, Omega):
        X = 0
        for i in range(3):
            X += b[i]*tl.fold(M[i],i,T.shape)
        X = X/np.sum(b)
        for i in range(T.shape[0]):
            for j in range(T.shape[1]):
                #if Omega[i, j, :].all() != 0:
                if Omega[i, j, :].any() != 0:
                        X[i, j, :] = T[i, j, :]
    return X
    ```


    ```

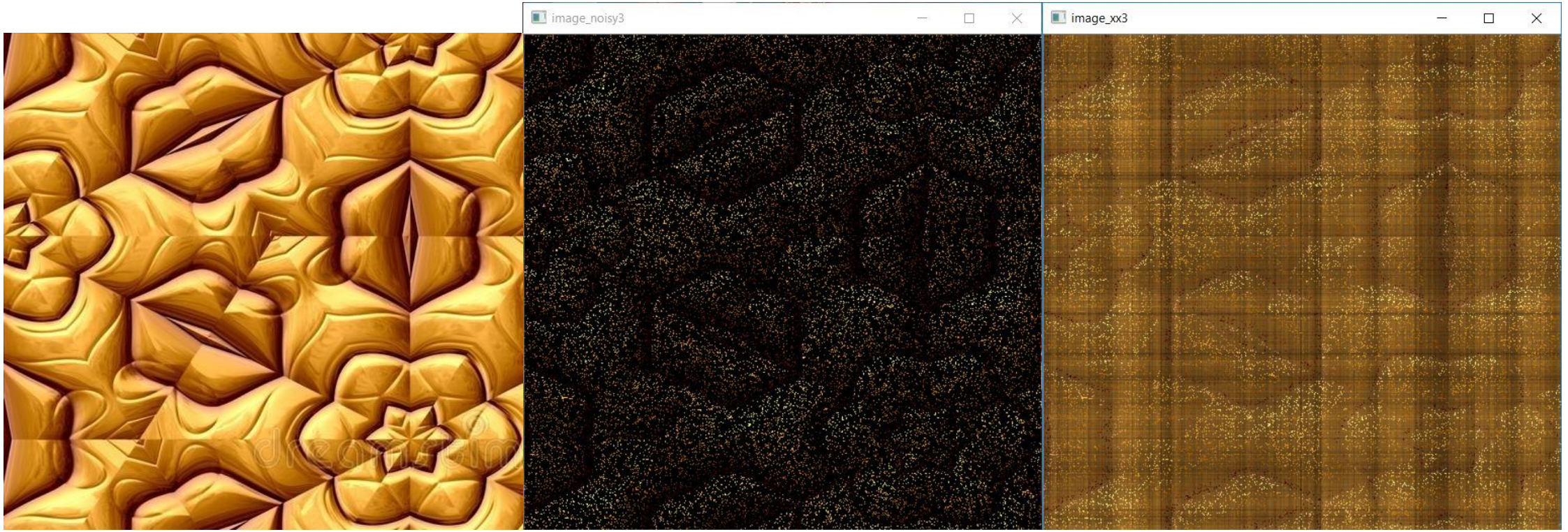
# RESULTS FROM SILRTC

- 512 loops with 75% of pixels missing from the original image:



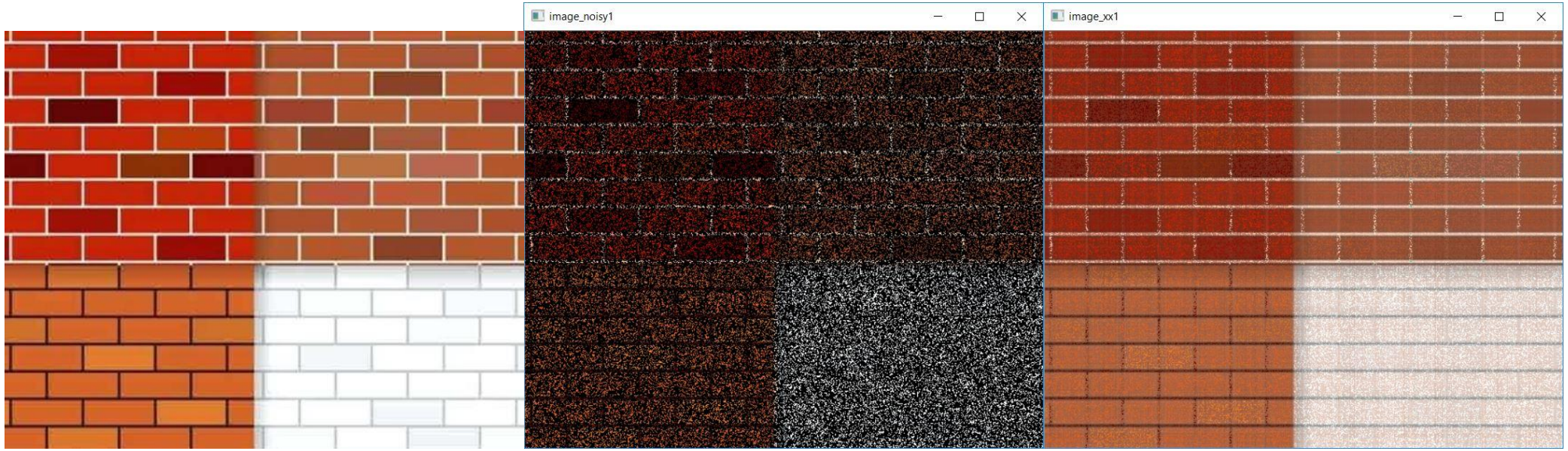
# RESULTS FROM SILRTC

- 512 loops with 90% of pixels missing from the original image:



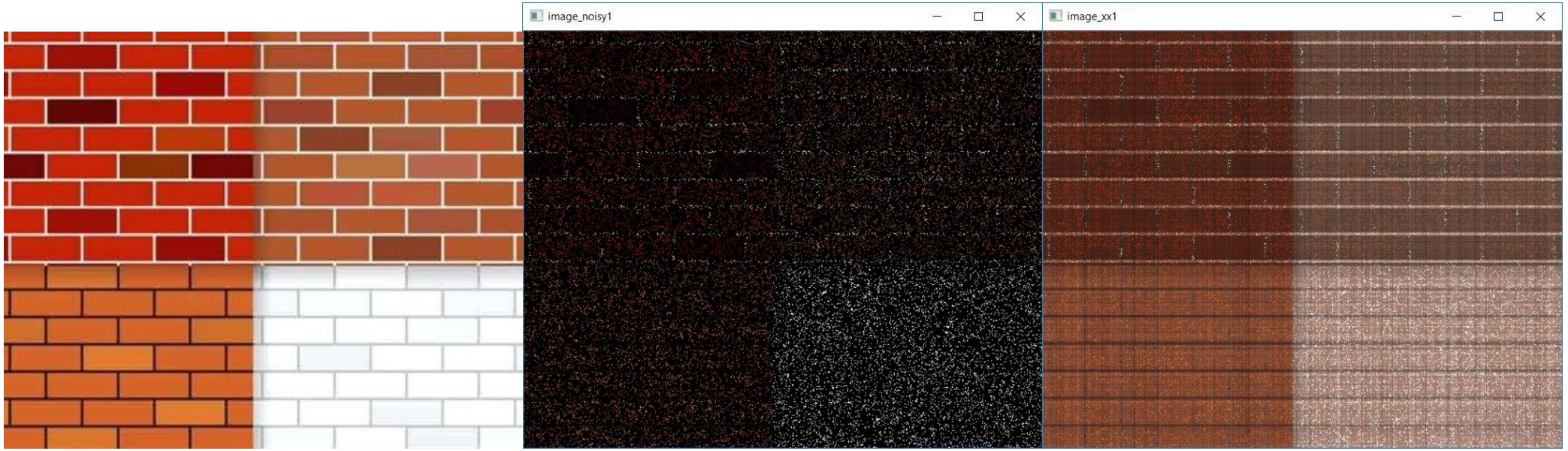
# RESULTS FROM SILRTC

- 512 loops with 75% of pixels missing from the original image:



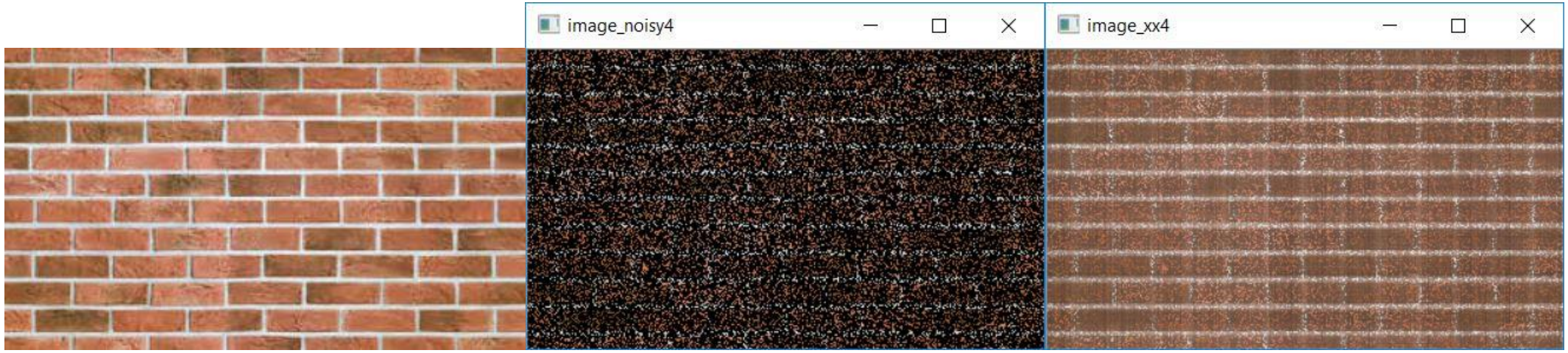
# RESULTS FROM SILRTC

- 512 loops with 90% of pixels missing from the original image:



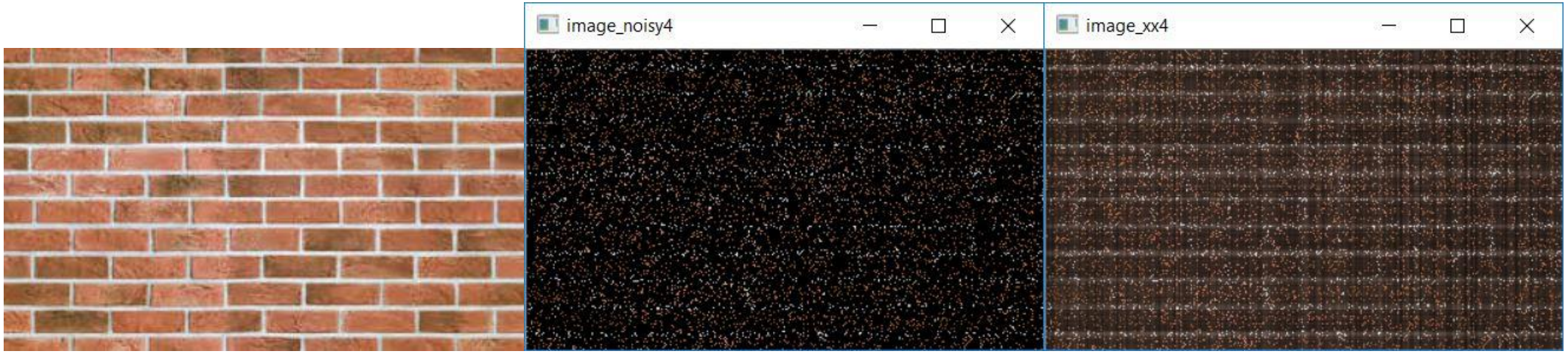
# RESULTS FROM SILRTC

- 512 loops with 75% of pixels missing from the original image:



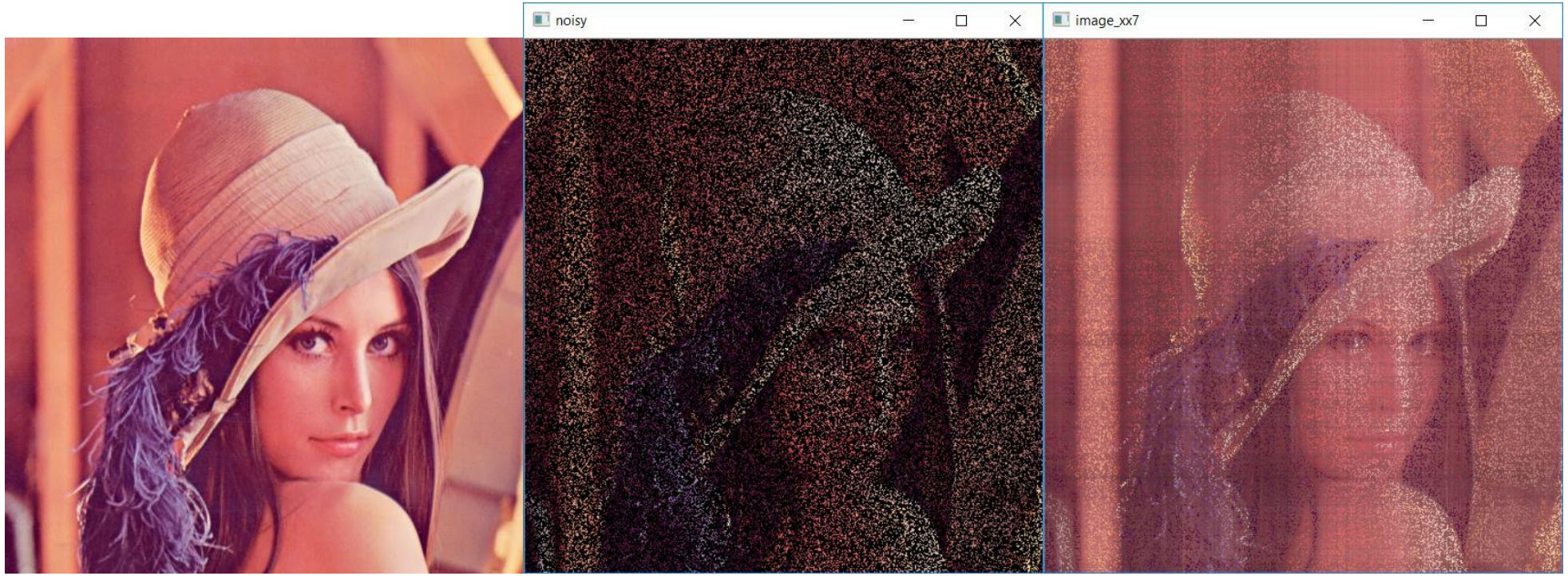
# RESULTS FROM SILRTC

- 512 loops with 90% of pixels missing from the original image:



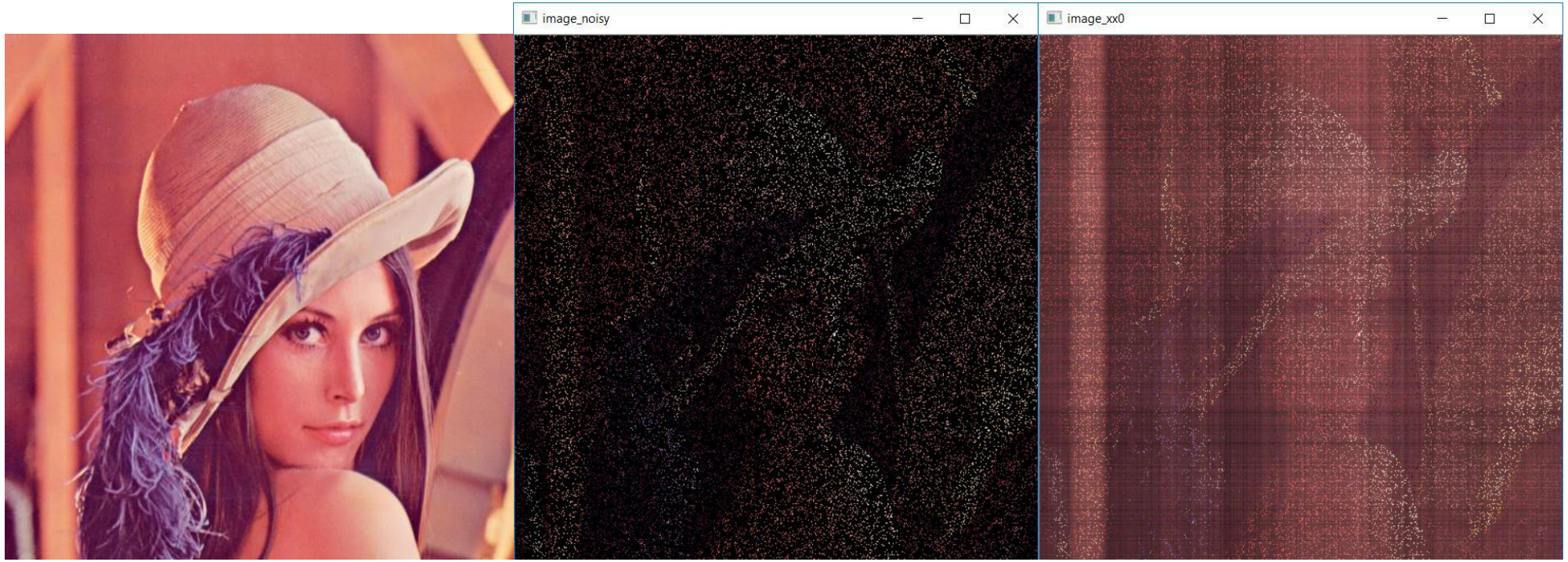
# RESULTS FROM SILRTC

- 512 loops with 75% of pixels missing from the original image:



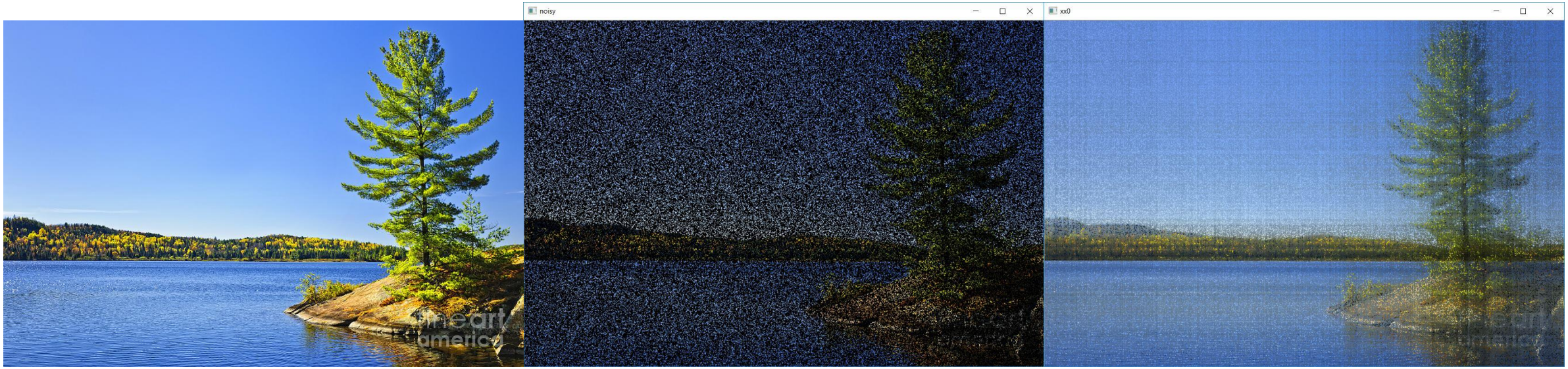
# RESULTS FROM SILRTC

- 200 loops with 90% of pixels missing from the original image:



# RESULTS FROM SILRTC

- 16 loops with 75% of pixels missing from the original image:

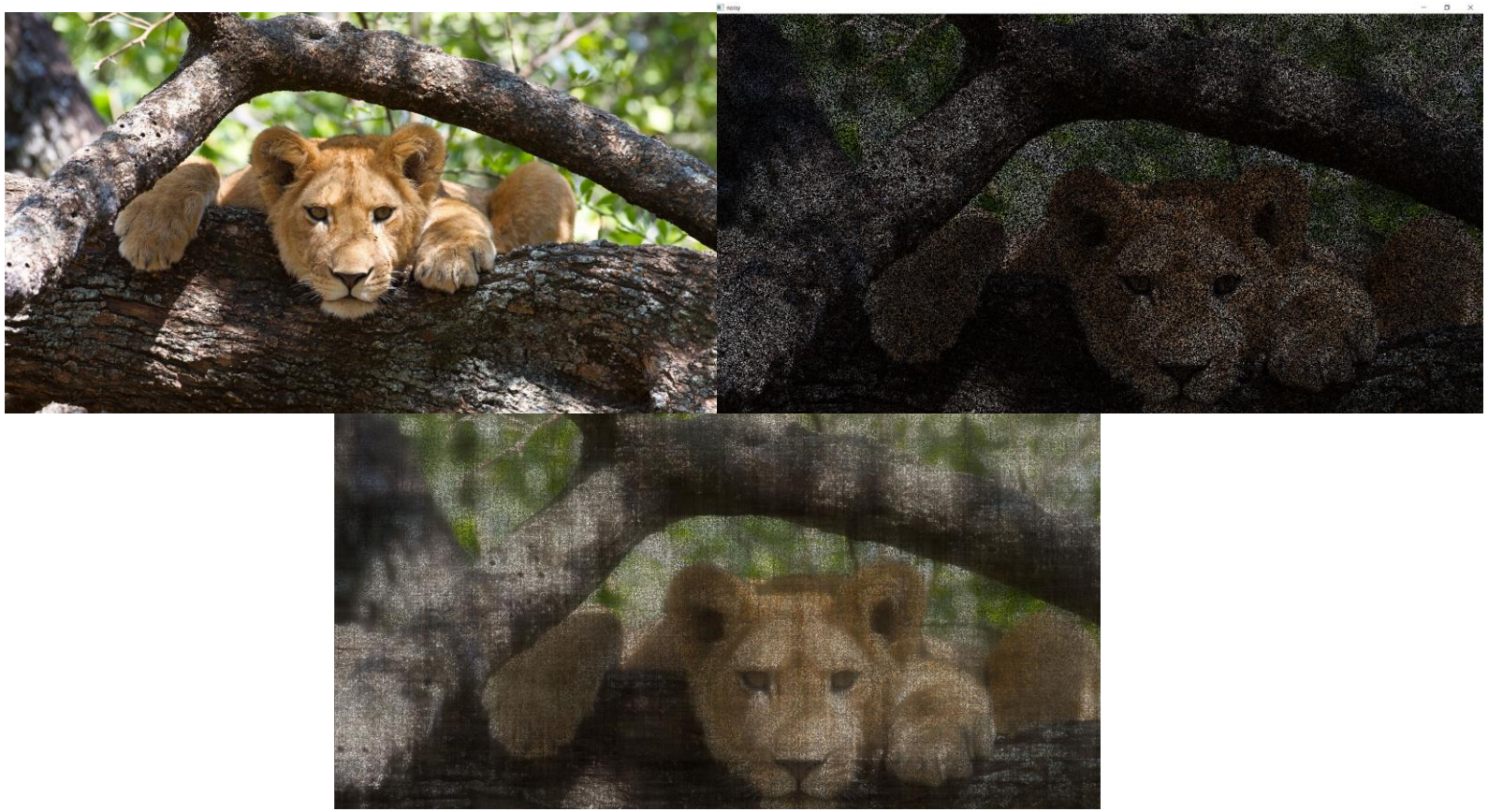


# RESULTS FROM SILRTC

- 16 loops with 90% of pixels missing from the original image:

# RESULTS FROM SILRTC

- 16 loops with 75% of pixels missing from the original image:



# RESULTS FROM SILRTC

- 32 loops with 75% of pixels missing from the original image: