Compressed Sensing

Nicholas Chiapputo

EENG 5850.001

April 29, 2021

Motivation

- Conventional Image Capture:
  - Capture tons of data
  - ...throw most of it away

- Why not only capture the important information?
  - “Why go to so much effort to acquire **all** the data when **most** of what we get will be thrown away? Can’t we just **directly measure** the part that won’t end up being thrown away?” -- David Donoho
  - Problem: What data is important?

- Solution: Compressed Sensing
  - Compression during capture (on the fly)
  - Randomly select data to retrieve
  - Sparse optimization

Compressed Sensing

- Process
  - Take random projections (measurements) of the original signal
  - Recover the signal using optimization techniques
- How can we sample below the Nyquist rate?
  - Nyquist-Shannon theorem assumes a broadband signal
  - The signal is assumed to be “sparse” (most natural signals are) or compressible
    - Sparse: only a few frequencies are present
  - Transform into a domain in which it has a sparse representation
  - A signal is sparse if it most of its components are at or near zero
- Reconstruction
  - Sparse optimization

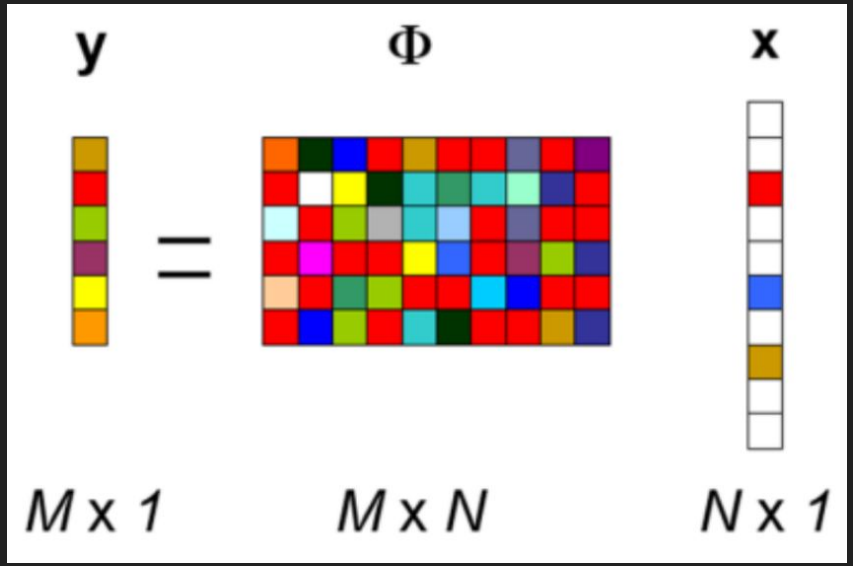
Problem

- Underdetermined linear system:

- $k \ll M \ll N$
- **x** is sparse with *k* non-zero entries
- ***A*** is often also called **Φ**
- Underdetermined -- infinite solutions
  - Projection of *N*-dimensional vector into an *M* dimensional space

Problem

- Underdetermined linear system:

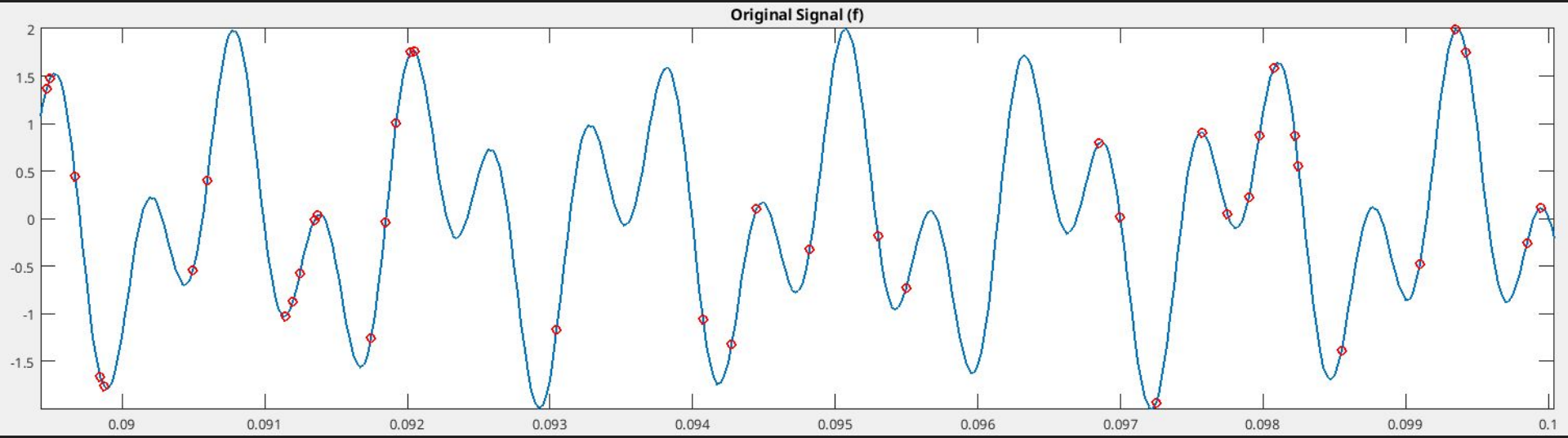


Visualized problem from [2]

How do we reconstruct?

- Solve the underdetermined equation $\boldsymbol{y} = \boldsymbol{\Phi x}$
- Ideal optimization: minimize $\|\boldsymbol{x}\|_0$ (number of non-zero elements in $\boldsymbol{x}$)
  - This is an NP-hard problem!
- Other Options:
  - $\|\boldsymbol{x}\|_1$: L_1-norm -- least absolute deviations or Basis Pursuit (sparse solutions)
  - $\|\boldsymbol{x}\|_2$: L_2-norm -- least squares (fastest)
- Method is known as *Basis Pursuit*

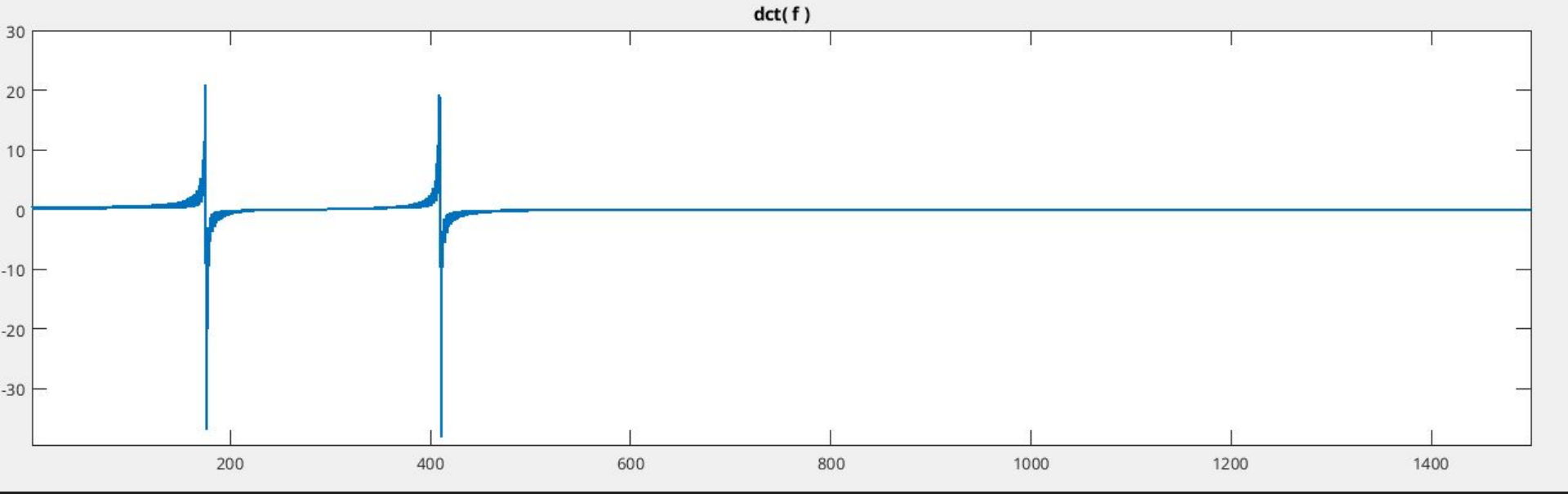
Comparing L^1- and L^2-norm



Audio signal

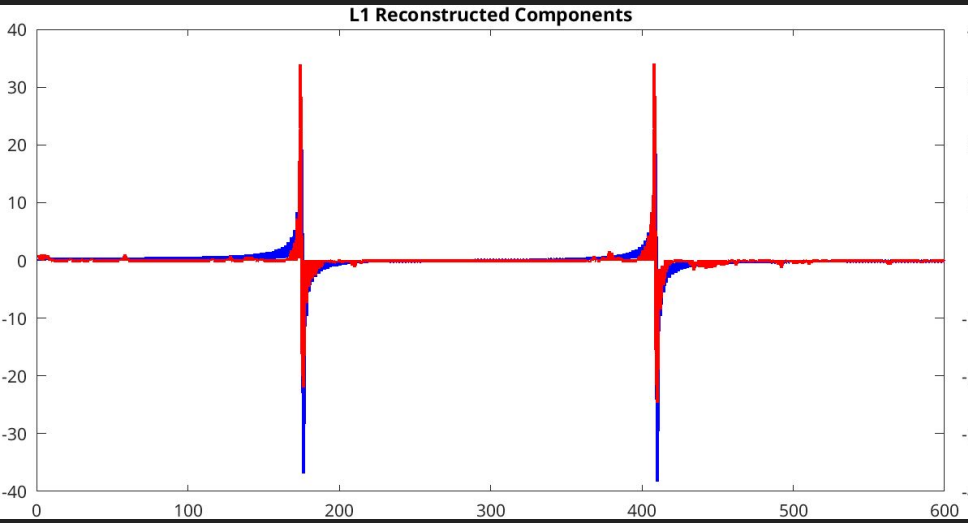
$f = \sin(1394*\pi*t) + \sin(3266*\pi*t)$

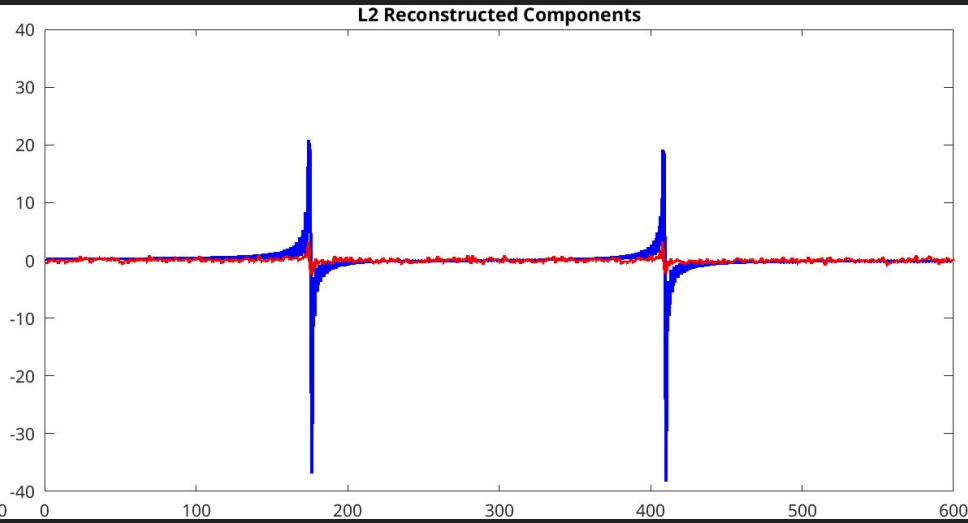
Comparing L^1- and L^2-norm



DCT Components

Comparing L^1- and L^2-norm





DCT Components

Comparing L_1- and L_2-norm

- L_2-norm is more sensitive to outliers, results in many small components
- L_1-norm does not punish outliers, results in more sparse results
  - We want sparsity!
- Solution: Constrain solution to minimum L_1 norm:

$$\text{minimize } \|\mathbf{x}\|_1$$
$$\text{s.t. } \boldsymbol{y} = \boldsymbol{\Phi x}$$

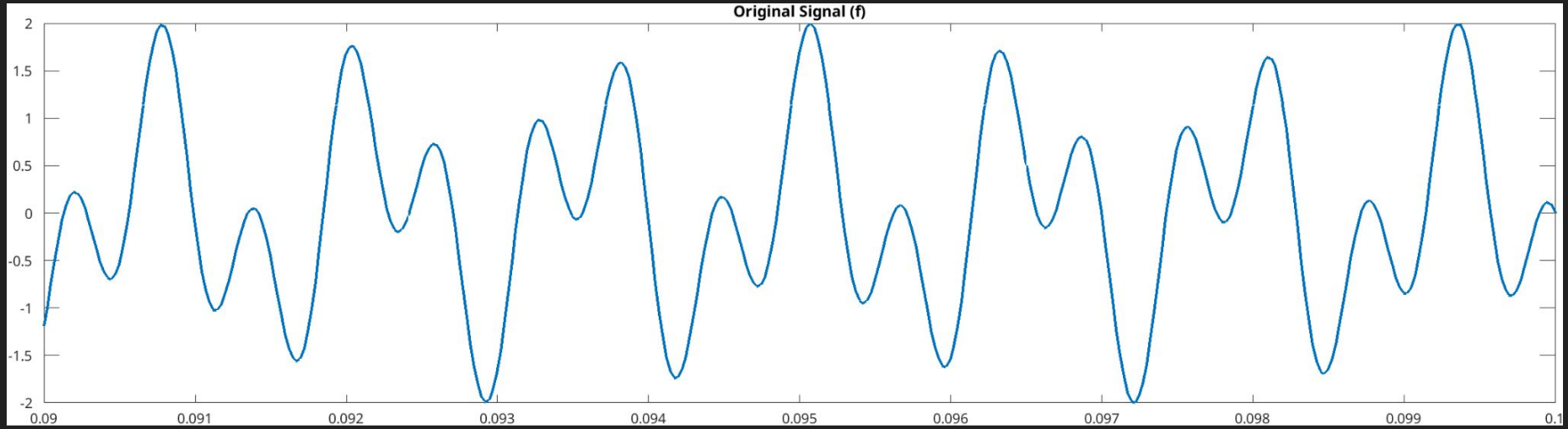
Calculating the Sensing Matrix (A or Φ)

- Start with $\boldsymbol{y} = \boldsymbol{\phi f}$
  - $\boldsymbol{f}$: Target signal (Nx1)
  - $\boldsymbol{\phi}$: Sampling/sparsifying matrix (MxN)
  - $\boldsymbol{y}$: Compressed/undersampled signal (Mx1)
- Let $\boldsymbol{f} = \boldsymbol{\psi x}$
  - $\boldsymbol{f}$: Reconstructed signal (Nx1)
  - $\boldsymbol{\psi}$: Transform matrix from spectral to temporal (frequency to time) domains (NxN)
    - Inverse DCT of $\boldsymbol{I_N}$
  - $\boldsymbol{x}$: Sparse frequency components (Nx1)
    - Also the solution to the optimization problem
- Substitute $\boldsymbol{f}$: $\boldsymbol{y} = \boldsymbol{\phi\psi x}$
- From $\boldsymbol{y} = \boldsymbol{Ax} \rightarrow \boldsymbol{A} \equiv \boldsymbol{\phi\psi}$
  - $\boldsymbol{A}$ is just subsampled rows of the transform matrix!

1-Dimensional CS

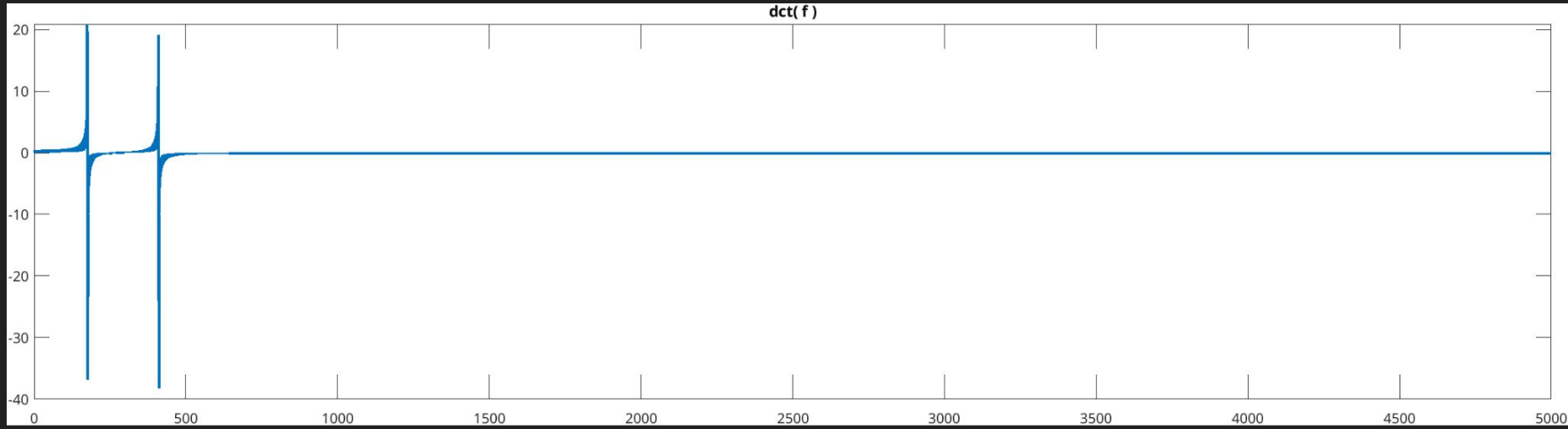
- Steps:
  1. Generate full signal of *N* samples
  2. Compute DCT components of original signal
  3. Extract *M* (e.g., 0.5*N*) samples from original signal -- ***y***
  4. Generate sensing matrix ***A*** (or **Φ**)
  5. Perform convex optimization
  6. Perform inverse DCT on optimized **x** to retrieve reconstructed signal

Audio Signal Example



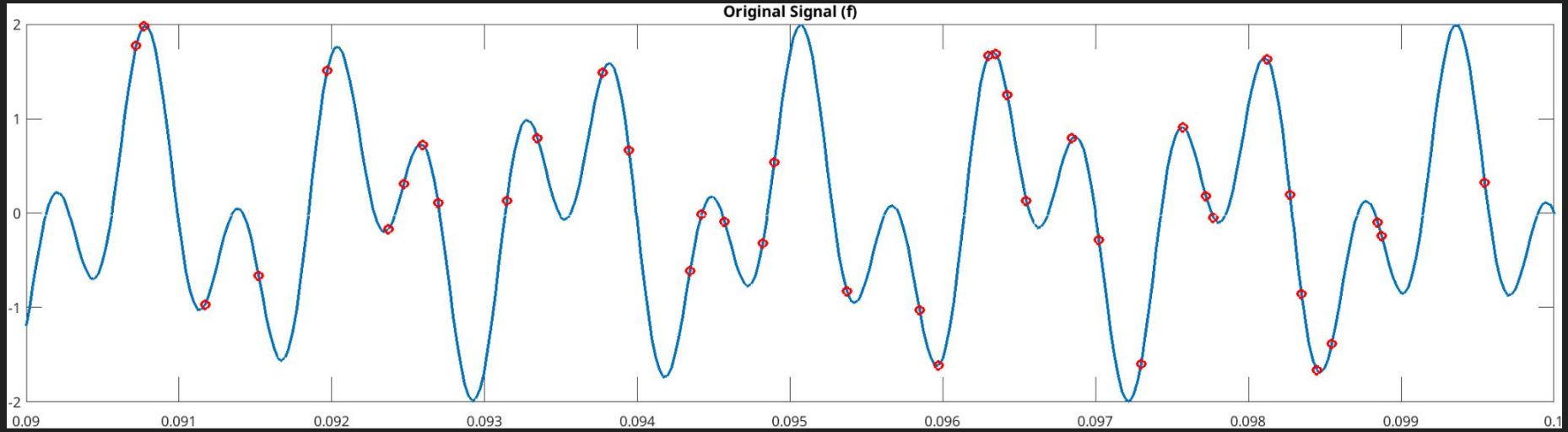
1. Generate full signal of $N = 5{,}000$ samples

Audio Signal Example



2. Compute DCT components of original signal

Audio Signal Example



3. Extract $M = 0.10N = 500$ samples from original signal

Audio Signal Example

4. Generate sampling matrix A (or Φ)

```
temp = randperm( n );    % Random permutation of [n]
ind = temp( 1:m );       % Grab the first 500 random indexes.

% Figure out mapping from time to frequency domain.
D = dct( eye( n, n ) );
A = D( ind, : );
```

5. Perform convex optimization

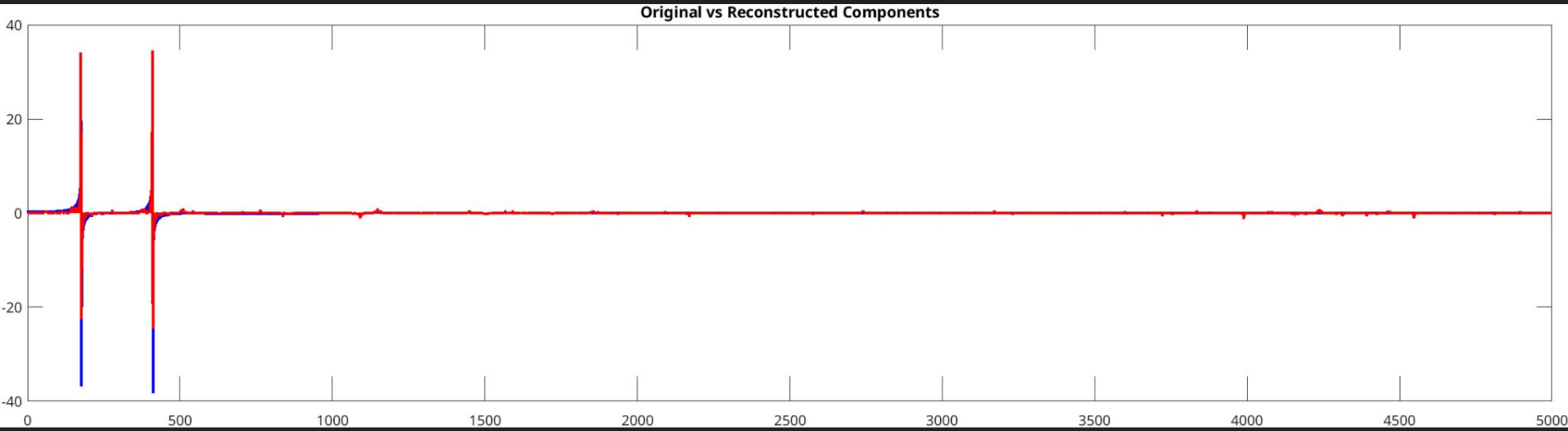
```
cvx_begin
    variable x( n );
    minimize( norm( x, 1 ) );
    subject to
        A*x == fr.';
cvx_end
```

6. Perform DCT on optimized x to retrieve reconstructed signal

```
sig1 = dct( x );
```

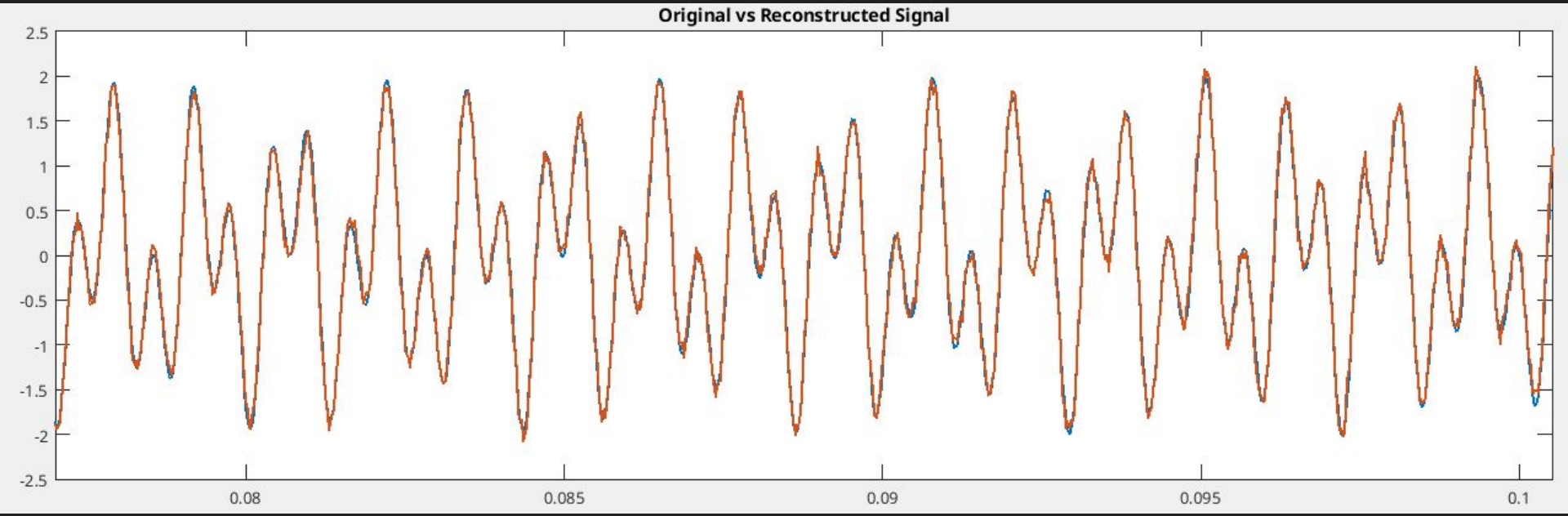
Audio Signal Example - Results

- Verify reconstructed DCT components



Audio Signal Example - Results

- Verify reconstructed signal



Application to 2D Signals (Images)

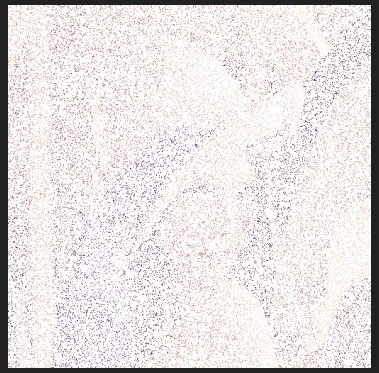
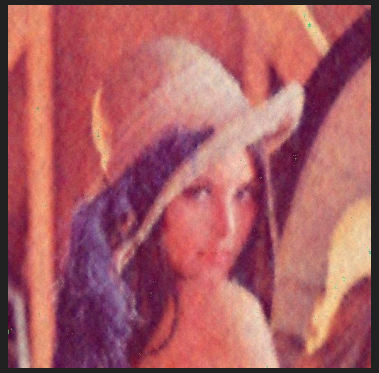
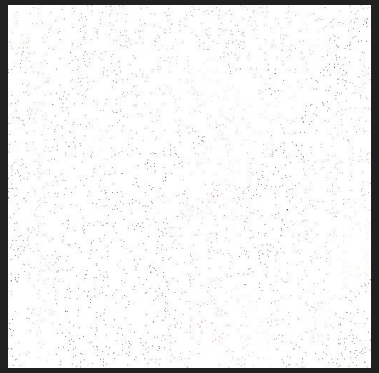
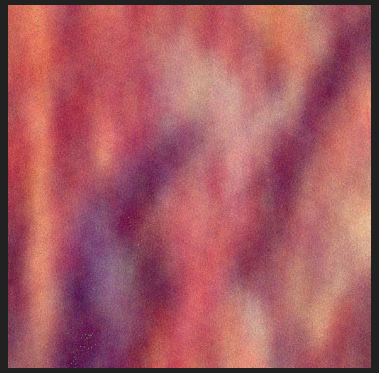
- CS in images requires flattening the images
  - Example: A 512x512 image becomes a 262,144x1 vector
  - Each color (RGB) is calculated separately (three vectors)
- Problem?
  - Requires *massive* memory!
  - Example: N = 512x512 = 262,144, M = 0.5N = 131,072
  - ***A*** is 262,144x131,072, **x** is 262,144x1, ***y*** is 131,072x1
    - That's 34.4 *billion* values
    - Assuming uint8, 34GB of RAM required (not including the requirements for the optimization problem!). What if we had color? 3 times that!
  - We can only do very small portions of an image at a time because of this.
- Solution?
  - L-BFGS and OWL-QN!

L-BFGS and OWL-QN

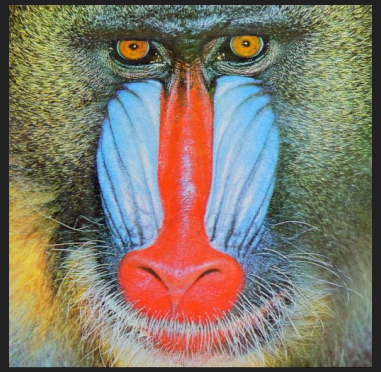
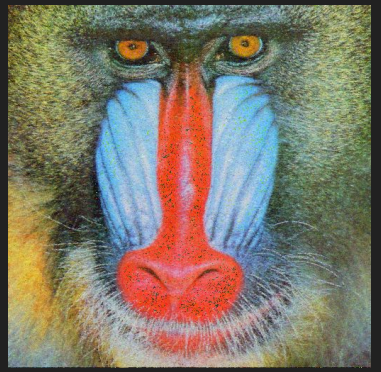
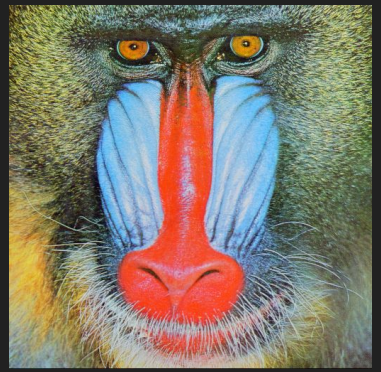
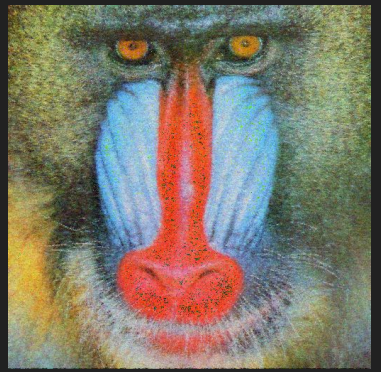
- Limited memory BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm
  - Approximates BFGS with a focus on memory optimization
  - Minimizes function *f(x)*
- Orthant-Wise Limited-Memory Quasi-Newton (OWL-QN)
  - L-BFGS variant specific for fitting L_1 models using their sparsity properties
  - Developed by Microsoft Research
  - C library is available. Using numpy wrappers, can be included in Python
  - Minimizes the function $f(\boldsymbol{x}) = g(\boldsymbol{x}) + C\|\boldsymbol{x}\|_1$ where *g* is a differentiable convex loss function
  - Let's use the least squares objective function as our *g*:
    - $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$
    - $\nabla g = 2(\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^T\boldsymbol{y})$

	Original	Mask	Reconstructed
50%			
30%			

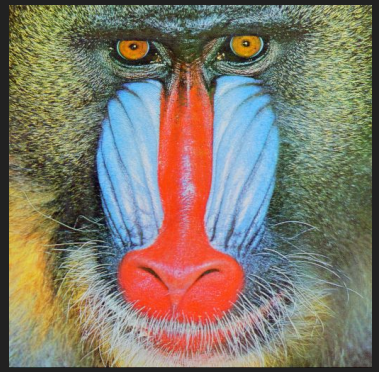
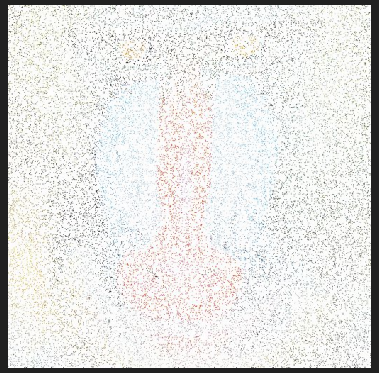
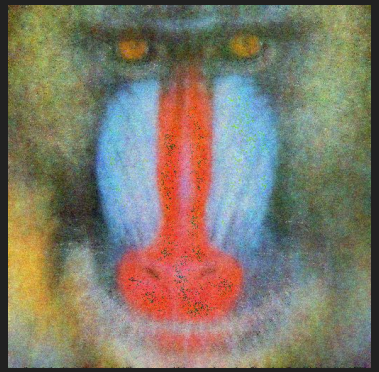
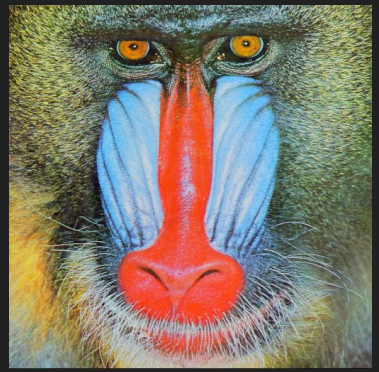
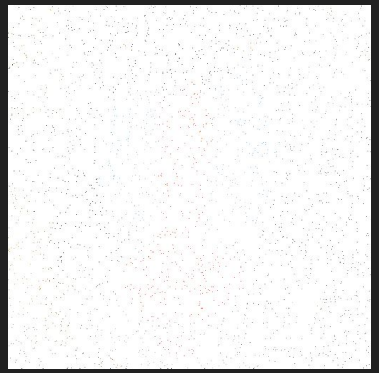
Comparison - Lena (512x512)

	Original	Mask	Reconstructed
10%			
1%			

Comparison - Lena (512x512)

	Original	Mask	Reconstructed
50%			
30%			

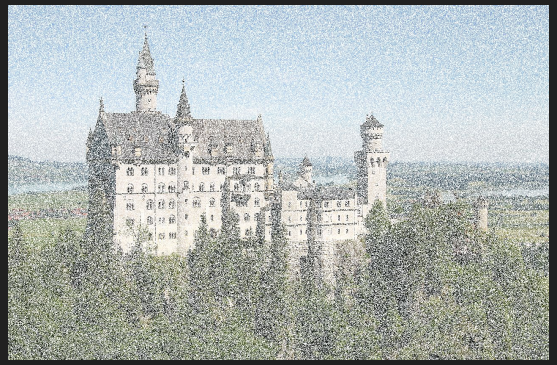
Comparison - Mandrill (512x512)

	Original	Mask	Reconstructed
10%			
1%			

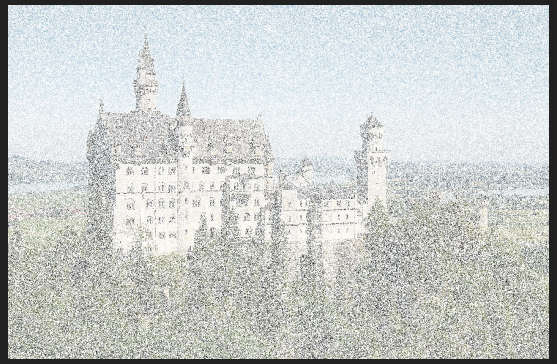
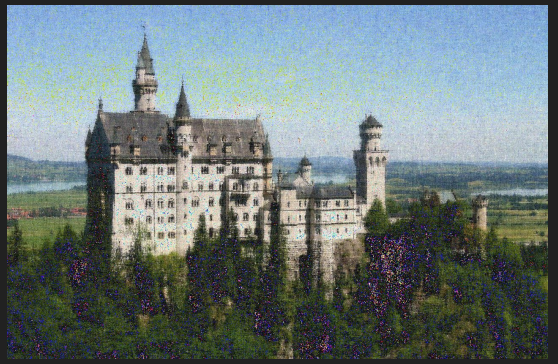
Comparison - Mandrill (512x512)

Original Mask Reconstructed

50%


30%

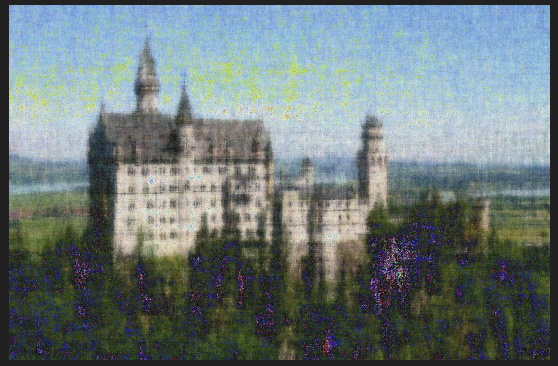




Comparison - Neuschwanstein (1,280x838)

Original Mask Reconstructed

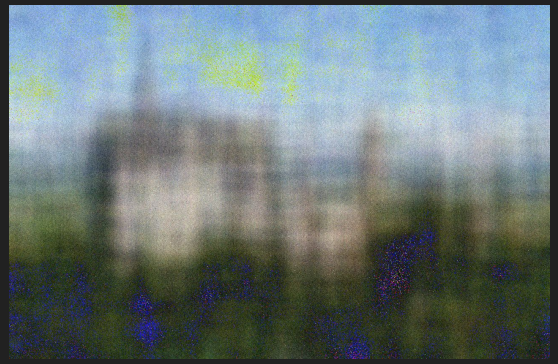
10%





1%



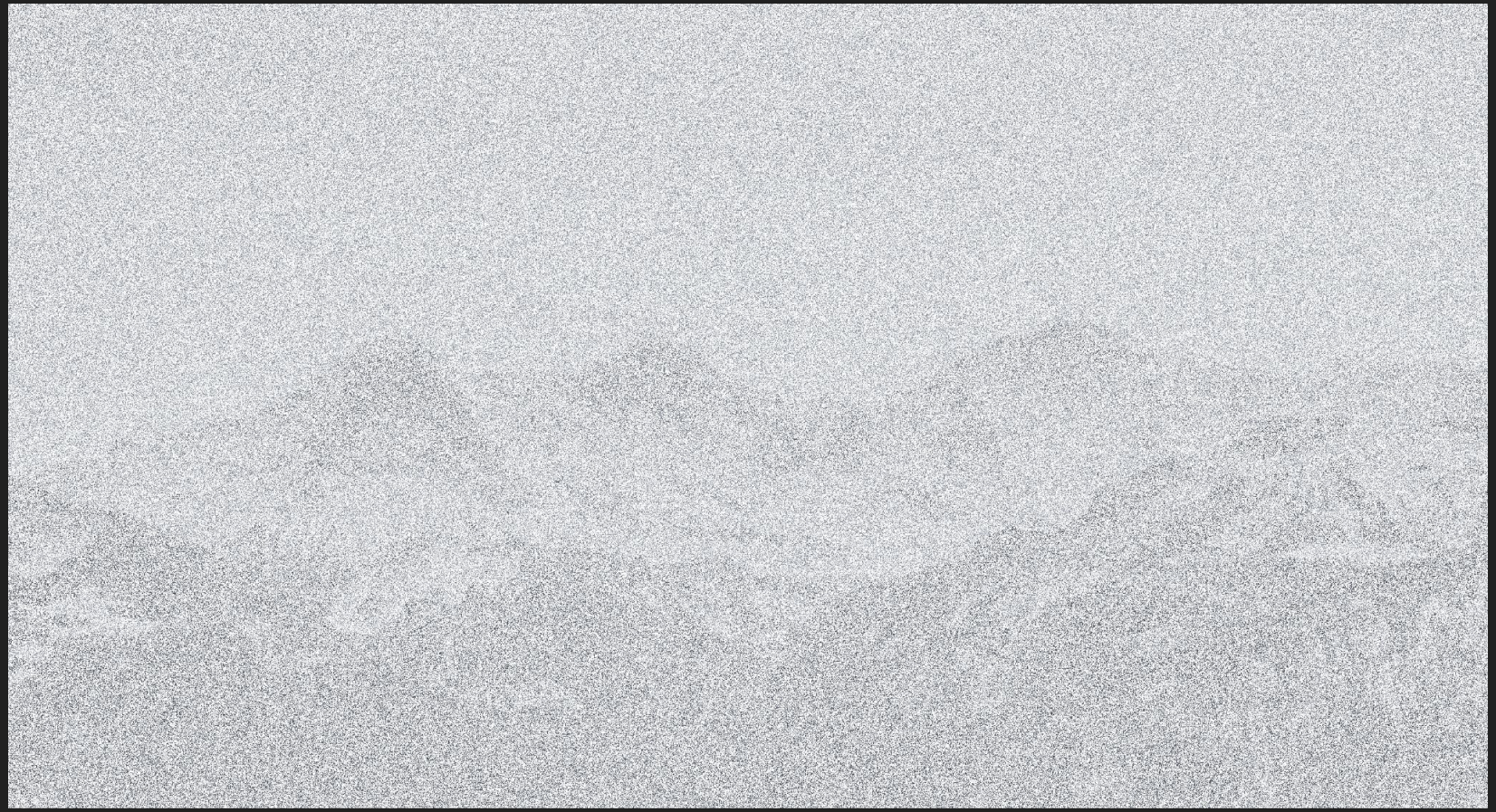


Comparison - Neuschwanstein (1,280x838)

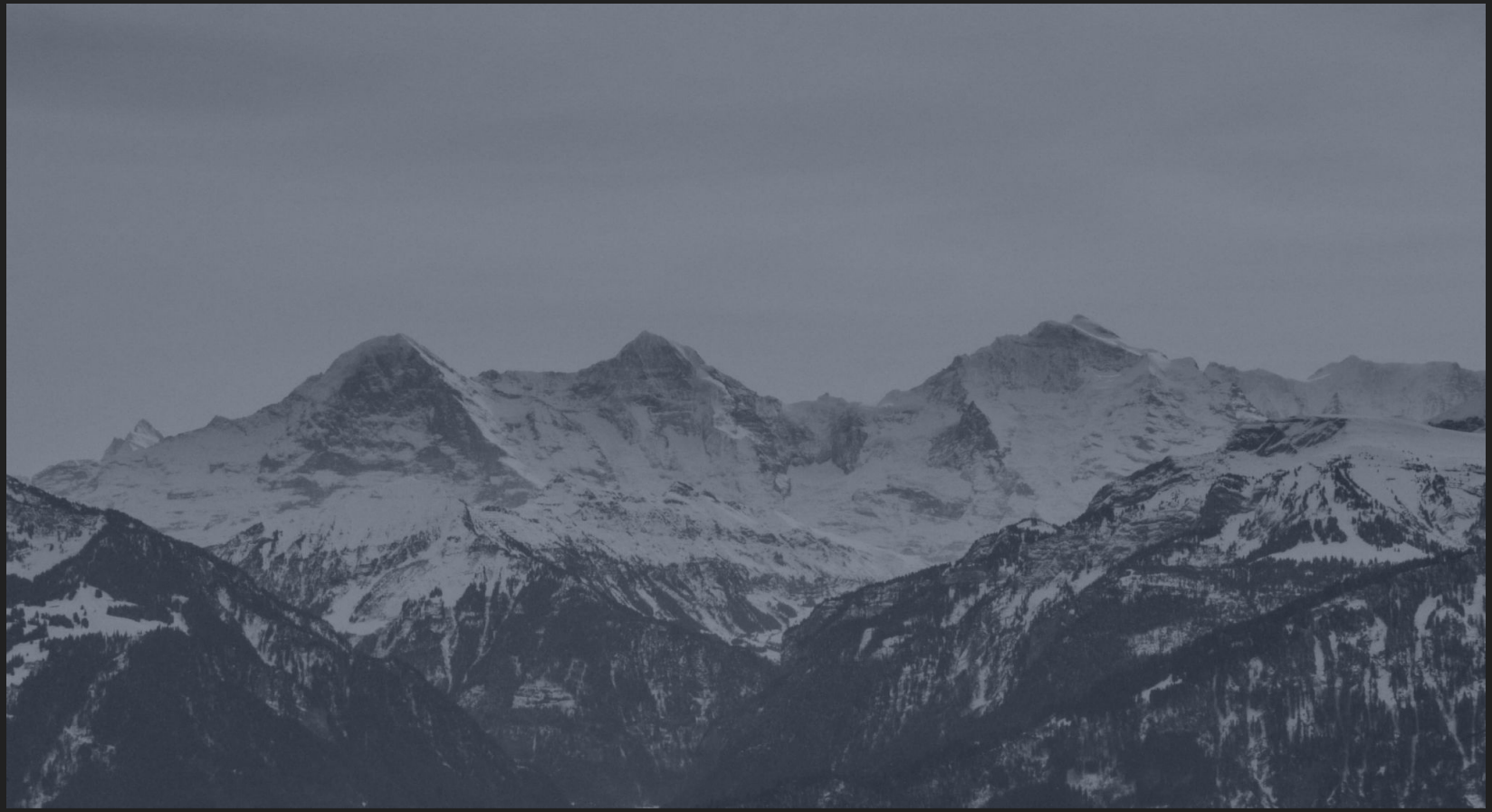
Results

Sample Percentage	SSIM			MSE			PSNR		
	Lena	Mandrill	Neuschwanstein	Lena	Mandrill	Neuschwanstein	Lena	Mandrill	Neuschwanstein
100%	0.978	0.993	0.933	5.226	6.991	157.099	40.949	36.685	26.169
90%	0.959	0.932	0.770	10.243	74.015	376.106	38.026	29.438	22.378
80%	0.937	0.870	0.648	15.987	154.363	617.950	36.093	26.245	20.221
70%	0.911	0.800	0.554	24.849	255.127	871.792	34.178	24.063	18.727
60%	0.879	0.725	0.476	34.154	368.031	1147.691	32.796	22.472	17.533
50%	0.840	0.643	0.407	50.981	497.685	1426.921	31.057	21.161	16.587
40%	0.792	0.555	0.344	70.938	642.209	1728.594	29.622	20.054	15.754
30%	0.726	0.457	0.281	102.855	790.105	2021.594	28.009	19.154	15.074
20%	0.640	0.349	0.217	157.736	966.865	2319.439	26.151	18.277	14.477
10%	0.506	0.227	0.149	275.905	1163.247	2599.152	23.723	17.474	13.982
1%	0.251	0.105	0.093	1067.362	1685.916	2760.950	17.848	15.862	13.720

Original image (5,472x2,976)



30% sampled image

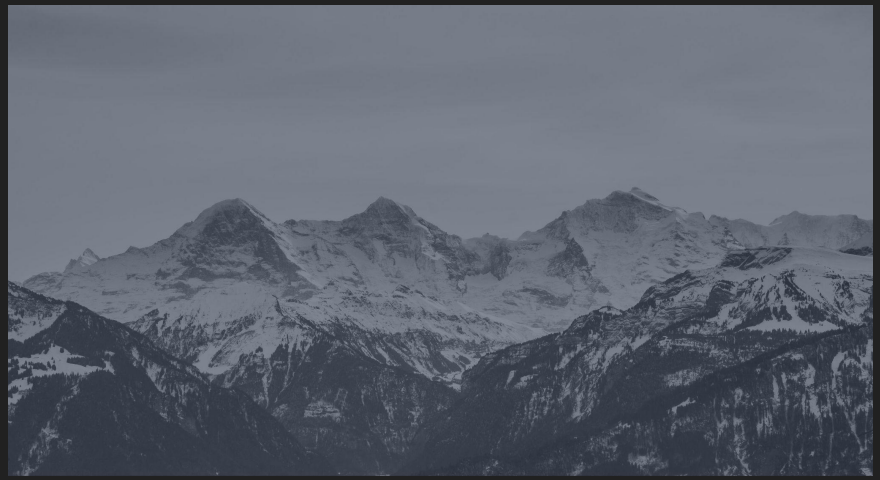


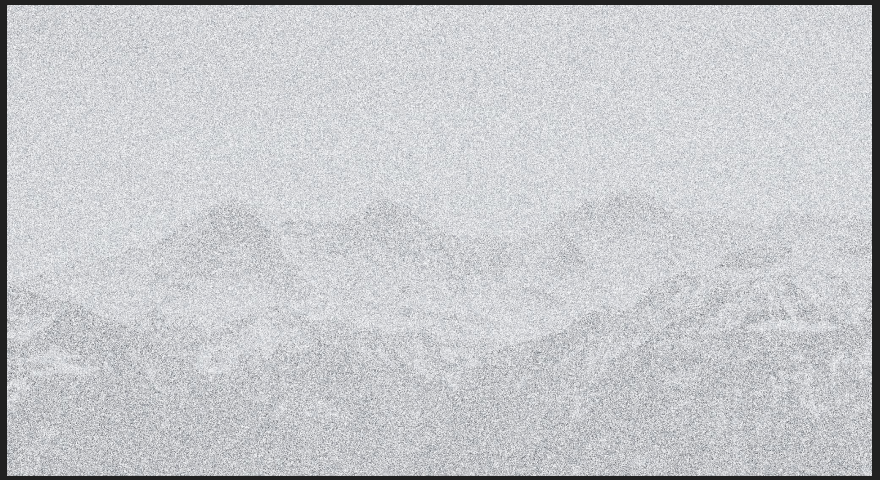
Recovered image

Original

Mask

30%



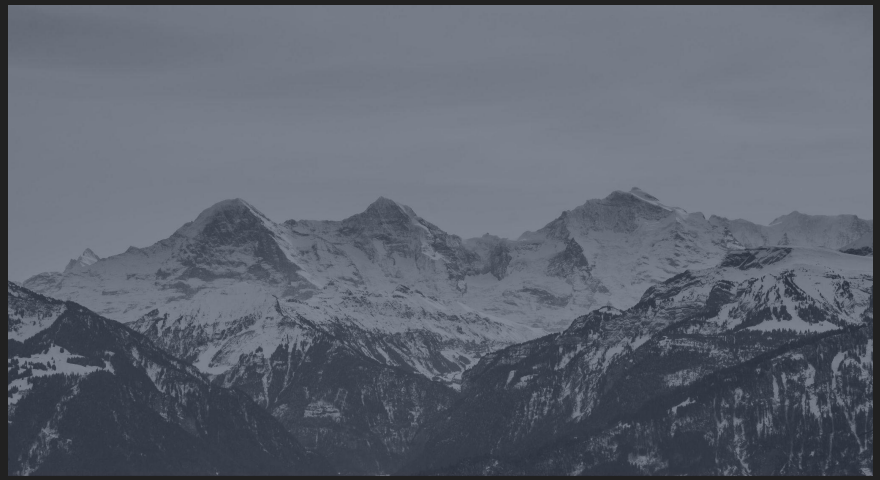


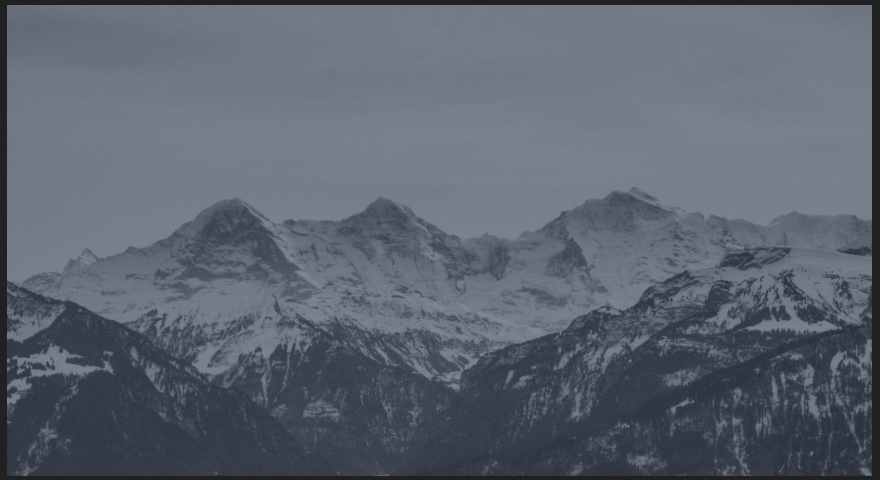
Comparison -- Mountains (5,472x2,976)

Original

Reconstructed

30%



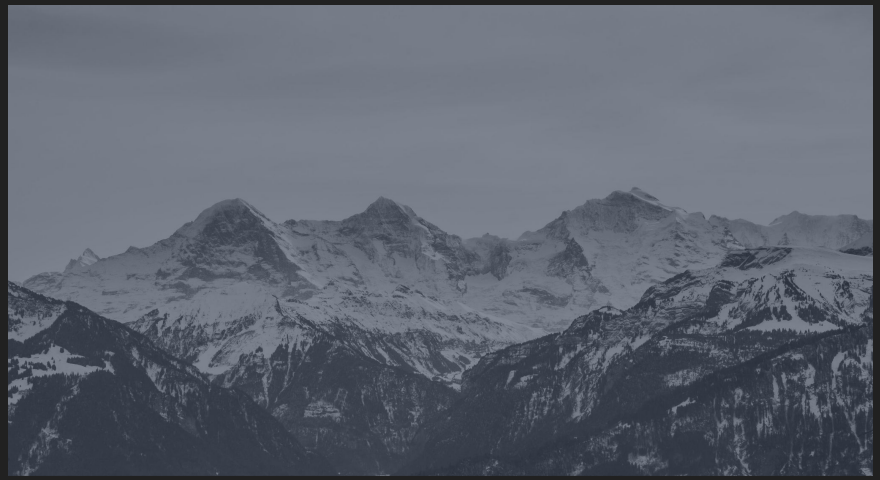


Comparison -- Mountains (5,472x2,976)

Original

Mask

10%

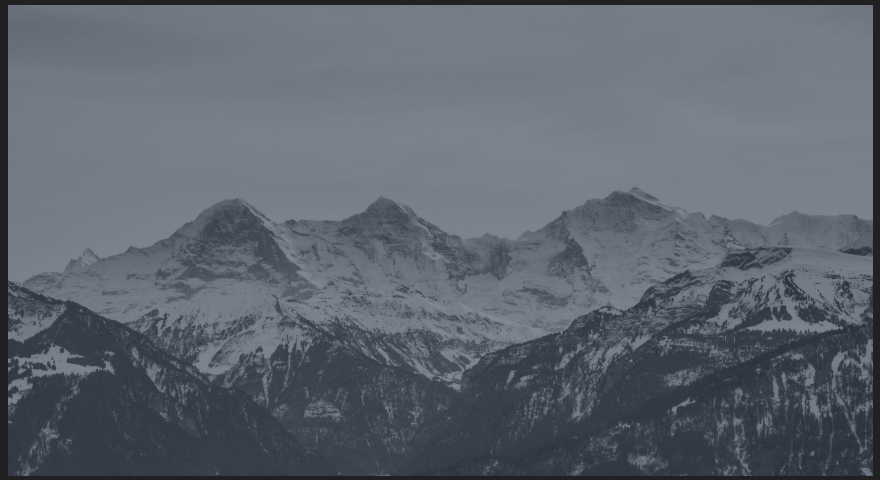


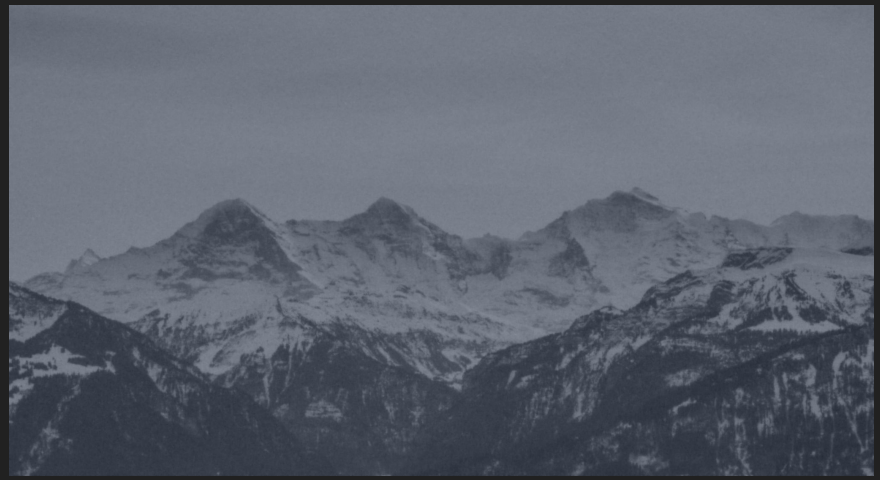
Comparison -- Mountains (5,472x2,976)

Original

Reconstructed

10%



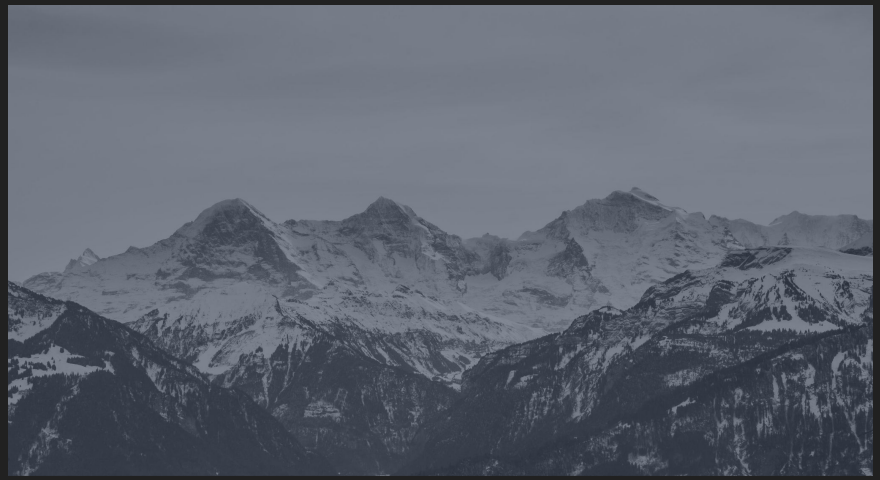


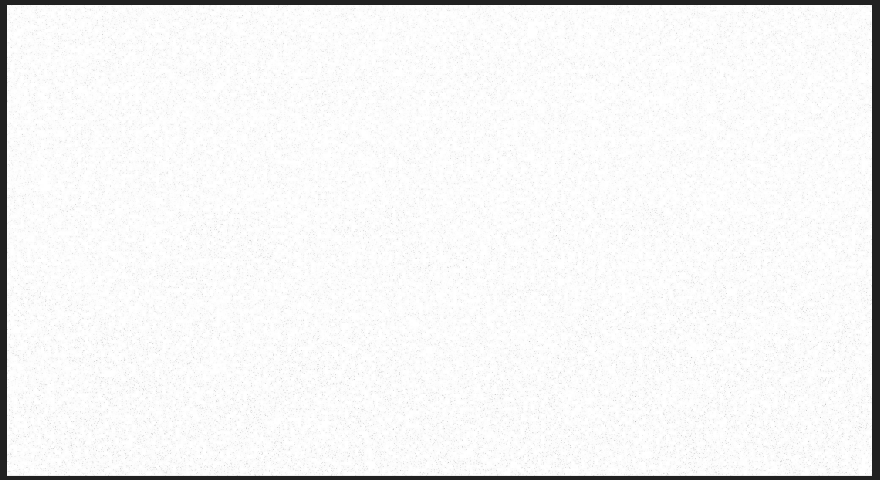
Comparison -- Mountains (5,472x2,976)

Original

Mask

1%



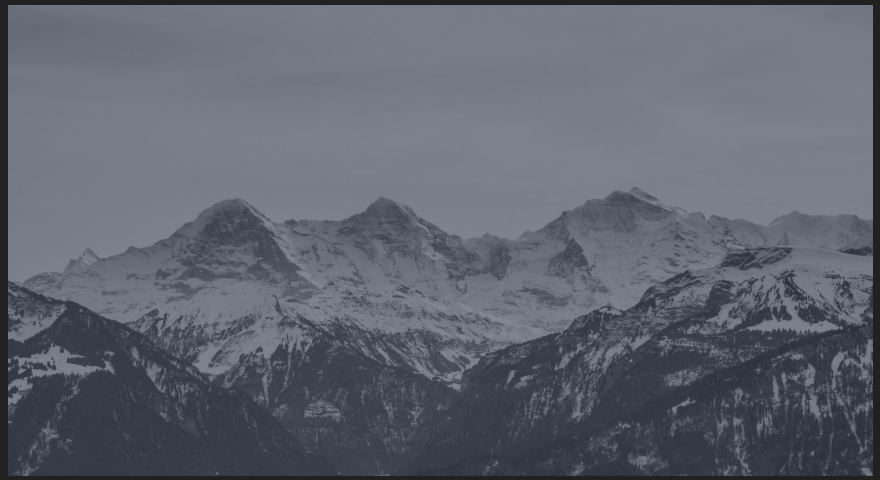


Comparison -- Mountains (5,472x2,976)

Original

Reconstructed

1%



Comparison -- Mountains (5,472x2,976)

Applications

- Data compression
- Single pixel camera
- Medical imaging
- Wireless Sensor Networks
- Channel coding

Conclusion

- Compressive Sensing is a great tool for compression of data
  - Assumption is that data can be represented sparsely in some domain
- Algorithms need some optimization for efficient computation
  - Memory management
  - Long processing times
- Can greatly reduce the amount of time it takes to capture some data
  - Example: MRI

References

[1] D. L. Donoho, "Compressed sensing," In *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289-1306, April 2006. doi: 10.1109/TIT.2006.871582.

[2] A. Hladnik, P. Saksida, "Compressed sensing and some image processing applications," In *Int. Symp. on Graphic Engineering and Design*, 2018, pp. 567-572. doi: 10.24867/GRID-2018-p68.

[3] Nathan Kutz. *Compressive Sensing*. (Feb. 6, 2021). Accessed: Apr. 29, 2021. [Online Video]. Available: https://www.youtube.com/watch?v=rt5mMEmZHfs

[4] P. C. Nahar, M. T. Kolte, "An introduction to compressive sensing and its applications," *Int. Jour. of Scientific and Research Publications*, vol. 4, no. 6. Jun. 2014. [Online]. Available: http://www.ijsrp.org/research-paper-0614/ijsrp-p3076.pdf

[5] "Compressed Sensing in Python." Pyrunner. http://www.pyrunner.com/weblog/2016/05/26/compressed-sensing-python/ (retrieved Apr. 29, 2021).

[6] G. Andrew, J. Gao, "Scalable training of L1-regularized log-linear models," *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 33-40, doi: 10.1145/1273496.1273501.

[7] P. Boufonos, *Sparsity, randomness and compressed sensing* [PDF]. Available: http://mlsp.cs.cmu.edu/courses/fall2009/class24/CS.slides.pdf. (accessed Apr. 29, 2021).