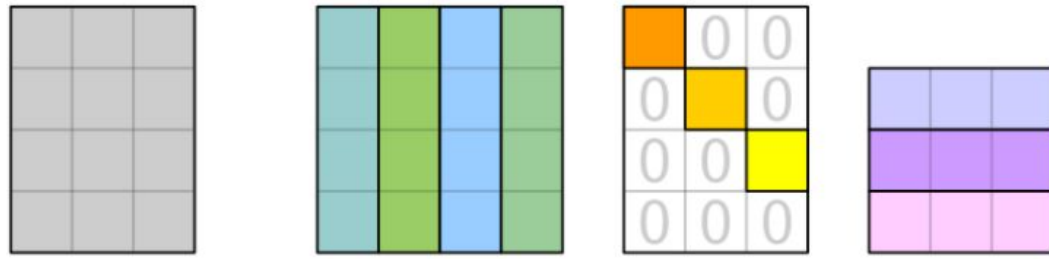


# Image Compression

Matthew Futch and Soham Purushan

# SVD

Any matrix  $A$  can be factored into  $U$ ,  $\Sigma$ ,  $V^t$ , where  $U$  contains the left singular vectors of  $A$ ,  $\Sigma$  contains the singular values in descending order, and  $V^t$  contains the right singular vectors.

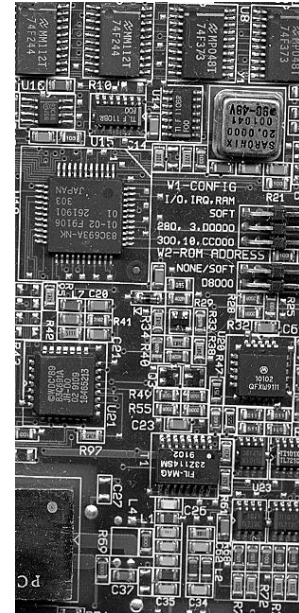
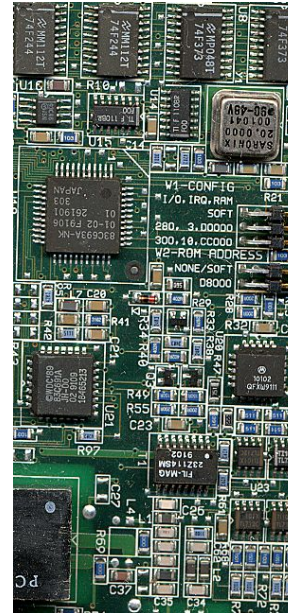
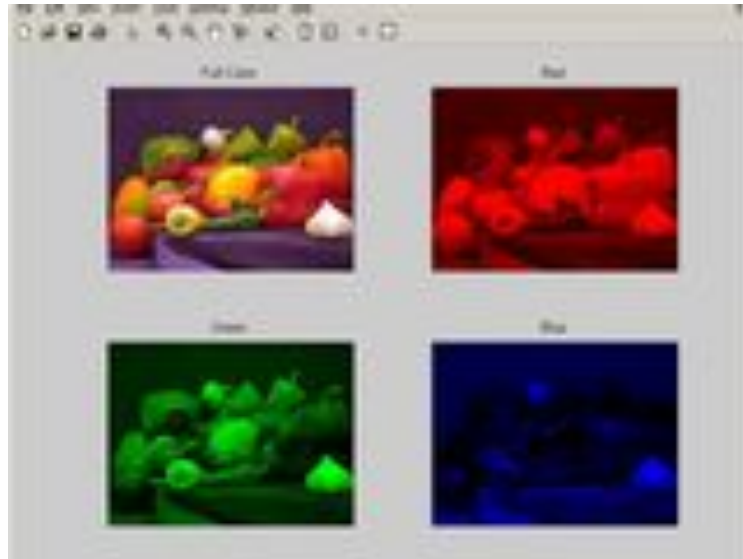


The diagram illustrates the SVD factorization of matrix  $M$  into matrices  $U$ ,  $\Sigma$ , and  $V^*$ . Matrix  $M$  is a 4x3 grid of gray squares. Matrix  $U$  is a 4x4 grid with four columns of different colors: teal, green, blue, and light green. Matrix  $\Sigma$  is a 4x3 grid where the top-left 3x3 submatrix contains colored squares (orange, yellow, and light yellow) and the rest of the grid contains zeros. Matrix  $V^*$  is a 3x3 grid with three rows of different colors: light blue, purple, and pink.

$$\begin{matrix} \mathbf{M} \\ m \times n \end{matrix} = \begin{matrix} \mathbf{U} \\ m \times m \end{matrix} \begin{matrix} \mathbf{\Sigma} \\ m \times n \end{matrix} \begin{matrix} \mathbf{V}^* \\ n \times n \end{matrix}$$

# What this means for images

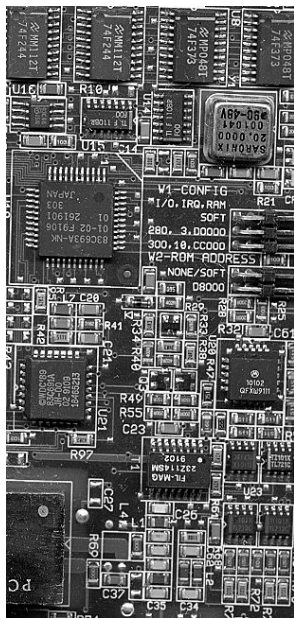
A m by n colored image is stored as m by n by 3 matrix such that each pixel corresponds to (Red,Green,Blue) where  $0 \leq R \leq 255$ ,  $0 \leq G \leq 255$ ,  $0 \leq B \leq 255$



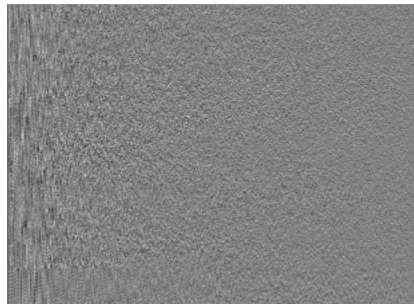
# Methods

- SVD for images and upper bound of singular values
- Dimensionality reduction and randomized SVD
- Discrete Wavelet Transform
- Discrete Cosine Transform
- Optimization

# SVD for images



=



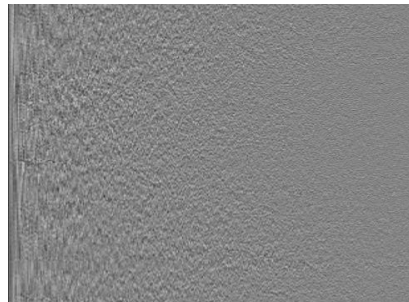
$U$

\*



$\Sigma$

\*



$V^t$

# Memory

An  $m$  by  $n$  grayscale image will need to be stored in a matrix of size  $m$  by  $n$ .

We want SVD with  $k$  singular values such that  $U$ ,  $\Sigma$ , and  $V^t$  have fewer than  $m \cdot n$  elements.

$U$  is size  $m$  by  $k$ ,  $\Sigma$  is size  $k$  since a diagonal matrix can be stored as an array,  $V^t$  is size  $k$  by  $n$ . Summing the sizes of the matrices,

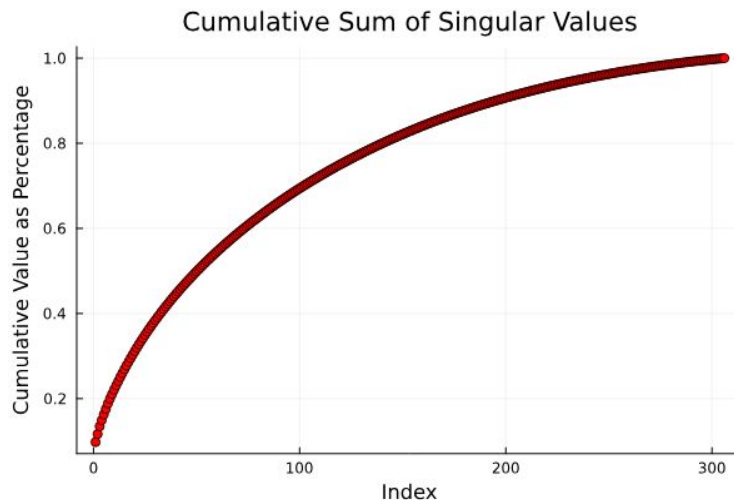
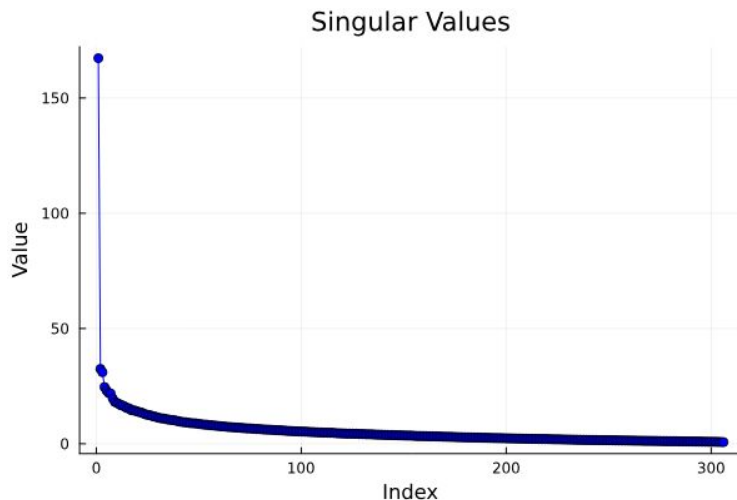
$$mk + k + nk = k(m + n + 1)$$

$$k(m + n + 1) < mn \implies k < \frac{mn}{m+n+1} \qquad k < \frac{n^2}{2n+1}$$

The circuit image is  $648 \times 306$  so  $m = 648$  and  $n = 306 \rightarrow k < \lfloor 207.6 \rfloor = 207$

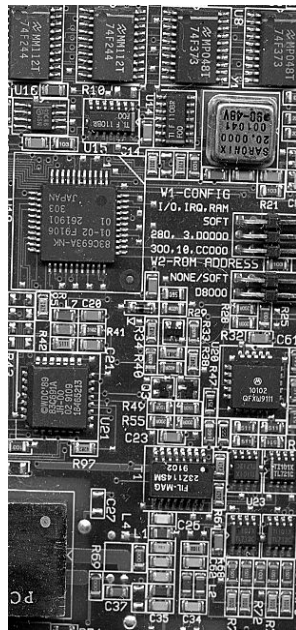
# Singular Value Trends

- Sharp decline after first few values, suggesting a low effective rank.
- Steep initial curve - few singular values dominate the total sum
- Small subset of singular values captures most of the matrix information
  - Should be good for compression



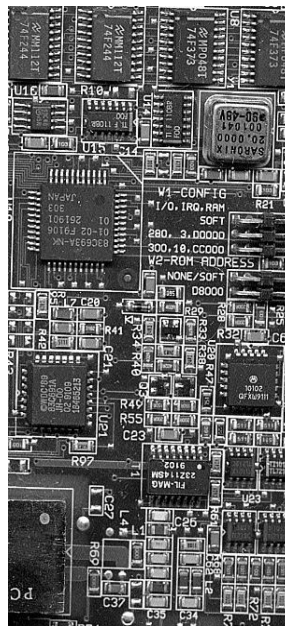
# Effects of Compression

Uncompressed



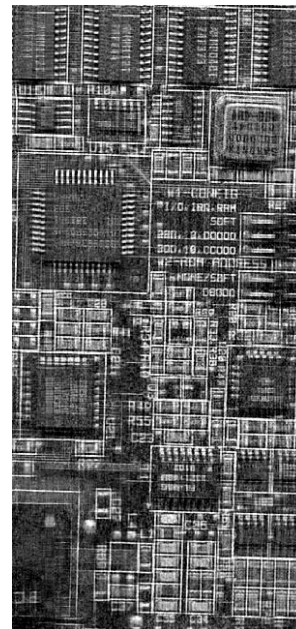
$$648 \times 306 = 198288$$

$k = 200$



$$200(648 + 306 + 1) = 191000$$

$k = 50$



$$50(648 + 306 + 1) = 47750$$



# Randomized SVD

## Algorithm:

1. Draw an  $n \times k$  Gaussian random matrix  $\Omega$ .
2. Form the  $m \times k$  sample matrix  $Y = A \Omega$ .
3. Form an  $m \times k$  orthonormal matrix  $Q$  such that  $Y = QR$ .
4. Form the  $k \times n$  matrix  $B = Q^* A$ .
5. Compute the SVD of the small matrix  $B$ :  $B = \hat{U} \Sigma V^*$ .
6. Form the matrix  $U = Q \hat{U}$ .

**Step 1:** Compute an approximate basis for the range of  $A$ . We want a matrix  $Q$  with  $\ell$  orthonormal columns ( $k \leq \ell \leq n$ ) that captures the action of  $A$ . Formally,  $A \approx QQ^* A$ .

**Step 2:** Given such a matrix  $Q$ —which is much smaller than  $A$ —use it to compute our desired matrix factorization.

```
• function rsvd(X, k)
•     m, n = size(X)
•     Omega = rand(n, k)
•     Y = X * Omega
•     Q, R = qr(Y)
•     Qm = Matrix(Q)
•     B = Qm' * X
•     Uhat, S, Vt = svd(B)
•     Uk = Uhat[:, 1:k]
•     Vk = Vt'[1:k, :]
•     U = Qm * Uk
•     return U, Diagonal(S), Vk
• end
```

# Why randomized SVD works

## Johnson-Lindenstrauss Lemma

- A set of points in a high-dimensional space can be embedded into a lower dimension such that distances between the points are nearly preserved up to a small factor
- This allows us to take a random projection of  $A$  in a lower dimension and still achieve a solution relatively close to how it was in the higher dimension

**Lemma** For any  $0 < \epsilon < 1$  and any integer  $n$  let  $k$  be a positive integer such that

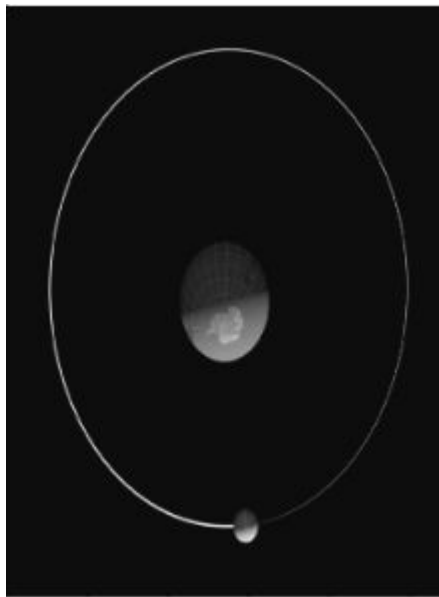
$$k \geq \frac{24}{3\epsilon^2 - 2\epsilon^3} \log n \quad (2)$$

then for any set  $A$  of  $n$  points  $\in \mathbb{R}^d$  there exists a map  $f : \mathbb{R}^d \rightarrow \mathbb{R}^k$  such that for all  $x_i, x_j \in A$

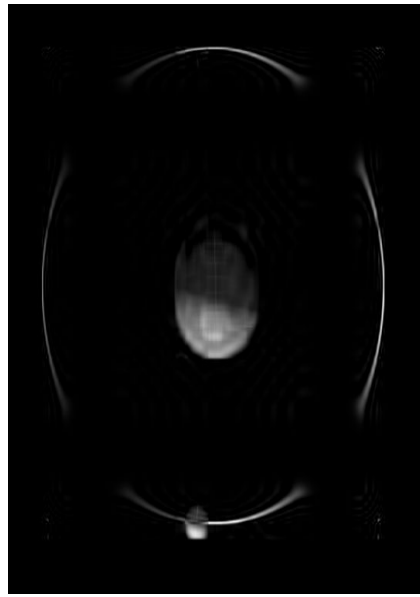
$$(1 - \epsilon) \|x_i - x_j\|^2 \leq \|f(x_i) - f(x_j)\|^2 \leq (1 + \epsilon) \|x_i - x_j\|^2 \quad (3)$$

# Effects of Randomized SVD

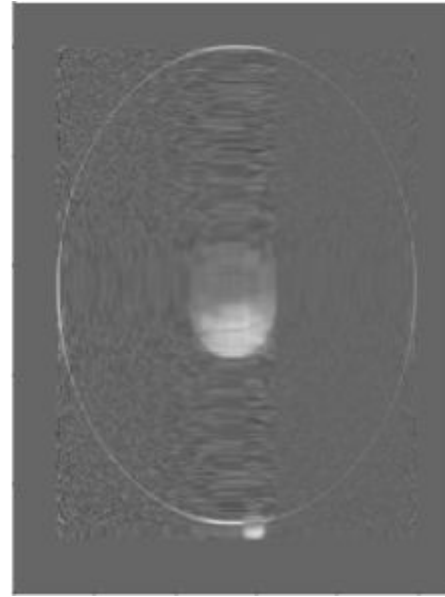
Uncompressed



SVD  $k=20$



RSVD  $k=20$



# Batch Processing with SVD

- Batch processing allows us to handle large datasets efficiently by processing multiple images simultaneously, reducing time and memory overhead compared to processing images one-by-one.

```
function batch_preprocess_and_svd(image_paths, batch_size)
    num_images = length(image_paths) # 'length' should be lowercase
    for i in 1:batch_size:num_images
        # Get the current batch of image paths
        batch_end = min(i+batch_size-1, num_images)
        batch_paths = image_paths[i:batch_end]

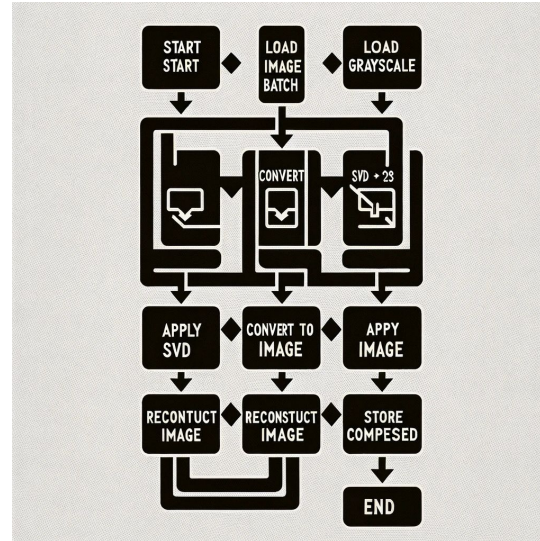
        # Preprocess the images
        batch = map(preprocess_image, batch_paths)

        # Apply SVD
        U_matrices, S_vectors, Vt_matrices = apply_svd_on_batch(batch)

        # Here you can save the SVD results or further process them

        println("Processed batch $(i):$(batch_end)")

    return U_matrices, S_vectors, Vt_matrices
end
```



# Optimization of SVD

- Decided to create an optimization Algorithm that finds the best “k” values or number of singular values to find the best balance between image quality and compression ratio
- To being this we first need to convert the RGB image to Grayscale.:

```
function custom_rgb2gray(image::Array{<:ColorTypes.RGB,2})  
    # Apply the luminosity method  
    return 0.299 .* red.(image) .+ 0.587 .* green.(image) .+ 0.114 .* blue.(image)  
end
```

# Optimization of SVD

- In the algorithm we used the concept of PSNR as a measure of quality for compressed images.
- Peak Signal-to-Noise Ratio (PSNR) is a widely used metric to measure the quality of reconstructed images (or videos) compared to the original ones. It is expressed in decibels (dB)
- PSNR is used to quantify the level of degradation (noise) introduced by compression or by other alterations such as blurring or noise addition

$$\begin{aligned} PSNR &= 10 \cdot \log_{10} \left( \frac{MAX_I^2}{MSE} \right) \\ &= 20 \cdot \log_{10} \left( \frac{MAX_I}{\sqrt{MSE}} \right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE). \end{aligned}$$

# Optimization of SVD

- MAX\_I: The maximum possible pixel value of the image. For images normalized between 0 and 1, MAX\_I = 1
- MSE (Mean Squared Error): The average of the squares of the differences between the original and the compressed image pixels.
- A higher PSNR indicates better quality. In typical settings, a PSNR value of 30 dB or higher is considered good

$$\text{MSE} = \frac{1}{mn} \sum (I_{\text{original}} - I_{\text{compressed}})^2$$

## Cont.

- Use a truncated SVD representation to recompose the image after an optimization
- Only the first  $k$  singular values are used in the reconstruction of the matrix  $A_k$ . Quality of approximation is based on the value of  $k$ .

$$A_k = U_k \cdot S_k \cdot V_k^T$$



- Function aims to find the smallest  $k$  that retains a quality of the reconstructed image above a predefined threshold.
- Inputs: original image matrix, the maximum number of steps for reducing ' $k$ ', and the quality threshold relative to the original image quality
- Uses an iterative process where ' $k$ ' starts at its maximum possible value (equal to the number of singular values in the original SVD) and is reduced stepwise
- PSNR is used after each iteration to measure the quality of the reconstructed image compared to the original

```
function optimize_k(original_image, max_k_reduction_steps, quality_threshold)
    U, S, Vt = svd(original_image)
    best_k = length(S)
    current_k = best_k
    step_size = max(1, best_k ÷ max_k_reduction_steps)
    original_quality = calculate_psnr(original_image, original_image)

    println("Starting optimization with initial k: $best_k, step size: $step_size")

    for i in 1:max_k_reduction_steps
        reconstructed_image = reconstruct_image(U, S, Vt, current_k)
        current_quality = calculate_psnr(original_image, reconstructed_image)

        println("Iteration: $i, current k: $current_k, PSNR: $current_quality")

        if current_quality < quality_threshold * original_quality
            println("Quality below threshold at k: $current_k, stopping optimization.")
            break
        end

        best_k = current_k
        current_k -= step_size
    end

    best_reconstructed_image = reconstruct_image(U, S, Vt, best_k)
    return best_k, best_reconstructed_image
end
```

# Issues with Optimization with SVD

- Found that with each iteration, as  $k$  decreases, the PSNR fluctuates but generally stays constant. This is unusual because we would expect the PSNR to decrease as  $k$  gets smaller, indicating a loss of quality.
- The quality of image reconstruction using SVD may not always decline linearly with  $k$ . At certain points, the loss of certain singular values may cause a significant drop in perceived quality, not fully captured by PSNR
- PSNR is influenced by the maximum pixel value, it can be disproportionately affected by outliers in the image data

- There are limitations especially for images with high levels of detail or noise.
- Some images may compress well with a small number of singular values, while others, especially those with intricate textures or noise, might require more singular values to maintain quality

```
no changes to C:\Users\sonali\Documents\Julia\ENVIRONMENTS\julia-1.10.4\bin\julia.exe
Iteration: 1, current k: 2333, PSNR: 5.997827207445445
Iteration: 2, current k: 2100, PSNR: 5.998082596087283
Iteration: 3, current k: 1867, PSNR: 5.998675222369483
Iteration: 4, current k: 1634, PSNR: 5.999723459631394
Iteration: 5, current k: 1401, PSNR: 6.001400139375934
Iteration: 6, current k: 1168, PSNR: 6.003903188932021
Iteration: 7, current k: 935, PSNR: 6.007595856966313
Iteration: 8, current k: 702, PSNR: 6.013003421016228
Iteration: 9, current k: 469, PSNR: 6.021580932564809
Iteration: 10, current k: 236, PSNR: 6.039880831167485
The best k value found is: 236
```

- Alternative quality metrics, such as Structural Similarity Index (SSIM) or Visual Information Fidelity (VIF), may provide a more comprehensive assessment of image quality for SVD-based compression

# Wavelet Transform

Discrete Wavelet Transform (DWT) with the Daubechies wavelet ('db4'). The compression process involves transforming the image into the wavelet domain, thresholding small coefficients to zero to reduce data, and then reconstructing the image from the modified coefficients.

$$W(f)(a, b) = \sum_{x,y} f(x, y) \psi_{a,b}(x, y)$$

$f(x,y)$  is the image function.

$\psi_{a,b}(x,y)$  represents the wavelet basis function at scale  $a$  and position  $b$ .

# DWT: Discrete Wavelet Transformation

- Perform a multi-level wavelet decomposition on the image using `pywt.wavedec2`. This function breaks down the image into a set of wavelet coefficients at different scales and orientations.

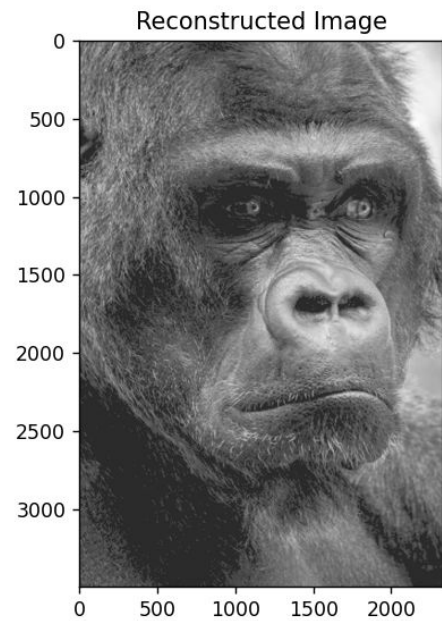
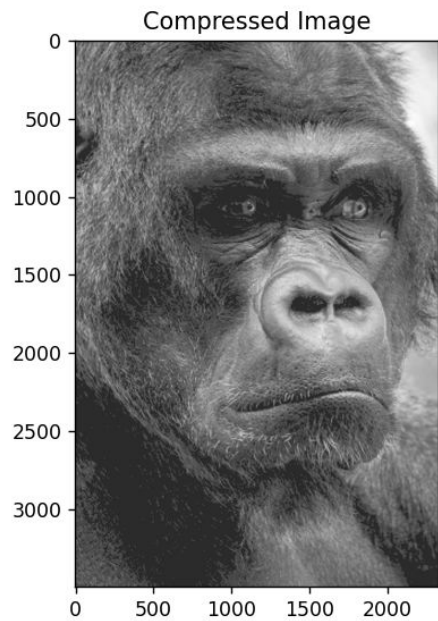
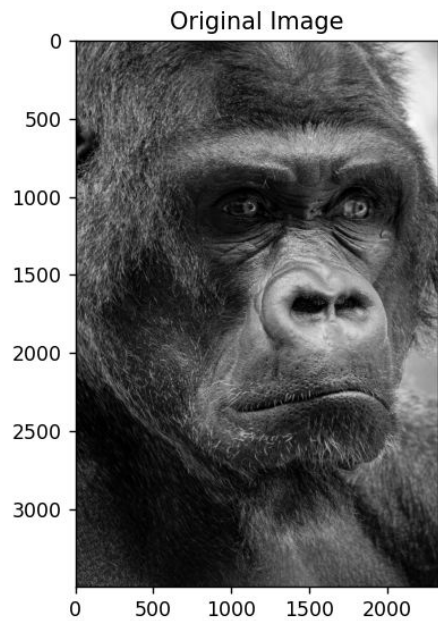
$$c_{A,D} = \sum_k h_k \cdot x[2i - k] + g_k \cdot x[2i - k]$$

$c_A$  are the approximation coefficients.  $c_D$  are detail coefficients.  $h$  is the low pass filter,  $g$  is the high-pass filter, and  $x$  is the input signal.

To achieve compression, small coefficients (those below a defined threshold) are set to zero, effectively reducing the amount of data

$$\text{threshold} = 0.1 \times \max(|c|)$$

C represents the wavelet coefficients.





# Optimization with DWT

- Created a process where DWT coefficients of an image are quantized and evaluated at different levels of decomposition to determine the optimal level for image compression while maintaining quality above a specified threshold.
- The wavelet coefficients of an image are obtained through a DWT.
- Quantization is applied to the detail coefficients:  $C_d$ . With thresholds  $T$  which are determined dynamically based on the standard deviation of coefficients.

$$C_{d,q} = \begin{cases} \text{sign}(C_d) \cdot (|C_d| - T), & \text{if } |C_d| > T \\ 0, & \text{otherwise} \end{cases}$$

Thresholds:

$$T = \sigma(C_d) \times 0.5$$

The 0.5 is there to essentially remove noise in the algorithm as the information in the detail coefficients is primarily concentrated in the coefficients with larger magnitudes. Thus, the measure spread can be a good indicator of the significant energy levels in the detail coefficients.

# Finding optimal compression level

- Each level's compression is evaluated using PSNR, quantifying the quality loss relative to the uncompressed image
- The algorithm iteratively tests each level of decomposition, quantizes the detail coefficients, reconstructs the image, and calculates its PSNR.
- It records the highest PSNR that surpasses a predefined quality threshold, ensuring the final image quality is acceptable
- The "best level" of decomposition is the one that maximizes PSNR while staying above the quality threshold, indicating the most efficient compression without significant quality degradation

Level 1, PSNR: 36.64954714069674, Thresholds: [5.205981731414795]

Level 2, PSNR: 34.12847589640344, Thresholds: [9.471710205078125, 5.221951484680176]

Level 3, PSNR: 31.851487706634853, Thresholds: [24.536636352539062, 9.473797798156738, 5.234094619750977]

Level 4, PSNR: 30.16931757905345, Thresholds: [52.230613708496094, 24.536636352539062, 9.473797798156738, 5.234094619750977]

Level 5, PSNR: 29.219109055074874, Thresholds: [88.69855499267578, 52.1959342956543, 24.470348358154297, 9.442546844482422, 5.214404106140137]

Best level: 1 with PSNR: 36.64954714069674

# DCT: Discrete Cosine Transform

- Apply DCT to the image to get frequency coefficients
- Set high-frequency coefficients (often contain less important visual information) to zero to reduce detail and size
- Apply IDCT to the modified coefficients to reconstruct a compressed version of the original image
- DCT:

$$F(u, v) = \frac{2}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cos \left[ \frac{(2x+1)u\pi}{2M} \right] \cos \left[ \frac{(2y+1)v\pi}{2N} \right]$$

$f(x,y)$ : pixel intensity of the original image

$M$  and  $N$ : Dim. of the image

$F(u,v)$ : DCT coefficient at frequencies of  $u$  and  $v$ .

$C(u)$  and  $C(v)$ : scaling factors that depend on the frequency indices and ensure the transformation matrix is orthonormal

IDCT: Formula:

$$f(x, y) = \frac{2}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} C(u)C(v)F(u, v) \cos\left(\frac{(2x+1)u\pi}{2M}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right)$$

Compression Algorithm Steps:

Original Image: Spatial domain with pixel values.

After DCT: Frequency domain with high and low frequencies.

After Thresholding: Preserved low frequencies, high frequencies set to zero.

After IDCT: Reconstructed image with reduced detail

# DCT Optimization

- Used block processing:
- allows independent manipulation of different parts of the image, which can be tailored to local characteristics
- the image is divided into uniform smaller sections, such as 8x8 pixel blocks. Each block is processed independently, allowing for tailored compression techniques that adapt to the content within each block
- mimics the JPEG compression technique, making it relevant and widely applicable

# Adaptive Quantization

- Quantization reduces the precision of the transformed coefficients and is a critical step in compressing the data. Adaptive quantization adjusts the quantization thresholds based on local image characteristics, such as variance within a block
- High Variance Blocks: These contain more details (like edges). A lower threshold is used to preserve more coefficients to retain detail.
- Low Variance Blocks: These are smoother, and a higher threshold can be applied, simplifying the block and increasing compression
- This method aims to optimize the balance between compression rate and image quality by minimizing perceptual loss of detail, particularly in critical areas of the image



PSNR: 39.5526635

Original Image



Compressed Image



- Unlike other methods where a specific PSNR threshold (e.g., 30 dB) might be targeted to ensure a minimum quality standard, this adaptive approach focuses on optimizing quality dynamically across different image areas
- It doesn't set a uniform threshold but adapts based on content, aiming for the highest possible quality at the lowest possible bitrate.
- Absence of a fixed threshold allows the algorithm to maximize compression efficiency while maintaining quality dynamically, making it suitable for diverse types of images with varying levels of detail and noise.

# References

<https://cs.stanford.edu/people/mmahoney/cs369m/Lectures/lecture1.pdf>

<https://gregorygundersen.com/blog/2019/01/17/randomized-svd/>

<https://www.mathworks.com/matlabcentral/fileexchange/18125-rgb-image-decomposition>

[https://en.wikipedia.org/wiki/Singular\\_value\\_decomposition](https://en.wikipedia.org/wiki/Singular_value_decomposition)

<https://math.mit.edu/~gs/learningfromdata/>