Quantization-free Lossy Image Compression Using Integer Matrix Factorization

Pooya Ashtari[1*†]
pooya.ashtari@esat.kuleuven.be

Pourya Behmandpoor[1†]
pourya.behmandpoor@esat.kuleuven.be

Fateme Nateghi Haredasht[2]
fnateghi@stanford.edu

Jonathan H. Chen[2]
jonc101@stanford.edu

Lieven De Lathauwer[1]
lieven.delathauwer@kuleuven.be

Sabine Van Huffel[1]
sabine.vanhuffel@esat.kuleuven.be

[1]Department of Electrical Engineering (ESAT), STADIUS Center, KU Leuven, Belgium
[2]Department of Medicine, Stanford University, Stanford, CA, USA

Abstract

one paragraph

1 Introduction

- Importance of image compression - Common current compression methods, such as JPEG and JPEG 2000, rely on - kind of transforms, such as discrete cosine transform (DCT) and wavelet transform - carefully designed quantization techniques - Image compression based on low-rank approximation, particularly SVD and NMF - Drawback of SVD or NMF-based - can only process matrices with floating point elements, so requires the conversion of the input image to floating point - therefore, requires quantization, but it is sensitive to quantization - therefore, does not not perform well especially at low bite rates compared to JPEG - Therefore, we introduce an algorithm based on integer matrix factorization (IMF), which: - low-rank approximates a matrix as a product of two factor matrices with integer elements within a specified interval - is quantization-free, bypassing the need for quantization layer and extra complications - in fact integrates the quantization and factorization into a single step - has a very low-complexity decoder and low decoding time

- mention the experiments and results on Kodak, clic, ImageNet

Your outline provides a structured overview of your paper on "Quantization-free Lossy Image Compression Using Integer Matrix Factorization." Here are some suggestions to improve clarity and flow:

1. **Introduction** - Emphasize the importance of image compression in digital communication and data storage. - Mention commonly used compression methods, such as JPEG and JPEG 2000, which utilize: - Specific types of transforms, including the discrete cosine transform (DCT) and wavelet transform. - Carefully designed quantization techniques.

2. **Current Approaches and Their Limitations** - Discuss image compression strategies based on low-rank approximation methods like Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF). - Highlight drawbacks of using SVD or NMF in image compression:

*Corresponding author. Emails:pooya.ashtari@esat.kuleuven.be, pooya.ash@gmail.com
†Equal contribution

Preprint. Under review.

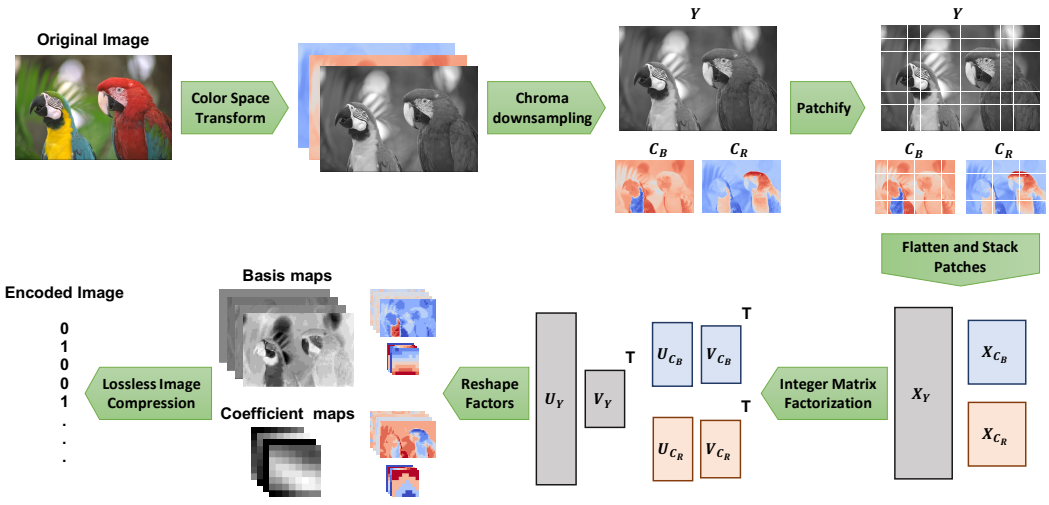


Figure 1 An illustration of the encoder for our image compression method, based on integer matrix factorization.

- Requirement to process matrices with floating point elements, necessitating the conversion of input images to a floating-point format. - Sensitivity to quantization, which leads to suboptimal performance, particularly at low bit rates compared to established methods like JPEG.

3. **Proposed Solution: Integer Matrix Factorization (IMF)** - Introduce an innovative algorithm based on integer matrix factorization (IMF), characterized by: - Low-rank approximation of a matrix as a product of two factor matrices containing integer elements within a specified range. - Elimination of the quantization layer, simplifying the compression process by integrating quantization and factorization into a single step. - Advantages such as a very low-complexity decoder and reduced decoding time, improving efficiency.

This restructured introduction should help convey your key points more clearly and logically, setting a strong foundation for the detailed discussion in your paper.

2 Related Work

Transform-based compression.

Low-rank approximation for compression.

Integer Matrix Factorization.

3 Method

3.1 Overall Encoding Framework

Figure 1 illustrates an overview of the encoding pipeline for our proposed image compression method using integer matrix factorization (IMF). The encoder accepts an RGB image with dimensions $H \times W$ and a color depth of 8 bits, represented by the tensor $\boldsymbol{\mathcal{X}} \in \{0, \ldots, 255\}^{3 \times H \times W}$. Each step of encoding is described in the following.

Color Space Transformation. Analogous to the JPEG standard, the image is initially transformed into the $\mathrm{YC_BC_R}$ color space. Let $\boldsymbol{Y} \in [0, 255]^{H \times W}$ represent the *luma* component, and $\boldsymbol{C}_B, \boldsymbol{C}_R \in [0, 255]^{\frac{H}{2} \times \frac{W}{2}}$ represent the blue-difference and red-difference *chroma* components, respectively. Note that as a result of this transformation, the elements of the *luma* ($\boldsymbol{Y}$) and *chroma* ($\boldsymbol{C}_B$, $\boldsymbol{C}_R$) matrices are not limited to integers and can take any value within the interval $[0, 255]$.

Chroma Downsampling. After conversion to the $\mathrm{YC_BC_R}$ color space, the *chroma* components $\boldsymbol{C}_B$ and $\boldsymbol{C}_R$ are downsampled using average-pooling with a kernel size of $(2, 2)$ and a stride of $(2, 2)$, similar to the process used in JPEG. This downsampling exploits the fact that the human visual system perceives far more detail in brightness information (*luma*) than in color saturation (chroma).

Patchification. After *chroma* downsampling, we have three components: the *luma* component $\boldsymbol{Y} \in [0, 255]^{H \times W}$ and the *chroma* components $\boldsymbol{C}_B, \boldsymbol{C}_R \in [0, 255]^{\frac{H}{2} \times \frac{W}{2}}$. Each of the matrices is split into non-overlapping 8×8 patches. If a dimension of a matrix is not divisible by 8, the matrix is first padded to the nearest size divisible by 8 using reflection of the boundary values. These patches are then flattened into row vectors and stacked vertically to form matrices $\boldsymbol{X}_Y \in [0, 255]^{\frac{HW}{64} \times 64}$, $\boldsymbol{X}_{C_B} \in [0, 255]^{\frac{HW}{256} \times 64}$, and $\boldsymbol{X}_{C_R} \in [0, 255]^{\frac{HW}{256} \times 64}$. Later, these matrices will be low-rank approximated using integer matrix factorization (IMF). Note that this patchification technique differs from the block splitting in JPEG, where each block is subject to discrete cosine transform (DCT) and processed independently. This patchification technique not only captures the locality and spatial dependencies of neighboring pixels but also performs better with the matrix decomposition approach to image compression.

Low-rank approximation. We now apply a low-rank approximation to the matrices $\boldsymbol{X}_Y$, $\boldsymbol{X}_{C_B}$, and $\boldsymbol{X}_{C_R}$, which is the core of our compression method that provides a lossy compressed representation of these matrices. The low-rank approximation [4] aims to approximate a given matrix $\boldsymbol{X} \in \mathbb{R}^{M \times N}$ by

$$\boldsymbol{X} \approx \boldsymbol{U}\boldsymbol{V}^\mathsf{T} = \sum_{r=1}^{R} U_{:r} {V_{:r}}^\mathsf{T}, \tag{1}$$

where $\boldsymbol{U} \in \mathbb{R}^{M \times R}$ and $\boldsymbol{V} \in \mathbb{R}^{N \times R}$ are *factor matrices* (or simply *factors*), and $R \leq \min(M, N)$ represents the *rank*. We refer to $\boldsymbol{U}$ as the *basis matrix* and $\boldsymbol{V}$ as the *coefficient matrix*. By selecting a sufficiently small value for R, the factor matrices $\boldsymbol{U}$ and $\boldsymbol{V}$, with a combined total of $(M + N)R$ elements, offer a compact representation of the original matrix $\boldsymbol{X}$, which has MN elements, capturing the most significant patterns in the image. Depending on the loss function used to measure the reconstruction error between $\boldsymbol{X}$ and the product $\boldsymbol{U}\boldsymbol{V}^\mathsf{T}$, as well as the constraints on the factor matrices $\boldsymbol{U}$ and $\boldsymbol{V}$, various formulations and variants have been proposed for different purposes. In Section 3.3, we introduce and elaborate on our variant, termed integer matrix factorization (IMF), and argue why it is well-suited and effective for image compression.

Reshape factors. IMF yields integers factor matrices $\boldsymbol{U}_Y \in \{0, \ldots, 255\}^{\frac{HW}{64} \times R}$ and $\boldsymbol{V}_Y \in \{0, \ldots, 255\}^{64 \times R}$; $\boldsymbol{U}_{C_B} \in \{0, \ldots, 255\}^{\frac{HW}{256} \times R}$ and $\boldsymbol{V}_{C_B} \in \{0, \ldots, 255\}^{64 \times R}$; and $\boldsymbol{U}_{C_R} \in \{0, \ldots, 255\}^{\frac{HW}{256} \times R}$ and $\boldsymbol{V}_{C_R} \in \{0, \ldots, 255\}^{64 \times R}$ that correspond to $\boldsymbol{X}_Y$, $\boldsymbol{X}_{C_B}$, and $\boldsymbol{X}_{C_R}$. We reshape these matrices by unfolding their first dimension to obtain R-channel 2D spatial maps, referred to as *factor maps* and represented by the following tensors:

$$\begin{aligned} \boldsymbol{\mathcal{U}}_Y &\in \{0, \ldots, 255\}^{R \times \frac{H}{8} \times \frac{W}{8}}, \\ \boldsymbol{\mathcal{U}}_{C_B}, \boldsymbol{\mathcal{U}}_{C_R} &\in \{0, \ldots, 255\}^{R \times \frac{H}{16} \times \frac{W}{16}}, \\ \boldsymbol{\mathcal{V}}_Y, \boldsymbol{\mathcal{V}}_{C_B}, \boldsymbol{\mathcal{V}}_{C_R} &\in \{0, \ldots, 255\}^{R \times 8 \times 8}. \end{aligned} \tag{2}$$

Lossless image compression. Since *factor maps* are already integer tensors, we do not need a quantization step, which is commonly present in other lossy image compression methods and adds extra complications. This is the main advantage of our approach. Moreover, each channel of a *factor map* can be treated as a 8-bit grayscale image and therefore can be directly encoded by any standard lossless image compression method, such as PNG or WebP. For images with a resolution of $H, W \gg 64$, which are most common nowadays, the basis maps ($\boldsymbol{\mathcal{U}}$) are significantly larger than the coefficient maps ($\boldsymbol{\mathcal{V}}$), accounting for the majority of the storage space. Interestingly, in practice, the IMF basis maps turn out to be meaningful images, each capturing some visual semantic of the image (see Figure 2 for an example). Therefore, our IMF approach can effectively leverage the power of existing lossless image compression algorithms, offering a significant advantage over current methods.

3.2 Decoding

The decoder receives an encoded image and reconstructs the RGB image by applying the inverse of the operations used by the encoder, starting from the last layer and moving to the first. Initially, the factor maps are produced by losslessly decompressing the encoded image. These maps are then transformed into factor matrices by flattening their spatial dimensions. The matrices $\boldsymbol{X}_Y$, $\boldsymbol{X}_{C_B}$, and $\boldsymbol{X}_{C_R}$ are calculated through the product of the corresponding factor matrices. The *luma* and downsampled *chroma* components are then obtained by reshaping $\boldsymbol{X}_Y$, $\boldsymbol{X}_{C_B}$, and $\boldsymbol{X}_{C_R}$ back into their spatial forms. Subsequently, the downsampled *chroma* components are upsampled to their original size using bilinear interpolation. Finally, the $\mathrm{YC_BC_R}$ image is converted back into an RGB image.

3.3 Integer Matrix Factorization (IMF)

The main building block of our method is integer matrix factorization (IMF), which is responsible for lossy compression of matrices obtained through patchification. IMF can be framed as an optimization problem, aiming to minimize the reconstruction error between the original matrix $\boldsymbol{X} \in \mathbb{R}^{M\times N}$ and the product $\boldsymbol{U}\boldsymbol{V}^{\mathsf{T}}$, while ensuring that the elements of the factor matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are integers within a specified interval $[\alpha, \beta]$. Formally, the IMF problem can be expressed as:

$$\begin{aligned} &\min_{\boldsymbol{U},\boldsymbol{V}} \ \|\boldsymbol{X} - \boldsymbol{U}\boldsymbol{V}^{\mathsf{T}}\|_{\mathrm{F}}^2 \\ &\text{s.t. } \boldsymbol{U} \in \mathbb{Z}_{[\alpha,\beta]}^{M\times R}, \boldsymbol{V} \in \mathbb{Z}_{[\alpha,\beta]}^{N\times R}, \end{aligned} \tag{3}$$

where $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm; $R \leq \min(M, N)$ represents the *rank*; and $\mathbb{Z}_{[\alpha,\beta]} \triangleq [\alpha, \beta] \cap \mathbb{Z}$ denotes the set of integers within $[\alpha, \beta]$. Without constraints on the factors, the problem would have an analytic solution through the singular value decomposition (SVD), as addressed by the Eckart–Young–Mirsky theorem [4]. If only a nonnegativity constraint were applied (without integrality), variations of nonnegative matrix factorization (NMF) would emerge [7, 5]. The IMF problem (3) poses a challenging integer program, with finding its global minima known to be NP-hard [3, 9]. Only a few iterative algorithms [3, 8] have been proposed to find a "good solution" for some IMF variants in contexts other than image compression. In Section 3.4, we propose an efficient iterative algorithm for the IMF problem (3).

The application of SVD and NMF in image compression is problematic mainly because the resulting factors contain continuous values that must be represented as arrays of floating-point numbers. This necessitates a quantization step that not only adds extra complications but also significantly degrades compression performance due to quantization errors (as demonstrated in Section 4). Conversely, our IMF formulation produces integer factor matrices that can be directly stored and losslessly processed without incurring roundoff errors. The reason for limiting the feasible region to $[\alpha, \beta]$ in our IMF formulation is to enable more compact storage of the factors using standard integral data types, such as `int8` and `int16`, supported by programming languages. Given that the elements of the input matrix $\boldsymbol{X}$ are in $[0, 255]$, we found the signed `int8` type, which represents integers from -128 to 127, suitable for image compression applications. As a result, our IMF formulation is well-suited for image compression, effectively integrating the factorization and quantization steps into a single, efficient compression process.

3.4 Block Coordinate Descent Scheme for IMF

We propose an efficient algorithm for IMF using the block coordinate descent (BCD) scheme (aka alternating optimization). Starting with some initial parameter values, this approach involves sequentially minimizing the cost function with respect to a single column of a factor at a time, while keeping the other columns of that factor and the entire other factor fixed. This process is repeated until a stopping criterion is met, such when the change in the cost function value falls below a predefined threshold or the maximum number of iterations is reached. Formally, this involves solving one of the following subproblems at a time:

$$\boldsymbol{u}_r \leftarrow \operatorname*{arg\,min}_{\boldsymbol{u}_r \in \mathbb{Z}_{[\alpha,\beta]}^{M}} \|\boldsymbol{E}_r - \boldsymbol{u}_r\boldsymbol{v}_r^{\mathsf{T}}\|_{\mathrm{F}}^2, \tag{4}$$

$$\boldsymbol{v}_r \leftarrow \operatorname*{arg\,min}_{\boldsymbol{v}_r \in \mathbb{Z}_{[\alpha,\beta]}^{N}} \|\boldsymbol{E}_r - \boldsymbol{u}_r\boldsymbol{v}_r^{\mathsf{T}}\|_{\mathrm{F}}^2, \tag{5}$$

Algorithm 1: The proposed block coordinate descent (BCD) algorithm for IMF.

Input: $\boldsymbol{X} \in \mathbb{R}^{M\times N}$, factorization rank R
Output: Factor matrices $\boldsymbol{U} \in \mathbb{Z}_{[\alpha,\beta]}^{M\times R}$ and $\boldsymbol{V} \in \mathbb{Z}_{[\alpha,\beta]}^{N\times R}$

Initialize $\boldsymbol{U}, \boldsymbol{V}$ using the truncated SVD method, provided by (10) and (11)
while *stopping criterion not satisfied* **do**
  $\boldsymbol{A} \leftarrow \boldsymbol{X}\boldsymbol{V}, \boldsymbol{B} \leftarrow \boldsymbol{V}^{\mathsf{T}}\boldsymbol{V}$
  for $r = 1, \ldots, R$ **do**
    $U_{:r} \leftarrow \mathrm{clamp}_{[\alpha,\beta]}\left(\mathrm{round}\left(\frac{A_{:r} - \sum_{s\neq r} B_{sr}U_{:s}}{\|V_{:r}\|^2}\right)\right)$
  end
  $\boldsymbol{A} \leftarrow \boldsymbol{X}^{\mathsf{T}}\boldsymbol{U}; \boldsymbol{B} \leftarrow \boldsymbol{U}^{\mathsf{T}}\boldsymbol{U}$
  for $r = 1, \ldots, R$ **do**
    $V_{:r} \leftarrow \mathrm{clamp}_{[\alpha,\beta]}\left(\mathrm{round}\left(\frac{A_{:r} - \sum_{s\neq r} B_{sr}V_{:s}}{\|U_{:r}\|^2}\right)\right)$
  end
end
return $(\boldsymbol{U}, \boldsymbol{V})$

where $\boldsymbol{u}_r = U_{:r}$ and $\boldsymbol{v}_r = V_{:r}$ represent the r-th columns of $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively. $\boldsymbol{E}_r = \boldsymbol{X} - \sum_{s\neq r}^{R} \boldsymbol{u}_s\boldsymbol{v}_s^{\mathsf{T}}$ is the residual matrix. We define one iteration of BCD as a complete cycle of updates across all the columns of both factors. In fact, the proposed algorithm is a $2R$-block coordinate descent procedure, where at each iteration, first the columns of $\boldsymbol{U}$ and then the columns of $\boldsymbol{V}$ are updated (see Algorithm 1). Note that subproblem (5) can be transformed into the same form as (4) by simply transposing its error term inside the Frobenius norm. Therefore, we only need to find the best rank-1 approximation with integer elements constrained within a specific interval. Fortunately, this problem has a closed-form solution, as addressed by Theorem 1 below.

Theorem 1 (Integer rank-1 approximation)**.** *Let* $\boldsymbol{E} \in \mathbb{R}^{M\times N}$, $\boldsymbol{u} \in \mathbb{Z}_{[\alpha,\beta]}^{M\times 1}$, *and* $\boldsymbol{v} \in \mathbb{R}^{N\times 1}$. *A global solution to the problem*

$$\boldsymbol{u}^* \in \underset{\boldsymbol{u}\in\mathbb{Z}_{[\alpha,\beta]}^{M\times 1}}{\arg\min} \|\boldsymbol{E} - \boldsymbol{u}\boldsymbol{v}^{\mathsf{T}}\|_F^2 \tag{6}$$

is given by

$$\boldsymbol{u}^* = \mathrm{clamp}_{[\alpha,\beta]}\left(\mathrm{round}\left(\frac{\boldsymbol{E}\boldsymbol{v}}{\|\boldsymbol{v}\|^2}\right)\right), \tag{7}$$

where $\mathrm{round}(\boldsymbol{Z})$ *denotes the operator that rounds each element of* $\boldsymbol{Z}$ *to the nearest integer, and* $\mathrm{clamp}_{[\alpha,\beta]}(\boldsymbol{Z}) \triangleq \max(\alpha, \min(\boldsymbol{Z}, \beta))$ *denotes the operator that clamps each element of* $\boldsymbol{Z}$ *to the interval* $[\alpha, \beta]$.

Proof. See Appendix A for the proof. □

Using Theorem 1, we obtain the following update formulas for our BCD algorithm:

$$\boldsymbol{u}_r \leftarrow \mathrm{clamp}_{[\alpha,\beta]}\left(\mathrm{round}\left(\frac{\boldsymbol{E}_r\boldsymbol{v}_r}{\|\boldsymbol{v}_r\|^2}\right)\right), \tag{8}$$

$$\boldsymbol{v}_r \leftarrow \mathrm{clamp}_{[\alpha,\beta]}\left(\mathrm{round}\left(\frac{\boldsymbol{E}_r^{\mathsf{T}}\boldsymbol{u}_r}{\|\boldsymbol{u}_r\|^2}\right)\right). \tag{9}$$

It is noteworthy that the combination of $\mathrm{round}(\cdot)$ and $\mathrm{clamp}_{[\alpha,\beta]}(\cdot)$ can be interpreted as the element-wise projector to $\mathbb{Z}_{[\alpha,\beta]}$. Thanks to Theorem 1, the proposed algorithm offers an encouraging property that guarantees the cost function is monotonically non-increasing with each update.

Initialization. The initial values of factors can significantly impact the convergence performance of the BCD algorithm. We found that the convergence with naive random initialization can be too slow. To address this issue, we propose an initialization method using SVD. The procedure

is straightforward. First, the truncated SVD of the input matrix $\boldsymbol{X} \in \mathbb{R}^{M \times N}$ is computed as $\tilde{\boldsymbol{U}}\boldsymbol{\Sigma}\tilde{\boldsymbol{V}}^{\mathsf{T}}$, where $\boldsymbol{\Sigma} \in \mathbb{R}^{R \times R}$ is a diagonal matrix corresponding to the R largest singular values. $\tilde{\boldsymbol{U}} \in \mathbb{R}^{M \times R}$ contains the corresponding left-singular vectors in its columns, and $\tilde{\boldsymbol{V}} \in \mathbb{R}^{N \times R}$ contains the corresponding right-singular vectors in its columns. The initial factors are then calculated as follows:

$$\boldsymbol{U}^0 = \text{clamp}_{[\alpha,\beta]}(\text{round}(\tilde{\boldsymbol{U}}\boldsymbol{\Sigma}^{\frac{1}{2}})), \tag{10}$$

$$\boldsymbol{V}^0 = \text{clamp}_{[\alpha,\beta]}(\text{round}(\boldsymbol{\Sigma}^{\frac{1}{2}}\tilde{\boldsymbol{V}})). \tag{11}$$

Essentially, this means we first low-rank approximate $\boldsymbol{X}$ and then project the elements of the resulting factor matrices into $\mathbb{Z}_{[\alpha,\beta]}$. Algorithm 1 provides the pseudocode for the proposed BCD algorithm for IMF.

3.5 Implementation Details

Parameter Setup. We used a patch size of 8×8 for patchification. The number of BCD iterations for IMF is by default set to 10 (although our ablation studies in Section 4.4 suggests that even 2 iterations may be sufficient in practice). For lossless compression of factor maps, we employed the WebP codec implemented in the Pillow library [1].

Evaluation. For comparison, we experimented with the widely-used Kodak dataset [6], with 24 lossless images of 768 × 512 resolution. To assess the robustness of our proposed method, we also tested it using the CLIC 2024 validation dataset [**?**], consisting of 30 high-resolution, high-quality images. To evaluate the rate-distortion performance, we measured the bit rate in bits per pixel (bpp) and assessed the quality of reconstructed images using the PSNR and SSIM metrics. We plotted rate-distortion (RD) curves, such as the PSNR-bpp curve, for each method to demonstrate the compression performance.

Baseline Codecs. For JPEG compression, we employed the Pillow library [1]. Our SVD-based compression baseline follows the same framework as the proposed IMF compression algorithm. However, it employs truncated SVD instead of IMF, followed by uniform quantization of the factors and lossless compression using the zlib library [2]. This differs from the IMF algorithm where factors are first reshaped into factor maps and then compressed losslessly as images using WebP.

4 Experiments

In this section, the proposed IMF-based compression method is assessed against SVD-based and JPEG [] methods. The performance is reported qualitatively and also based on two criteria, namely rate-distortion performance and image classification performance. Moreover, ablation studies are presented to investigate the effect of different hyperparameters in IMF.

4.1 Qualitative Performance

Qualitative performance is shown on an image selected from the Kodak [] dataset. Fig.2 depicts the first IMF components U_Y, U_{Cb}, and U_{Cr} which are extracted following the procedure elaborated in Section 3.5. It is evident in the figure that the IMF components with higher energy maintain the overall texture of the image in each channel, while components with lower energy focus more on subtle changes.

The qualitative comparison is made in Fig.**??** on an image selected from the Kodak [] dataset. The images compressed by the considered compression methods are shown in different bits per pixel (bpp) values, a standard compression measure in the literature []. As can be seen, the IMF-based compression method is capable of maintaining quality compressed images in bpp values as low as ?, outperforming JPEG and SVD-based methods which suffer from maintaining balanced colors as soon as bpp drops below ? and ?, respectively. The artifacts in color are visible, e.g. by JPEG in bpp values starting ?.

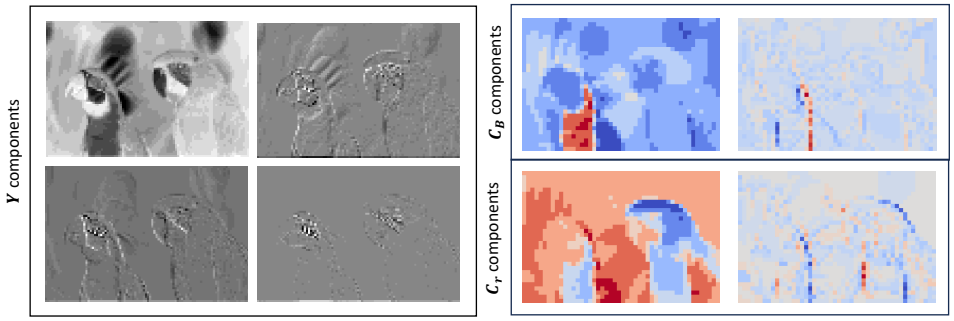


Figure 2 IMF components of the `kodim23` image from the Kodak dataset, corresponding to luma (Y), blue-difference (Cb), and red-difference (Cr) chroma.

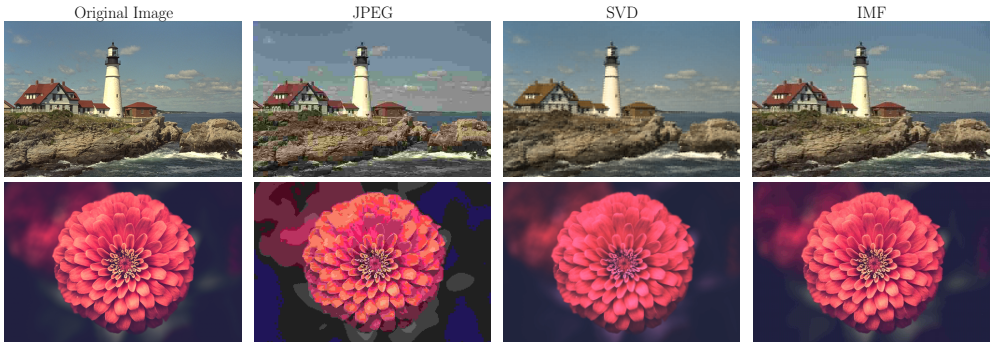


Figure 3 Qualitative performance comparison between various compression methods. The top row shows performance on kodim21 from the Kodak dataset in the bpp value of 0.3, and the bottom row shows one example of the Clic dataset in the bpp values of 0.14, 0.12, and 0.1 for JPEG, SVD, and IMF, respectively.

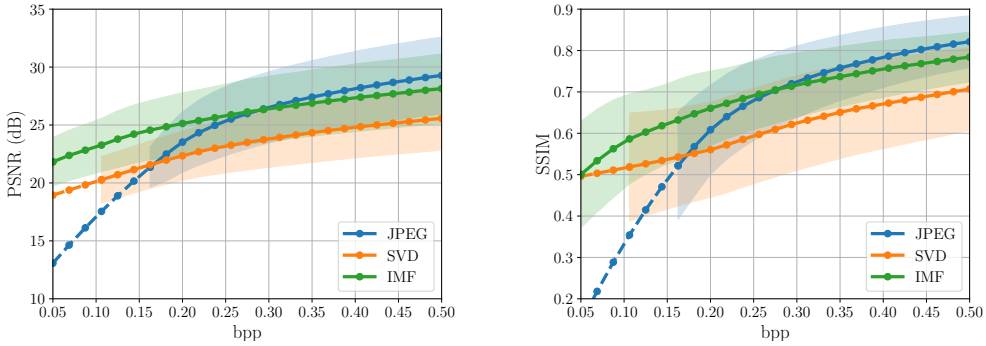
4.2 Rate-Distortion Performance

Peak signal-to-noise ratio (PSNR), as well as structural similarity index measure (SSIM) [], are reported versus bpp for the considered methods. The considered datasets are Clic [] and Kodak, consisting of respectively 32 and 24 colored images of various sizes. For each image compressed by any of the considered methods, bpp, PSNR, and SSIM are calculated. Then, for each compression method, PSNR and SSIM values are interpolated linearly in the fixed bpp values in the range $(0.05, 0.5)$. In the following plots, the average performance over all images is reported, along with the standard deviation presented as shadows. For the missing bpp values, the average is extrapolated quadratically and is shown by dashed lines.

In Fig.4, the compression performance on the Kodak dataset is reported. In this figure, the proposed IMF-based method outperforms the SVD-based method, which can be attributed to the quantization errors that SVD is prone to during encoding and decoding, deteriorating its performance in both criteria. In this view, the quantization-free property of IMF effectively guarantees higher performance in different bpp values. It is also evident that the IMF-based method outperforms JPEG in low bpp values. The same performance regime can also be concluded for all the mentioned compression methods on the Clic dataset in Fig.5.

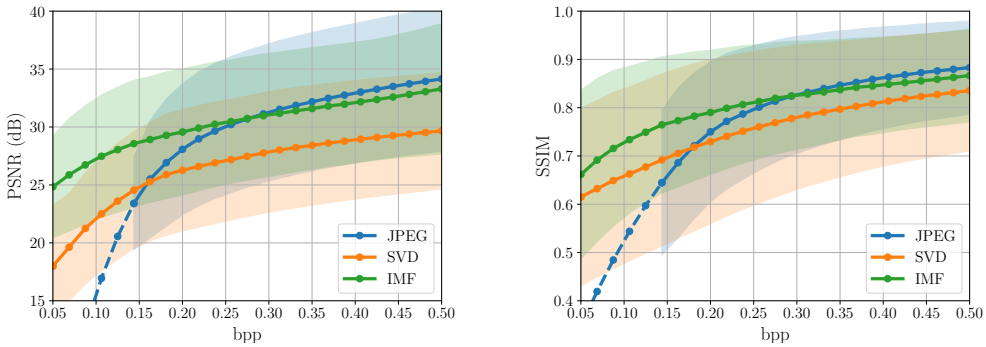
4.3 ImageNet Classification Performance

As another criterion, classification performance is investigated on the images compressed by the considered compression methods. This criterion focuses on the higher-level information required for object recognition and classification embedded in each image. Furthermore, it highlights the importance of image compression where various vision tasks such as classification are the main objective—rather than maintaining the perceived image quality—while keeping the requirement of resources such as memory, communication bandwidth, computation power, latency budget, etc. as



(a) **(b)**

Figure 4 Rate-distortion performance on the Kodak dataset. In panels (a) and (b), the average PSNR and SSIM are plotted against bpp, respectively.



(a) **(b)**

Figure 5 Rate-distortion performance on the CLIC dataset. In panels (a) and (b), the average PSNR and SSIM are plotted against bpp, respectively.

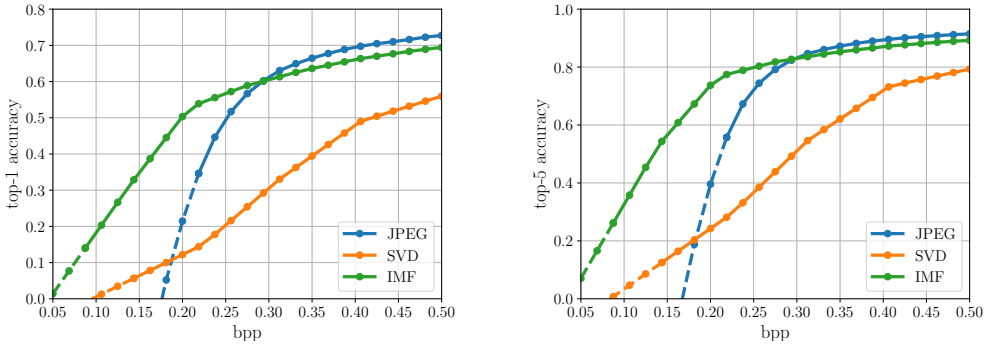
limited as possible. ImageNet [] validation set, consisting of 50000 224×224 colored images in 1000 classes, is considered for this classification task done by a ResNet-50 classifier [], pre-trained on the original ImageNet dataset. The classification performance comparison is made in Fig.6, showing the higher compression performance of IMF for classification, reaching top-1 accuracy of 70% in bpp values as low as 0.2.

4.4 Ablation Studies

In this section, ablation studies are performed, focusing on various hyper-parameters in the IMF-based compression method and their effect on the compression performance.

Patchification. In Fig.7a, patchification effect with different patch sizes is investigated. First, it can be concluded that patchification has a positive effect on performance. This performance boost is mainly due to the fact that in each patch (local) pattern variation is limited and hence IMF has more representation power to reconstruct the original pattern in each patch with fewer components. The performance on various datasets has shown that patches of size 8×8 lead to the best performance. The same conclusion is evident for the Kodak dataset example presented in Fig.7a.

Factor bounds. As elaborated in Section 3.5, during the IMF optimization, IMF components are constrained into a bound $(-\alpha, \alpha - 1)$. Fig.7b studies the compression performance versus different bounds. According to the results, the bound $(-16, 15)$ leads to the best performance since the value distribution of IMF components lies mostly in this bound. Hence, dedicating fewer bits to



(a) **(b)**

Figure 6 Impact of different compression methods on ImageNet classification accuracy. Panels (a) and (b) show the validation top-1 and top-5 accuracy plotted against bits per pixel (bpp), respectively. A ResNet-50 model pre-trained on the original ImageNet images is evaluated using validation images compressed by different methods.

represent this narrower bound compared to the other bounds results in higher compression rates without sacrificing performance.

BCD iteration. The next parameter to study is the inner iteration number required in IMF BCD updates. According to the numerical results on various datasets, the objective value in the IMF optimization drops drastically after 2 iterations, while more iteration numbers have marginal improvement. This observation is shown for the Kodak dataset in Fig.7c. This feature makes the IMF computationally efficient since with a limited number of iterations a high compression performance can be achieved.

Color space. The compression performance of IMF is studied with two color spaces, namely RGB and YCbCr, in Fig.7d. Although the compression performance remains unchanged in terms of PSNR, qualitative results reported in Fig.? indicate that YCbCr color space can maintain natural colors and brightness more effectively. Consistently, the JPEG method employs this color space as well.

5 Conclusion and Future Work

Acknowledgments and Disclosure of Funding

References

[1] Alex Clark. Pillow (pil fork) documentation, 2015. URL https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf.

[2] Peter Deutsch and Jean-Loup Gailly. Zlib compressed data format specification version 3.3. Technical report, 1996.

[3] Bo Dong, Matthew M Lin, and Haesun Park. Integer matrix approximation and data mining. *Journal of scientific computing*, 75:198–224, 2018.

[4] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1 (3):211–218, 1936.

[5] Nicholas Gillis. *Nonnegative Matrix Factorization*. SIAM, 2020.

[6] Eastman Kodak. Kodak lossless true color image suite, 1993. URL https://r0k.us/graphics/kodak/.

[7] Daniel Lee and H. Sebastian Seung. Algorithms for Non-negative Matrix Factorization. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000. URL https://proceedings.neurips.cc/paper_files/paper/2000/file/f9d1152547c0bde01830b7e8bd60024c-Paper.pdf.

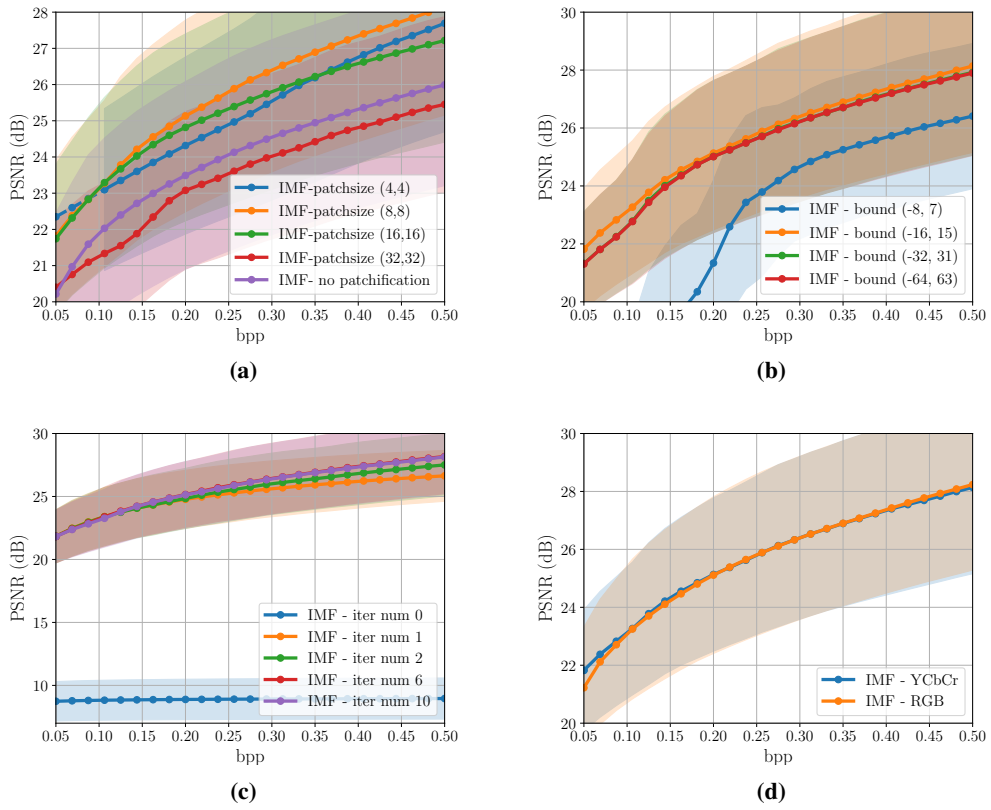


Figure 7 Ablation experiments for the IMF compression method. In all cases, we plot PSNR as a function of bits per pixel (bpp) on the Kodak dataset. (a) Compares IMF compression performance without patchification and different patch sizes. (b) Compares IMF compression performance for different bound values of factor matrices. (c) Compares IMF compression performance for different numbers of BCD iterations. (d) Compares IMF compression performance between RGB and YCbCr color space transform.

[8] Matthew M Lin, Bo Dong, and Moody T Chu. Integer matrix factorization and its application.

[9] Peter van Emde Boas. Another np-complete problem and the complexity of computing short vectors in a lattice. *Tecnical Report, Department of Mathmatics, University of Amsterdam*, 1981.

A Proof of Theorem ??