# PCA-Based Lossy Image Compression with Generalized Hebbian Algorithm

Tomáš Belák

May 7, 2025

**Abstract** In this project, I explore the use of the Generalized Hebbian Algorithm (GHA) for principal component analysis (PCA)-based lossy image compression. The principal components were learned from an 8×8 block decomposition of a 256×256 grayscale portrait. These components were then used to encode and reconstruct three different images - two portraits and one image of a leopard. Despite the variation in image content, the encoding generalized well, producing visually coherent reconstructions across all test cases.

## Contents

**1 Introduction** **2**

**2 Methods** **2**
- 2.1 Image Preparation . . . . . 2
- 2.2 Mean Centering . . . . . 3
- 2.3 Principal Component Extraction with GHA . . . . . 3
- 2.4 Compression and Encoding . . . . . 4
- 2.5 Image Reconstruction . . . . . 4
- 2.6 Generalization to Other Images . . . . . 4

**3 Results** **5**
- 3.1 Compression Results for Varying $k$ . . . . . 5
- 3.2 Learned Components and Their Coefficient Maps . . . . . 7
- 3.3 Eigenvalue Plot . . . . . 9
- 3.4 Generalization to Other Images . . . . . 10

**4 Conclusion** **12**

# 1 Introduction

In this project, I used the Generalized Hebbian Algorithm (GHA) to do image compression with Principal Component Analysis (PCA). The goal was to reduce the size of images by keeping only the most important information.

I worked with a 256×256 grayscale portrait of a woman. This portrait was divided into 8×8 non-overlapping blocks. I used GHA to learn the first $k$ principal components in these blocks. Then, I used these principal components to compress the image by projecting the image on to these principal components - I will refer to this process as encoding. I tried different number of principal components, however it is important to mention that because of the size of the blocks only if $k = 88 = 64$ the compression has the potential to be lossless.

To test how well this works, I used the same principal components to encode and reconstruct three different images — two portraits and one of a leopard. Despite the variation in image content, the encoding generalized well, producing visually coherent reconstructions.

This report explains how the method works, shows the results, and discusses how well GHA can be used for this kind of image compression.

# 2 Methods

## 2.1 Image Preparation

The main image used in this project was a 256×256 grayscale portrait called "Elaine" which can be seen on the figure 1. The image was loaded using the Pillow library and normalized to values between 0 and 1. It was then divided into 1024 non-overlapping blocks of size 8×8 pixels. Each 8×8 block was flattened into a 64-dimensional vector. These vectors were used as the input for PCA training.

Figure 1: Learning rate schedule over 600 epochs.

## 2.2 Mean Centering

Before training, the 64-dimensional input vectors were mean-centered. This means the average value across all blocks was calculated and subtracted from each vector. This step is important for PCA.

## 2.3 Principal Component Extraction with GHA

The Generalized Hebbian Algorithm (GHA) was used to find the first $k = 8$ principal components.

For learning the principal components I used sequential GHA, that finds components one by one, ensuring that each new component is orthogonal to the previous. Each component was trained for 600 epochs as that seemed to offer a nice balance between good convergence to orthonormal basis and training time . The figure 2 shows how the learning rate decreased over time with starting value of 0.001 and finishing value 0.0001. The weights were initialized with zero mean. Also, the training method utilized shuffling of the inputs in each epoch which was done with random seed equal to 24.

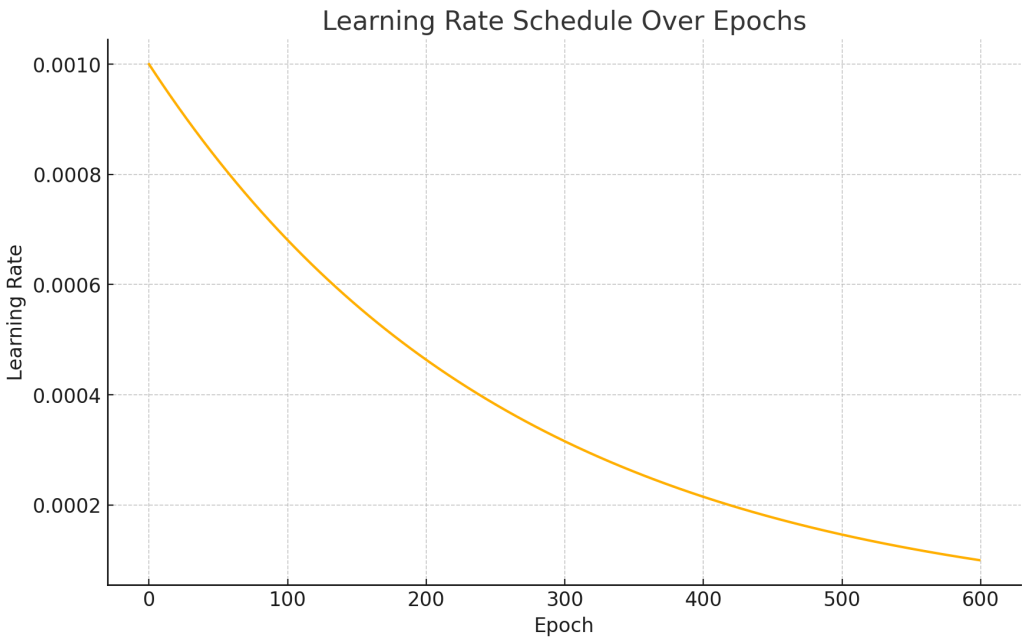


Figure 2: Learning rate schedule over 600 epochs.

## 2.4 Compression and Encoding

Once the components were learned, the image was compressed. Each flattened 8×8 block was encoded by projecting it onto the first $k$ principal components. This gave $k$ values per block, which formed the compressed data. This process was done for $k = 8$.

## 2.5 Image Reconstruction

To reconstruct the image, the compressed values were multiplied back with the first $k$ learned components and the mean vector was added again. Each reconstructed 64-dimensional vector was reshaped into an 8×8 block. All blocks were placed back into their original positions to create the full image.

## 2.6 Generalization to Other Images

To test how well the learned components worked on new data, three other grayscale images were loaded and resized to 256×256. These images were also divided into 8×8 blocks, flattened, and encoded using the components learned from the Elaine image. The same reconstruction process was used

to rebuild these images, and the quality was evaluated mainly visually and supported by using mean squared error (MSE).

# 3 Results

## 3.1 Compression Results for Varying $k$

To evaluate the quality of reconstruction, the "Elaine" image was compressed and then reconstructed using different numbers of principal components: $k = 1...8$. As expected, the quality of the reconstructed image improved with higher values of $k$ which can be seen on the figure 3.

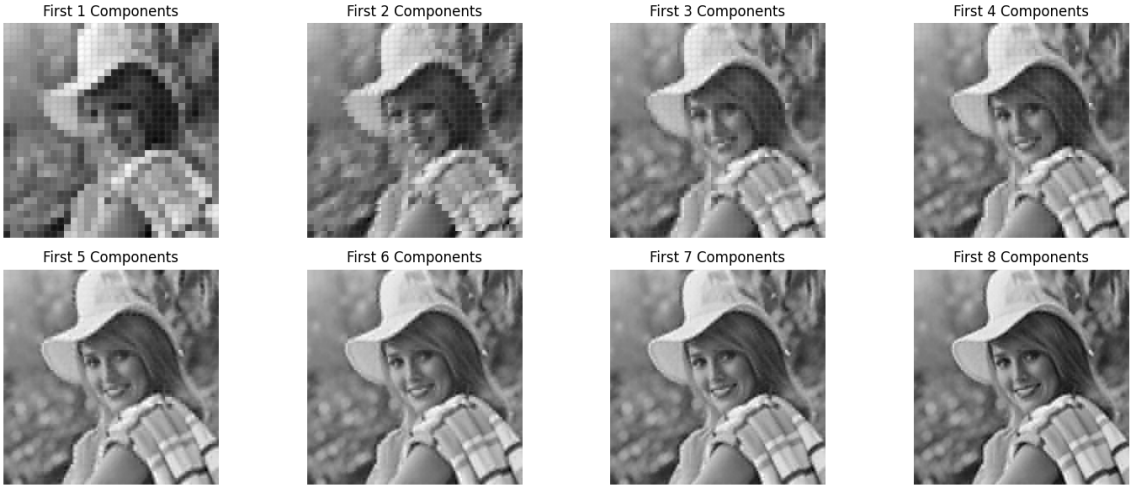


Figure 3: Reconstructed using first 1...8 principal components

In the figure 4, we can see the comparison of the original image to the one encoded and reconstructed using the first 8 learned principal components.


Figure 4: The original image compared to reconstructed image using $k = 8$ components

In the figure 5, we can see how the MSE evolved with each added principal component. As $k$ increases, more principal components are included in the reconstruction, which naturally leads to a lower reconstruction error. However, the improvement in mean squared error (MSE) does not grow linearly. The largest gains in quality come from the first few components, which capture the most important patterns and variations in the image. Each additional component contributes less and less to the total explained variance, which is also reflected in the flattening curve of the eigenvalues that can be viewed in a later subsection in the figure 8.

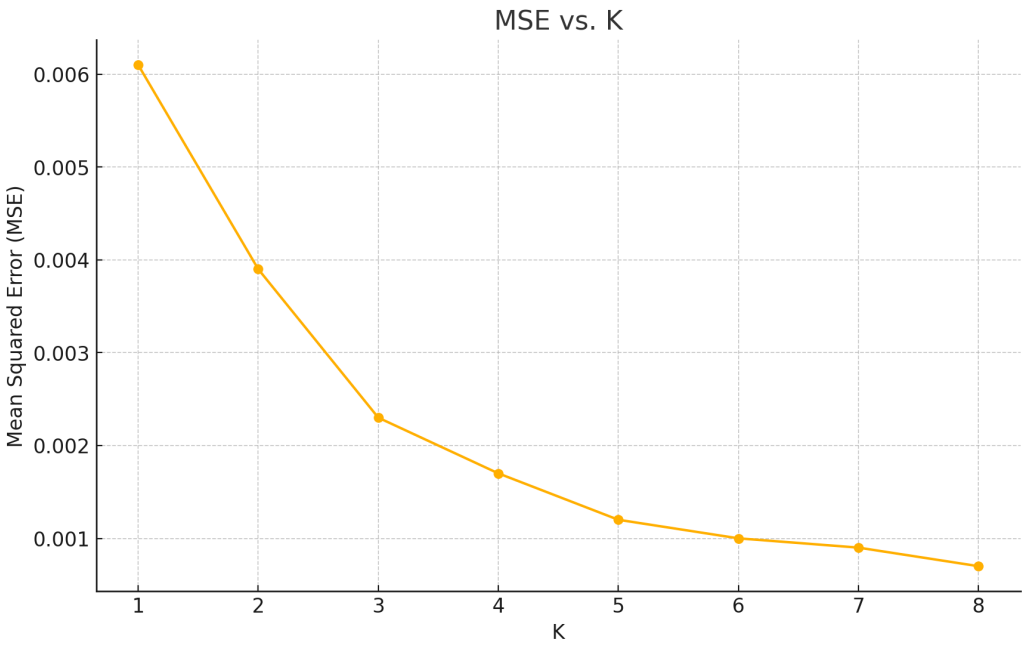


Figure 5: Evolution of MSE with growing k

## 3.2 Learned Components and Their Coefficient Maps

The eight most significant principal components learned from the Elaine image were reshaped into 8×8 blocks and visualized as grayscale images. These patterns represent the directions of greatest variance across the 8×8 image blocks and form the basis vectors used for compression.

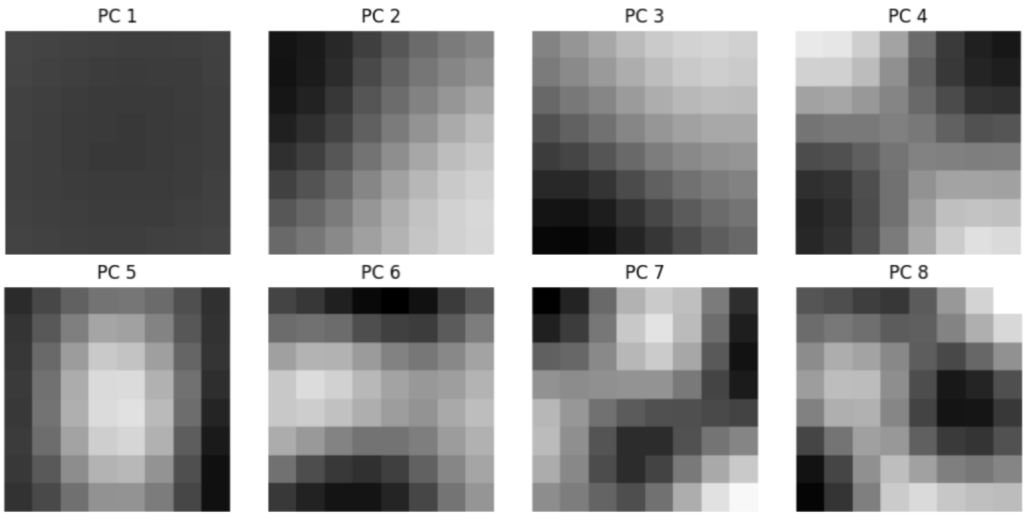


Figure 6: First eight principal components visualized as 8×8 grayscale patterns

Each 8×8 block in the image is encoded using a weighted combination of these components. The weight (or coefficient) for each component indicates how strongly that pattern appears in a given block. To better understand where and how each component is used across the image, the corresponding coefficient maps are shown below. Each map is a 32×32 grid (since the image has 1024 blocks), where lighter values mean higher usage of that component in that region.

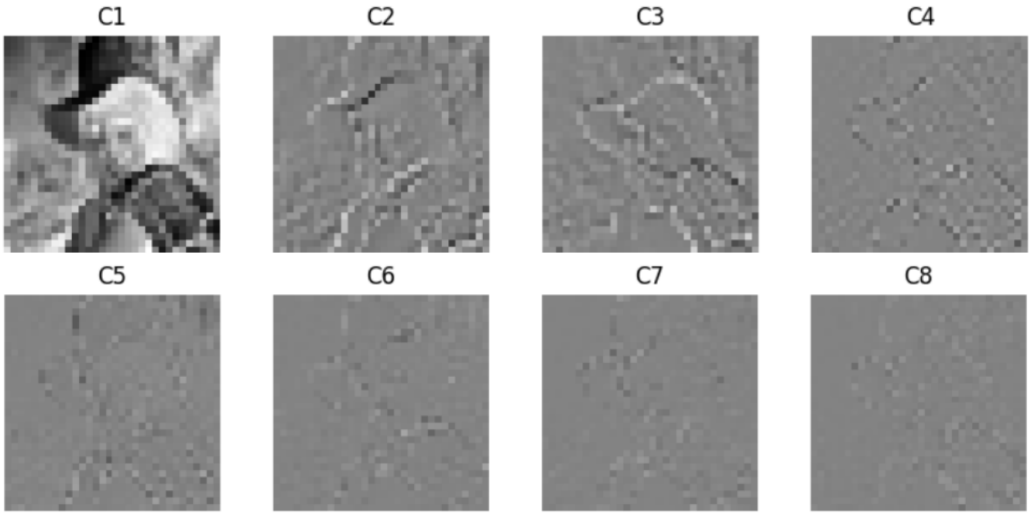


Figure 7: Coefficient maps for the first eight principal components (C1 to C8)

These coefficient maps clearly show how different components contribute to different areas of the image. For example, some components are more active in textured regions, while others highlight edges or smooth areas.

## 3.3 Eigenvalue Plot

The eigenvalues of the PCA correlation matrix, sorted in decreasing order, indicate how much variance each principal component explains. The plot in the figure 8 shows that most of the information is captured by the first few components (circa 8). We can see that the learned eigenvalues are identical to the correlation matrix eigenvalues.

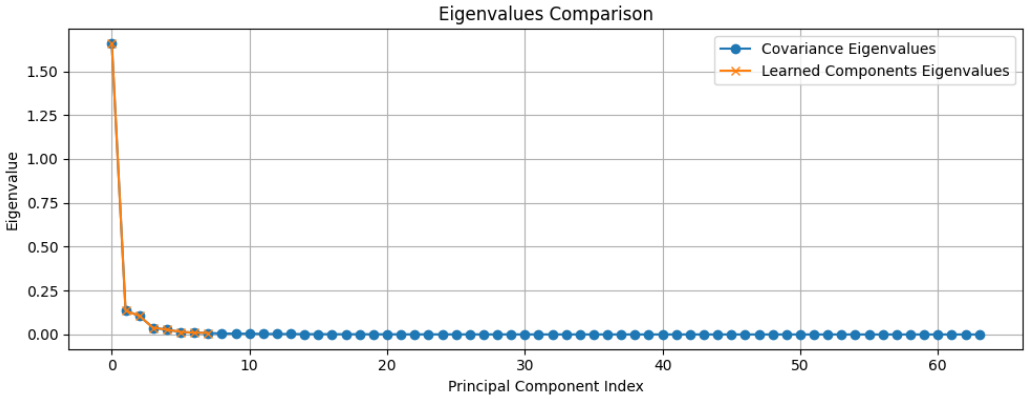


Figure 8: Comparison of learned eigenvalues and eigenvalues of the PCA correlation matrix in decreasing order

## 3.4 Generalization to Other Images

To test generalization, three other grayscale images (two portraits and one of a leopard) were encoded and reconstructed using the components learned from the Elaine image, with $k = 8$. Although the images were not part of the training data, the reconstructions were visually reasonable, showing that the principal components captured useful patterns that generalized well.


Figure 9: Example 1 – portrait reconstructed using Elaine's components ($k = 8$)


Figure 10: Example 2 – portrait reconstructed using Elaine's components ($k = 8$)

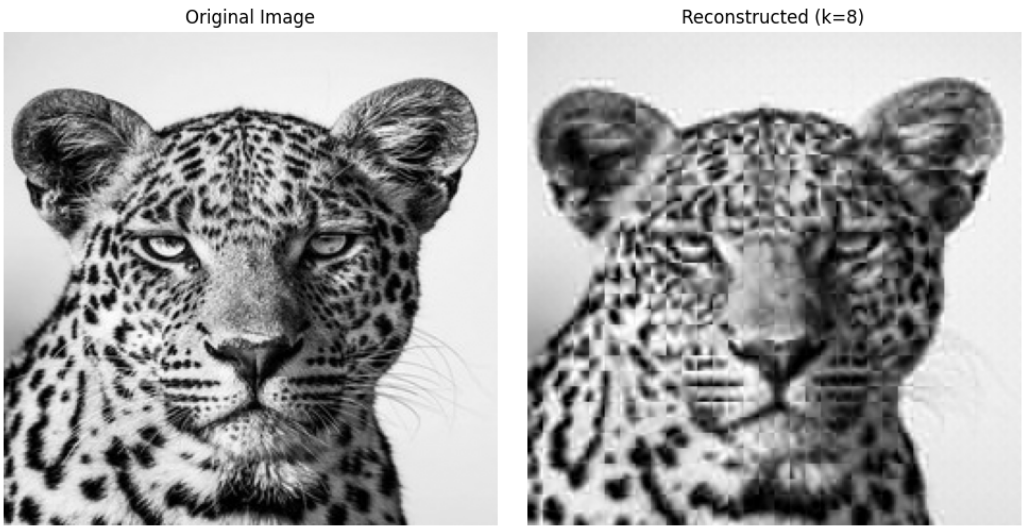


Figure 11: Example 3 – leopard reconstructed using Elaine's components ($k = 8$)

# 4 Conclusion

In this project, the Generalized Hebbian Algorithm (GHA) was successfully used to perform PCA-based image compression. The goal was to reduce image size by keeping only the most important information from small 8×8 image blocks.

Using a 256×256 grayscale portrait image, the first $k = 8$ principal components were learned from the flattened 8×8 blocks. These components were then used to encode and reconstruct the original image, as well as three additional images—two portraits and one image of a leopard. The results showed that GHA could effectively learn meaningful patterns in the image data, and the reconstructions maintained visual coherence, even for images that were not used during training.

The reconstruction quality improved as more components were used, and the generalization to new images demonstrated that the learned components captured reusable features. Although the compression is lossy for $k < 64$, the visual quality remained surprisingly good, especially for low values of $k$.

Overall, this project showed that GHA is a suitable method for PCA-based image compression, and that a small number of learned components can generalize well across similar types of images.

# Appendix

Here is a link to my github repository with all the files needed: `https://github.com/tomasbelak24/fmfiuk-ann-pca-project`