

# Trabalho 5

Tiago Loureiro Chaves (187690)

*MC920A - Introdução ao Processamento de Imagem Digital - 2s2019*

## Resumo

Este projeto teve como objetivo explorar o uso do algoritmo de Análise de Componentes Principais (PCA) para redução de dimensionalidade em imagens digitais, avaliando-o como uma técnica de compressão com perdas (*lossy*).

Para isso, obtém-se o sistema de eixos ortogonais (referentes às componentes principais do PCA) pela fatoração SVD da matriz real que representa cada canal de cor da imagem de entrada. Então, um número reduzido de componentes é utilizado para se computar a representação comprimida da imagem.

Analisa-se tanto a taxa de compressão — razão entre os tamanhos do arquivo comprimido e original — como também o valor RMSE entre as imagens. A relação entre essas métricas e o número de componentes principais utilizadas pode ser vista nos gráficos da Seção 3.

## 1 Problema

A Análise de Componetes Principais (PCA – *Principal Component Analysis*) é uma das técnicas de redução de dimensionalidade de dados mais difundidas e utilizadas, apesar de ter sido desenvolvida há mais de um século, em 1901 [1].

O algoritmo do PCA consiste em:

- 1) centralizar os dados  $n$ -dimensionais em zero (i.e. subtrair a média de cada *feature*);
- 2) computar a matriz de covariância dos dados;
- 3) encontrar os autovalores e autovetores da matriz;
- 4) escolher os autovetores associados aos  $k$  maiores autovalores (que correspondem às componentes principais que melhor explicam a variância dos dados);
- 5) projetar os dados no espaço formado pelas  $k$  componentes principais selecionadas.

Para calcularmos os autovetores e autovalores da matriz de covariância usaremos a técnica de decomposição em valores singulares (SVD – *Singular Value Decomposition* [2]).

Neste trabalho estamos interessados em explorar uma aplicação não usual do PCA: a compressão de imagens digitais. Tal uso não é comum pois, como veremos na Seção 3, os resultados somente passam a apresentar boa qualidade visual quando a imagem comprimida ocupa praticamente o mesmo espaço que a original, e mesmo ao utilizarmos uma única componente principal para reconstrução ( $k = 1$ ), a taxa de compressão não é de apenas 3:1 — enquanto métodos *lossy* como JPEG chegam a alcançar taxas de 10:1, com pouca perda perceptível na qualidade da imagem [3].

## 2 Programa

### 2.1 Dependências e Organização

O trabalho foi desenvolvido em Python 3.7.4, com as bibliotecas Numpy 1.17.0, Matplotlib 3.1.1 e OpenCV 4.1.1, e é dividido em três arquivos:

- `main.py` – compressão de imagens por PCA (utilizando SVD)
- `metrics.py` – cálculo da taxa de compressão e RMSE
- `utils.py` – funções gerais para processamento de imagens

As imagens utilizadas para teste estão na pasta `i/`, e exemplos de saída encontram-se na pasta `o/`. O código também pode ser visto em [github.com/laurelkeys/image-processing](https://github.com/laurelkeys/image-processing).

### 2.2 Execução

O projeto pode ser executado como:

```
$ python main.py in_image.png n_of_components out_image.png
```

Sendo os parâmetros:

- `in_image.png`: imagem  $M \times N$  que será comprimida.
- `out_image.png`: imagem  $M \times N$  comprimida.
- `n_of_components`: número (inteiro) de componentes principais utilizadas.

O nome com o qual a imagem comprimida deve ser salva é um parâmetro opcional, sendo `in_image-k.png` o nome padrão do arquivo de saída (salvo no mesmo diretório em que o script é executado), onde  $k \in [0, \min(M, N)]$  é o valor passado no argumento `n_of_components`.

## 3 Resultados

As imagens comprimidas foram salvas em formato PNG (compressão sem perdas) com:

```
cv2.imwrite(out_image.png, __img, [cv2.IMWRITE_PNG_COMPRESSION, 9])
```

onde a flag `[cv2.IMWRITE_PNG_COMPRESSION, 9]` da função `imwrite()` faz com que a biblioteca OpenCV salve a imagem com o maior nível de compressão *lossless* possível, apesar de tomar mais tempo (o valor padrão é 1, por ser o mais rápido [4]).



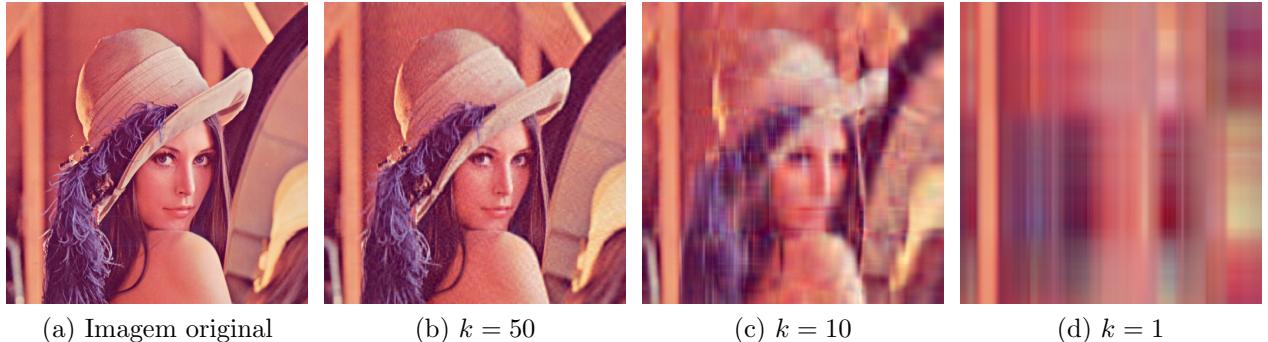
(a) Imagem original

(b)  $k = 50$

(c)  $k = 10$

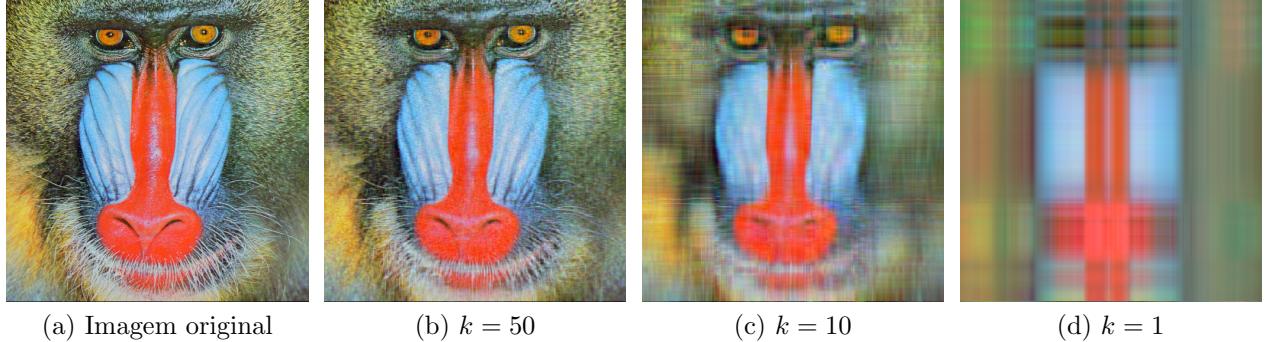
(d)  $k = 1$

Figura 1: Exemplos de resultados de compressão para a imagem `peppers.png`



(a) Imagem original (b)  $k = 50$  (c)  $k = 10$  (d)  $k = 1$

Figura 2: Exemplos de resultados de compressão para a imagem lena.png



(a) Imagem original (b)  $k = 50$  (c)  $k = 10$  (d)  $k = 1$

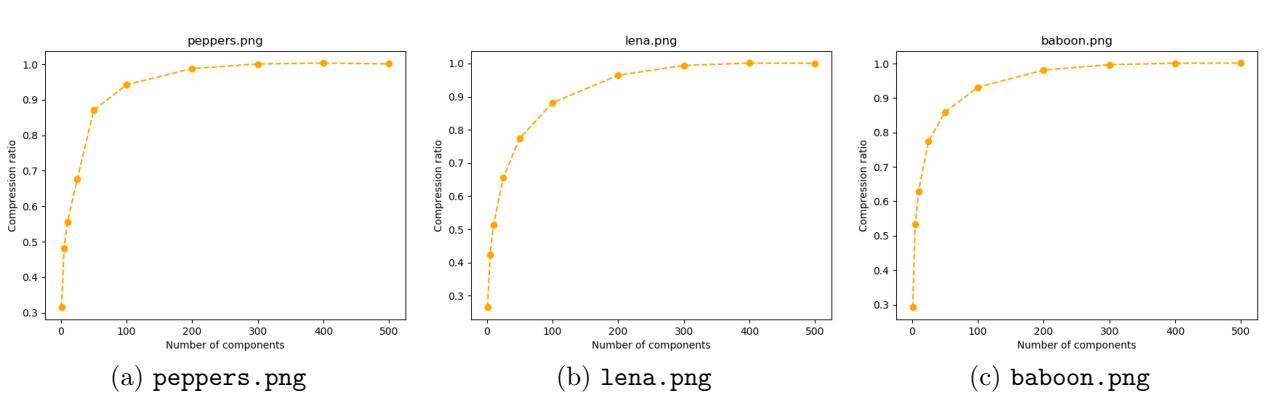


Figura 4: Taxa de compressão ( $\rho$ ) vs. número de componentes ( $k$ )

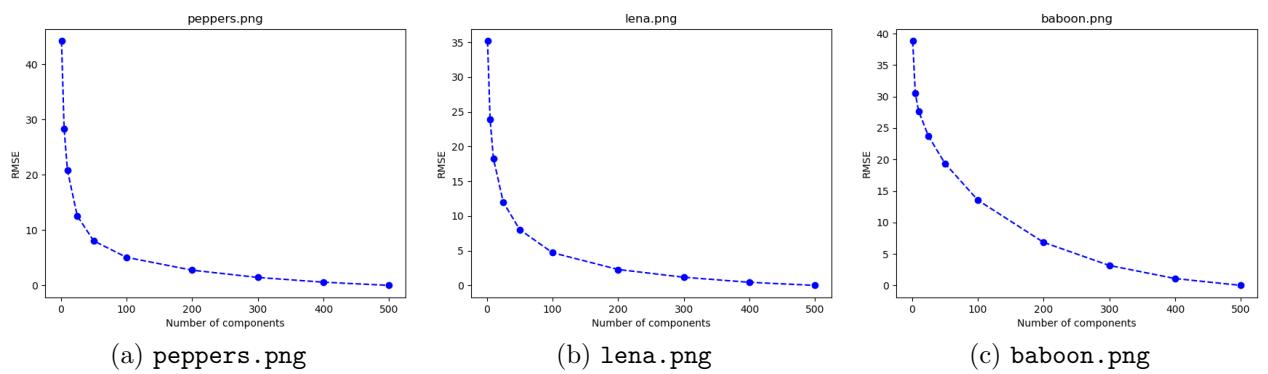


Figura 5: RMSE vs. número de componentes ( $k$ )

$k$	peppers.png		lena.png <sup>1</sup>		baboon.png	
	$\rho$	RMSE	$\rho$	RMSE	$\rho$	RMSE
1	31.48	44.21	26.48	35.17	29.18	38.81
5	48.14	28.38	42.37	23.86	53.44	30.53
10	55.62	20.86	51.27	18.27	62.93	27.71
50	87.07	8.05	77.42	8.04	85.88	19.39
100	94.24	5.08	88.15	4.70	93.11	13.55
200	98.82	2.75	96.47	2.29	98.11	6.84
300	100.11	1.44	99.42	1.17	99.68	3.16
400	100.35	0.58	100.13	0.45	100.10	1.09
500	100.14	0.00	100.08	0.00	100.14	0.00

Tabela 1: Taxa de compressão ( $\rho$ ) e RMSE para diferentes valores de  $k$  utilizados

Ressalta-se que, antes de aplicar-se a técnica SVD em cada canal das imagens originais, subtraí-se de todos os pixels a média de intensidade da sua coluna, conforme o passo 1) da Análise de Componentes Principais, mencionado na Seção 1. Então, após construirmos a imagem resultante considerando-se  $k$  componentes, adicionamos a ela as médias que foram inicialmente subtraídas da imagem original.

Sendo  $k$  o número de componentes principais utilizadas na redução,  $f$  a imagem de entrada e  $g$  a imagem de saída (ambas  $M \times N$ ), temos o pseudo-código do algoritmo:

---

**Algorithm 1** Compressão com PCA utilizando SVD

---

```

for  $i \in \{1, 2, 3\}$  :
  for  $x \in [1, N]$  :
     $f(:, x, i) \leftarrow f(:, x, i) - mean(f(:, x, i))$ 
  for  $i \in \{1, 2, 3\}$  :
     $[U_f(:, :, i), S_f(:, :, i), V_f^T(:, :, i)] \leftarrow svd(f(:, :, i))$ 
     $g = zeros(M, N)$ 
    for  $i \in \{1, 2, 3\}$  :
       $U_g(:, 1:k, i) \leftarrow U_f(:, 1:k, i)$ 
       $S_g(1:k, 1:k, i) \leftarrow S_f(1:k, 1:k, i)$ 
       $V_g^T(1:k, :, i) \leftarrow V_f^T(1:k, :, i)$ 
       $g(:, :, i) \leftarrow U_g(:, :, i) \cdot S_g(:, :, i) \cdot V_g^T(:, :, i)$ 
  return  $g$ 

```

---

## 4 Análise e Discussão

Considerando os métodos usuais de compressão *lossy* de imagens, gostaríamos de no mínimo uma taxa de compressão  $\rho$  de 50%, sendo:

$$\rho = \frac{\text{espaço utilizado pela imagem original}}{\text{espaço utilizado pela imagem comprimida}}$$

onde medimos o espaço como a quantidade de memória ocupada (em bytes). Em geral, nos referimos a essa taxa pela fração inversa de  $\rho$ , assim  $\rho = 50\%$  equivale a 2:1 ( $\frac{1}{\rho} = \frac{1}{0.5} = \frac{2}{1}$ ).

Vemos então, pela Tabela 1, que precisaríamos usar no máximo  $k = 10$  para que a imagem comprimida ocupasse metade do tamanho da original, ou seja, menos de 2% das 512 componentes principais (uma vez que as imagens são todas de  $512 \times 512$ ).

---

<sup>1</sup>[homepages.cae.wisc.edu/~ece533/images/lena.png](http://homepages.cae.wisc.edu/~ece533/images/lena.png)

Tomando como medida quantitativa da qualidade de imagens a raiz do erro quadrático médio (RMSE – *Root Mean Squared Error*), podemos avaliar os resultados de forma mais objetiva. A relação entre o RMSE e o valor de  $k$  escolhido é mostrada na Figura 5.

Como um método geral de compressão, os resultados com 100 componentes já são aceitáveis, e podemos até utilizar  $k = 50$  se uma perda considerável de nitidez for admissível. Diferenças perceptuais passam a ser imperceptíveis para valores de RMSE menores que 4.00 (próximos ao “cotovelo” da curva).

Entretanto, como vemos nas Figuras 5 e 4, precisamos de 150 a 250 componentes principais (dependendo da imagem) para que o RMSE chegue em 4.00, mas com esse número de componentes o espaço ocupado pelas imagens comprimidas já é cerca de 90% do original, o que torna o PCA um método inviável para compressão. Além disso, ao utilizarmos  $k = 1$  (apenas 0.2% da quantidade de componentes principais para imagens  $512 \times 512$ ), conseguimos um arquivo comprimido com cerca de 30% do tamanho original, enquanto algoritmos usuais de compressão com perdas chegam facilmente a 10%.

Algo interessante de se notar é que ao utilizarmos de 400 a 500 componentes o arquivo resultante fica maior do que o original. Isso provavelmente é decorrência de pequenos ruídos gerados por não termos  $k = 512$ , o que pode aumentar a entropia da imagem e, assim, fazer com que seja mais difícil de comprimí-la do que a original.

## 5 Conclusões

Vemos que ao utilizarmos mais de 95% das componentes principais encontradas pelo método PCA temos uma diferença praticamente nula entre as imagens (observe os valores de RMSE na Tabela 1 e Figura 5), conforme era esperado, uma vez que a variância explicada pelas últimas componentes é muito pequena.

Ao analisarmos os resultados para diferentes valores de  $k$ , verificamos que as imagens que são comprimidas considerando-se mais de 200 componentes (40% do número total) são praticamente indistinguíveis das originais. Mas, infelizmente, elas economizam apenas de 2% a 4% do espaço ocupado em memória, de modo que o PCA é uma forma extremamente ruim de se fazer compressão quando precisamos de imagens resultantes sem artefatos visuais.

Além disso, mesmo se um pouco de ruído for aceitável (RMSE entre 10.00 e 20.00), o valor de  $\rho$  não fica abaixo de 50%. Ainda, para que a taxa ficasse menor do que 3:1, foi necessário utilizar apenas as primeiras 1 ou 2 componentes principais, mas como vemos nas Figuras 1d, 2d e 3d, esses resultados só seriam interessantes para aplicações muito específicas — porém, outras formas de compressão resultariam em imagens semelhantes, só que ocupando muito menos espaço.

## Referências

- [1] “Principal component analysis — Wikipedia, The Free Encyclopedia.” [Online]. Available: [https://en.wikipedia.org/wiki/Principal\\_component\\_analysis](https://en.wikipedia.org/wiki/Principal_component_analysis)
- [2] “Singular value decomposition — Wikipedia, The Free Encyclopedia.” [Online]. Available: [https://en.wikipedia.org/wiki/Singular\\_value\\_decomposition](https://en.wikipedia.org/wiki/Singular_value_decomposition)
- [3] “JPEG — Wikipedia, The Free Encyclopedia.” [Online]. Available: <https://en.wikipedia.org/wiki/JPEG>
- [4] OpenCV, “Image file reading and writing (cv::IMWRITE\_PNG\_COMPRESSION).” [Online]. Available: [https://docs.opencv.org/master/d4/da8/group\\_imgcodecs.html](https://docs.opencv.org/master/d4/da8/group_imgcodecs.html)

Links acessados em novembro de 2019.