

Trabalho 5 - Análise de Componentes Principais

MC920 - Introdução ao Processamento de Imagem Digital

Victor Ferreira Ferrari
RA 187890
vferrari@mpc.com.br

21 de Novembro de 2019

1 Introdução

Em análise de dados, uma técnica muito utilizada e conhecida para **redução de dimensionalidade** é o PCA: *Principal Component Analysis*, ou Análise de Componentes Principais. Como a ideia da redução de dimensionalidade é reduzir o espaço dos dados enquanto ainda mantendo a maior parte da informação, essa técnica pode se tornar útil na área de **compressão com perdas**.

Assim, podemos utilizar apenas alguns componentes principais de uma imagem para realizar uma redução do espaço em disco, perdendo qualidade no processo, mas ainda mantendo boa parte da informação.

O PCA consiste da projeção dos dados em um subespaço gerado por eixos ortogonais, que podem ser obtidos a partir de uma técnica chamada SVD: *Singular Value Decomposition*. A SVD pode ser feita pela fatoração:

$$A = U\Sigma V^T$$

Se A for uma imagem quadrada, U e V possuem as mesmas dimensões. Se A for uma matriz $n \times p$, U é uma matriz $n \times n$ e V é uma matriz $p \times p$. Σ é uma matriz $n \times p$ diagonal.

Nessa decomposição, U e V^T são matrizes ortogonais, e são compostas de **autovetores**, das matrizes AA^T e A^TA , respectivamente. A matriz Σ é composta da raiz quadrada dos autovalores das matrizes em sua diagonal, *em ordem decrescente*.

Assim, pode-se ver a matriz A como o somatório de termos que consistem da multiplicação entre dois autovetores de AA^T e A^TA e a raiz quadrada do autovalor. Pela ordenação dos autovalores, temos que os primeiros termos desse somatório contribuem mais, ou seja, armazenam mais informação da imagem.

Assim, o PCA se dá pela reconstrução da matriz A com apenas os k componentes principais, ou seja, com apenas parte dos termos do somatório. Isso reconstrói a imagem de modo que requer menos espaço de armazenamento, porém com perdas, sendo um método possível de compressão com perdas.

Em imagens coloridas, a técnica deve ser aplicada em cada banda separadamente e depois a imagem é montada de novo.

O intuito do projeto é implementar a técnica de PCA para compressão com perdas de imagens, com número variável de componentes, avaliando o resultado em compressão e similaridade com a imagem original.

2 Características do Programa

O programa foi feito na linguagem *Python*, com auxílio das bibliotecas externas NumPy, Matplotlib e OpenCV (CV2). Os argumentos são passados na execução via `argv`, permitindo modificar diversas condições de execução. A imagem deve ser colorida.

Os argumentos podem ser:

- `file`: Imagem para aplicar a técnica de PCA;
- `comp`: Quantidade de componentes da imagem de saída.
- `--plot`: Modo "*plot*": fazer um gráfico com os valores de erro para compressão de 1 componente a `comp` componentes.
- `--folder`: opcional, pasta de saída (precisa existir).

3 Implementação

Como visto na seção 1, foi implementada a técnica de PCA com o método SVD para decomposição da matriz original. A leitura da imagem colorida é feita via biblioteca OpenCV, em matrizes NumPy.

Com a imagem lida, é feita a decomposição de cada banda de cor por SVD. Isso é feito com pela função de mesmo nome, `svd`, do NumPy, especialmente o módulo `linalg`, para operações de álgebra linear. A função retorna matriz V já transposta.

Então, as matrizes completas U , V e S são feitas a partir da junção entre as obtidas para cada banda de cor. A partir disso, são escolhidos apenas os k primeiros componentes de cada uma, para depois ser feita a junção em uma imagem de saída por meio do produto de Hadamard entre U e S , e depois a multiplicação matricial com V . O produto de Hadamard é utilizado pois a matriz S não é diagonal.

É importante notar que todos esses processos são feitos com as matrizes em formato de ponto flutuante, então para mostrar o resultado na tela é necessário arredondamento.

A avaliação é feita a partir de duas métricas, visando avaliar a compressão e a similaridade em relação à imagem original. Para a primeira finalidade, foi utilizado o **fator de compressão**, que é a razão entre o espaço de armazenamento da imagem comprimida e o espaço de armazenamento da imagem original. Para a segunda finalidade, foi utilizado o **erro médio quadrático**, dado por:

$$RMSE = \sqrt{\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f(x, y) - g(x, y)]^2}$$

É desejado que o RMSE seja o menor possível, e que o fator de compressão, dado por ρ , seja o mais próximo possível de 1.

Para conseguir o espaço de armazenamento das imagens, elas são salvas no diretório recebido. São salvas tanto a imagem comprimida quanto a original, pelo mesmo método, para coerência no resultado.

Ainda foi implementado um modo chamado de *plot mode*, no qual em vez de realizar o processo descrito acima para o número de componentes desejado, o processo é feito para todo k entre 0 e o número desejado, e um gráfico é feito com o RMSE para cada resultado.

Esse gráfico normalmente é feito com a variância para cada conjunto de n componentes, mas por uma escolha de projeto foi feito utilizando o RMSE, que forma uma curva com informações

similares. Essa curva pode ser usada para definir o número de componentes a ser utilizado na compressão. Nesse modo, a única imagem salva é o gráfico.

As imagens comprimidas são salvas na pasta "Outputs/" (caso outra pasta não tenha sido passada como argumento na chamada).

4 Resultados e Comparaçāo

Foram obtidos resultados para 4 imagens coloridas. As imagens de entrada estão disponíveis em https://www.ic.unicamp.br/~helio/imagens_coloridas/. As imagens comprimidas e gráficos estão na pasta "Outputs/". As imagens originais podem ser vistas na figura 1, em ordem de tamanho.

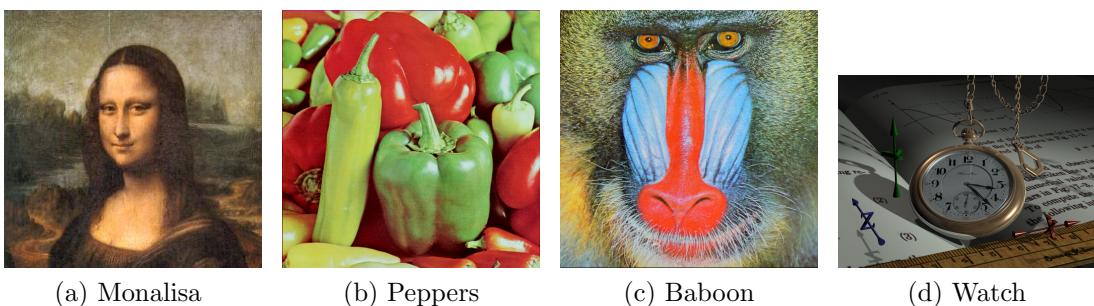


Figura 1: Imagens para esconder textos.

4.1 Testes Iniciais

A primeira imagem analisada será a Figura 1b. Os testes iniciais tiveram a finalidade de verificar se o programa realiza o esperado: comprimir a imagem de acordo com o número de componentes desejado. Assim, o programa foi executado com diversos valores de componentes, e o resultado pode ser visto na Figura 2. Se o número de componentes desejado for maior que o disponível, todos os disponíveis serão utilizados. Os valores do fator ρ e RMSE estão na tabela 1.

| Número de Componentes | Fator de Compressão (ρ) | RMSE |
|-----------------------|--------------------------------|---------------------|
| 1 | 0,482 | 89,78 |
| 30 | 0,826 | 19,73 |
| 100 | 0,983 | 8,85 |
| 300 | 0,997 | 2,47 |
| 512 | 1,0 | 6×10^{-13} |

Tabela 1: Avaliação da Compressão para `peppers.png`.

Pode-se ver claramente que a qualidade das imagens comprimidas são proporcionais à quantidade de componentes utilizados na compressão. Pela tabela, vemos que o fator de compressão acompanha a qualidade da imagem comprimida, e o erro é inversamente proporcional, como esperado.

A última imagem, de 512 componentes, é a reconstrução total da imagem original, já que é uma imagem de dimensões 512×512 . Isso pode ser comprovado pelo fator de compressão

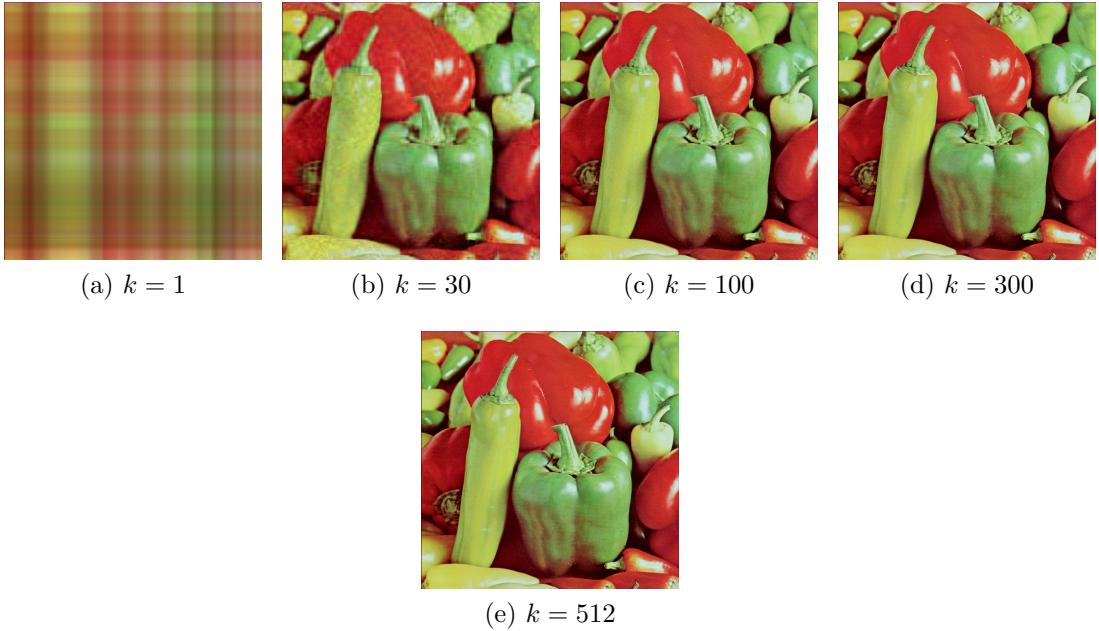


Figura 2: Testes na imagem `peppers.png`.

encontrado, que indica que ambas imagens utilizam a mesma quantidade de espaço de armazenamento, e o RMSE quase nulo. Já a primeira imagem possui apenas pistas de como deve ser a original.

Porém, vê-se que a relação entre número de componentes, erro e fator de compressão não é linear. Por isso, não é fácil descobrir qual a melhor relação para uma boa compressão e boa fidelidade. Para isso, podemos analisar essas relações, especificamente a entre RMSE e número de componentes.

Enfim, podemos ver que o programa se comporta como esperado, sendo bem-sucedida a implementação da técnica PCA.

4.2 Otimizando Compressões

Observando a tabela 1, vemos que a queda de qualidade é bem mais acentuada quanto menor o fator de compressão, mas que a partir de um certo ponto a qualidade perceptível muda muito pouco, pois o erro já está muito baixo. Infelizmente, também são os pontos com menor fator de compressão. Observando as relações entre duas dessas três variáveis, obtemos os gráficos da figura 3.

Nota-se que a queda no erro é muito acentuada no início, mas a partir de um certo ponto o decaimento diminui muito. Então, teoricamente, os valores ótimos se encontram no "cotovelo" da curva, ou um pouco depois dele. Porém, existem dois problemas com essa abordagem:

- A preservação da fidelidade é muito importante em compressão de imagens com perdas, então a relação "custo-benefício" deve ser mais voltada ao "benefício", o que acarreta em mais custo, ou seja, menor compressão.
- A queda do erro nem sempre possui um "cotovelo" muito bem sinalizado, então a melhor relação é mais difícil de ser encontrada, pois a qualidade perceptível sempre melhora significativamente.

Ainda assim, alguns possíveis bons valores, de acordo com o gráfico, para relação "custo-benefício" proporcional, estão na figura 4. Os resultados estão na tabela 2.

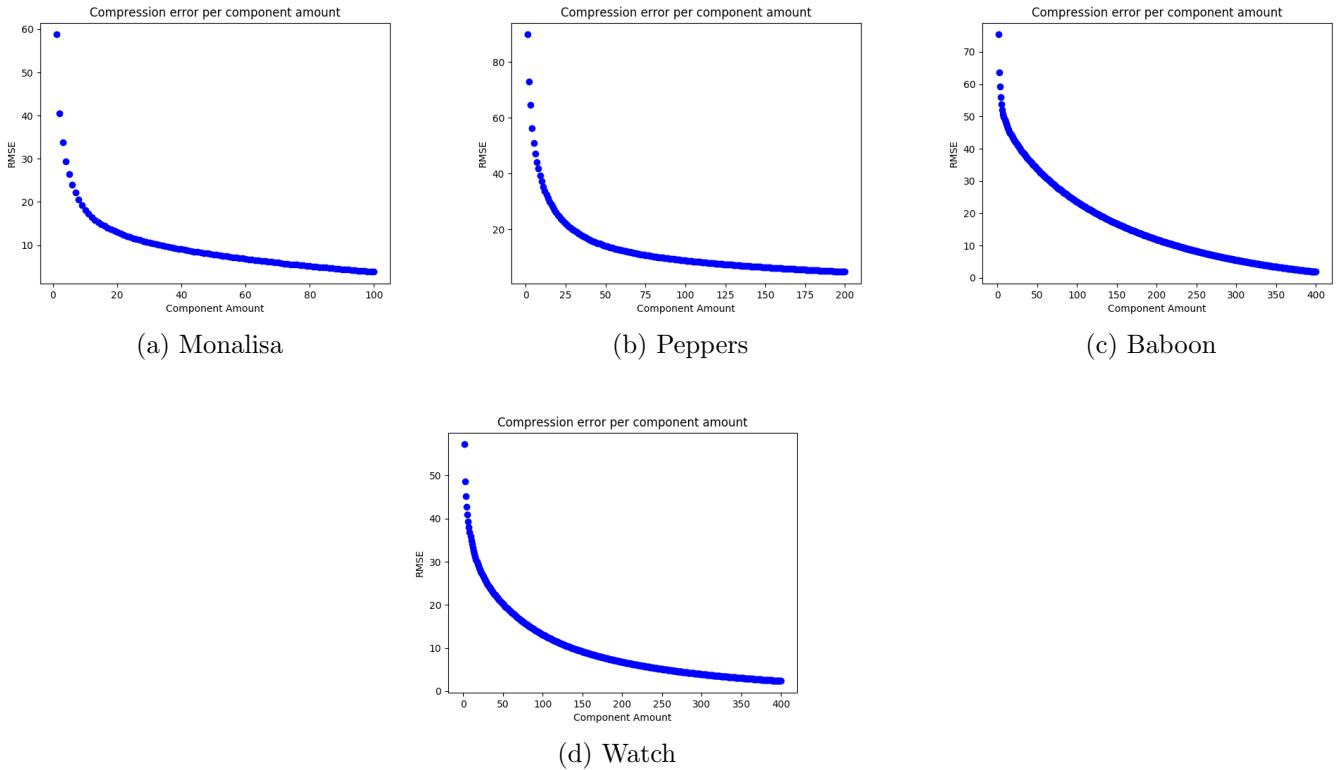


Figura 3: Gráficos das relações entre número de componentes e RMSE para cada imagem.

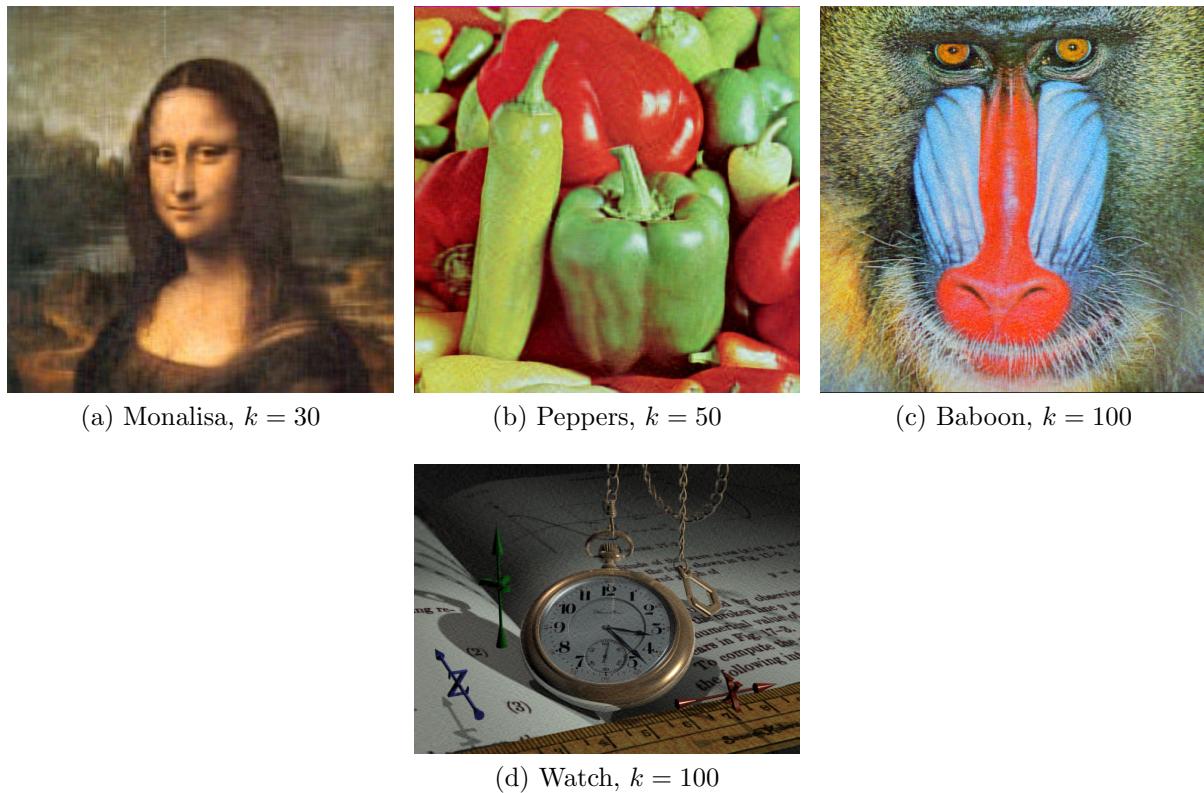


Figura 4: Imagens com compressão referente a um número de componentes próximo ao "ótimo" a partir da relação "custo-benefício".

Pode-se notar que na figura 4d o fator de compressão é maior que 1. Isso será melhor

| Imagen | Fator de Compressão (ρ) | RMSE |
|-----------------------|--------------------------------|-------|
| Monalisa ($k = 30$) | 0,895 | 10,61 |
| Peppers ($k = 50$) | 0,898 | 14,07 |
| Baboon ($k = 100$) | 0,935 | 23,51 |
| Watch ($k = 100$) | 1,545 | 13,12 |

Tabela 2: Avaliação da compressão para todas as figuras com número de componentes próximo ao "ótimo" a partir da relação "custo-benefício".

explorado na seção 4.3.

É perceptível que há uma grande perda na qualidade em muitas dessas imagens, principalmente na 4b e na 4d. Isso se dá pois a relação "custo-benefício" não é ótima para a aplicação em compressão de imagens. A compressão atingida foi baixa, cerca de 10% nas 3 primeiras imagens, mas com muita perda de fidelidade.

Uma imagem particularmente interessante é a 4c, pois embora das 3 primeiras tenha sido a que sofreu menor compressão, com menos de 7%, é a que possui maior erro, por uma grande margem. Ainda assim, é a imagem cuja perda de qualidade é menos perceptível. Isso provavelmente se deve a características intrínsecas da imagem.

Assim, podemos tentar outros valores para, reduzindo ainda mais a compressão, diminuir o erro consideravelmente. Alguns valores podem ser próximos aos vistos na curva, mas outros não. Os valores foram decididos empiricamente, utilizando também da curva. As imagens resultantes estão em 5 e os resultados estão na tabela 3.

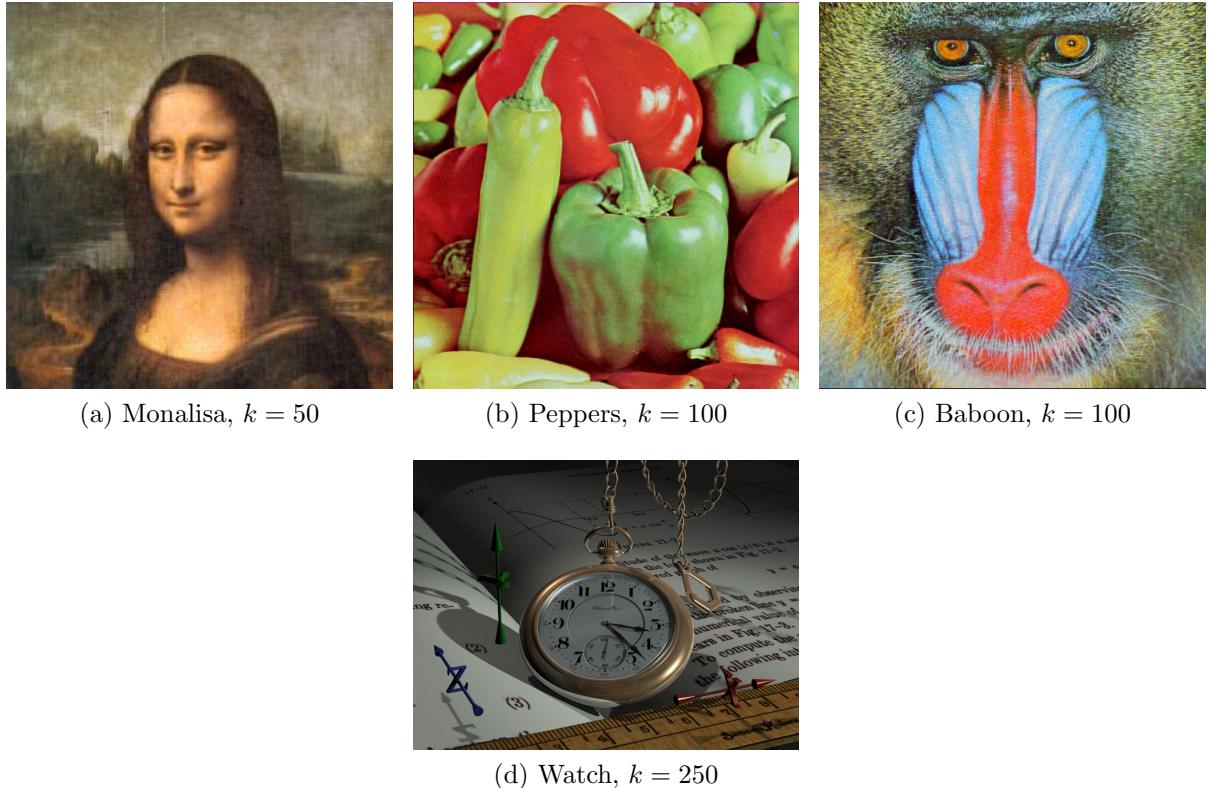


Figura 5: Imagens com compressão referente a um número de componentes escolhido com foco em qualidade.

O mesmo problema do fator de compressão persiste na figura 5d. Será mais explorado na próxima seção.

| Imagen | Fator de Compressão (ρ) | RMSE |
|-----------------------|--------------------------------|-------|
| Monalisa ($k = 50$) | 0,942 | 7,85 |
| Peppers ($k = 100$) | 0,940 | 8,85 |
| Baboon ($k = 100$) | 0,935 | 23,51 |
| Watch ($k = 250$) | 1,569 | 5,13 |

Tabela 3: Avaliação da compressão para todas as figuras com número de componentes escolhido com foco em qualidade.

No geral, houve uma grande melhora na fidelidade das imagens, evidenciado pela queda no erro de todas exceto a figura 5c, abaixo de 10. Porém, isso trouxe uma compressão menor, em torno de 6% para as 3 primeiras, o que pode não ter um grande impacto no armazenamento, e assim expõe a técnica PCA como pouco eficiente para compressão.

A imagem 5c continuou a mesma, pois por maior que seja o erro, a perda de fidelidade é menos perceptível. A imagem com piores resultados, mesmo com erro baixo, foi a 5b, provavelmente pois é uma imagem muito clara, fácil de ver detalhes.

Ainda assim, é fácil ver que foi possível reconstruir a imagem com baixo erro, com 1/4 ou até 1/5 do total de componentes da imagem, o que explicita as altas redundâncias interpíxel e psicovisual.

4.3 Problemas

Nos testes do programa, alguns problemas foram detectados. Como visto nas tabelas 2 e 3, o fator de compressão para a figura 1d foi maior que 1, ou seja, a imagem comprimida ocupa mais espaço em disco que a imagem original. Não foi encontrado motivo para isso acontecer, mas ocorre com outras imagens maiores também. Supõe-se que poderia estar relacionado a não ser quadrada, mas nos testes com a imagem `cookie.jpg` (presente na pasta "Outputs/") não foi evidenciado o mesmo problema (note que a imagem está em outro formato).

Outro problema que pode ser visto, esse em quase toda a execução, está na exibição da imagem comprimida na tela antes de salvar e fazer as avaliações. Nesse caso, foi necessário realizar um arredondamento da imagem de `float` para `uint8`, o que gera diversos efeitos estranhos na exibição da imagem. Esse problema não ocorre com a imagem salva, que não é arredondada *a priori*, e isso é tratado pela biblioteca.

5 Conclusão

Apesar de alguns erros no arredondamento de ponto flutuante a inteiro, que não estão presentes nas imagens salvas, a compressão foi bem-sucedida, com uma queda de qualidade inversamente proporcional ao número de componentes da imagem de saída.

Os gráficos se provaram razoáveis para ajudar a encontrar o melhor número de componentes para cada imagem, com claras diferenças nas curvas entre imagens. Porém, do jeito que foi feita, o tempo de execução para gerar esses gráficos foi elevado, o que não ocorreria se fossem feitos com variância, como usual. Os gráficos ainda não foram perfeitos, e testes empíricos foram necessários para conseguir uma compressão que mantém qualidade.

O principal problema do programa foi a sua instabilidade com relação ao fator de compressão, que ao depender da imagem de entrada e do número de componentes pode ser maior que 1, ou seja, a imagem comprimida ocupa mais espaço em disco que a original, o que vai de encontro ao propósito da implementação. Não foi possível entender o motivo para esse problema.

Um próximo passo para o programa seria implementar o SVD a partir de apenas operações matriciais, sem uso do módulo `linalg` da biblioteca `NumPy`, o que forneceria mais informações que poderiam ser exploradas, porém na versão atual foi escolhido utilizar a função da biblioteca.

Enfim, temos que quando o método cria imagens que ocupam menos espaço de armazenamento que a original, o PCA pode ser um simples método de compressão com perdas, embora tenha compressão significativamente baixa pela perda de informação.

Outros testes estão presentes na pasta "Outputs/", como visto anteriormente.