

Estudo sobre o Teorema da Melhor Aproximação de uma matriz

Presentado por

"Nelson Roldan Condori Colquehuanca"

Disciplina:

Álgebra Linear para Ciencia de Dados

Idea del Trabajo:

Desenvolvemos um exemplo de compressão de imagem, levando em consideração o método de decomposição de imagens SVD, variamos o valor de truncamento r que nos ajuda a entender melhor este método.

Professor: Cássio Machiaveli Oishi

Aproximação de uma matriz pelo Método SVD

Introdução à Decomposição por Valor Singular (SVD)

A Decomposição por Valor Singular, conhecida pela sigla SVD (do inglês, *Singular Value Decomposition*), é uma das fatorações matriciais mais importantes da álgebra linear. Sua relevância se estende por diversas áreas da ciência e engenharia, incluindo processamento de sinais, estatística, e, notavelmente, processamento de imagens. A técnica permite decompor uma matriz em componentes que revelam informações fundamentais sobre sua estrutura, possibilitando, entre outras coisas, a compressão de dados de forma eficiente.

Representação Matricial de Imagens

Para aplicar métodos de álgebra linear a uma imagem digital, primeiro é necessário representá-la em um formato matricial. Uma imagem pode ser vista como uma matriz A de dimensões $m \times n$, onde m e n são, respectivamente, a altura e a largura da imagem em pixels.

- **Imagens em Escala de Cinza:** Para uma imagem em tons de cinza, cada elemento A_{ij} da matriz corresponde a um único valor, geralmente um inteiro de 8 bits (variando de 0 a 255), que representa a intensidade da luz (brilho) no pixel da linha i e coluna j . O valor 0 representa o preto, e 255, o branco.
- **Imagens Coloridas (RGB):** Imagens coloridas são comumente representadas pelo modelo de cores RGB (Red, Green, Blue). Nesse caso, a imagem é tratada como três matrizes separadas, A_R , A_G , e A_B , uma para cada canal de cor. A SVD pode ser aplicada a cada uma dessas matrizes de forma independente.

O Teorema da Decomposição por Valor Singular

Seja $A \in \mathbb{C}^{m \times n}$ uma matriz com posto p , com $U \in \mathbb{C}^{m \times m}$ e $V \in \mathbb{C}^{n \times n}$ matrizes unitárias com colunas ortonormais, tais que

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

onde a matriz $\Sigma \in \mathbb{R}^{m \times n}$ é uma matriz diagonal (ou retangular diagonal) com entradas não negativas na diagonal e zeros fora da diagonal.

$$\Sigma = \begin{bmatrix} \hat{\Sigma} & 0 \\ 0 & 0 \end{bmatrix}$$

onde $\hat{\Sigma} = \text{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{p \times p}$ e $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p > 0$. Os σ_i são chamados de valores singulares de A .

Compressão de Imagem via SVD

A principal vantagem da SVD para a compressão de imagens reside na propriedade de que os primeiros valores singulares ($\sigma_1, \sigma_2, \dots$) correspondem às componentes mais significativas da imagem, ou seja, à sua macroestrutura. Os valores singulares menores estão associados a detalhes mais finos e, muitas vezes, a ruídos. A SVD permite uma **aproximação de posto reduzido** da matriz original, que é a base da compressão.



Fig. 1.1 Imagem de um girassol e sua conversão para escala de cinza.

Como visto acima, podemos carregar uma imagem e representá-la como uma matriz de números inteiros, onde cada número representa o brilho do pixel em sua posição.

A matriz possui 360 valores singulares, com o menor e o maior separados por várias ordens de grandeza. Isso indica que há muitas direções principais que têm um impacto mínimo na imagem, ou seja, nos termos $\sigma_i u_i v_i^T$. Os componentes correspondentes a esses valores provavelmente podem ser eliminados. Vejamos o que acontece quando removemos todos os componentes, exceto o primeiro ($\sigma_1 \approx 37093.40$).



Fig. 1.2 À medida que aumentamos o número de valores singulares utilizados na reconstrução, a imagem torna-se muito mais nítida. É possível ver claramente a imagem original com os primeiros 100 valores singulares. A imagem final demonstra a mudança na taxa de compressão ao variar o valor de r .

Não se vê muito do girassol na primeira imagem da Figura 1.2, mas o brilho parece estar nos pontos corretos. O padrão desta imagem comprimida pareceu interessante e digno de investigação.

Com r igual a 150, já estamos bastante próximos da qualidade da imagem original, mas o tamanho da imagem comprimida é aproximadamente $2/5$ do tamanho original.

Código adaptado da referência [1] com incorporações para a visualização:

```

1 # =====
2 # Pacotes
3 # =====
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from matplotlib.image import imread
7 import os
8 import math
9 #
10 # ESTILO E PALETA DE CORES PARA OS GRÁFICOS
11 #
12 plt.style.use('seaborn-v0_8-whitegrid')
13 colors = {'blue': '#3498db', 'green': '#2ecc71', 'red': '#e74c3c',
14           'purple': '#9b59b6', 'orange': '#f39c12', 'background': '#f7f9fc' }
15 plt.rcParams.update({'font.size': 12})
16 #
17 # Carregamos a imagem
18 #
19 image_path="/content/drive/MyDrive/datasets_gerados_flores/dataset_flores_25/flor_17.jpg"
20 A = imread(image_path)
21 X = np.mean(A, -1) # Converter para escala de cinza
22 fig, axsreal = plt.subplots(1, 2, figsize=(12, 5))
23 fig.suptitle('Imagen Original vs. Escala de Cinza', fontsize=16,
24               fontweight='bold')
25 axsreal[0].imshow(A)
26 axsreal[0].set_title('Imagen Colorida (RGB)')
27 axsreal[0].axis('off')
28 axsreal[1].imshow(X, cmap='gray')
29 axsreal[1].set_title('Imagen em Escala de Cinza')
30 axsreal[1].axis('off')
31 plt.show()
32 #
33 # Aplicação do SVD
34 #
35 U, S_vec, VT = np.linalg.svd(X, full_matrices=False)
36 S = np.diag(S_vec)
37 ranks = [5, 20, 50, 100, 150]
38 frobenius_errors = []
39 relative_errors = []
40 mse_errors = []
41 norm_X = np.linalg.norm(X, 'fro')
42 #
43 # Visualização das Imagens
44 #
45 fig, axs = plt.subplots(1, len(ranks), figsize=(20, 5))
46 fig.suptitle('Imagens Reconstruídas com Diferentes Valores de (r)', fontsize=18,
47               fontweight='bold', y=1.02)
48 for i, r in enumerate(ranks):

```

```

49 Xapprox = U[:, :r] @ S[0:r, :r] @ VT[:r, :]
50 error_rel = np.linalg.norm(X - Xapprox, 'fro') / norm_X
51 frobenius_errors.append(np.linalg.norm(X - Xapprox, 'fro'))
52 relative_errors.append(error_rel)
53 mse_errors.append(np.mean((X - Xapprox)**2))
54 axs[i].imshow(Xapprox, cmap='gray')
55 title_text = (f'r={r}\nErro Relativo: {error_rel:.2%}')
56 axs[i].set_title(title_text, fontsize=14, pad=10)
57 axs[i].axis('off')
58 plt.show()
# =====
# Visualização dos Erros
# =====
59 fig, axs_err = plt.subplots(1, 3, figsize=(20, 6))
60 fig.suptitle('Análise de Erros de Compressão SVD vs. Valores de (r)', fontweight='bold')
61 fig.set_facecolor(colors['background'])
62 # Gráfico de Erro Absoluto
63 axs_err[0].plot(ranks, frobenius_errors, 'o-', color=colors['blue'],
64                  markersize=10, linewidth=2.5, mfc='white', mew=2)
65 axs_err[0].set_title('Erro Absoluto (Norma de Frobenius)', fontsize=14,
66                      fontweight='bold')
67 axs_err[0].set_xlabel('Valores de (r)', fontsize=12)
68 axs_err[0].set_ylabel('Magnitude do Erro', fontsize=12)
69 # Gráfico de Erro Relativo
70 axs_err[1].plot(ranks, relative_errors, 'o-', color=colors['green'],
71                  markersize=10, linewidth=2.5, mfc='white', mew=2)
72 axs_err[1].set_title('Erro Relativo', fontsize=14, fontweight='bold')
73 axs_err[1].set_xlabel('Valores de (r)', fontsize=12)
74 axs_err[1].set_ylabel('Percentual de Erro (%)', fontsize=12)
75 axs_err[1].yaxis.set_major_formatter(plt.FuncFormatter(lambda y, _: f'{y:.1%}'))
76 # Gráfico de MSE
77 axs_err[2].plot(ranks, mse_errors, 'o-', color=colors['red'], markersize=10,
78                  linewidth=2.5, mfc='white', mew=2)
79 axs_err[2].set_title('Erro Quadrático Médio (MSE)', fontsize=14, fontweight='bold')
80 axs_err[2].set_xlabel('Valores de (r)', fontsize=12)
81 axs_err[2].set_ylabel('Valor do MSE', fontsize=12)
82 plt.tight_layout(rect=[0, 0.03, 1, 0.95])
83 plt.show()
# =====
# Análise dos Valores Singulares
# =====
84 fig, axs_sin = plt.subplots(1, 2, figsize=(14, 6))
85 fig.suptitle('Análise dos Valores Singulares', fontsize=18, fontweight='bold')
86 fig.set_facecolor(colors['background'])
87 # Gráfico da magnitude dos valores singulares (escala logarítmica)
88 axs_sin[0].semilogy(S_vec, color=colors['purple'], linewidth=2.5)
89 axs_sin[0].set_title('Decaimento dos Valores Singulares', fontsize=14,
90                      fontweight='bold')
91 axs_sin[0].set_xlabel('Índice do Valor Singular', fontsize=12)
92 axs_sin[0].set_ylabel('Magnitude (Escala Log)', fontsize=12)
93 axs_sin[0].set_facecolor(colors['background'])
94 # Gráfico de energia acumulada
95 cumulative_energy = np.cumsum(S_vec) / np.sum(S_vec)
96 axs_sin[1].plot(cumulative_energy, color=colors['orange'], linewidth=3)
97

```

```

104     axs_sin[1].set_title('Energia Acumulada por Valores Singulares', fontsize=14,
105                           fontweight='bold')
106     axs_sin[1].set_xlabel('Número de Valores Singulares', fontsize=12)
107     axs_sin[1].set_ylabel('Fração de Energia Total', fontsize=12)
108     axs_sin[1].set_facecolor(colors['background'])
109     # Marcar quantos valores são necessários para 90% e 95% da energia
110     for percentage in [0.90, 0.95, 0.99]:
111         idx = np.where(cumulative_energy >= percentage)[0][0]
112         axs_sin[1].axhline(y=percentage, linestyle='--', color='gray', alpha=0.8)
113         axs_sin[1].axvline(x=idx, linestyle='--', color='gray', alpha=0.8)
114         axs_sin[1].text(idx+5, percentage-0.036, f'{percentage:.0%} de energia (r={idx})',
115                         fontsize=10, color='black')
116     plt.tight_layout(rect=[0, 0.03, 1, 0.95])
117     plt.show()

```

Como se observa no código anterior, a imagem reconstruída é obtida a partir da multiplicação de matrizes. A partir da fórmula da SVD, simplesmente adicionamos mais matrizes deste tipo (de posto 1) para obter uma imagem mais próxima da original.

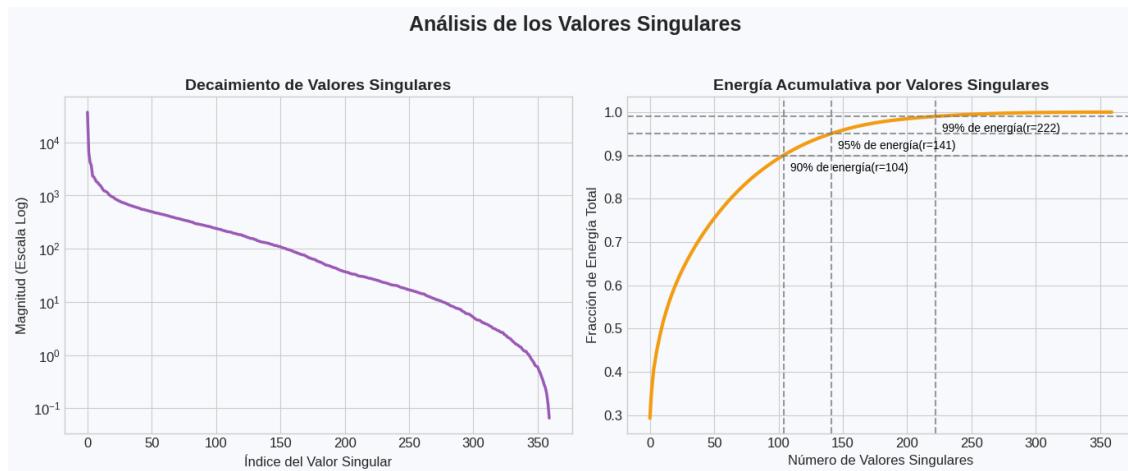


Fig. 1.3 Análise do Decaimento de Valores Singulares e Energia Acumulada.

Algumas questões importantes são: para que tipo de imagens essa compressão é útil? Ou que parâmetro ‘r’ escolher ao comprimí-las? Analisar os valores singulares, como mostrado anteriormente, nos permite determinar a importância de cada um. Dado que os valores singulares apresentam uma diminuição significativa após os primeiros, podemos comprimir a imagem consideravelmente (eliminando os dados associados aos valores singulares menores). Se quiséssemos criar um formato de arquivo e um sistema de armazenamento baseados neste algoritmo, poderíamos escolher um limiar para a magnitude mínima do valor singular. Isso nos permitiria ter um critério consistente para matrizes de baixa informação entre todas as imagens que armazenarmos.

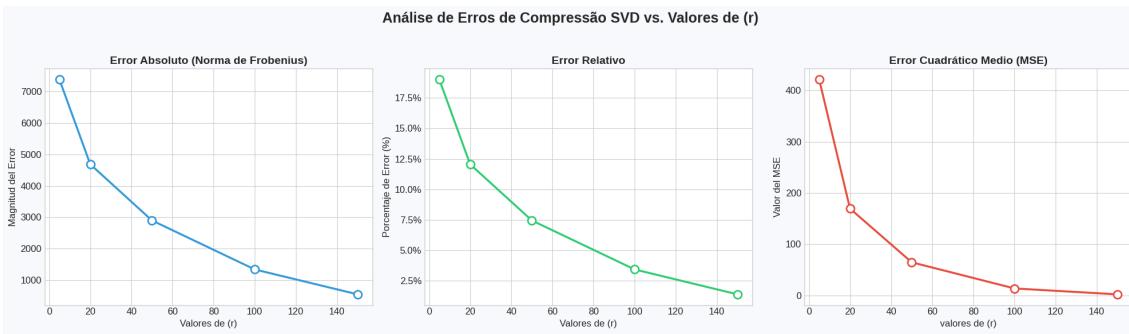


Fig. 1.4 Relação entre o número de valores singulares r utilizados na compressão SVD e o erro de reconstrução resultante.

Na Figura 1.4, o erro absoluto entre a imagem original e a comprimida é muito alto quando se utiliza um número baixo de valores singulares (r). À medida que ' r ' aumenta, o erro diminui drasticamente, indicando que a imagem reconstruída está se tornando progressivamente mais fiel à original.

De forma semelhante, o erro relativo — expresso em porcentagem — começa alto (próximo de 20%) para valores pequenos de ' r ' e cai rapidamente. Isso demonstra que a proporção do erro em relação à "energia" total da imagem original diminui à medida que mais informações (valores singulares) são incluídas na reconstrução.

O Erro Quadrático Médio (MSE), que mede a média dos quadrados das diferenças de pixels, também exibe um decaimento acentuado. Um MSE alto com ' r ' baixo significa grandes diferenças de brilho pixel a pixel, enquanto um MSE baixo com ' r ' alto indica que a imagem comprimida é visualmente muito mais próxima da original.

```

1 def gerar_e_salvar_dataset(num_imagens, caminho_base):
2     caminho_salvar = os.path.join(caminho_base, f'dataset_flores_{num_imagens}')
3     os.makedirs(caminho_salvar, exist_ok=True)
4     print(f"Pasta de destino configurada em: {caminho_salvar}")
5     dataset, info = tfds.load(
6         'tf_flowers',
7         split='train',
8         with_info=True,
9         shuffle_files=True
10    )
11    dataset_subset = dataset.take(num_imagens)
12    for i, exemplo in enumerate(dataset_subset):
13        imagem_tensor = exemplo['image']
14        imagem_np = np.array(imagem_tensor)
15        imagem_redimensionada = cv2.resize(imagem_np, (360, 360),
16                                            interpolation=cv2.INTER_AREA)
17        imagem_bgr = cv2.cvtColor(imagem_redimensionada, cv2.COLOR_RGB2BGR)
18        nome_arquivo = f'flor_{i+1}.jpg'
19        caminho_completo = os.path.join(caminho_salvar, nome_arquivo)
20        cv2.imwrite(caminho_completo, imagem_bgr)
21 if __name__ == "__main__":
22     drive.mount('/content/drive')
23     pasta_base = '/content/drive/MyDrive/datasets_gerados_flores'
24     tamanhos_datasets = [25, 30, 50]
25     for tamanho in tamanhos_datasets:
26         gerar_e_salvar_dataset(num_imagens=tamanho, caminho_base=pasta_base)

```

O código acima define uma função para gerar imagens, automatizando a criação de datasets de flores com um número específico de imagens e dimensões fixas.

A função primeiro cria uma pasta de destino, nomeando-a de acordo com o número de imagens a serem geradas. Em seguida, utiliza a biblioteca **tensorflow_datasets** para carregar o conhecido conjunto de dados, selecionando aleatoriamente o número de imagens solicitado.

Posteriormente, o código itera sobre cada imagem selecionada. Para cada uma, realiza três transformações principais utilizando a biblioteca **OpenCV**: a imagem é convertida para um array **NumPy**, redimensionada para um tamanho quadrado de 360x360 pixels e seu formato de cor é ajustado de RGB para BGR, que é o padrão utilizado pelo **OpenCV** para salvamento. Finalmente, cada imagem processada é salva como um arquivo JPG na pasta de destino.

O bloco principal define uma pasta base dentro do Google Drive chamada **datasets_gerados_flores** e especifica a criação de três datasets distintos, com 25, 30 e 50 imagens cada.

Na Figura 1.5, as duas imagens mostram uma comparação lado a lado de 25 imagens de flores. Cada linha exibe a imagem original, seguida por suas reconstruções usando um número crescente de valores singulares ($r = 5$, $r = 20$ e $r = 100$). Fica evidente que com $r = 5$, apenas a estrutura mais básica da imagem é capturada, resultando em uma imagem muito borrada e com um erro relativo alto. Ao aumentar para $r = 20$ e $r = 100$, a qualidade da imagem melhora drasticamente, tornando-se muito próxima da original, enquanto o erro relativo diminui significativamente.

A Figura 1.6 apresenta uma análise estatística dos valores singulares de todo o conjunto de imagens. A magnitude dos valores singulares diminui exponencialmente; os primeiros são muito grandes, enquanto os subsequentes se tornam rapidamente insignificantes. Isso confirma por que os primeiros r valores contêm a maior parte da informação visual.

O gráfico também mostra a porcentagem da informação total da imagem que é capturada à medida que mais valores singulares são usados. A curva de "Média" mostra que, em média, cerca de 90% da "energia" da imagem é reconstruída com apenas 97 valores singulares, e 95% é alcançado com $r = 143$. Isso justifica matematicamente por que é possível descartar a maioria dos valores singulares e ainda assim obter uma reconstrução de alta fidelidade.



Fig. 1.5 Comparação visual da compressão SVD para um dataset de 25 imagens, com ranks $r=5$, $r=20$ e $r=100$.

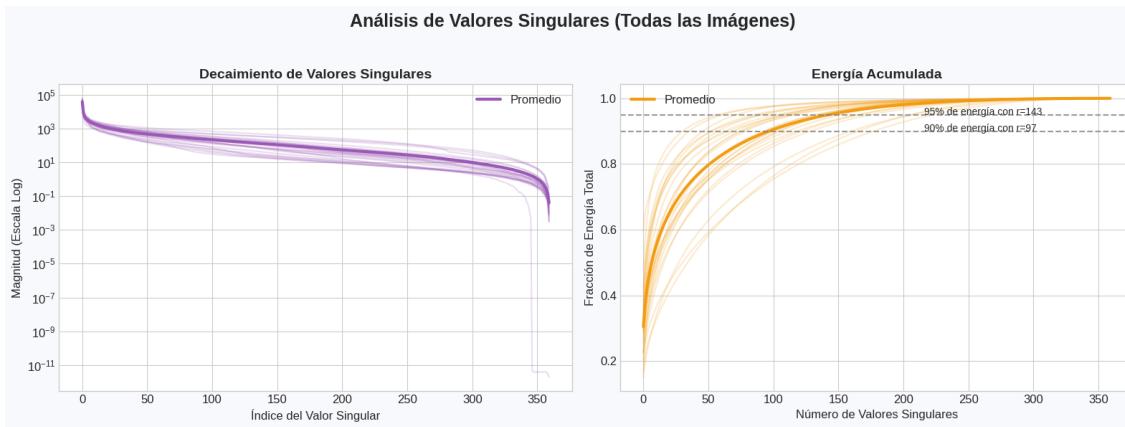


Fig. 1.6 Análise do Decaimento de Valores Singulares e Energia Acumulada para o conjunto de 25 imagens.

A seguir, mostramos o código utilizado para analisar os *datasets*. A finalidade é compreender a aplicação da Decomposição de Valor Singular. Este código pode ser reutilizado para os outros *datasets* (de 30 e 50 imagens) simplesmente alterando a pasta de onde as imagens são carregadas.

```

1 # -----
2 # ESTILO E PALETA DE CORES
3 # -----
4 plt.style.use('seaborn-v0_8-whitegrid')
5 colors = {
6     'blue': '#3498db', 'green': '#2ecc71', 'red': '#e74c3c',
7     'purple': '#9b59b6', 'orange': '#f39c12', 'background': '#f7f9fc'
8 }
9 plt.rcParams.update({'font.size': 12, 'figure.facecolor': colors['background']})
10 # -----
11 # CARREGAMENTO DE IMAGENS
12 # -----
13 dataset_path = '/content/drive/MyDrive/datasets_gerados_flores/dataset_flores_25'
14 image_files = sorted([f for f in os.listdir(dataset_path) if f.lower().endswith('.jpg')])
15 images = [imread(os.path.join(dataset_path, f)) for f in image_files]
16
17 # -----
18 # APLICAÇÃO DO SVD E CÁLCULO DE ERROS
19 # -----
20 if images:
21     ranks = [5, 20, 100]
22     approximations = {r: [] for r in ranks}
23     errors = {r: [] for r in ranks}
24     for img_color in images:
25         img_gray = np.mean(img_color, axis=-1)
26         norm_gray = np.linalg.norm(img_gray, 'fro')
27         U, S_vec, VT = np.linalg.svd(img_gray, full_matrices=False)
28         S = np.diag(S_vec)
29         for r in ranks:
30             Xapprox = U[:, :r] @ S[0:r, :] @ VT[:r, :]
31             approximations[r].append(Xapprox)
32             error_rel = np.linalg.norm(img_gray - Xapprox, 'fro') / norm_gray
33             mse = np.mean((img_gray - Xapprox)**2)
```

```

34         errors[r].append({'rel': error_rel, 'mse': mse})
35 # =====
36 # 4. VISUALIZAÇÃO COMPARATIVA EM DUAS PARTES
37 # =====
38 if images:
39     num_filas_total = len(images)
40     ponto_divisao = (num_filas_total + 1) // 2
41     num_cols = 1 + len(ranks)
42     figsize1 = (num_cols * 1.5, ponto_divisao * 1.7)
43     fig1, axes1 = plt.subplots(ponto_divisao, num_cols, figsize=figsize1)
44     # Bloco 1 de imagens
45     for i in range(ponto_divisao):
46         axes1[i,0].imshow(np.mean(images[i], axis=-1), cmap='gray')
47         axes1[i,0].set_title('Original', fontsize=10, fontweight='bold')
48         axes1[i,0].set_ylabel(f'Imagem {i+1}', rotation=0, fontsize=10, labelpad=40
49                               , va='center')
50         for j, r in enumerate(ranks):
51             ax_approx = axes1[i, j + 1]
52             approx_img = approximations[r][i]
53             error_info = errors[r][i]
54             ax_approx.imshow(approx_img, cmap='gray')
55             ax_approx.set_title(f'r={r} Erro:{error_info["rel"]:.1%}', fontsize=10)
56     for ax in axes1.flat:
57         ax.set_xticks([]); ax.set_yticks([])
58     plt.tight_layout(rect=[0, 0, 1, 0.96])
59     plt.show()
60     # Bloco 2 de imagens
61     num_filas_fig2 = num_filas_total - ponto_divisao
62     if num_filas_fig2 > 0:
63         figsize2 = (num_cols * 1.5, num_filas_fig2 * 1.7)
64         fig2, axes2 = plt.subplots(num_filas_fig2, num_cols, figsize=figsize2, squeeze=False)
65         for i in range(ponto_divisao, num_filas_total):
66             idx_fig2 = i - ponto_divisao
67             axes2[idx_fig2, 0].imshow(np.mean(images[i], axis=-1), cmap='gray')
68             axes2[idx_fig2, 0].set_title('Original', fontsize=10, fontweight='bold')
69             axes2[idx_fig2, 0].set_ylabel(f'Imagem {i+1}', rotation=0, fontsize=10,
70                                         labelpad=40, va='center')
71             for j, r in enumerate(ranks):
72                 ax_approx = axes2[idx_fig2, j + 1]
73                 approx_img = approximations[r][i]
74                 error_info = errors[r][i]
75                 ax_approx.imshow(approx_img, cmap='gray')
76                 ax_approx.set_title(f'r={r} Erro:{error_info["rel"]:.1%}', fontsize=10)
77         for ax in axes2.flat:
78             ax.set_xticks([]); ax.set_yticks([])
79         plt.tight_layout(rect=[0, 0, 1, 0.96])
80         plt.show()
81 # =====
82 # 5. ANÁLISE AGREGADA DE VALORES SINGULARES
83 # =====
84 if images:
85     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
86     fig.suptitle('Análise de Valores Singulares (Todas as Imagens)',
87                  fontsize=18, fontweight='bold')
88     all_s_vecs = []

```

```
89 all_energies = []
90 for img_color in images:
91     img_gray = np.mean(img_color, axis=-1)
92     _, s_vec, _ = np.linalg.svd(img_gray, full_matrices=False)
93     all_s_vecs.append(s_vec)
94     all_energies.append(np.cumsum(s_vec) / np.sum(s_vec))
95     ax1.semilogy(s_vec, color=colors['purple'], alpha=0.2)
96     ax2.plot(all_energies[-1], color=colors['orange'], alpha=0.2)
97 mean_s_vecs = np.mean(all_s_vecs, axis=0)
98 mean_energy = np.mean(all_energies, axis=0)
99 ax1.semilogy(mean_s_vecs, color=colors['purple'], linewidth=3, label='Média')
100 ax2.plot(mean_energy, color=colors['orange'], linewidth=3, label='Média')
101 ax1.set_title('Decaimento dos Valores Singulares', fontsize=14, fontweight='bold')
102 ax1.set_xlabel('Índice do Valor Singular')
103 ax1.set_ylabel('Magnitude (Escala Log)')
104 ax1.legend()
105 ax2.set_title('Energia Acumulada', fontsize=14, fontweight='bold')
106 ax2.set_xlabel('Número de Valores Singulares')
107 ax2.set_ylabel('Fração de Energia Total')
108 ax2.legend()
109 for percentage in [0.90, 0.95]:
110     idx = np.where(mean_energy >= percentage)[0][0]
111     ax2.axhline(y=percentage, linestyle='--', color='gray', alpha=0.8)
112     ax2.text(len(mean_energy)*0.6, percentage,
113             f'{percentage:.0%} de energia com r={idx}', fontsize=10)
114 plt.tight_layout(rect=[0, 0, 1, 0.95])
115 plt.show()
```



Fig. 1.7 Exemplo da análise para 30 imagens.

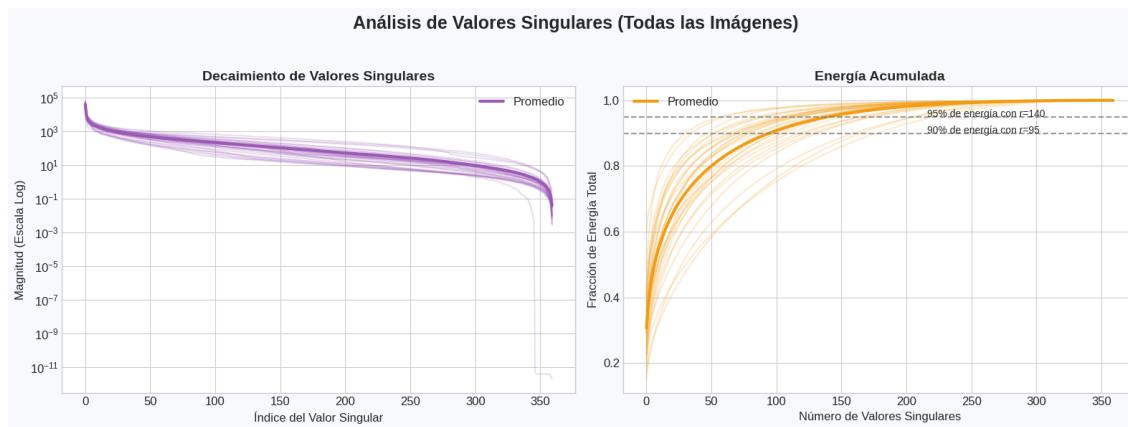


Fig. 1.8 Análise do Decaimento de Valores Singulares e Energia Acumulada para 30 imagens.



Fig. 1.9 Exemplo da análise para 50 imagens.

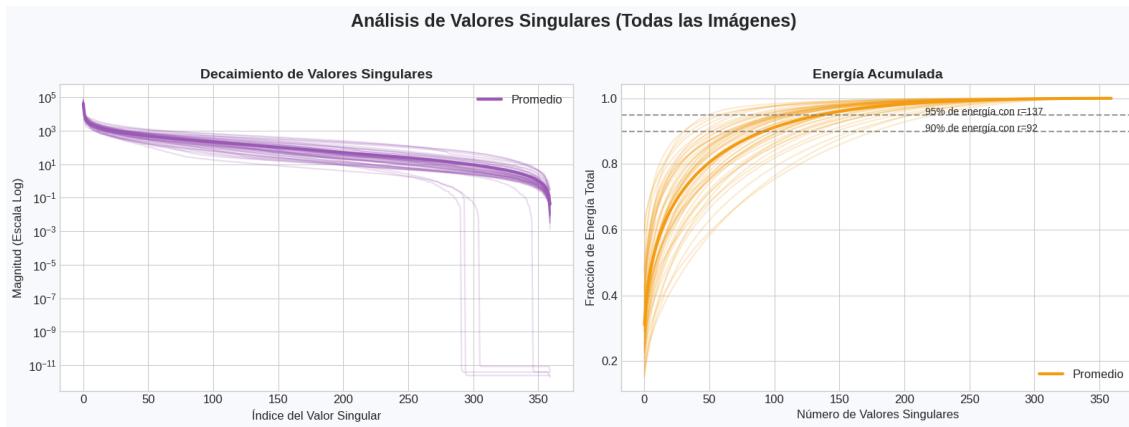


Fig. 1.1 Análise do Decaimento de Valores Singulares e Energia Acumulada para 50 imagens.

Conclusão

A Decomposição por Valor Singular oferece um método robusto e matematicamente elegante para a compressão de imagens. Ao representar uma imagem como uma matriz e decompô-la, a SVD isola e hierarquiza os componentes estruturais da imagem através dos seus valores singulares. Isso permite a criação de uma aproximação de alta fidelidade da imagem original usando uma fração dos dados, descartando informações menos relevantes. A flexibilidade na escolha do número de valores singulares a serem retidos (r) torna a SVD uma ferramenta poderosa para ajustar o balanço entre a qualidade visual e a eficiência do armazenamento.

Referências Bibliográficas

- [1] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2 edition, 2022.