Algoritmo de codificação e decodificação para vídeos brutos de vigilância usando SVD/PCA.

Renan Mendanha [1]

[1]Instituto de Computação – Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brasil

renanma@dcc.ufrj.br

Abstract. *This article is a report that explores the utilization of the SVD/PCA algorithm [1] for building a Python package [2] capable of encoding and decoding unprocessed surveillance video files (RAW). The main objective is to elucidate each step of the encoding and decoding process, as well as present the results of experiments conducted with the application of the algorithm.*

Resumo. *Este artigo é um relatório que explora a utilização do algoritmo SVD/PCA [1] para a construção de um pacote em Python [2] capaz de codificar e decodificar arquivos de vídeo de vigilância não processados (RAW). O objetivo principal é explicitar cada etapa do processo de codificação e decodificação, além de expor os resultados dos experimentos realizados a partir da aplicação do algoritmo.*

1. Introdução

Nos dias atuais, deparamo-nos com enormes volumes de dados sendo gerados constantemente, e os sistemas de vigilância inteligentes não são exceção a esse fato. Portanto, um dos principais desafios reside diretamente na capacidade de armazenar e processar esses dados.

Neste artigo, apresenta-se um relatório que explora a aplicação dos algoritmos SVD/PCA na construção de um pacote em Python que implementa o processo de codificação e decodificação, com o objetivo primordial de comprimir os dados através da separação entre primeiro plano e plano de fundo de um vídeo.

2. Ferramentas

Para a construção do pacote, foi necessária a compreensão do funcionamento de alguns algoritmos, Estes que foram utilizados como ferramenta para atingir o objetivo final esperado pelo codificador.

2.1 SVD/PCA

Como ideia básica, utilizamos o SVD (Singular value decomposition) [1] para obter a informação redundante dos quadros do vídeo na forma de uma matriz de posto baixo. De forma semelhante, o PCA (principal component analysis) [1] busca a melhor estimativa de posto k, que chamamos de L, de uma matriz M.

Neste projeto, foi utilizada a biblioteca “fbpca” [6] criada pelo Facebook que implementa as funções citadas nesta seção de maneira computacional eficiente.

2.2 Robust PCA

Apesar do PCA clássico já ser uma ferramenta bastante sofisticada, ele pode apresentar dificuldades ao tentar estimar a matriz L a partir de uma matriz M que apresenta muito ruído. Para lidar com esse problema, utiliza-se o "Robust PCA" [3], o qual fatora a matriz M na soma de duas matrizes (L + S), das quais L é a matriz de posto baixo e S é uma matriz esparsa.

Para a aplicação abordada neste relatório, as matrizes L, S e M possuem significados bem definidos, sendo a matriz M o vídeo não processado, a matriz L equivalente ao plano de fundo do vídeo, pois é a informação dos frames redundantes e a matriz S representando o primeiro plano do vídeo, que contém somente as informações de eventuais pessoas, animais ou carros que estão transitando sobre uma determinada área de observação.

2.3 Robust PCA como um problema de otimização

O "Robust PCA" [3] pode ser formulado e interpretado como um problema de otimização, sendo equivalente à:

$$minimize\ ||L||_* + \lambda||S||_1 ;$$

Sujeito a: $L + S = M$.

Onde $||L||_*$ é a norma nuclear de L e $||S||_1$ a norma L1 de S. Ou seja, descrevemos a matriz esparsa minimizando sua norma L1 e a matriz de posto baixo minimizando a norma nuclear.

2.4 Primary Component Pursuit

O algoritmo "Primary Component Pursuit" (PCP) [4] nos permite resolver a otimização do Robust PCA, desta maneira, obtendo ao mesmo tempo e de forma conveniente as fatorações L e S da matriz M.

Dentre as possíveis implementações do algoritmo PCP, estaremos utilizando uma que nos dá o resultado através do método "Inexact Augmented Lagrange Multiplier" (IALM) [5]. Existem diversos outros métodos que possibilitam a implementação do PCP, porém, foi observado que o método IALM possui melhor desempenho dentre os que foram citados no paper indicado pela referência número 5.

3. Metodologia

Nesta seção, abordaremos passo a passo a metodologia aplicada para a codificação e decodificação dos dados. Podemos representar de maneira bem definida os processos pelos quais o vídeo de entrada percorre, seguindo um fluxo que é observado nas figuras 1 e 2.

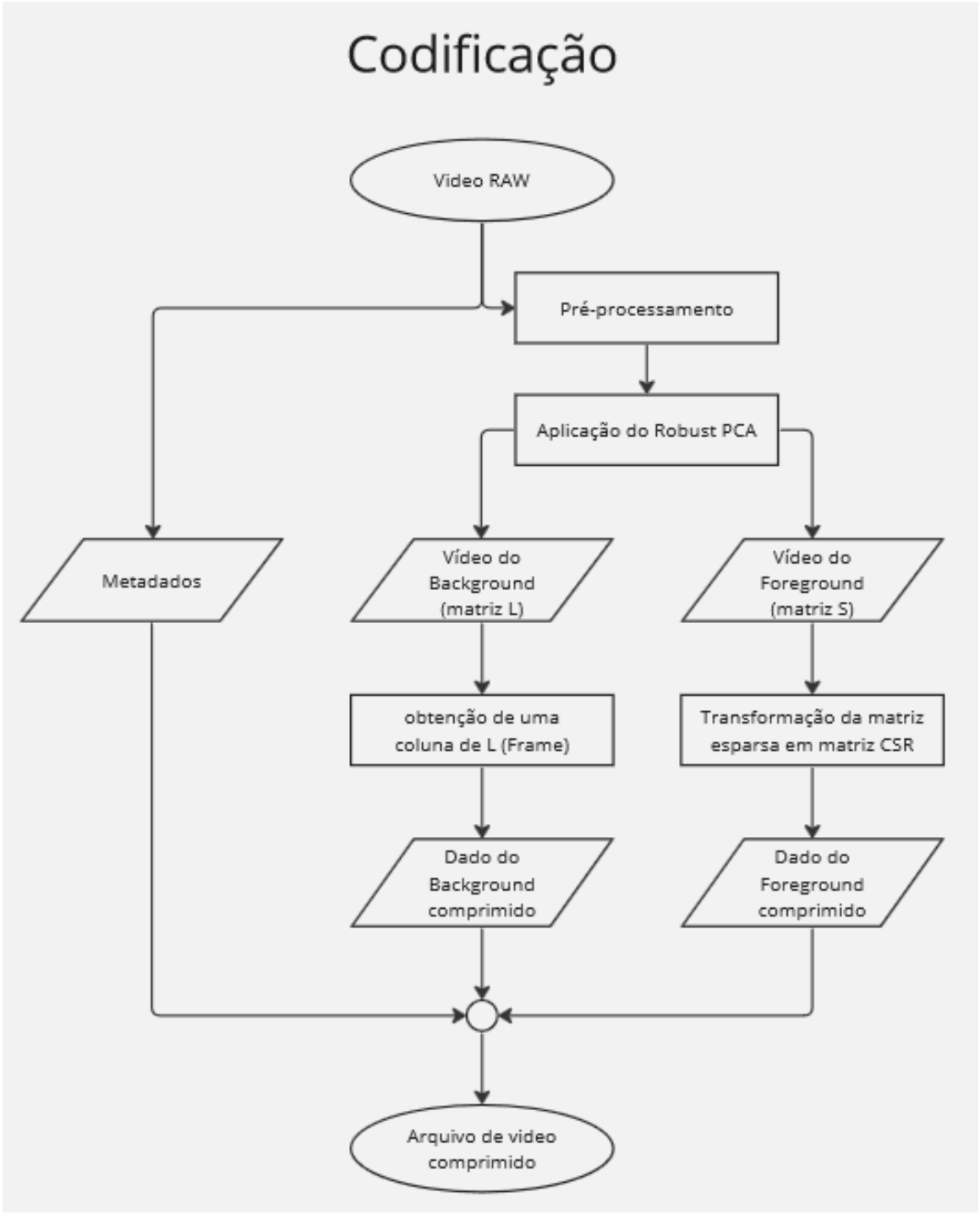

Figura 1. Fluxograma do Codificador

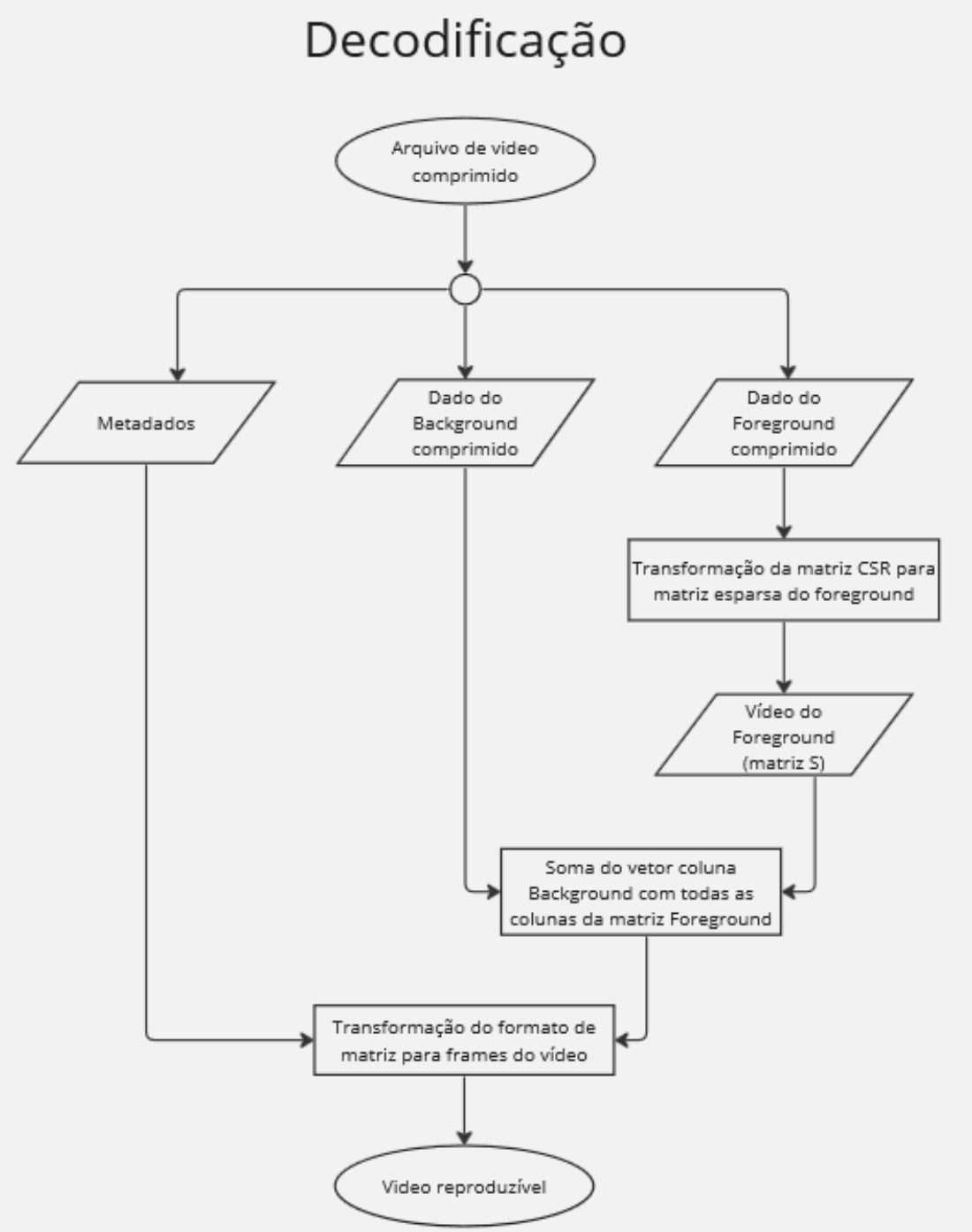

Figura 2. Fluxograma do Decodificador

A partir desses fluxogramas, podemos detalhar cada passo dos processos de codificação e decodificação.

3.1 Codificação

A partir do vídeo RAW, inicialmente extraímos seus metadados para utilização em um momento posterior, aplicando também um pré-processamento sobre este vídeo. Tendo o vídeo

pré-processado, executamos o Robust PCA, o que gera as matrizes L e S. Com as matrizes geradas, seguimos para os passos de compressão das mesmas, assim produzindo os arquivos comprimidos que, juntos aos metadados, são aglomerados em uma estrutura e guardados em um só arquivo que é salvo no disco.

Podemos aprofundar a explicação sobre os passos de maior complexidade nas subseções 3.1.1, 3.1.2 e 3.1.3:

3.1.1 Pré-processamento

O passo de pré-processamento consiste inicialmente em transformar o vídeo de entrada em escala de cinza, o que facilita a manipulação dos dados. Com o vídeo transformado para essa escala, extraímos todos os quadros do vídeo e, para cada quadro, conseguimos armazená-los na forma de matriz $I_{n,m}$, os quais n e m descrevem altura e largura, respectivamente, da dimensão do vídeo (obtidas através dos metadados).

A partir das matrizes dos quadros, redimensionamos as imagens $I_{n,m}$ para um vetor coluna $V_{n \bullet m,1}$, de forma que consigamos representar toda a informação de um quadro em um único vetor coluna. Supondo que o vídeo possua k quadros, teremos k vetores V, o que torna possível empilhá-los ao longo das colunas de uma matriz M, que consequentemente possui dimensão $M_{n \bullet m,k}$. Isso nos permite representar completamente os dados do vídeo em uma matriz bidimensional, tornando possível observar tanto a redundância quanto a diferenciação dos quadros entre si.

3.1.2 Aplicação do Robust PCA

Após o pré-processamento, com a obtenção da matriz $M_{n \bullet m,k}$, conseguimos aplicar diretamente o Robust PCA, efetuando o processo que foi explicitado na seção 2. O produto obtido após esta etapa do processo são duas matrizes:

- $L_{n \bullet m,k}$ Uma matriz de baixo posto que representa a informação redundante dos quadros do vídeo, ou seja, o plano de fundo que se mantém estático.
- $S_{n \bullet m,k}$ Uma matriz esparsa que representa a informação tratada como ruído no vídeo, ou seja, o primeiro plano que contém variações pontuais em relação ao plano de fundo, representando o movimento de pessoas, animais, carros, etc.

3.1.3 Compressão das matrizes

Com o intuito de reduzir o tamanho ocupado pelos dados no disco, realizamos a compressão dos dados das duas matrizes obtidas a partir da aplicação do Robust PCA.

Para a matriz L do plano de fundo, selecionamos simplesmente um quadro específico (uma coluna da matriz) pois ela contém informações redundantes em todas as suas colunas, já que o plano de fundo é estático e o "ruído" foi removido.

Com a matriz esparsa S, a fim de guardar somente as informações relevantes, devemos separar somente suas células não nulas. Este objetivo pode ser atingido ao utilizar a estrutura

matricial CSR [7] [8], produzindo resultados que reduzem consideravelmente o tamanho final dos dados à medida que aumenta-se a esparsidade de S. Estes resultados serão abordados na seção 4.

3.2 Decodificação

Com o vídeo codificado, ao iniciarmos a decodificação, obtemos inicialmente os metadados e os dados do plano de fundo e primeiro plano comprimidos. Computamos em seguida a descompressão dos dados, com a transformação do foreground de matriz CSR para matriz esparsa.

Com a matriz esparsa S, realizamos a soma de todas as colunas de S com o vetor coluna do dado comprimido do plano de fundo, ou seja, somamos todos os frames do primeiro plano à imagem do plano de fundo. Com isso, é possível transformar a matriz gerada para o formato de frames de um vídeo, tornando possível sua reprodução. Essa transformação é explicada na seção 3.2.1:

3.2.1 Transformação de matriz para vídeo

A matriz representa em suas colunas os quadros do vídeo, com isso, para gerarmos um clipe reproduzível, precisamos extrair cada coluna dessa matriz e transformá-las em imagens de dimensão indicada pelos metadados, ou seja, estaremos redimensionando um vetor coluna $V_{n \bullet m,\, 1}$ para uma imagem $I_{n,\, m}$.

Além disso, precisa-se ordenar essa sequência de imagens, indicando o número de quadros por segundo, informação essa que também é retirada dos metadados. Com isso, temos a sequência de quadros sendo reproduzida corretamente como vídeo.

3.2.2 Alternativas de composição

De maneira alternativa ao processo listado anteriormente, onde se usa ambos os planos para a composição final do vídeo decodificado, podemos optativamente indicar somente o plano de fundo, ignorando o passo de soma de matrizes e utilizando somente a matriz L de baixo posto para composição. Analogamente, podemos indicar o primeiro plano e obter o vídeo de saída somente com a matriz esparsa S.

4. Experimentos e Resultados

Foram realizados experimentos com dois vídeos diferentes, um deles consiste em um círculo se movimentando por um fundo preto (que chamamos de Ball Bounce Test) e o outro exemplo ilustra a aplicação real sobre um vídeo retirado de uma das câmeras de vigilância da prefeitura do Rio de Janeiro (nomeado de CCTV Test), na qual podemos observar o movimento dos carros, da água e a influência de ruídos intrínsecos da câmera de vigilância. As características numéricas de cada teste são:

- Ball Bounce Test: Como já foi introduzido, temos um simples teste, consistindo de um vídeo de 30 segundos com 144 pixels de altura e 256 de largura, a taxa de reprodução é de 30 quadros por segundo, formando um total de 900 quadros.

- CCTV Test: Neste teste simulando um caso real, temos um vídeo de 6 segundos com 424 pixels de altura e 600 de largura, a taxa de reprodução é de 30 quadros por segundo, formando um total de 180 quadros.

Podemos observar os resultados relacionados ao tamanho do dado, comparando o vídeo não processado (RAW) com o vídeo codificado, na Tabela 1. A esparsidade se refere à matriz S, gerada após a aplicação do Robust PCA e a taxa de redução indica a porcentagem que consegue-se reduzir do tamanho total do arquivo RAW ao codificá-lo com o processo em pauta neste relatório. Todos os tamanhos dos dados medidos e apresentados na unidade Kibibyte (KiB).

Métricas \ Teste	Ball Bounce Test	CCTV Test
Tamanho do vídeo RAW	259200,000 KiB	357750,000 KiB
Esparsidade de S	99,357%	99,695%
Tamanho do vídeo Codificado	1465,416 KiB	2310,144 KiB
Taxa de redução	99.435%	99.354%

Tabela 1. Resultados RAW x Codificado (valores aproximados com 3 casas decimais)

Além dos testes numéricos, podemos verificar a qualidade da codificação e compressão visualmente, foram retirados 2 quadros do vídeo utilizado no CCTV Test com a intenção de comparar a qualidade entre o vídeo original e o processado. Os resultados podem ser observados na Tabela 2.

Composição\Quadro	Quadro A	Quadro B
Primeiro plano		
Plano de fundo		
Ambos os planos		
RAW		

Tabela 2. Resultados da qualidade visual

5. Conclusão

Neste relatório, é exposto o pacote em Python que implementa o processo de codificação e decodificação de vídeos de câmeras de vigilância através da aplicação dos algoritmos SVD/Robust PCA. Pode-se observar altos ganhos em relação à habilidade de redução do espaço utilizado para armazenamento, tendo todo esse ganho sem comprometer a qualidade da mídia.

Vale ressaltar que, apesar dos resultados obtidos serem bastante satisfatórios em relação ao arquivo não processado, existem codificadores extremamente avançados, modernos e amplamente utilizados que abordam outros métodos, conseguindo resultados ainda melhores. Exemplos de codificadores amplamente utilizados que possuem alta eficiência são o H.264 e H.265.

Pode-se salientar também que existem diversos pontos que podem ser melhorados no algoritmo, o principal deles está relacionado ao desempenho computacional, já que o código está sendo executado de maneira sequencial, não se beneficiando do fato que processamento de vídeo é uma tarefa altamente paralelizável, tendo em vista GPUs com seus números de unidades de processamento (“cores”) cada vez mais altos.

Os arquivos de código que contêm exemplos de utilização da biblioteca, assim como todos os materiais utilizados para experimentos, estão disponíveis no GitHub [2] do pacote.

Referências

[1] D.P. Berrar, W. Dubitzky, M. Granzow, eds. (2003). “Singular value decomposition and principal component analysis”. A Practical Approach to Microarray Data Analysis, pp 91-109. https://www.cs.cmu.edu/~tom/10701_sp11/slides/pca_wall.pdf

[2] Renan Mendanha. (2023). “CCTV-Encoder”. GitHub. https://github.com/RenanMALV/CCTV-Encoder

[3] Emmanuel J. Candès, Xiaodong Li, Yi Ma, John Wright. (December 17, 2009). “Robust Principal Component Analysis?”. Seção 1.1. https://arxiv.org/pdf/0912.3599.pdf

[4] Emmanuel J. Candès, Xiaodong Li, Yi Ma, John Wright. (December 17, 2009). “Robust Principal Component Analysis?”. Seção 1.2. https://arxiv.org/pdf/0912.3599.pdf

[5] Zhouchen Lin, Minming Chen, Yi Ma. (October, 2013). “The Augmented Lagrange Multiplier Method for Exact Recovery of Corrupted Low-Rank Matrices”. Seção 3.1. https://arxiv.org/pdf/1009.5055.pdf

[6] Facebook Archive. (Nov 1, 2020). “fbpca”. https://fbpca.readthedocs.io/en/latest/

[7] SciPy documentation. (2023). “scipy.sparse.csr_matrix”. Versão 1.11.1. https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html

[8] Liu Weifeng, Vinter Brian. (2015). “CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication”. Seção 2.1. https://www.researchgate.net/publication/273788746_CSR5_An_Efficient_Storage_Format_for_Cross-Platform_Sparse_Matrix-Vector_Multiplication