Remoção de Fundo e Segmentação de Objetos em vídeo via PCA

Aluno: Victor Pereira de Lima

Link para o repositório (Github):
https://github.com/VictorPLima/Projeto_Final_COCADA_2025-1

Introdução

A ideia deste projeto veio de uma aula de inspirações de projetos onde o professor apresentou uma das aplicações de álgebra linear expostas no curso Computational Linear Algebra, esta aplicação era a de Background Removal with Robust PCA, onde a decomposição por Análise de Componentes Principais (PCA) é utilizada para obter e remover o fundo em uma filmagem de rua com pessoas andando. Em situações como esta a câmera normalmente encontra-se estática filmando apenas um certo ângulo, o que resulta em imagens com fundo praticamente constante, onde a única variabilidade fora a iluminação ocorre devido a objetos em movimento (veículos, pessoas, animais, …), como a cada componente da PCA temos a variância do que é decomposto mais explicada, podemos tirar proveito disso para buscar formas de decompor as imagens de um vídeo para obter seus detalhes principais e tentar segmentá-los.

Transformando vídeos em matrizes

O projeto foi desenvolvido utilizando a linguagem de programação Python e suas bibliotecas numéricas (NumPy, scikit-learn), de visualização de dados (matplotlib), e de tratamento de imagens e vídeos (PIL, moviepy, OpenCV). O vídeo utilizado neste projeto encontra-se no repositório com o nome “v1.mp4” e é uma gravação de 50 segundos onde pessoas são registradas caminhando na rua, como pode-se observar na Imagem 1.

Imagem 1: Frame do vídeo na posição 0:17 / 0:50.

Fonte: https://nbviewer.org/github/fastai/numerical-linear-algebra/blob/master/nbs/3.%20Background%20Removal%20with%20Robust%20PCA.ipynb#Load-and-view-the-data

As imagens são representadas no computador como matrizes de pixels, onde cada entrada possui as cores do pixel codificadas, no caso de imagens coloridas temos que as cores são representadas em código RGB (*Red, Green, Blue*), onde temos números entre 0 e 255 para indicar o quanto de vermelho, verde, e azul tem numa mesma cor e assim o computador conseguir representá-la, consequentemente, uma imagem colorida de dimensão mxn é representada por três matrizes mxn, cada uma para uma das cores RGB, conforme é mostrado abaixo.

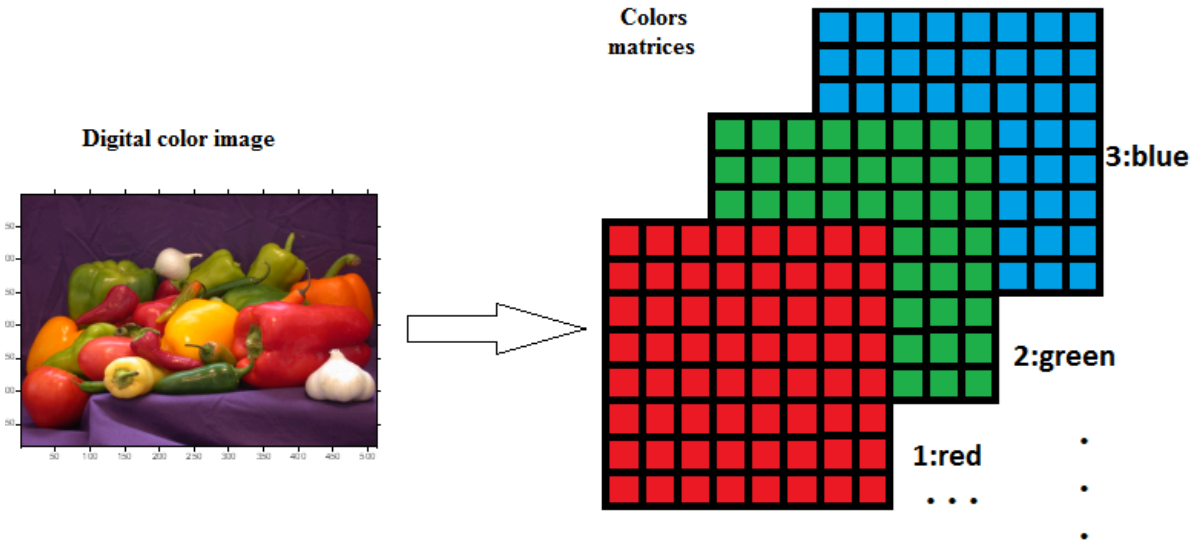


Imagem 2: Representação digital de imagem em código RGB.
Fonte: https://www.researchgate.net/figure/Color-image-representation_fig1_371071395

Um vídeo é composto por várias imagens (frames) obtidas por segundo, pelo menos 30 por segundo (30 fps) normalmente, supondo uma resolução básica de 360p (dimensão 640x360 pixels) e considerando que um pixel RGB requer 3 bytes (quantos bits tem um pixel), um vídeo de 1min requer (60s)*(30fps)*(640*360 pixel/frame)*(3 byte/pixel) = 1.244.160.000 bytes ≈ 1,24 GB, ou seja, trabalhar com vídeo requer muito armazenamento, para mitigar esse problema reduzimos a dimensão dos frames e os passamos para preto e branco (*grayscale*) onde temos apenas um 1 byte por pixel e uma matriz para codificar cor em vez de três.

A PCA busca decompor uma matriz A conforme $A_{mxn} = B_{mxk}C_{kxn}$, onde B, a matriz de componentes principais, possui como vetores coluna os k autovetores associados aos k maiores autovalores da matriz de covariância AA^T ou A^TA ordenados decrescentemente, e C possui as projeções dos vetores coluna de A nas componentes contidas em B. Portanto, para usar a PCA precisamos de uma só matriz que represente todo o vídeo, como este nada mais é do que uma lista de matrizes de pixels (frames) ordenada no tempo, podemos ver o vídeo como uma matriz onde cada vetor coluna é um frame e a quantidade de colunas é a de frames, no entanto, um frame ainda é uma matriz, e não um vetor, para contornar este problema "achatamos" cada frame em um vetor usando a função flatten() do NumPy, conforme abaixo.

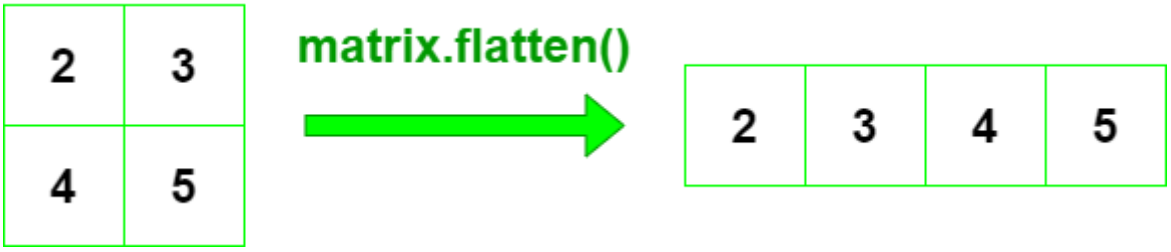
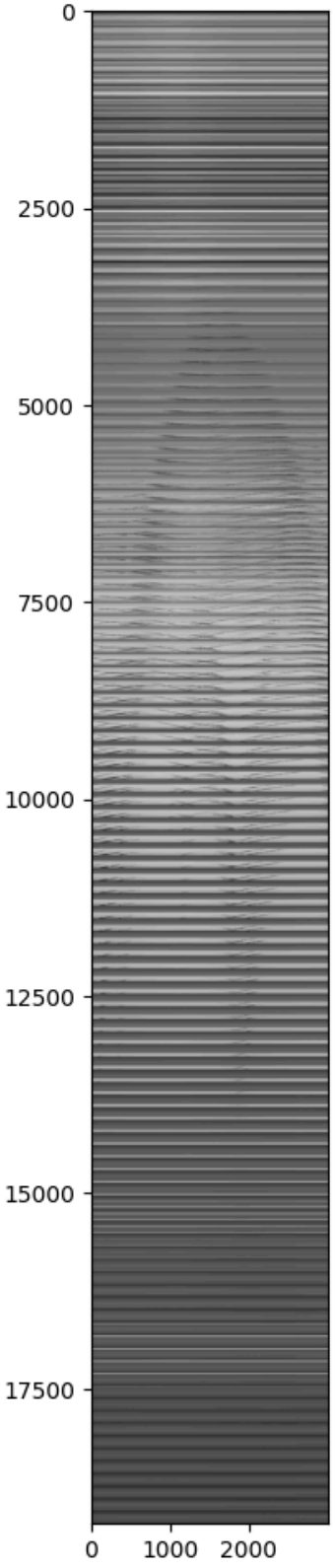
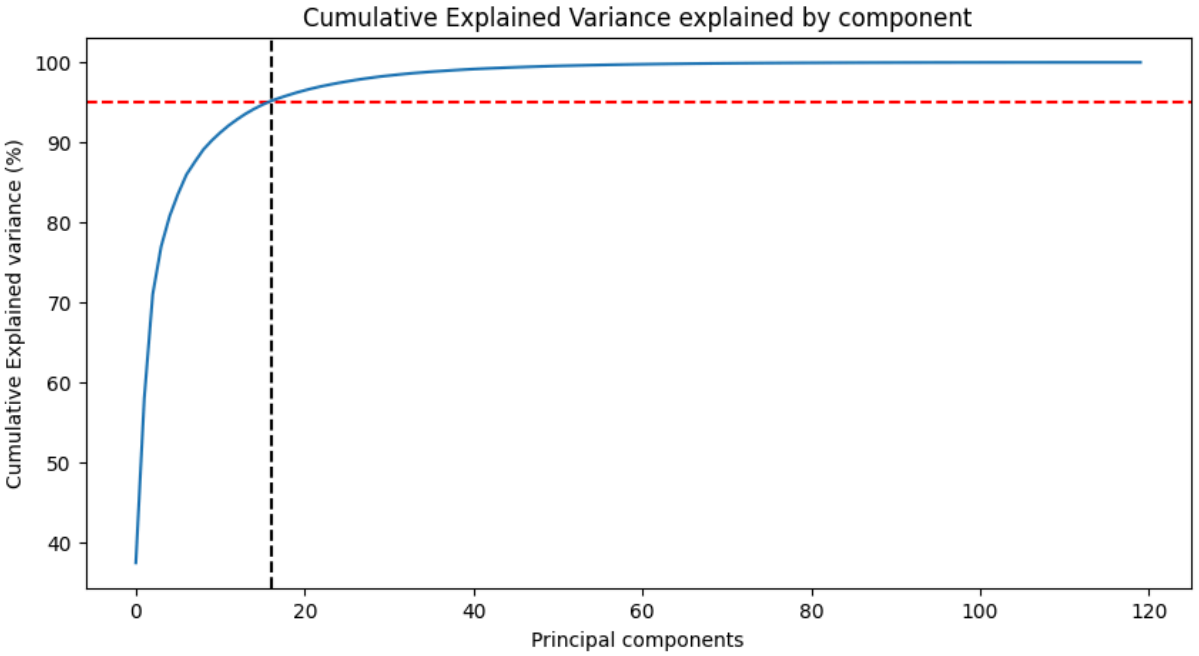


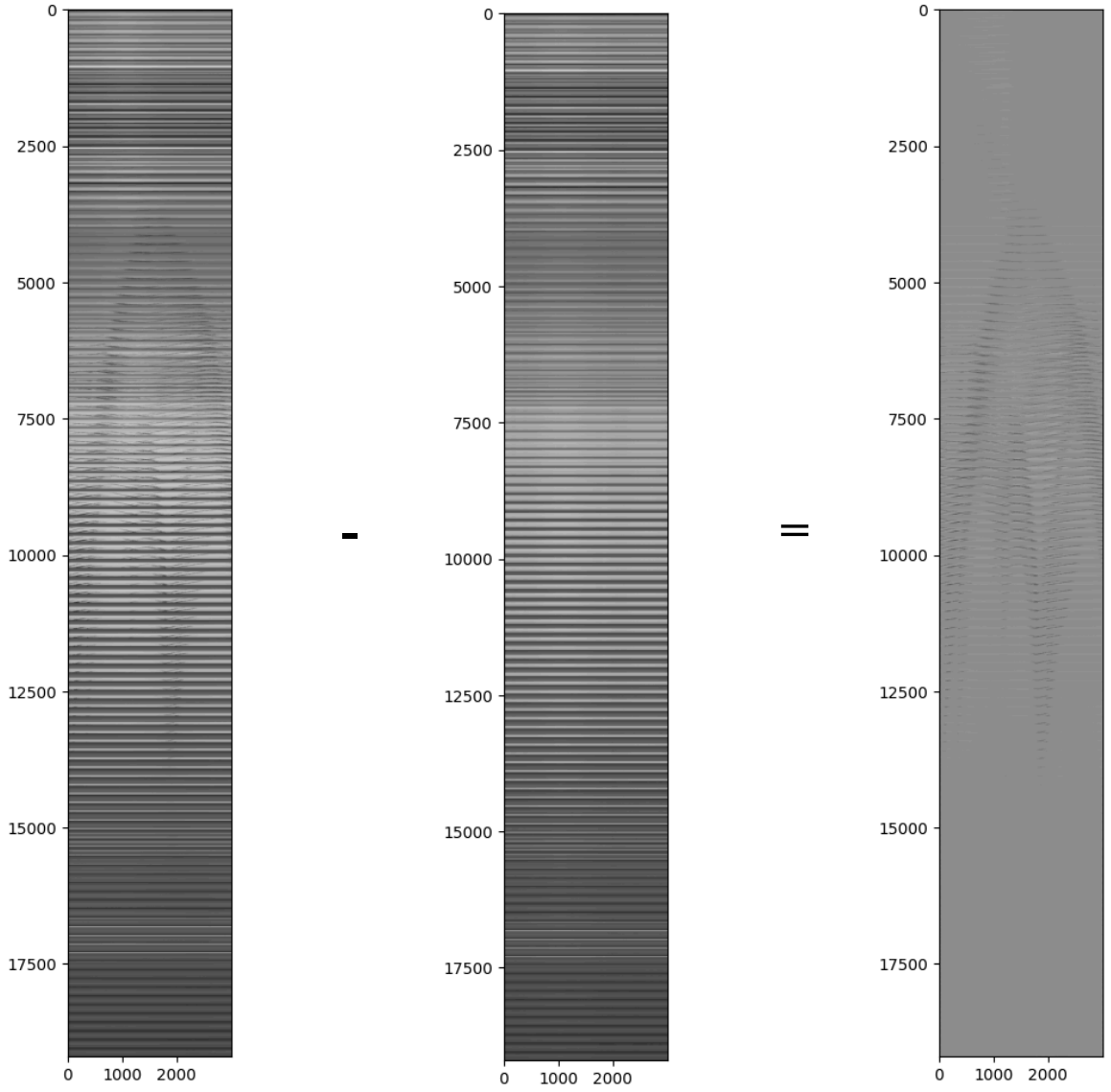
Imagem 3: Achatando matrizes em vetores por meio da função flatten do NumPy.
Fonte: https://www.geeksforgeeks.org/python/flatten-a-matrix-in-python-using-numpy/

Definimos uma taxa de fps para escolher quantos frames pegaremos por segundo do vídeo, já que pegar todos os frames pode ser muito custoso quando lidamos com vídeos com altas taxas de fps (>60 fps), e onde muitas vezes não há mudanças significativas entre frames

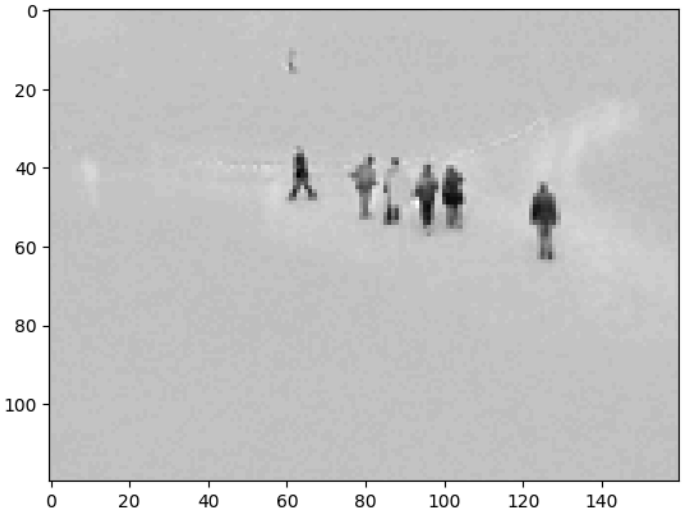
consecutivos. Feito isso, tornamos cada vetor-frame (como chamaremos frames achatados em vetores) em vetor coluna de uma matriz D e temos o seguinte resultado:



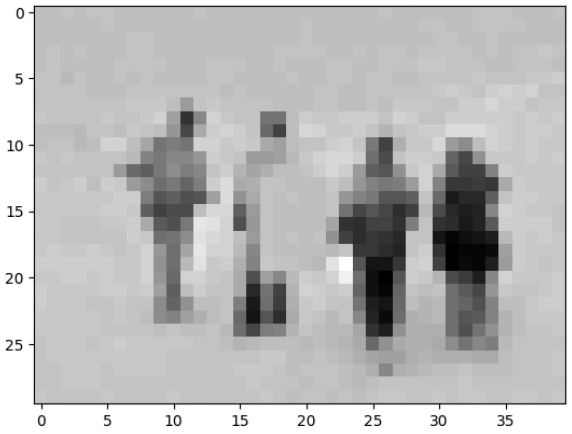

Cumulative Explained Variance explained by component
Cumulative Explained variance (%)
100
90
80
70
60
50
40
0
20
40
60
80
100
120
Principal components

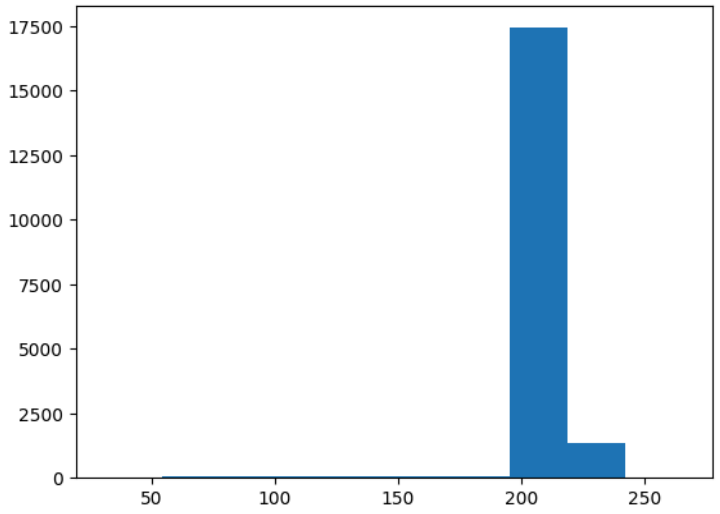

0
2500
5000
7500
10000
12500
15000
17500
0 1000 2000
-
=

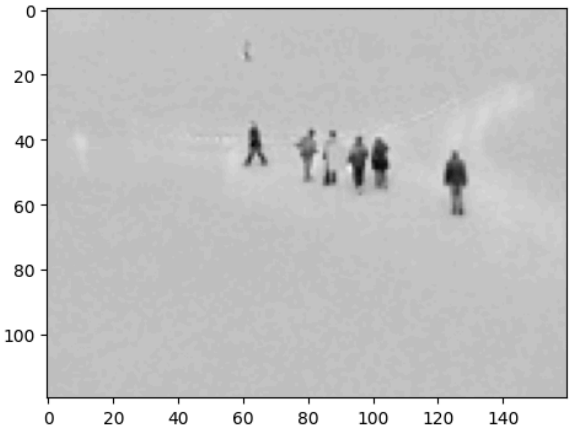

0
20
40
60
80
100
0
20
40
60
80
100
120
140

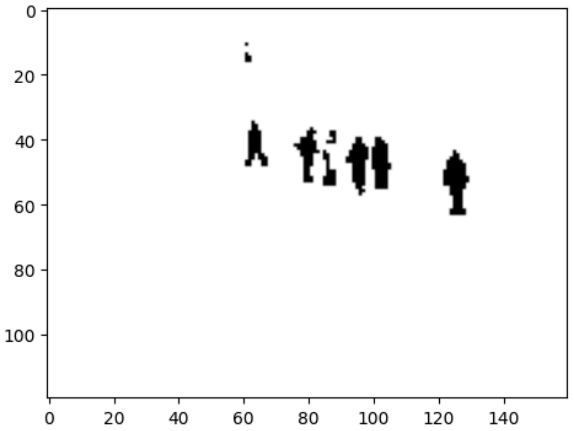

0
20
40
60
80
100
0
20
40
60
80
100
120
140

Segmentando as pessoas obtendo suas posições na imagem






0
20
40
60
80
100
0
20
40
60
80
100
120
140


0
20
40
60
80
100
0
20
40
60
80
100
120
140


0:02 / 0:50