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 no Google Colab 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.

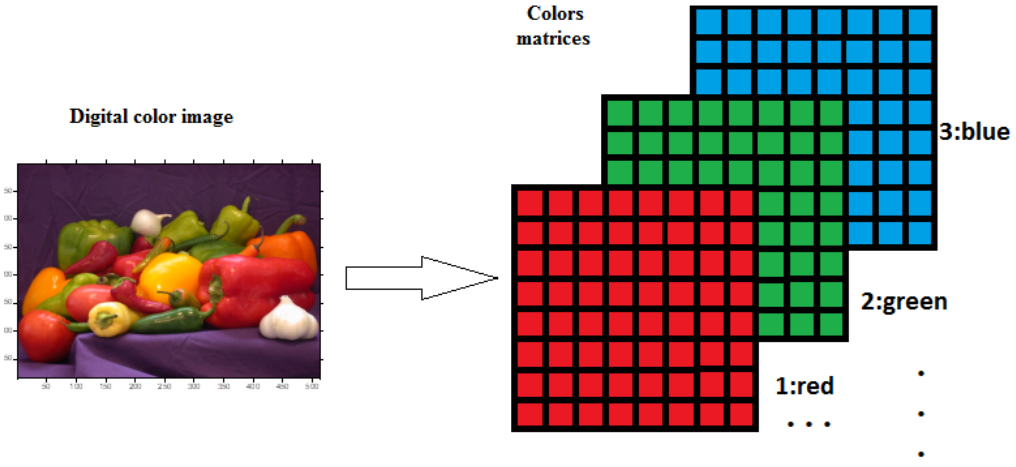


Imagem 2: Representação digital de imagem em código RGB.

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 (com os dados padronizados, ou centrados na média) 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.

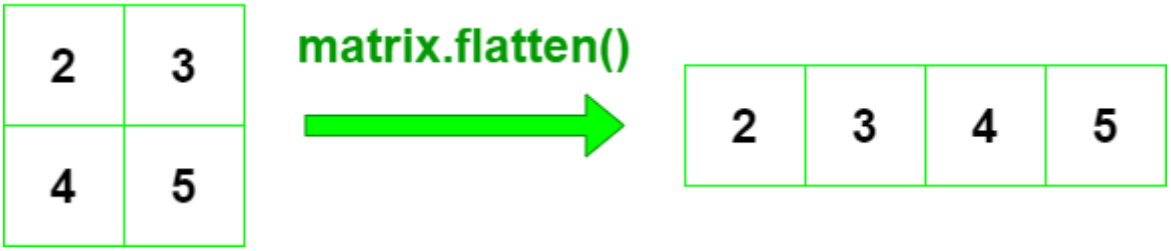


Imagem 3: Achatando matrizes em vetores por meio da função flatten do NumPy.
Fonte: https://www.geeksforgeeks.org/python/flatten-a-matrix-in-python-using-numpy/

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 a mesma plotada na Imagem 4.

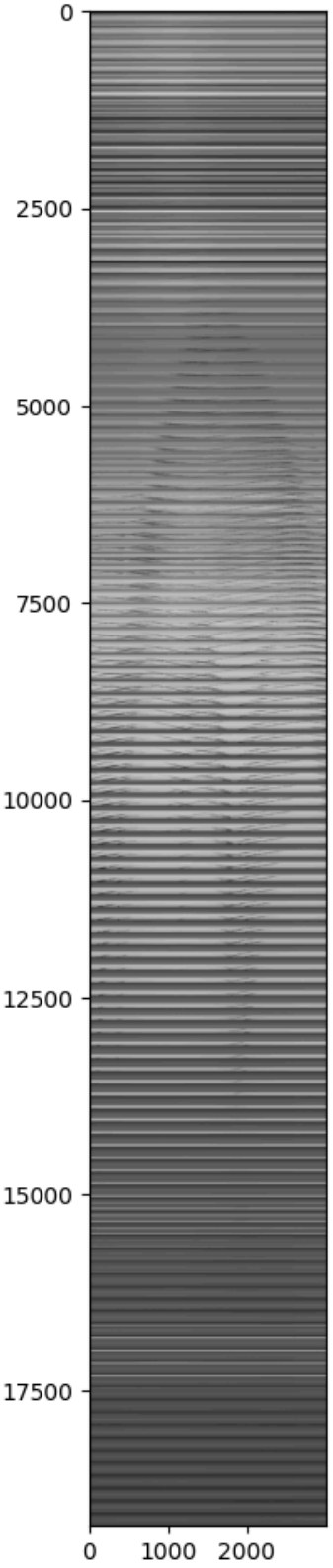


Imagem 4: Matriz de dados do vídeo (D) plotada.

O vídeo original tem duração de 50s e dimensão de 320x240 pixels, para trabalharmos com ele reduzimos sua dimensão em 50% e pegamos seus frames em uma taxa de 60fps, portanto, passamos a trabalhar com (50*60) frames de dimensão 160x120 pixels, ou seja, 3000 matrizes 120x160, que tornam-se os 3000 vetores colunas de dimensão 19200x1 (120*160=19200) que compõem D, que como podemos ver ao lado possui dimensão 19200x3000. A plotagem de D utilizando matplotlib toma 17s no Colab, uma vez que mesmo reduzindo em 50% os frames ainda temos como resultado uma matriz de 19200*3000B=57,6MB, como consequência é recomendado sempre salvá-la e trabalhar sobre cópias, para evitar ter que gerá-la novamente, o que pode levar muito tempo e exigir muito dos recursos de hardware (ou seja, também muito gasto de energia) dependendo do tamanho do vídeo.

Ao observarmos D e suas colunas, que correspondem aos frames do vídeo a cada (1/60)s, podemos reparar que há um grande padrão de fundo que não muda, e padrões sutis em forma de curvas definidas por sombras que mudam de posição ao longo das colunas de D, ou seja, ao longo do tempo. Acreditamos que este padrão de fundo é justamente o fundo da imagem contido em um vetor coluna que se repete horizontalmente em D, e as sombras são os objetos (pessoas) em movimento, como o fundo pode ser visto como um único vetor que engloba a maior parte de variabilidade de D, ele deve corresponder à primeira componente principal (PCA 1) de D, mas será que somente com uma componente principal é possível obter mesmo toda a variância contida no fundo?

Porque a PCA é interessante para imagens e vídeos?

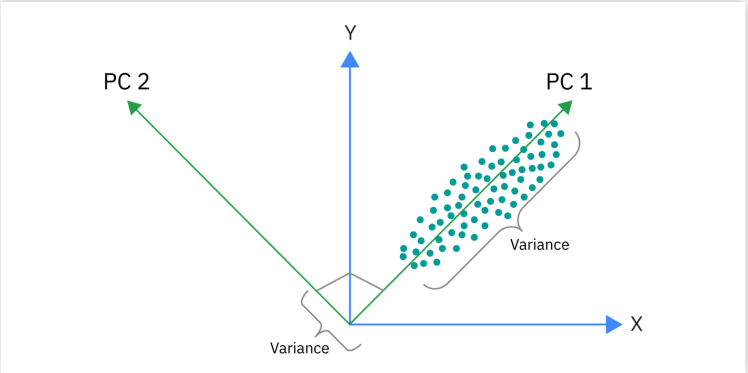


Imagem 5: Visualização da PCA.
Fonte: https://www.ibm.com/br-pt/think/topics/principal-component-analysis

A PCA, conforme observado acima, busca reduzir a dimensionalidade de um conjunto de dados buscando os vetores, neste caso as componentes principais ("PC"), que melhor representam os dados ao explicar a variância dos mesmos. O exemplo da Imagem 6 visa expor melhor isso ao tentar usar PCA para decompor a matriz de pixels em *grayscale* de uma imagem de tabuleiro de xadrez. Observando os padrões da imagem e lembrando que neste padrão de codificação de cores preto e branco correspondem a 0 e 255, respectivamente, vemos que esta é simplesmente composta de vetores coluna que seguem um padrão de branco e preto ($v_{bp} = [255, 0, \ldots, 255, 0]^T$), preto e branco ($v_{pb} = [0, 255, \ldots, 0, 255]^T$), e apenas preto (vetor nulo), sendo que os dois primeiros vetores podem ser obtidos um a partir do outro por meio da transformação linear $v_{bp} = 255I - v_{pb}$, e o último ao multiplicar por 0 (zero) qualquer vetor, ou seja, faz sentido imaginar que a partir de apenas duas componentes principais seja possível representar o tabuleiro e toda a sua variância, o que é confirmado abaixo, onde somando as variâncias explicadas por PCA 1 e 2 chegamos ao acumulado de 99,96% da imagem original, praticamente 100%, e a partir de PCA 3 temos os dados originais completamente explicados. É interessante notar que a imagem, de dimensão 128x128 pixels corresponde a uma matriz de posto 128, ou seja, que poderia ser decomposta em até 128 componentes, mas com apenas 3 (aproximadamente **2,34%**) foi possível representar 100% da imagem, isto se deve à natureza da mesma, que por possuir um padrão simples, torna-se "explicada" com facilidade pela análise de componentes principais.

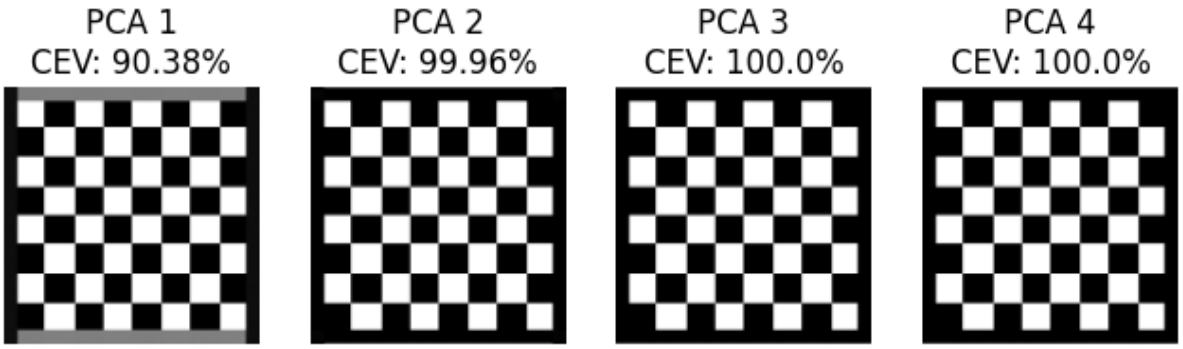


Imagem 6: Exemplo de uso da PCA em imagem de tabuleiro de xadrez 128x128px. CEV corresponde a *Cumulative Explained Variance*.

Ao observar as componentes principais do tabuleiro de xadrez na Imagem 7 observamos que os padrões de branco e preto realmente aparecem como parte dos vetores, e buscando analisar semanticamente vemos que PC1 representa a borda do tabuleiro nos extremos (127, cinza) e o xadrez no meio (255 e 0, branco e preto), PC2 as bordas pretas e a matriz de transição entre branco e preto (v_{bp}) e preto e branco (v_{pb}) ao fazer-se $PC2 - PC1$, e PC3 a complementar de PC2 que deve ser utilizada provavelmente para ajustar as cores da borda, já que $0.5PC3 - PC1$ faz as bordas cinzas se tornarem pretas sem mudar o padrão interno do xadrez.

```
PC1 = [127 127 127 127 127 127 127 255 255 255 255 255 255 255 255 255 255
 255 255   0   0   0   0   0   0   0   0   0   0   0   0   0   0 127
 255 255 255 255 255 255 255 255 255 255 255 255 255 255   0   0   0   0
   0   0   0   0   0   0   0   0   0   0 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255   0   0   0   0   0   0   0   0   0   0   0
   0   0   0 127 255 255 255 255 255 255 255 255 255 255 255 255 255 255
   0   0   0   0   0   0   0   0   0   0   0   0   0 127 127 127 127 127
 127 127]
PC2 = [  0   0   0   0   0   0   0 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
 255 255 255 255 255 255 255 255 255 255 255 255 255   0   0   0   0   0
   0   0]
PC3 = [255 255 255 255 255 255 170   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 170 255 255 255 255
 255 255]
```

Imagem 7: PCA 1, 2, e 3 com valores inteiros em *grayscale* obtidas para a imagem do tabuleiro de xadrez.

Finalmente, vamos analisar o quando da variância da nossa matriz de dados do vídeo (D) é explicada por suas componentes, como sua dimensão é 19200x3000, significa que tem posto 3000 e com isso no máximo 3000 componentes, no entanto, boas notícias surgem da interpretação do gráfico da Imagem 8: PCA 1 explica **90%** da variância, e a partir de PCA 350 100% é explicado, ou seja, com apenas 350/3000≈**11,67%** das componentes principais conseguimos representar 100% do vídeo. As porcentagens obtidas realmente nos fazem crer que PCA 1 poderá representar o fundo da imagem.

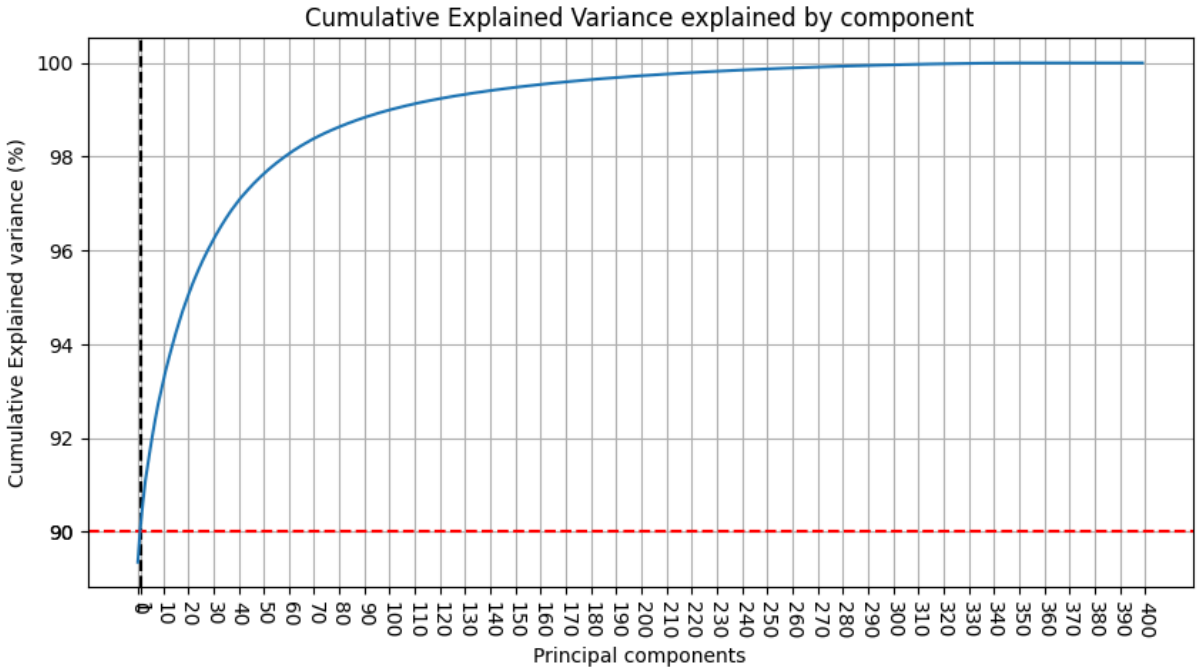


Imagem 8: Gráfico da porcentagem da variância explicada cumulativamente pelas componentes principais de D.

Remoção de Fundo

Chegamos finalmente à etapa principal deste projeto. Tendo D, plotada na Imagem 4, calculamos PCA1, a matriz gerada a partir da primeira componente de D, e observando o resultado de sua plotagem na Imagem 9 encontramos uma matriz que corresponde a D sem as sombras observadas, como D e PCA1 são matrizes de mesma dimensão, basta executar $D - PCA1$ para obter justamente apenas as sombras, que acreditamos ser os objetos em movimento.

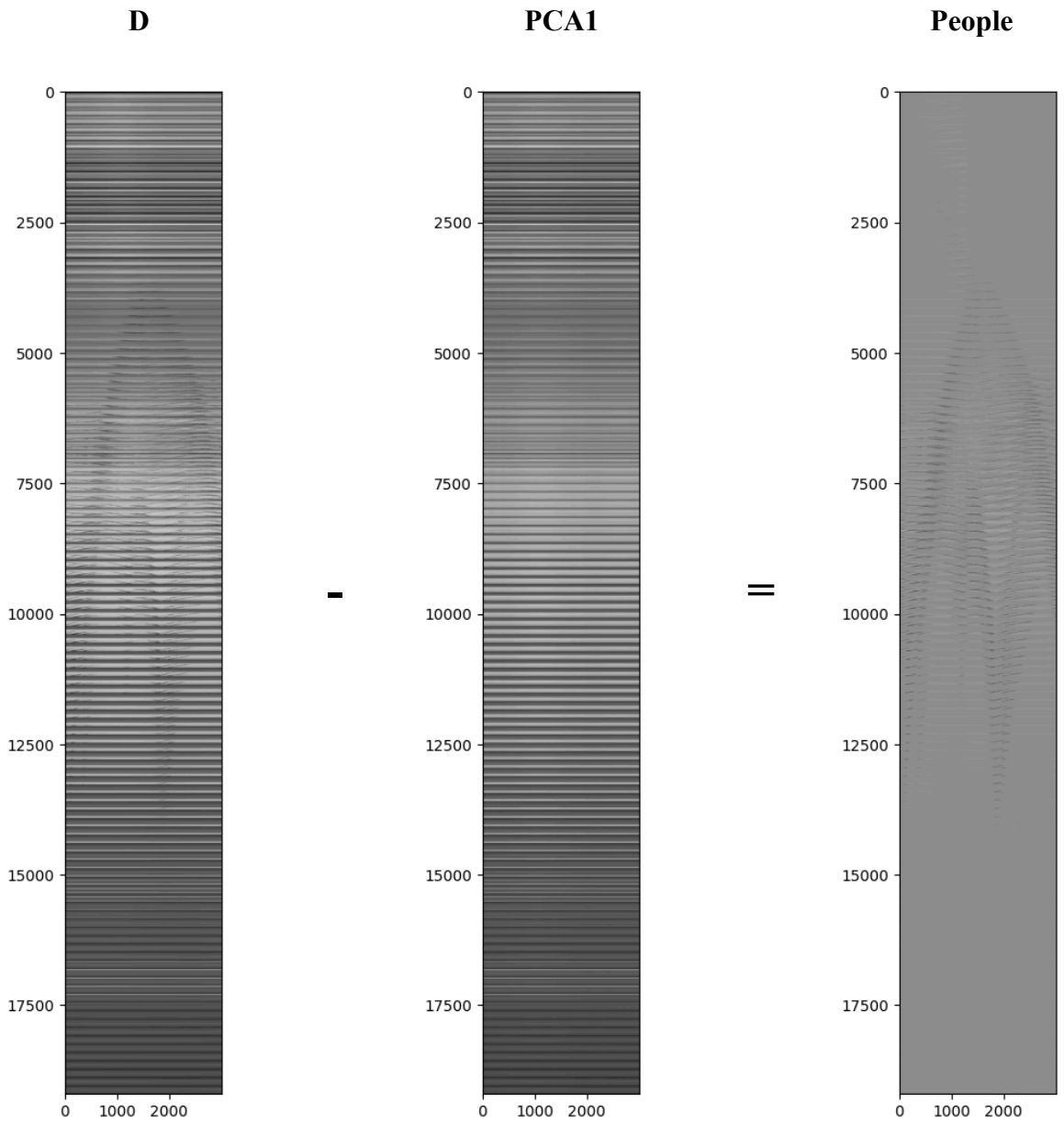


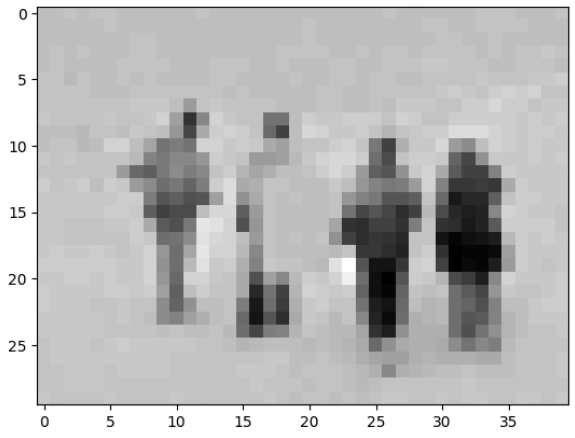
Imagem 9: Operações matriciais da remoção de fundo.

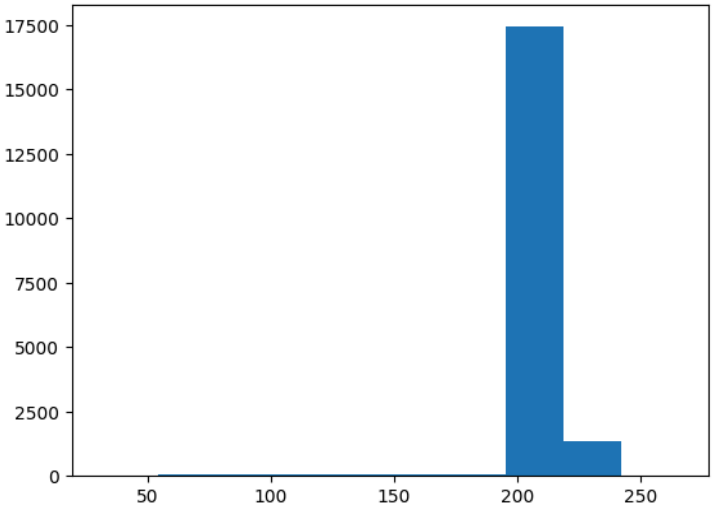
Agora vamos “desachatar” o vetor-frame da posição correspondente ao momento 0:10 (10s de filme) obtido das matrizes D, PCA1, e $D - PCA1$ (People) para ver se alcançamos os resultados desejados. Para obter o vetor-frame do segundo S do filme basta pegar o vetor coluna da coluna (S*60), pois obtemos 60fps para montar nossa matriz de vetores-frames.

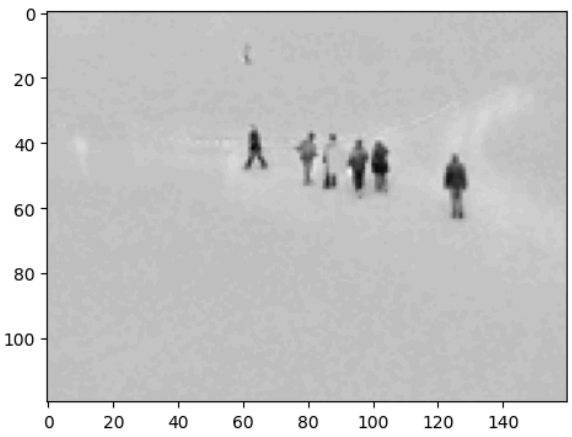
Observando os resultados plotados expostos na Imagem 10 vemos que em grande parte alcançamos nosso objetivo.

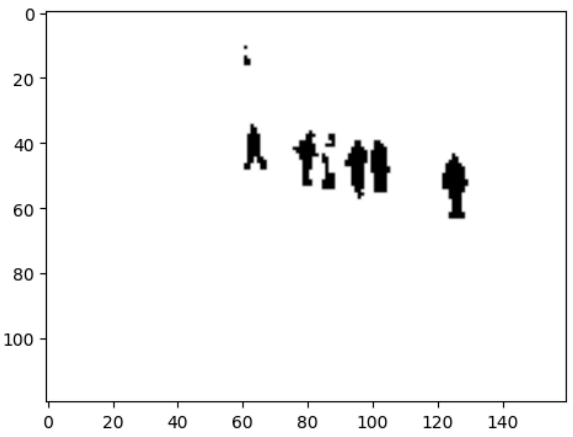
Imagem 10: Imagem do frame no momento 0:10 original, apenas com fundo, e sem fundo, respectivamente.

Segmentando as pessoas obtendo suas posições na imagem




17500
15000
12500
10000
7500
5000
2500
0
50
100
150
200
250


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