Resumo de Vídeos

Introdução ao Processamento de Imagem Digital

Percy Maldonado Quispe
203968

1. Introdução

A técnica de sumarização é conhecida por extrair resumo de vídeos, ou seja, é a capacidade de reduzir os quadros de um vídeo, mantendo os pontos mais importantes, onde estes são conhecidos como picos locais ao longo do comportamento do vídeo. A figura 1 mostra a representação de um vídeo em quadros, tomadas e cenas. Para realizar esta técnica, foi feita a sumarização utilizando os quadros de um vídeo, especificamente nas diferenças de características entre 2 quadros consecutivos.

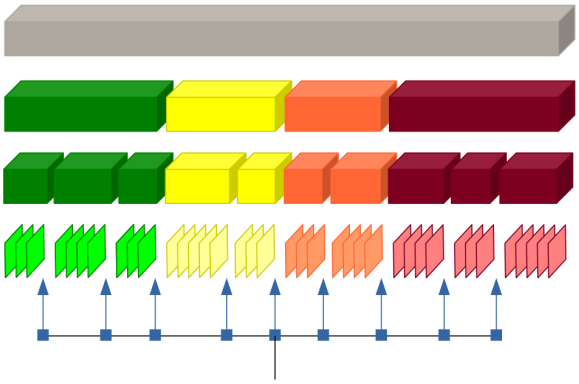


Figura 1: Representação de um vídeo

Para encontrar as diferenças entre dois quadros consecutivos, usaremos 4 diferenças:

- Diferença entre pixels.
- Diferença entre Blocos.
- Diferença entre Histogramas.
- Diferença entre Bordas.

Onde a diferença de pixels 3.2.1 se refere à diferença entre dois quadros consecutivos, a diferença entre os blocos 3.2.2 é o mesmo caso, mas não apenas levando em consideração um pixel, mas um bloco de *nxn* de pixels, para a diferença de histogramas 3.2.3, obtemos o histograma de cada quadro, e comparamos os 2 consecutivos com base em média e desvio padrão, finalmente a diferença de bordas **??**, refere-se ao número de pixels que foram detectados como bordas em um quadro que diferem do próximo quadros.

2. Características do Programa

O programa foi feito em linguagem Python, com bibliotecas externas NumPy e OpenCV. Os argumentos são passados em tempo de execução via *argv*, que permite modificar várias condições em tempo de execução.

Os argumentos do programa podem ser:

Figura 2: Numpy, Python e OpenCV

- *video*: Vídeo para aplicar a técnica.
- *difference*: Nome do tipo de diferença entre quadros consecutivos.
- –*gray*: Indica a pasta onde estão os vídeos em tons de cinza.
- –*out*: Indica a pasta onde serão salvos os resumos.

3. Implementação

Para a implementação da sumarização, foram feitas 4 diferenças entre os quadros consecutivos. Mas antes de aplicar as técnicas, foi realizado um pré-processamento dos vídeos para diminuir o tempo de sumarização.

3.1. Pré-processamento

Para reduzir o tempo de processamento, o vídeo foi convertido para tons de cinza, pois na hora de aplicar a técnica, leva muito tempo para converter o quadro RGB, por isso o programa a seguir foi feito em python *convert.py*

Os argumentos podem ser:

- *video*: Vídeo para converter de RGB para tons de cinza, se o texto digitado for 'all', todos os vídeos na pasta './videos/color' serão convertidos, o resultado será salvo na pasta ' pasta ./videos/gray'

3.2. Diferença entre quadros

Abaixo mostramos a implementação das seguintes 4 diferenças entre dois quadros consecutivos, para realizar este processamento, foi realizado em um conjunto de quadros consecutivos e não apenas 2 quadros, o funcionamento é o mesmo, porém é mais rápido devido ao facilidades que oferece Numpy.

3.2.1. Dif. Píxeis

Para a diferença entre pixels, subtraímos os últimos $n-1$ quadros de uma tomada dos primeiros $n-1$ quadros e comparamos com um limite.

Listing 1: Diferença de Pixel

```
def diff_pixels(frames, params):
    # subtraction between two consecutive frames in a shot
    difference = np.abs(frames[1:] - frames[:-1])

    # number of pixels greater than a threshold
    metrics = np.sum(difference > params[0], axis=(1, 2))

    return metrics
```

3.2.2. Dif. Blocos

Na diferença entre blocos, extraímos os blocos n utilizando a função *reshape*, que realiza operações entre arrays e já é eficiente com Numpy, o teste também foi realizado através de um *loop* **for**, e vimos a diferença no tempo de execução, é bastante.

Listing 2: Diferença de Bloco

```
def diff_blocks(frames, params):
    # parmas[0] is block size
    rows, cols = frames.shape[1], frames.shape[2]

    # trows is number os blocks
    trows = rows // params[0]
    tcols = cols // params[0]

    # use new frame dimensions
    # diff between two consecutive frames
    # squared difference
    frames = frames[:, :trows * params[0], :tcols * params[0]]
    difference = np.abs(frames[1:] - frames[:-1])
    difference = difference * difference

    metrics = []
    for diff in difference:
        # vectorized sum blocks
        # vectorized sqrt soma
        soma = np.sum(utils.reshape(
            diff, trows, tcols, params[0]), axis=(1, 2))
        metrics.append((np.sqrt(soma) > params[1]).sum())
    return metrics
```

3.2.3. Dif. Histogramas

Os histogramas de cada quadro foram feitos usando o Numpy, então calculamos o valor T, que é definido pela equação 1.

$$T = \mu + \alpha\sigma \tag{1}$$

Listing 3: Diferença de Histogramas

```
def diff_histogram(frames, params):
    total = frames.shape[0]
    # array of histograms
    histograms = np.empty((total, 256))

    # numpy histograms
    for i in range(total):
        hist, _ = np.histogram(frames[i], bins=256, range=(0, 255))
        histograms[i] = hist

    # diff between two consecutive frames
    # T = mean + 3*std
    difference = np.abs(histograms[1:] - histograms[:-1])
    means = np.mean(difference, axis=1)
    stds = np.std(difference, axis=1)
    metrics = means + stds*3

    return metrics
```

3.2.4. Dif. Bordas

A última diferença, é a do mapa de arestas, em que escolhemos o filtro Sobel 3.

Listing 4: Diferença de Bordas

```
def diff_edges(frames, params):
    metrics = []
    for frame in frames:
```

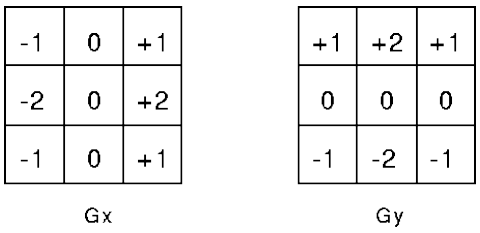


Figura 3: Filtro Sobel

```
        # apply sobel filter and count pixels
        # sobel filter is vectorized as addition
        mask = utils.normalize(utils.sobel(frame))
        metrics.append((mask > params[0]).sum())
    return metrics
```

3.3. Máximos locais

Para encontrar os quadros importantes no vídeo, ou seja, os máximos locais, utilizou-se o Numpy com a operação *np.diff*, também vista nas aulas do professor Hélio Pedrini. Com essa função podemos detectar todos os máximos locais na matriz de medidas dadas pelos métodos de diferença.

3.4. Execução

Para executar os métodos descritos acima, podemos fazê-lo a partir de um terminal que tenha os seguintes pacotes instalados:

- *opencv-python*
- *matplotlib*
- *numpy*

Aqui está um exemplo de como resumir um vídeo.

Listing 5: Example

```
python summarize.py ./videos/color/news.mpg edges
```

Se não houver vídeo em tons de cinza pré-processado, cada quadro será convertido em tons de cinza antes de fazer a diferença.

4. Resultados e Comparações

Os resultados foram obtidos para 11 vídeos diferentes (n), com 4 métodos para encontrar as medidas de diferença (m), dando um total de $n * m$ vídeos resumidos, onde n é o número de vídeos e m o número de métodos para encontrar a diferença entre 2 quadros consecutivos. Os vídeos de entrada estão disponíveis em https://www.ic.unicamp.br/~helio/videos_mp4/.

4.1. Comparação entre métodos

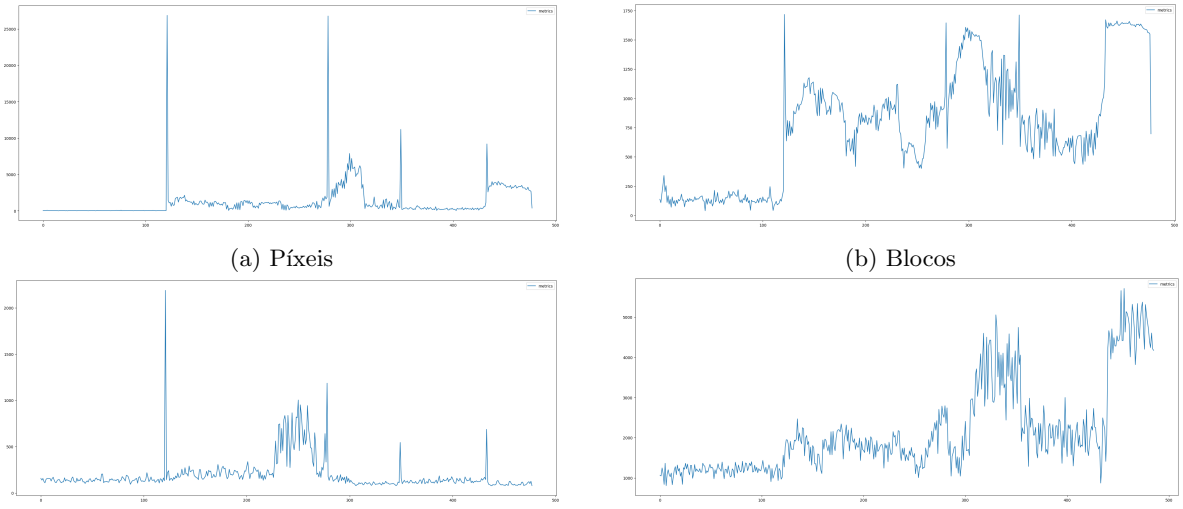
Para comparar os métodos, será feito de acordo com as métricas e como isso representa as transições entre os quadros de vídeo, também avaliaremos o tempo de execução entre os 4 métodos desenvolvidos.

4.1.1. Comparação entre medidas

Para comparar as medidas e as características dos vídeos apresentamos os gráficos de como os frames mudam em relação ao tempo, para o vídeo *news.mpg* temos os gráficos que são apresentados em 5.

Pelos gráficos podemos ver que todas as medições não encontram alterações relevantes nos primeiros frames do vídeo, também concorda com o vídeo original, onde apenas o apresentador aparece quase sem movimentos, então existe um pico que é a transição entre o apresentador e o player que está gravando, a diferença por arestas 4d acaba sendo menos sensível a mudanças bruscas, isso porque leva informações de todo o quadro, a diferença entre histogramas 4c não apresenta muita informação sobre mudanças bruscas, como 4d e 4b fazem.

De tudo isso, pode-se concluir que tirar informações de um grupo de pixels ou de todo o quadro dá melhores resultados do que pegar um único pixel ou ter o histograma.



(a) Píxeis

(b) Blocos

(c) Histogramas

(d) Bordas

Figura 4: Medições de todos os quadros consecutivos no vídeo *news.mpg*

4.1.2. Comp. entre o tempo de execução

Como visto na seção 3, para a implementação utilizamos a vetorização, que facilita muito as operações e processamentos, evitando *loops* desnecessários para que esses métodos sejam viáveis para uso em vídeos maiores, abaixo mostramos o tempos de execução em cada método para cada vídeo processado, conforme pode ser visto na tabela 1.

	Píxeis	Blocos	Histogramas	Bordas
indi009.mpg	0.32942	0.35593	0.70758	0.47934
lisa.mpg	0.13613	0.17306	0.35721	0.27032
news.mpg	0.27429	0.52565	0.83441	0.71206
toy.mp4	0.14846	0.25556	0.45476	0.38641
umn.mp4	0.07495	0.12078	0.24430	0.19460
xylophone.mp4	0.06069	0.09344	0.18142	0.14408
08024.mpg	29.07321	56.90182	118.14064	96.27925
08386.mpg	66.39135	85.03457	143.63174	127.36379
08401.mpg	17.40648	30.79790	58.59955	44.06276
36553.mpg	30.05942	50.39042	93.20687	78.58836
UGS09.mpg	28.57321	46.50043	98.95662	72.39124

Cuadro 1: Tempo de execução

A partir da tabela 1, pode-se observar que o método que mais consome tempo é a diferença entre histogramas, ainda mais do que aplicar um filtro como o caso de Sobel. O segundo mais caro, mas com bons resultados, é a diferença do mapa de arestas, resultados que podem ser comparados com a diferença entre blocos. A diferença entre os pixels dá resultados não tão descritivos quando o vídeo cresce.

4.2. Comparação entre resumo

Quando abrimos um vídeo bruto, é possível saber quantos quadros o vídeo possui, e quando realizamos os métodos de sumarização podemos obter o número de quadros finais, com essas duas medições conseguimos ver em quanto tempo os vídeos foram resumidos , a métrica que indica o valor para o qual foi reduzido pode ser vista em 2. Os resultados da aplicação dessa métrica estão na tabela 2.

$$T_r = \frac{N_{final}}{N_{initial}} \tag{2}$$

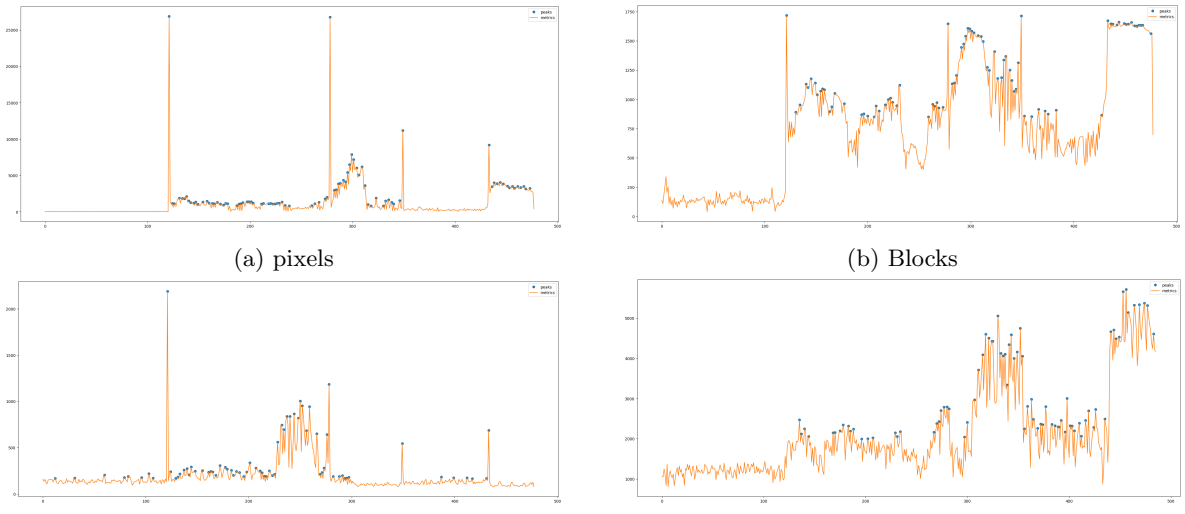
	Píxeis	Blocos	Histogramas	Bordas
indi009.mpg	0.14741	0.14542	0.15936	0.14314
lisa.mpg	0.14176	0.16795	0.15716	0.15606
news.mpg	0.17992	0.17155	0.14854	0.16667
toy.mp4	0.03636	0.07879	0.13333	0.14881
umn.mp4	0.13228	0.14286	0.15873	0.15104
xylophone.mp4	0.12857	0.11429	0.14286	0.12587
08024.mpg	0.09518	0.17297	0.15824	0.15272
08386.mpg	0.12099	0.18432	0.16614	0.15146
08401.mpg	0.03291	0.16121	0.15255	0.16298
36553.mpg	0.11557	0.17711	0.16032	0.15424
UGS09.mpg	0.07025	0.11081	0.15994	0.15347

Cuadro 2: Taxa em que foi resumido

Observando a tabela 2, podemos ver que o vídeo que teve a maior taxa de resumo foi *toy.mp4*, isso porque o vídeo dificilmente apresenta mudanças bruscas, é um vídeo em que há um brinquedo girando em algum lugar, mas não indica mudanças abruptas de quadro para quadro.

4.3. Threshold

Para observar os quadros que foram extraídos, temos os seguintes gráficos que representam os máximos locais que indicam transições abruptas entre 2 quadros consecutivos, os pontos locais para o vídeo *news.mpg* podem ser vistos no gráfico a seguir, onde eles são os máximos locais nas diferentes métricas obtidas.

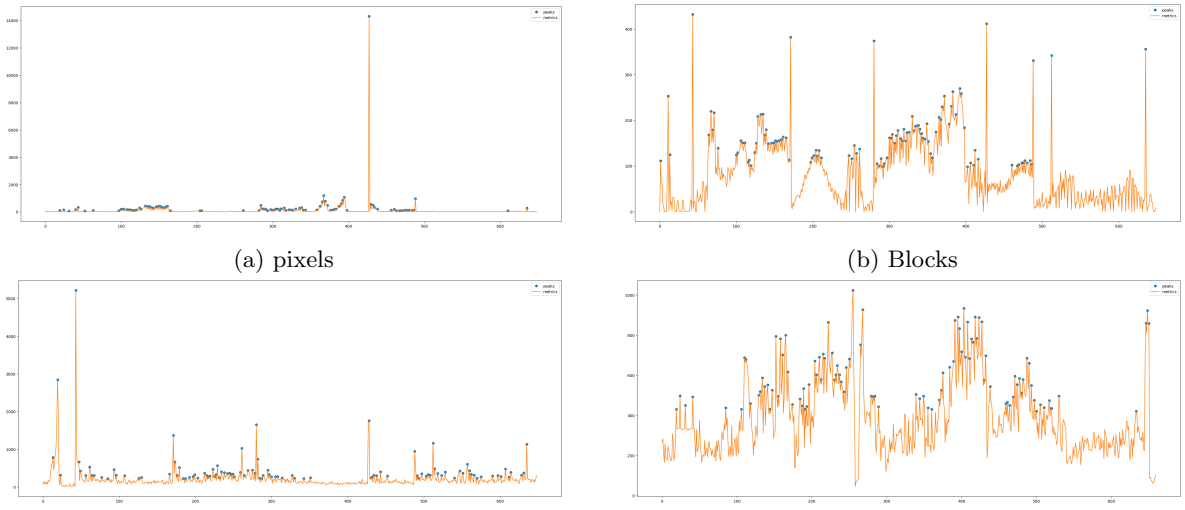


(a) pixels (b) Blocks

(c) Histograms (d) Edges

Figura 5: Pontos com transições abruptas para o vídeo *news.mpg*

Além do vídeo *news.mpg*, apresentamos gráficos para outros vídeos 6, 7, 8, 9 10.



(a) pixels

(b) Blocks

(c) Histograms

(d) Edges

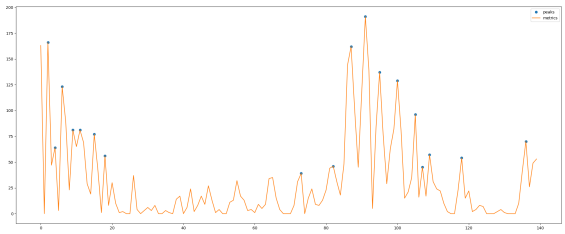
Figura 6: Pontos com transições abruptas para o vídeo *lisa.mpg*

5. Conclusões

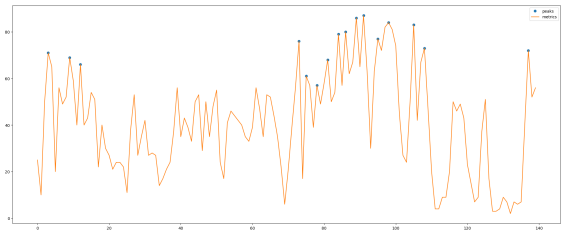
As técnicas de sumarização implementadas são capazes de observar diferentes comportamentos nos frames consecutivos de um vídeo. De acordo com os experimentos realizados, podemos concluir o seguinte:

- A diferença entre pixels, apresenta resultados aceitáveis mais não descreve bem o comportamento do vídeo, pois leva a informação de um pixel e não em sua totalidade.
- A diferença entre histogramas nem sempre apresenta os melhores resultados, embora haja uma diferença entre 2 quadros, isso porque um histograma mostra como os pixels estão distribuídos em uma imagem.
- Métodos que usam informações de pixel em blocos ou em todo o quadro dão os melhores resultados e também reduzem mais os quadros, então 3.2.2 e 3.2.4 são os melhores nesse processamento.

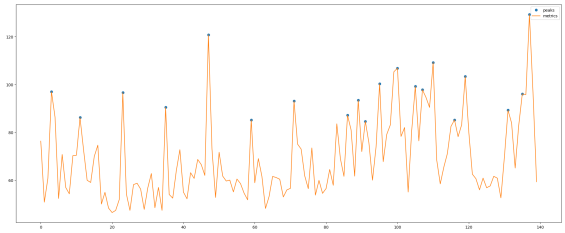
Além dos pontos citados, as implementações respondem de forma eficiente e escalável para vídeos com mais frames, também outro ponto importante a destacar é o processamento em tomadas de tamanho variável (s), isso ainda pode melhorar se tivermos uma máquina em que temos mais memória RAM.



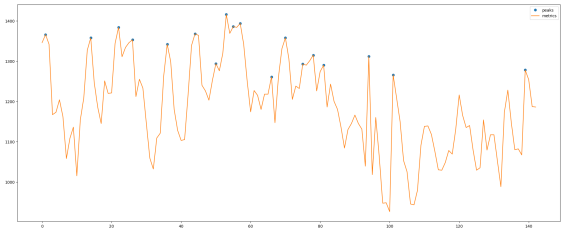
(a) pixels



(b) Blocks

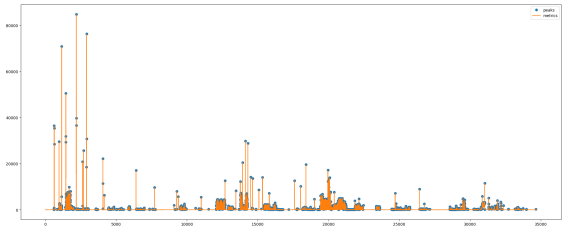


(c) Histograms

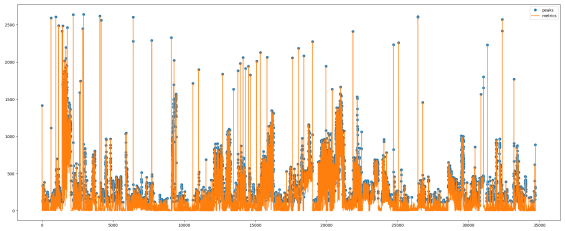


(d) Edges

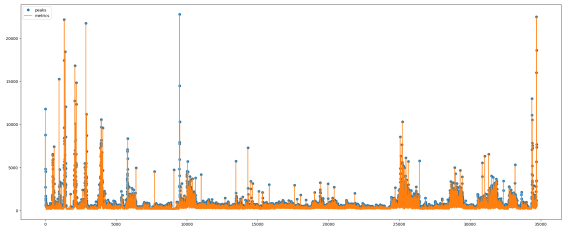
Figura 7: Pontos com transições abruptas para o vídeo *xylophone.mpg*



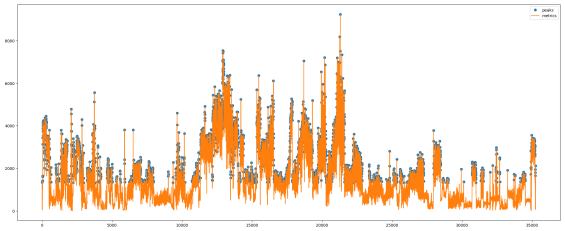
(a) pixels



(b) Blocks

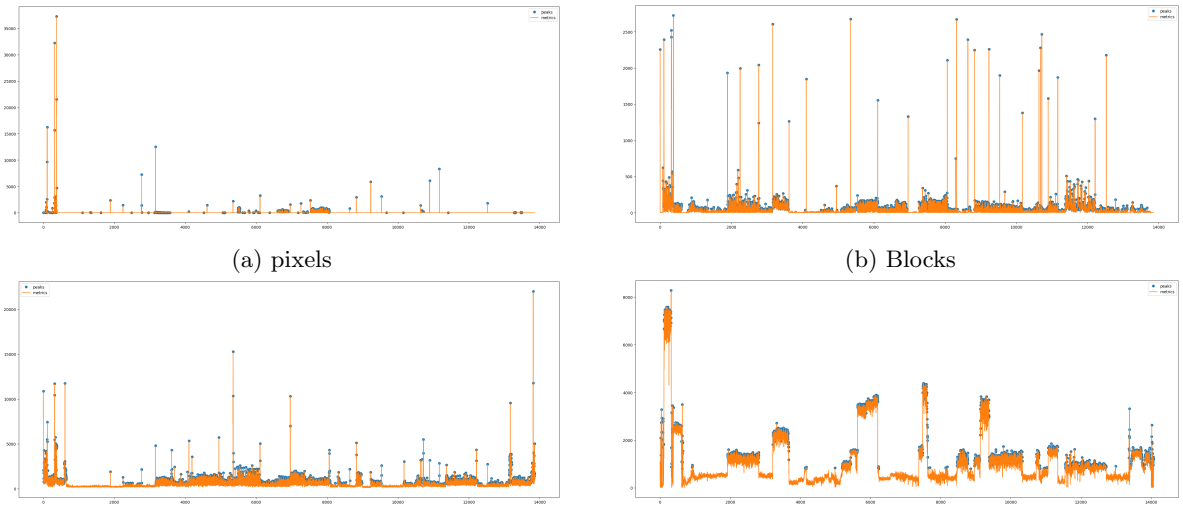


(c) Histograms



(d) Edges

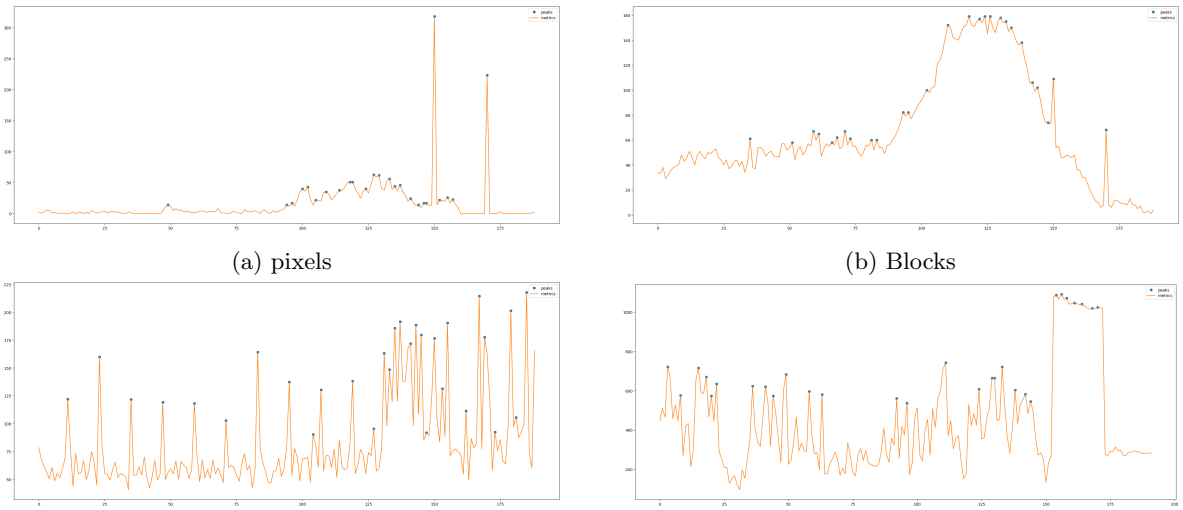
Figura 8: Pontos com transições abruptas para o vídeo *08024.mpg*



(a) pixels (b) Blocks

(c) Histograms (d) Edges

Figura 9: Pontos com transições abruptas para o vídeo *08401.mpg*



(a) pixels (b) Blocks

(c) Histograms (d) Edges

Figura 10: Pontos com transições abruptas para o vídeo *umn.mp4*