

Análise comparativa do impacto do vírus sincicial respiratório na célula humana através de técnicas de processamento de imagens

Bianca Aissa Santos
Universidade Estadual Paulista
São José do Rio Preto, Brasil
bianca.aissa@unesp.br

Bruno Vinicius Veronez de Jesus
Universidade Estadual Paulista
São José do Rio Preto, Brasil
bvv.jesus@unesp.br

Daphne Lie Haranaka Pereira
Universidade Estadual Paulista
São José do Rio Preto, Brasil
daphne.pereira@unesp.br

Gustavo Tomaello Juiz
Universidade Estadual Paulista
São José do Rio Preto, Brasil
gustavo.tomaello@unesp.br

Abstract—Relatório feito com o objetivo de explicar e analisar os procedimentos de processamento de imagens digitais aplicados ao problema proposta nos desafios 02 na disciplina de Processamentos de Imagens Digitais, visando responder as perguntas introduzidas pelo professor.

Index Terms—Processamento de imagem, Técnicas de análise, comparação de imagens, dimensão fractal, espaço de características

I. INTRODUÇÃO

Existe uma linhagem celular (Vero) infectada ou não com o vírus sincicial respiratório humano (hRSV). Foi criado uma sequência de imagens, a partir de padrões previamente estabelecidos, que representam a célula humana exposta ao vírus em um diferentes intervalos de tempo. São sete imagens, que mostram o controle celular (célula sem exposição ao vírus) e outras 6 observações (em intervalos de 21h, 29h, 44h, 53h, 73h, 96h) que mostram o progresso do vírus na célula humana. A partir dessas amostras, foram propostas diferentes técnicas de processamento de imagens para entender o impacto do vírus na célula humana. Foram utilizados os procedimentos indicados pelo desafio proposto na disciplina Processamento de Imagens Digitais.

II. BIBLIOTECAS E LINGUAGEM UTILIZADAS

Para realizar a manipulação das imagens foi utilizado a linguagem Python e algumas de suas bibliotecas e frameworks, como Tensor Flow, Python Imaging Library (PIL), OpenCV, NumPy, Matplotlib, e Skimage.

III. PROCEDIMENTOS REALIZADOS

A. Limiarização ótima de Otsu e Watershed

Com o objetivo de analisar as imagens, o método de Otsu foi aplicado para limiarização do que é considerado fundo e elemento. E posteriormente é aplicado o método de segmentação de Watershed, com objetivo de demarcar os núcleos celulares

e possíveis aglutinações dos mesmos. Ademais, foi utilizado o seguinte código em python para obtenção das imagens que serão mostradas a seguir.

Fica então visível as regiões com maior concentração de núcleos, de forma que na imagem original é representada pela região mais clara e nas pré-processadas é demarcado com diferentes tonalidades as regiões encontradas. Abaixo é possível visualizar a demarcação em 7 diferentes momentos. listings

```
import numpy as np
import cv2 as cv
import cv2
from matplotlib import pyplot as plt

def imshow(img, ax=None):
    if ax is None:
        plt.imshow(img)
        plt.show()

    else:
        ax.imshow(cv2.cvtColor(img,
cv2.COLOR_BGR2RGB))
        ax.axis('off')

image_filenames = ["Normal.jpg", "21h.jpg",
"29h.jpg", "44h.jpg", "53h.jpg",
"73h.jpg", "96h.jpg"]

for filename in image_filenames:
    img = cv.imread(filename)
    assert img is not None, "erro ao ler
    imagens"
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    ret, thresh = cv.threshold(gray, 0, 255,
cv.THRESH_BINARY_INV+cv.THRESH_OTSU)
```

```

kernel = np.ones((3,3),np.uint8)
opening =
cv.morphologyEx(thresh, cv.MORPH_OPEN,kernel,
iterations = 2)
sure_bg =
cv.dilate(opening,kernel,iterations=3)
dist_transform =
cv.distanceTransform(opening, cv.DIST_L2,5)
ret, sure_fg =
cv.threshold(dist_transform,0.1
*dist_transform.max(),255,0)
sure_fg = np.uint8(sure_fg)
unknown = cv.subtract(sure_bg,sure_fg)

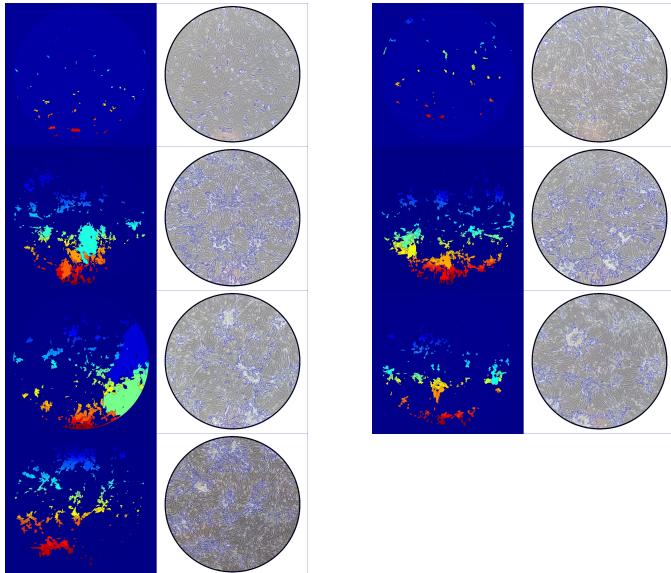
ret, markers =
cv.connectedComponents(sure_fg)
markers = markers+1
markers[unknown==255] = 0

markers = cv.watershed(img,markers)
img[markers == -1] = [255,0,0]
imshow(markers)
imshow(img)

markers_visual = cv.normalize(markers,
None, 0, 255,
cv.NORM_MINMAX).astype(np.uint8)
markers_color =
cv.applyColorMap(markers_visual,
cv.COLORMAP_JET)
concatenated_img =
np.hstack((markers_color, img))

output_filename =
f'concatenated_{filename}'
cv.imwrite(output_filename,
concatenated_img)

```



B. Matriz de co-ocorrência

É obtido uma matriz de co-ocorrência que representa a média das matrizes calculadas via $d=1$ e diferentes ângulos ($0, 90, 180, 270$). Com os seguintes descritores segundo momento angular, entropia, correlação, contraste e homogeneidade.

1) Código utilizado: geração da matriz de co-ocorrência

```

import cv2
import numpy as np
import pandas as pd
from skimage.feature import graycomatrix,
    graycoprops
from skimage.measure import shannon_entropy

# Funcao para calcular a matriz de
# co-ocorrencia media
def calcular_matriz_coocorrecnia_media(img,
    distance=1, angles=[0, 90, 180, 270]):
    glcms = [graycomatrix(img, [distance],
        [angle], levels=256, symmetric=True,
        normed=True) for angle in angles]
    glcm_media = np.mean(glcms, axis=0)
    return glcm_media

# Funcao para calcular os descritores de
# textura
def calcular_descritores_textura(glcm_media):

    entropy = shannon_entropy(glcm_media)

    descritores = {
        'Segundo Momento Angular':
            graycoprops(glcm_media, 'ASM')[0, 0],
        'Entropia': entropy,
        'Correlao': graycoprops(glcm_media,
            'correlation')[0, 0],
        'Contraste': graycoprops(glcm_media,
            'contrast')[0, 0],
        'Homogeneidade':
            graycoprops(glcm_media, 'homogeneity')[0,
            0]
    }
    return descritores

# Carregar imagens em escala de cinza
imgs_gray = [cv2.imread(img,
    cv2.IMREAD_GRAYSCALE) for img in
    ["Normal.jpg", "21h.jpg", "29h.jpg",
    "44h.jpg", "53h.jpg", "73h.jpg",
    "96h.jpg"]]

# Definir parametros da matriz de
# co-ocorrencia
distance = 1
angles = [0, 90, 180, 270]
a = 0

descritores_all = []

# Calcular matriz de co-ocorrecnia media e
# descritores de textura para cada imagem
for img_gray in imgs_gray:
    glcm_media =
        calcular_matriz_coocorrecnia_media(img_gray,

```

```

distance, angles)
descritores =
calcular_descritores_textura(glcm_media)
descritores_all.append(descritores)

desc_df = pd.DataFrame(descritores_all)
print(desc_df)

```

A matriz obtida a partir das 7 imagens:

	Segundo Momento Angular	Entropia	Correlação	Contraste	Homogeneidade
0	0.102721	4.817422	0.958439	214.376019	0.438339
1	0.102870	3.779268	0.964065	175.079033	0.445675
2	0.102510	5.668658	0.936722	358.862807	0.386584
3	0.102506	4.941417	0.947434	274.826536	0.399825
4	0.102611	4.579813	0.956674	226.075972	0.417073
5	0.102494	4.844868	0.956571	244.043664	0.410774
6	0.102439	6.782698	0.951693	408.093061	0.381035

Figure 1. Descritores para imagens em 7 momentos.

Analisando a matriz é possível chegar as seguintes conclusões:

- 1) **Uniformidade da Textura:** A uniformidade da textura, medida pelo Segundo Momento Angular, permanece praticamente constante ao longo do tempo de infecção, indicando que a textura das imagens não se torna significativamente mais ou menos uniforme.
- 2) **Complexidade da Textura:** A entropia crescente indica que a textura das imagens se torna mais complexa e desordenada conforme a infecção progride.
- 3) **Similaridade Linear:** A alta correlação constante sugere que os padrões lineares entre os pixels permanecem relativamente estáveis durante o tempo de infecção.
- 4) **Variação de Intensidade:** O aumento do contraste ao longo do tempo indica uma maior variação na intensidade dos pixels, o que pode estar associado à progressão da infecção e à degradação celular.
- 5) **Similaridade de Pixels:** A diminuição da homogeneidade sugere que a imagem se torna menos homogênea, refletindo uma maior diversidade na textura e possivelmente indicando áreas de infecção mais desenvolvidas.

Em resumo, esses resultados indicam que a progressão da infecção viral nas placas de Petri resulta em imagens com texturas mais complexas, maior variação de intensidade e menor homogeneidade, enquanto a uniformidade da textura e a similaridade linear entre os pixels permanecem relativamente estáveis.

C. Obtenção da Dimensão Fractal

A obtenção da Dimensão fractal (DF), usando Box-counting, foi feita para quantificar imagens monocromáticas (espaço tridimensional). Consideramos iterações realizadas enquanto lado da caixa ≥ 2 .

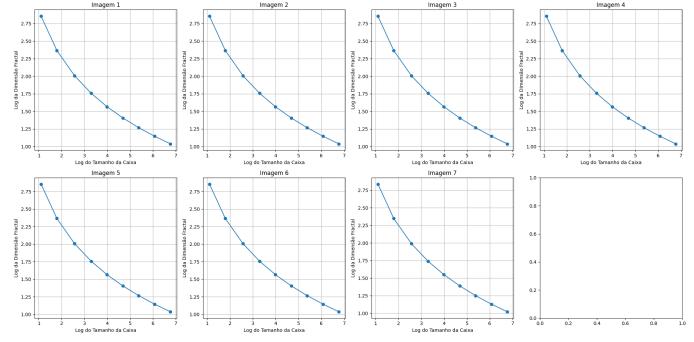


Figure 2. Caption

D. Espaço de características

Para analisar o espaço de características foi utilizado Otsu e Watershed para adequadamente separar o fundo e o objeto da imagem. Foi tratado como objeto os núcleos das células e fundo o resto da placa de petri. para se obter o resultado utilizou-se o seguinte código:

```

import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import peak_local_max
from skimage.segmentation import watershed
from scipy import ndimage as ndi
from skimage.morphology import
    remove_small_objects, disk
from skimage.measure import regionprops, label
from skimage.filters.rank import entropy

# Funcao para calcular a dimensao fractal
# usando o metodo de contagem de caixas
def fractal_dimension(Z):
    def boxcount(Z, k):
        S = np.add.reduceat(
            np.add.reduceat(Z, np.arange(0,
            Z.shape[0], k), axis=0),
            np.arange(0,
            Z.shape[1], k), axis=1)
        return len(np.where((S > 0) & (S <
        k*k))[0])
    Z = (Z < 255)
    if Z.sum() == 0:
        return np.nan
    p = min(Z.shape)
    n = 2**np.floor(np.log(p)/np.log(2))
    if n < 2:
        return np.nan
    sizes =
    2**np.arange(int(np.log(n)/np.log(2)), 1,
    -1)
    counts = [boxcount(Z, size) for size in
    sizes]
    if len(counts) < 2 or any(c == 0 for c in
    counts):
        return np.nan
    coeffs = np.polyfit(np.log(sizes),
    np.log(counts), 1)
    return -coeffs[0]

```

```

image_path = '96h.jpg'
image = cv2.imread(image_path)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

ret, binary_otsu = cv2.threshold(gray, 0,
    255, cv2.THRESH_BINARY_INV +
    cv2.THRESH_OTSU)

binary_otsu =
    remove_small_objects(binary_otsu.astype(bool)
        min_size=50).astype(np.uint8)

distance =
    ndi.distance_transform_edt(binary_otsu)

local_maxi = peak_local_max(distance,
    min_distance=20, labels=binary_otsu)

markers = np.zeros_like(distance, dtype=bool)
markers[tuple(local_maxi.T)] = True
markers, _ = ndi.label(markers)

labels = watershed(-distance, markers,
    mask=binary_otsu)

labels = remove_small_objects(labels,
    min_size=50)

regions = regionprops(labels,
    intensity_image=gray)

features_objects = []
features_background = []

entropy_image = entropy(gray, disk(3))

min_area = 200
valid_labels = [region.label for region in
    regions if region.area >= min_area]
filtered_labels = np.isin(labels,
    valid_labels)

# Separar objetos e fundo
object_mask = np.isin(labels, valid_labels)
background_mask = ~object_mask

# Calcular caracteristicas para os objetos
for region in regions:
    if region.area >= min_area:
        area = region.area
        perimeter = region.perimeter
        circularity = (perimeter ** 2) / (4 *
            np.pi * area) if area > 0 else 0
        mean_intensity = region.mean_intensity
        intensity_image =
            region.intensity_image
        std_intensity =
            np.std(intensity_image)
        minr, minc, maxr, maxc = region.bbox
        aspect_ratio = (maxc - minc) / (maxr -
            minr) if (maxr - minr) > 0 else 0
        region_entropy =
            np.mean(entropy_image[region.coords[:, 0],
                region.coords[:, 1]])
        region_df =
            fractal_dimension((region.image*255).
                astype(np.uint8))

        features_objects.append({
            'Area': area,
            'Perimeter': perimeter,
            'Circularity': circularity,
            'Mean Intensity': mean_intensity,
            'Standard Deviation Intensity':
            std_intensity,
            'Aspect Ratio': aspect_ratio,
            'Entropy': region_entropy,
            'Fractal Dimension': region_df
        })

# Calcular caracteristicas para o fundo
background_area = np.sum(background_mask)
background_perimeter =
    np.sum(np.pad(background_mask.astype(int),
        pad_width=1, mode='constant')[1:-1,
        1:-1]) # Ajuste no indice de fatiamento
background_mean_intensity =
    np.mean(gray[background_mask])
background_std_intensity =
    np.std(gray[background_mask])
background_entropy =
    np.mean(entropy_image[background_mask])
background_df =
    fractal_dimension(background_mask)

features_background = {
    'Area': background_area,
    'Perimeter': background_perimeter,
    'Mean Intensity':
    background_mean_intensity,
    'Standard Deviation Intensity':
    background_std_intensity,
    'Entropy': background_entropy,
    'Fractal Dimension': background_df
}

# Exibir as caracteristicas
print("Caracteristicas dos Objetos:")
for i, feature in enumerate(features_objects):
    print(f"Objeto {i + 1}:")
    for key, value in feature.items():
        if np.isnan(value):
            print(f"  {key}: NaN")
        else:
            print(f"  {key}: {value:.2f}")

print("\nCaracteristicas do Fundo:")
for key, value in features_background.items():
    if np.isnan(value):
        print(f"  {key}: NaN")
    else:
        print(f"  {key}: {value:.2f}")

total_features = len(features_objects)
mean_features_objects = {}

for feature_name in
    features_objects[0].keys():
    feature_sum = 0
    count = 0
    for feature in features_objects:
        if not

```

```

np.isnan(feature[feature_name]):
    feature_sum +=
feature[feature_name]
    count += 1
mean_features_objects[feature_name] =
feature_sum / count if count != 0 else
np.nan

print("Média das características dos objetos:")
for key, value in
mean_features_objects.items():
    print(f"{key}: {value:.2f}" if not
np.isnan(value) else f"{key}: nan")

fig, axes = plt.subplots(1, 4, figsize=(24,
6), sharex=True, sharey=True)
ax = axes.ravel()
ax[0].imshow(gray, cmap=plt.cm.gray)
ax[0].set_title('Imagem em escala de cinza')
ax[1].imshow(binary_otsu, cmap=plt.cm.gray)
ax[1].set_title('Binarização de Otsu')
ax[2].imshow(labels,
cmap=plt.cm.nipy_spectral)
ax[2].set_title('Segmentação por Watershed')
ax[3].imshow(object_mask, cmap=plt.cm.gray)
ax[3].set_title('Classificação de Núcleos e
Fundo')
for a in ax:
    a.axis('off')

plt.tight_layout()
plt.show()

# Salvar resultados
cv2.imwrite('gray_image.jpg', gray)
cv2.imwrite('binary_otsu.jpg', binary_otsu)
cv2.imwrite('watershed_labels.jpg', (labels *
(255 / labels.max())).astype(np.uint8))
cv2.imwrite('classified_image.jpg',
object_mask.astype(np.uint8) * 255)

```

Table I
TABELA COMPARATIVA DO VETOR DE CARACTERÍSTICAS AO LONGO DO TEMPO

Características	96h	Normal
Características do Fundo		
Área	9,214,630.00	8,427,371.00
Perímetro	9,214,630.00	8,427,371.00
Intensidade Média	247.10	252.36
Desvio Padrão da Intensidade	19.07	9.56
Entropia	0.63	0.29
Dimensão Fractal	1,15	1,13
Média das Características dos Objetos		
Área	5614.92	14854.63
Perímetro	318.63	459.48
Circularidade	1.84	1.67
Intensidade Média	135.95	153.20
Desvio Padrão da Intensidade	68.45	77.57
Razão de Aspecto	1.05	1.08
Entropia	3.73	2.97
Dimensão Fractal	0.90	0.90

E. Diferenças Estatísticas

COMPARAÇÃO DE IMAGENS

Para comparar as imagens, foram calculadas as seguintes métricas, similar as medidas vistas durante o cálculo da matriz de co-ocorrência, para as imagens normais e após 96 horas:

Entropia

- **Resultado**

Normal:

- Imagem Otsu: 1.4747613261793695
- Imagem Watershed: 9.64137115291814796h:
- Imagem Otsu: 1.8909819692079848
- Imagem Watershed: 10.812522453300026

- **Conclusão**

A entropia calcula o quanto aleatórios são os pixels distribuídos pela imagem. Imagens que possuem alta entropia possuem uma maior variedade de pixels. Ao analisar o resultado, é evidente que a imagem de Watershed tem uma distribuição de níveis de intensidade mais uniforme, o que faz sentido, pois a imagem de Otsu utilizada é binarizada.

Contraste

- **Resultado**

Normal:

- Imagem Otsu: 1192.63925115
- Imagem Watershed: 0.
- 96h:
- Otsu: 3070.20879726
- Watershed: 0.00624555

- **Conclusão**

O contraste mede a variação de intensidade entre pixels adjacentes. Como a imagem de Otsu só possui valores de branco ou preto, o seu contraste é significativamente mais alto.

Dissimilaridade

- **Resultado**

Normal:

- Imagem Otsu: 4.92634025
- Imagem Watershed: 0.
- 96h:
- Otsu: 12.57197286
- Watershed: 0.00624555

- **Conclusão**

A dissimilaridade também calcula a variação de intensidade entre pixels, a única diferença é que a dissimilaridade leva em conta a distância entre os pixels.

Homogeneidade

- **Resultado**

Normal:

- Imagem Otsu: 0.8995184
- Imagem Watershed: 1.

96h:

- Otsu: 0.78205042
- Watershed: 0.99687723

• Conclusão

A homogeneidade calcula a variabilidade na imagem. Analisando os resultados, podemos ver que o Watershed tem uma homogeneidade um pouco maior.

Energia

• Resultado

Normal:

- Imagem Otsu: 0.62188422
 - Imagem Watershed: 1.
- 96h:
- Otsu: 0.49575025
 - Watershed: 0.75590334

• Conclusão

A energia mede se existe uma concentração de intensidade em certas áreas da imagem. O Watershed, com um valor maior, indica que a energia está distribuída uniformemente.

ASM (Angular Second Moment)

• Resultado

Normal:

- Imagem Otsu: 0.38673999
 - Imagem Watershed: 1.
- 96h:
- Otsu: 0.24576831
 - Watershed: 0.57138986

• Conclusão

O Angular Second Moment mede a uniformidade da textura. O baixo valor na imagem de Otsu indica que a textura é complexa e variada.

Essas 5 últimas estatísticas foram feitas através do seguinte código:

```
import numpy as np
from skimage.feature import graycomatrix,
    graycoprops
from skimage.io import imread
# Distancia entre pixels
distances = [1]
# Angulos entre pixels
angles = [0]
glcm = graycomatrix(image, distances, angles,
    levels=256, symmetric=True, normed=True)
contrast = graycoprops(glcm, 'contrast')
dissimilarity = graycoprops(glcm,
    'dissimilarity')
correlation = graycoprops(glcm, 'correlation')
homogeneity = graycoprops(glcm, 'homogeneity')
energy = graycoprops(glcm, 'energy')
asm = graycoprops(glcm, 'ASM')
```

Correlação de Pearson

• Resultado

- Valor: -0.8496665596305685

• Conclusão

A correlação de Pearson varia de -1 a 1. Uma correlação próxima a 1 indica que as imagens são muito semelhantes em termos de padrões lineares, e -1 indica que são opostas.

Similaridade de Jaccard

• Resultado

- Valor: 2.6314404505026052e-05

• Conclusão

A similaridade de Jaccard calcula a similitude entre dois conjuntos. Ao ser aplicado em imagens, nos mostra a proporção de pixels que são iguais entre as duas imagens. Se for próximo de 1, indica que a maioria dos pixels corresponde entre as duas imagens, enquanto uma similaridade próxima de 0 indica que as imagens são muito diferentes. Podemos concluir, então, que as imagens são bastante diferentes.

O cálculo da correlação de Pearson e a similaridade de Jaccard foram feitas utilizando o seguinte código:

```
#Correlacao de Pearson
# Normalizar as imagens para ter valores
entre 0 e 1
image1_norm = cv2.normalize(image, None,
    alpha=0, beta=1,
    norm_type=cv2.NORM_MINMAX,
    dtype=cv2.CV_32F)
image2_norm = cv2.normalize(image2, None,
    alpha=0, beta=1,
    norm_type=cv2.NORM_MINMAX,
    dtype=cv2.CV_32F)

# Calcular a matriz de correlacao
corr_matrix =
    np.corrcoef(image1_norm.flatten(),
    image2_norm.flatten())

# Obter a correlacao de Pearson
pearson_corr = corr_matrix[0, 1]

#similaridade de jacard
def jaccard_similarity(set1, set2):
    # Interseo dos dois conjuntos
    intersection =
        len(set1.intersection(set2))
    # Unio dos dois conjuntos
    union = len(set1.union(set2))
    return intersection / union

# Converter as imagens em conjuntos de pixels
nicos
set1 = set(image1.flatten())
set2 = set(image2.flatten())

# Calcular a similaridade de Jaccard
similarity = jaccard_similarity(set1, set2)
print("Similaridade de Jaccard:", similarity)
```

```
MSE: 39000.009769, SSIM: 0.000082
```

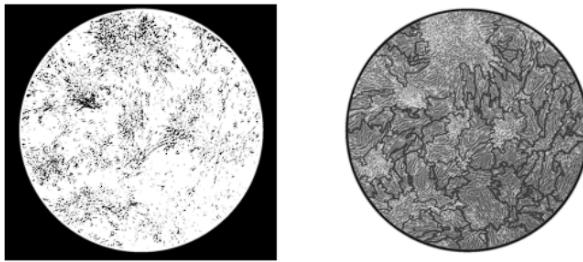


Figure 3. Valores de MSE e SSIM das duas imagens

MEAN SQUARED ERROR (MSE)

MSE é uma medida de erro quadrático médio entre duas imagens. Ele calcula a média dos quadrados das diferenças absolutas entre os pixels correspondentes das duas imagens. O MSE é sensível a pequenas alterações e variações entre as imagens. Um valor de MSE menor indica uma maior semelhança entre as imagens, enquanto um valor maior indica uma maior diferença.

STRUCTURAL SIMILARITY INDEX (SSIM)

SSIM é uma medida de semelhança estrutural entre duas imagens. O SSIM calcula a semelhança estrutural entre as imagens em três componentes: luminância, contraste e textura. O SSIM varia de -1 a 1, onde 1 indica uma perfeita semelhança estrutural, 0 indica ausência de semelhança estrutural, e valores negativos indicam semelhança estrutural negativa (ou seja, imagens opostas).

O código implementado para mostrar os valores de MSE e SSIM é mostrado a seguir:

```
def compare_images(imageA, imageB, title):
    # Calcular os valores de MSE e SSIM para
    # as imagens
    m = mse(imageA, imageB)
    s = ssim(imageA, imageB, data_range=1.0)
    diff = np.abs(imageA - imageB)
    cv2.imshow('Difference', diff)

    fig = plt.figure(title)
    plt.suptitle("MSE: %f, SSIM: %f" % (m, s))

    ax = fig.add_subplot(1, 2, 1)
    plt.imshow(imageA, cmap=plt.cm.gray)
    plt.axis("off")

    ax = fig.add_subplot(1, 2, 2)
    plt.imshow(imageB, cmap=plt.cm.gray)
    plt.axis("off")

    plt.show()
```

```
compare_images(image, image2, 'Comparison')
```

Além disso foi gerado uma imagem que apresenta a subtração da imagem de Otsu e a de Watershed, analisando ela, é possível perceber as associações distintas entre as duas imagens.

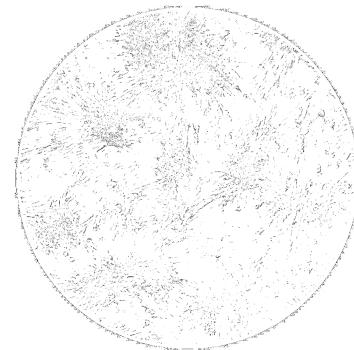


Figure 4. Diferença absoluta entre as imagens de 96h

Foi realizado o mesmo procedimento para as outras imagens, porém usando um algoritmo diferente para salientar as associações em branco.

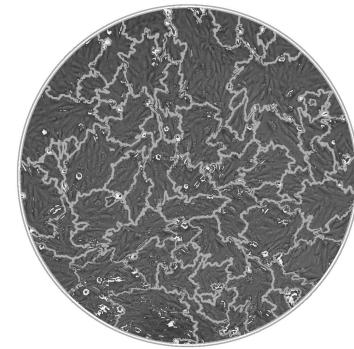


Figure 5. Diferença absoluta entre as imagens Normais

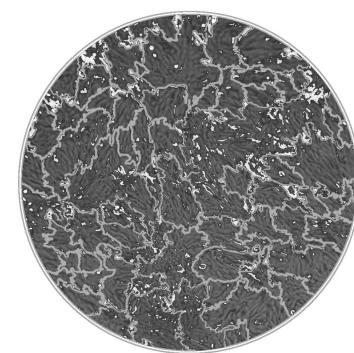


Figure 6. Diferença absoluta entre as imagens de 21h

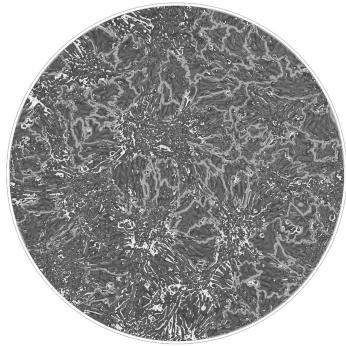


Figure 7. Diferença absoluta entre as imagens de 29h



Figure 8. Diferença absoluta entre as imagens de 44h



Figure 9. Diferença absoluta entre as imagens de 53h



Figure 10. Diferença absoluta entre as imagens de 73

Ademais os valores de todos esses parâmetros para as outras imagens podem ser encontradas nas seção de tabelas (figuras 11, 12, 13).

As funções utilizadas foram as seguintes:

```
def entropia(imagem):
    return -np.sum(np.log2(imagem + 1e-10) * 
        imagem / np.sum(imagem))

def contraste(imagem):
    return np.std(imagem)

def dissimilaridade(imagem1, imagem2):
    # Encontrar o tamanho minimo entre as
    # duas imagens
    min_dim = min(imagem1.shape[:2])

    # Redimensionar as imagens para o tamanho
    # minimo encontrado
    imagem1_resized = cv2.resize(imagem1,
        (min_dim, min_dim))
    imagem2_resized = cv2.resize(imagem2,
        (min_dim, min_dim))

    # Calcular a dissimilaridade
    return np.sum((imagem1_resized -
        imagem2_resized)**2) / (2 *
        np.prod(imagem1_resized.shape))

def homogeneidade(imagem):
    return np.sum(imagem**2) / np.size(imagem)

def energia(imagem):
    return np.sum(imagem**2)

def ASM(imagem):
    return
    np.mean(np.abs(feature.graycomatrix(imagem,
        [1], [0, np.pi/4, np.pi/2,
        3*np.pi/4])[0]))

def correlacao_pearson(imagem1, imagem2):
    min_dim = min(imagem1.shape[:2])

    imagem1_resized = cv2.resize(imagem1,
        (min_dim, min_dim))
    imagem2_resized = cv2.resize(imagem2,
        (min_dim, min_dim))
```

```

# Converter as imagens redimensionadas
# para arrays unidimensionais
imagem1_flat = imagem1_resized.ravel()
imagem2_flat = imagem2_resized.ravel()

# Calcular a correlacao de Pearson
return pearsonr(imagem1_flat,
imagem2_flat)[0]

def similaridade_jacard(imagem1, imagem2):
min_dim = min(imagem1.shape[:2])

imagem1_resized = cv2.resize(imagem1,
(min_dim, min_dim))
imagem2_resized = cv2.resize(imagem2,
(min_dim, min_dim))

# Converter as imagens redimensionadas
# para arrays binarios
imagem1_bin = (imagem1_resized >
128).astype(int) # Exemplo: considera
pixels acima de 128 como 1
imagem2_bin = (imagem2_resized >
128).astype(int) # Assumindo que ambas
as imagens sao binarias

# Calcular a interseccao e a uniao
intersection =
np.logical_and(imagem1_bin, imagem2_bin)
union = np.logical_or(imagem1_bin,
imagem2_bin)

# Calcular a similaridade de Jacard
jacard_index = np.sum(intersection) /
np.sum(union)
return jacard_index

def mse(imagem1, imagem2):
min_dim = min(imagem1.shape[:2])

imagem1_resized = cv2.resize(imagem1,
(min_dim, min_dim))
imagem2_resized = cv2.resize(imagem2,
(min_dim, min_dim))

# Calcular o erro quadratico medio (MSE)
return np.mean((imagem1_resized -
imagem2_resized)**2)

def ssim_color(imagem1, imagem2):
# Converter imagens coloridas para
grayscale se necessario
if len(imagem1.shape) == 3:
    imagem1_gray = cv2.cvtColor(imagem1,
cv2.COLOR_BGR2GRAY)
    imagem2_gray = cv2.cvtColor(imagem2,
cv2.COLOR_BGR2GRAY)
else:
    imagem1_gray = imagem1
    imagem2_gray = imagem2

# Verificar se as imagens tem o mesmo
tamanho
if imagem1_gray.shape!=
imagem2_gray.shape:
    # Redimensionar as imagens para o
tamanho da imagem menor
    min_dim = min(imagem1_gray.shape)
    imagem1_gray =
cv2.resize(imagem1_gray, (min_dim,
min_dim))
    imagem2_gray =
cv2.resize(imagem2_gray, (min_dim,
min_dim))

# Calcular a SSIM
return ssim(imagem1_gray, imagem2_gray)

```

IV. CONCLUSÃO

É evidente que a presença do vírus induz mudanças significativas na colônia. Essas alterações não são observáveis apenas em termos de morfologia, mas também em aspectos estatísticos e nos parâmetros de processamento de imagens, como textura, contraste e entropia. Tais mudanças permitem prever a evolução da contaminação, possibilitando a classificação da presença do vírus com base no estado da colônia e a avaliação do seu progresso.

V. TABELAS

Imagen	Entropia	Contraste	Homogeneidade	Energia	ASM
Normal	-7.619774717092037	60.15280616772569	76.47092175284146	69837834	14.21484375
21h	-7.64607293876376	58.9622079232108	76.31118191971618	69691950	14.5625
29h	-7.644607828428793	61.51772614478523	71.74843965573878	65524980	26.7421875
44h	-7.660999480264698	56.41683596163225	74.95780391126294	68455964	15.66796875
53h	-7.654666791655011	56.475505660700485	75.37632328143135	68838181	15.4765625
73h	-7.623768341018702	62.07008884392726	72.74156100124827	66431958	27.25390625

Figure 11. Dados calculados para a imagem após a técnica de Otsu

Imagen	Entropia	Contraste	Homogeneidade	Energia	ASM
Normal	-7.994353436859417	122.51914524559938	0.6383879867738377	14872433	32879.6123046875
21h	-7.994353436859423	123.83671180327659	0.6189935242764087	14420603	34644.5732421875
29h	-7.994353436859423	126.31526435298508	0.5680032103902776	13232689	39284.8623046875
44h	-7.994353436859432	125.80553858325715	0.5812452547245002	13541187	38079.7919921875
53h	-7.994353436859435	125.09537882615095	0.5966486636651744	13900038	36678.0302734375
73h	-7.994353436859425	124.19889103854278	0.6130395449068321	14281894	35186.4052734375

Figure 12. Dados calculados para a imagem após a técnica de Watershed

Imagen	Dissimilaridade	Correlação de Pearson	Similaridade de Jaccard	MSE	SSIM Value
Normal	38.440876636827454	-0.8017967864389869	0.5137306254578119	76.88175327365491	0.396008665117582
21h	38.25622237944359	-0.7812605359876894	0.5067710860599661	76.51244475888718	0.39331008545280355
29h	36.17682282521788	-0.7819390042761161	0.45765892834395644	72.35364565043577	0.22875296459663255
44h	37.77367497964849	-0.7965787807860685	0.5018672894132292	75.54734995929698	0.2846393306483424
53h	38.01362823465692	-0.8013556146365057	0.5181933152120137	76.02725646931384	0.34091511432676075
73h	36.61037173886072	-0.8045671819391293	0.4916583942712429	73.22074347772144	0.31909774976046584

Figure 13. Parâmetros que comparam as duas imagens