Segmentação de imagem com KMeans no processo de automatização de análise de viabilidade celular

1. Introdução

O teste Live/Dead de viabilidade celular á baseado no princípio de permeabilidade da membrana. É muito usado para identificar a potencialidade de dano celular frente a diferentes formulações. Para quantificar as células vivas e mortas o devemos separar as células de outros “ruídos” na imagem. No caso em questão pode ser considerado ruído: sujeira do ambiente, material usado para armazenamento da célula entre outros. Depois é quantificar as células totais, depois as mortas, marcadas com um reagente para mudar sua cor na imagem, assim temos a quantidade de células vivas e mortas. O cálculo de viabilidade celular se dá, portanto, através do quociente de células vivas pelas células totais multiplicado por 100.

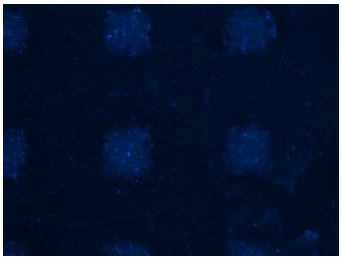
Existem softwares que fazem o trabalho de quantificar as células em amostras porém são inviáveis para o nosso caso, pois o trabalho para contagem de células é muito grande para uma única imagem e como a pesquisa cientifica é no sentido de desenvolver uma plataforma para que de uma única vez seja possível testar várias culturas de células, podemos ter dezenas e até centenas de culturas de células em um único experimento. O que torna o trabalho manual cansativo, estressante e suscetível a erros, já que é necessário fazer o *upload* de cada imagem de forma individualizada, identificar as áreas da imagem que não são de interesse, como impurezas, selecionar as áreas a serem analisadas e então realizar o comando de contagem.

Diante do exposto, foi definido como objetivo a automatização da tarefa, usando KMeans no processo para encontrar de forma automática os microambientes.

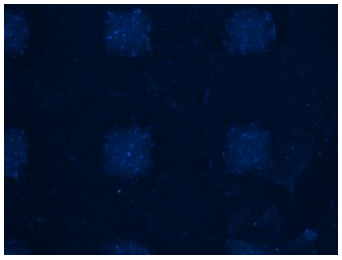
2. Dataset

O conjunto de dados consiste em 189 imagens que são recortes, de alta definição, da imagem [a], tiradas pelo microscópio e serão usadas para o treinamento do nosso algoritmo. Temos 3 grupos de imagens para cada quadrante. Tomando como exemplo o quadrante Area_1_9 temos uma imagem Area_1_9.tiff [c] que é a foto das células vivas e mortas com destaque do reagente que marca as células mortas, Area1_9_DAPI.tiff [b] que é a foto das células vivas e mortas sem o destaque do reagente e a imagem Area1_9_GFP.tiff [d] que é a foto somente das células que foram marcadas pelo reagente que identifica as células mortas.

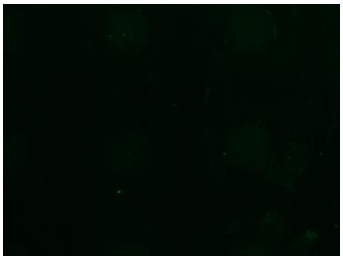
a - Foto Inteira tirada pelo microscópio.



b – Area1_9_DAPI.tiff



c – Area1_9.tiff



d - Area1_9_GFP.tiff

3. Metodologia

Nosso primeiro passo será identificar os microambientes usando o KMeans. Para isso vamos usar as imagens do grupo [c] Area_1_x.tiff por ter os microambientes bem definidos.
É sabido que ao trabalhar com imagem, recomenda-se reduzir a dimensão para ganhar em processamento. Partindo dessa premissa trabalhamos com a imagem normal e redimensionadas em 50% e 25%.
A métrica usada para avaliar o desempenho do algoritmo é a quantidade de células mortas reconhecidas corretamente e o tempo de execução para cada grupo de imagem original, reduzida em 50% e 25%.

3.1. Melhorando a imagem

Um consenso nos artigos usados como parâmetro para construção desse algoritmo é um método de melhoramento de contraste que deve ser implementado antes da segmentação, pois as imagens médicas possuem baixo nível de contraste. Foi usado dois algoritmos nos testes para obter o melhor.

Contrast Stretching [5] – aumenta o contraste e controla a curva de função, M é a linha média que usamos para mudar os valores escuros para valores claros. np.spacing(1) é uma constante que é a distância entra 1 e o próximo maior numero.

```
def contrast_stretching(img, E):

    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _mean = img.reshape((img.shape[0]*img.shape[1], 1)).mean(axis=0)
    M = _mean
    cs = 1 / ( 1 + (M / (img + np.spacing(1))) ** E)
    return np.float32(cs)
```

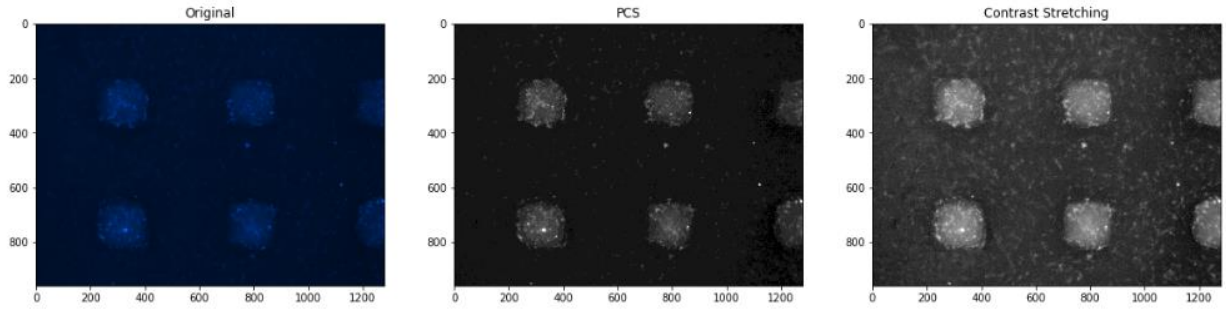
Partial Contrast Stretching [9] – Função de mapeamento linear que é usada para melhorar o contraste. Essa técnica é baseada no brilho e contraste original da imagem para fazer o ajuste.

r_max, b_max e g_max são os valores máximos de cada canal de cor Vermelho, azul e verde respectivamente. Assim como r_min, b_min e g_min são os valores mínimos de cada canal de cor. minTH é a média dos valores mínimos e maxTH a média dos valores máximos.

```
def pcs(img, tx = 0.1):

    r, g, b = cv2.split(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    r_min = r.reshape((r.shape[0]*r.shape[1], 1)).min(axis=0)
    g_min = g.reshape((g.shape[0]*g.shape[1], 1)).min(axis=0)
    b_min = b.reshape((b.shape[0]*b.shape[1], 1)).min(axis=0)
    r_max = r.reshape((r.shape[0]*r.shape[1], 1)).max(axis=0)
    g_max = g.reshape((g.shape[0]*g.shape[1], 1)).max(axis=0)
    b_max = b.reshape((b.shape[0]*b.shape[1], 1)).max(axis=0)
    minTH = (r_min + g_min + b_min)/3
    maxTH = (r_max + g_max + b_max)/3
    NminTH = minTH - minTH * tx
    NminTH = 0 if NminTH < 0 else NminTH
    NmaxTH = maxTH + (255 - maxTH) * tx
    NmaxTH = 255 if NmaxTH > 255 else NmaxTH
```

```
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _min = img.reshape((img.shape[0]*img.shape[1], 1)).min(axis=0)
    output = np.zeros_like(img)

    for x in range(img.shape[0]):
        for y in range(img.shape[1]):

            if img[x, y] > minTH :
                output[x,y] = (img[x,y]/minTH) * NminTH
            elif minTH <= img[x, y] and img[x, y] <= maxTH :
                output[x,y] = (((NmaxTH - NminTH)/(maxTH - minTH)) * (img[x,y
] - _min)) + _min
            elif img[x, y] < maxTH :
                output[x,y] = (img[x,y]/maxTH) * maxTH

    return output
```

Após alguns testes obtivemos que a função PCS obteve melhor resultado para melhorar o contraste de imagens coloridas, que é o nosso caso. Reduzindo o ruído do que não é microambiente na imagem.



Depois foi usada mais uma função para melhorar a imagem, antes de submeter a segmentação do KMeans. A função log transformation [4] pode ser usada para clarear as intensidades de uma imagem (como a Transformação Gama, onde gama <1). Mais frequentemente, é usado para aumentar o detalhe (ou contraste) de valores de intensidade mais baixos.

```
def log_transformation(constant, image):

    return constant * (np.log(1 + np.float32(image)))
```

Observando que, em comparação com outros métodos testados e com a imagem processada apenas pelo PCS, tivemos uma redução no tempo de segmentação da imagem pelo KMeans quando usamos a função log_transformation. Esse fator é muito importante já que iremos processar muitas imagens.

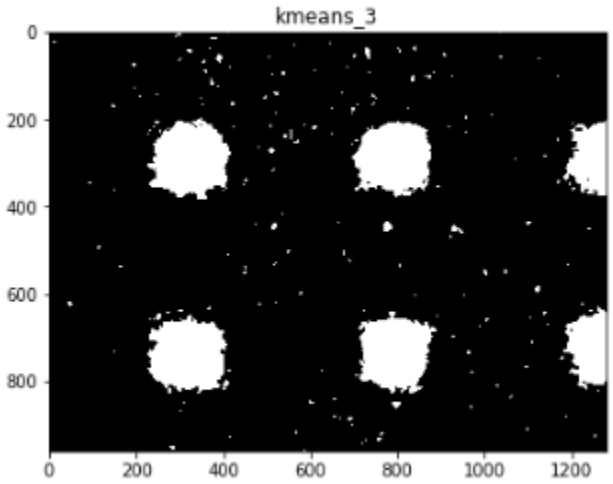
3.2. KMeans

O passo seguinte foi usar a biblioteca do Sklearn para segmentar a imagem pré processada e usar o resultado para criar uma mascara que será usada para encontrar os microambientes.

Para isso criamos a seguinte função

```
def kmeans(image):
    shape_2 = 1 if len(image.shape) == 2 else image.shape[2]
```

```
    reshaped = image.reshape(image.shape[0] * image.shape[1], shape_2)
    kmeans = KMeans(n_clusters=2, n_init=40, max_iter=500).fit(reshaped)
    clustering = np.reshape(np.array(kmeans.labels_, dtype=np.uint8), (ima
ge.shape[0], image.shape[1]))

    all_black = True
    for i in range(len(clustering)):
        if clustering[i][0] == 1:
            all_black = False
            break

    if(not all_black):
        ret,thresh2 = cv2.threshold(clustering,127,255,cv2.THRESH_BINARY_I
NV)
        clustering = thresh2

    return clustering
```



e – Resultado da segmentação de uma imagem do dataset

3.3. Identificando microambientes

Nessa etapa, foi usado visão computacional, com a biblioteca openCV, para identificar os microambientes. Usamos o retorno do método Kmeans para criar uma elipse para cada microambiente, destacados em branco e a partir deles criar mascaras que pudessem remover da imagem original somente os microambientes.

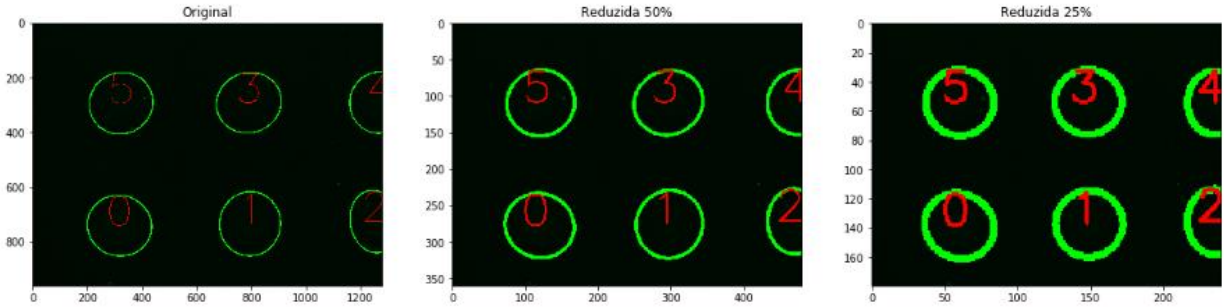
Segue abaixo o código da função.

```
def find_microenvironments(img2print, mask, type_img=1 ):
    mask = mask.copy()
    img2print = img2print.copy()
    _mask = np.zeros_like(img2print)
    microenvironments = []
```

```
    if type_img == 1:
        area_min = 3000
        count_cnt_min = 100
        plus = 50
        max_r = 200
        text_size = 5
    elif type_img == 2:
        area_min = 1500
        count_cnt_min = 50
        plus = 25
        max_r = 100
        text_size = 2
    elif type_img == 3:
        area_min = 300
        count_cnt_min = 10
        plus = 12
        max_r = 50
        text_size = 1


    _, contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL,cv2.CHA
IN_APPROX_SIMPLE)
    for cnt in contours:
        area = cv2.contourArea(cnt)

        if area < area_min:
            continue

        if len(cnt) < count_cnt_min:
            continue
        '''
        if len(microenvironments) > 0 :
            continue
        '''


        _e = list(cv2.fitEllipse(cnt))

        #excluindo raios muito grandes
        if(_e[1][0] > max_r or _e[1][1] > max_r):
            continue

        #aumentando os raios da elipse para exluir algum erro.
        #print(area, len(cnt), _e[1])
        _e[1] = (_e[1][0] + plus, _e[1][1] + plus)
        ellipse = tuple(_e)

        mask = np.zeros_like(img2print)
        cv2.ellipse(mask, ellipse, (255,255,255), -1)
        masked = np.bitwise_and(img2print, mask)
        microenvironments.append(masked)
        cv2.ellipse(img2print, ellipse, (0,255,0), 4)
        cv2.putText(img2print,
                    str(len(microenvironments) - 1),
                    (int(ellipse[0][0]) - plus, int(ellipse[0][1])),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    text_size,
```

```
                    255,
                    2)
        cv2.ellipse(_mask, ellipse, (255,255,255), -1)


    return (microenvironments, img2print, _mask)
```

Após comparar os resultados, identificamos que o algoritmo usando Kmean para segmentar e identificar os microambientes obteve sucesso em ambos os grupos de imagens, tanto na imagem original quanto nas reduzidas.



d – Comparação entre imagem original e reduzidas

3.4. Identificando células mortas

No diagnóstico de células mortas, é usado um reagente que dá destaque às células, como visto anteriormente. A função anterior nos forneceu uma imagem para cada microambiente, com isso é possível agora usar também visão computacional para identificar os pontos que se destacam em cada microambiente.

Essa função irá identificar as células mortas em cada microambiente e armazenar em um dataframe, para que possa dar um retorno mais prático, um arquivo csv por exemplo, para o pesquisador que for usar o algoritmo em analises futuras.

A seguir o algoritmo que identifica as células mortas em cada microambiente.

```
import pandas as pd
df = pd.DataFrame(columns = ['MICROAMBIENTE', 'me 0', 'me 1', 'me 2','me 3',
'me 4','me 5', 'me 6','me 7', 'me 8','me 9'])


for k, me in enumerate(microenvironments):

    nome = 'Original'
    if(k == 1):
        nome = 'Reduzido 50%'
    elif k == 2:
        nome = 'Reduzido 25%'

    row = ['{}'.format(nome), '-','-','-','-','-','-','-','-','-','-']
    for i, m in enumerate(me):
```

```
        _masked = cv2.cvtColor(m, cv2.COLOR_BGR2GRAY)
        #blur = cv2.medianBlur(_masked,5)
        ret,thresh1 = cv2.threshold(_masked,20,255,cv2.THRESH_TOZERO) #THRESH
_TOZERO     THRESH_BINARY

        _, contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        cells = 0
        for cnt in contours:
            area = cv2.contourArea(cnt)
            #print(k, cnt.shape)
            if(cnt.shape[0] < 5):
                continue

            #print(k, i)
            ellipse = cv2.fitEllipse(cnt)
            cv2.ellipse(m, ellipse, (0,255,0), 4)
            #plt.imshow(m)
            cells += 1

        row[i+1] = cells
        #print('{} Microambiente {}; {} células mortas encontradas'.format(k,
i, cells))

    df.loc[len(df)] = row
```

Após executar o algoritmo, foi possível observar que o mesmo consegue identificar as células mortas nos microambientes gerados a parti da imagem original, as imagens reduzidas não tiveram o mesmo sucesso em encontrar as células mortas.

	MICROAMBIENTE	me 0	me 1	me 2	me 3	me 4	me 5	me 6	me 7	me 8	me 9
0	Original	0	1	0	0	0	1	-	-	-	-
1	Reduzido 50%	0	0	0	0	0	1	-	-	-	-
2	Reduzido 25%	0	0	0	0	0	0	-	-	-	-

4. Resultados

O algoritmo acima, foi testado com varias imagens individualmente para ajustar alguns parâmetros. Depois dos testes, todas as imagens do dataset foram submetidas ao algoritmo.

Após o retorno do algoritmo, considero o objetivo alcançado pois como premissa tínhamos a automatização dessa tarefa, que se mostrou possível fazendo uso da segmentação de imagens com KMeans e com visão computacional, com ajuda da biblioteca openCV.

Abaixo segue a tabela com a contagem de todas as células mortas em cada imagem do dataset.

MICROAMBIENTE	me 0	me 1	me 2	me 3	me 4	me 5	me 6
1	0	-	-	-	-	-	-
2	0	1	0	0	0	0	-
3	14	5	0	0	0	4	-
4	-	-	-	-	-	-	-
5	0	1	0	-	-	-	-
6	0	0	0	-	-	-	-
7	0	-	-	-	-	-	-
8	0	0	0	1	0	-	-
9	0	0	2	1	0	0	3
10	0	0	0	0	3	0	-
11	0	0	0	0	1	1	0
12	0	0	0	0	0	0	1
13	0	1	0	0	2	3	-
14	0	0	2	-	-	-	-
15	1	0	1	0	1	-	-
16	2	0	0	0	-	-	-
17	2	0	4	0	-	-	-
18	0	1	0	0	0	-	-
19	0	0	0	0	-	-	-
20	-	-	-	-	-	-	-
21	0	0	1	-	-	-	-
22	0	0	0	0	0	-	-
23	1	0	0	0	-	-	-
24	2	0	0	1	-	-	-
25	0	0	0	0	1	0	-

26	0	0	0	0	0	-	-
27	0	0	0	0	0	1	-
28	0	0	1	0	-	-	-
29	0	0	0	0	1	0	-
30	3	1	0	-	-	-	-
31	-	-	-	-	-	-	-
32	0	0	0	0	-	-	-
33	0	0	0	1	0	0	-
34	0	0	0	0	1	0	-
35	0	0	-	-	-	-	-
36	1	1	1	0	-	-	-
37	-	-	-	-	-	-	-
38	-	-	-	-	-	-	-
39	1	1	0	0	-	-	-
40	1	1	0	0	1	0	-
41	-	-	-	-	-	-	-
42	0	0	0	0	-	-	-
43	0	0	1	-	-	-	-
44	1	1	0	0	0	-	-
45	2	0	1	0	1	0	-
46	0	1	0	0	1	0	-
47	1	0	5	0	0	2	-
48	-	-	-	-	-	-	-
49	-	-	-	-	-	-	-
50	0	0	0	0	0	1	-
51	1	0	0	0	0	0	-
52	0	0	0	1	-	-	-
53	0	1	0	0	0	0	-
54	-	-	-	-	-	-	-
55	0	0	0	0	0	0	-
56	1	0	0	0	-	-	-
57	1	-	-	-	-	-	-
58	0	0	0	-	-	-	-
59	4	0	0	-	-	-	-
60	0	0	-	-	-	-	-
61	0	0	1	0	3	-	-
62	0	1	2	1	0	0	-
63	-	-	-	-	-	-	-

5. Referencias

1. https://www.researchgate.net/publication/283185016_Image_Segmentation_Using_K_-means_Clustering_Algorithm_and_Subtractive_Clustering_Algorithm
2. https://docplayer.net/22673363-Colour-image-segmentation-approach-for-detection-of-malaria-parasites-using-various-colour-models-and-k-means-clustering.html
3. https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html
4. https://medium.com/@sonu008/image-enhancement-contrast-stretching-using-opencv-python-6ad61f6f171c
5. http://www.cs.uregina.ca/Links/class-info/425/Lab3/
6. https://www.researchgate.net/publication/224227539_Improving_colour_image_segmentation_on_acute_myelogenous_leukaemia_images_using_contrast_enhancement_techniques
7. http://www.indjst.org/index.php/indjst/article/viewFile/111353/78453
8. https://www.researchgate.net/publication/263618841_Colour_Image_Segmentation_Using_Unsupervised_Clustering_Technique_for_Acute_Leukemia_Images
9. http://interscience.in/IJPPT_Vol2Iss1/29-34.pdf