

## **Identificação, Contagem e Medição de Nanopartículas por Processamento Digital de Imagens**

### **1 Introdução**

Este trabalho tem como objetivo segmentar fotos de estruturas microscópicas para a identificação de nanopartículas.

#### **1.1 Representação digital de imagens**

Neste trabalho lidaremos com imagens em 256 níveis de cinza. Em processamento digital de imagens, elas são representadas como matrizes, onde cada valor vai de 0 a 255.

O Scilab não oferece suporte para ler imagens JPG em uma matriz, então utilizamos a biblioteca SciCV. Para instalar, utilizamos o seguinte comando:

```
--> atomsInstall("scicv")
```

Em seguida, reiniciamos o Scilab.

##### **1.1.1 Leitura de uma imagem em uma matriz**

```
--> scicv_Init()  
--> X = double(imread("caminho/para/imagem.jpg", CV_LOAD_IMAGE_GRAYSCALE)(:,:));
```

Onde a função `double` converte para matriz de reais.

Então, `X` será uma matriz como qualquer outra no Scilab. Podemos plotá-la:

```
--> matplot(X);
```

##### **1.1.2 Histograma**

O histograma de uma imagem é um vetor que representa a frequência de cada cor (no nosso caso, nível de cinza) na imagem.

```
function H = histograma(X)  
    // vetor de 256 elementos  
    H = [];  
    for cor = 0:255  
        // nº de pixels com esta cor  
        H(cor + 1) = size(find(X == cor), 2);  
    end  
endfunction
```

Visualizar o histograma de uma imagem nos permite inferir algumas propriedades.

```
--> plot(histograma(X));
```

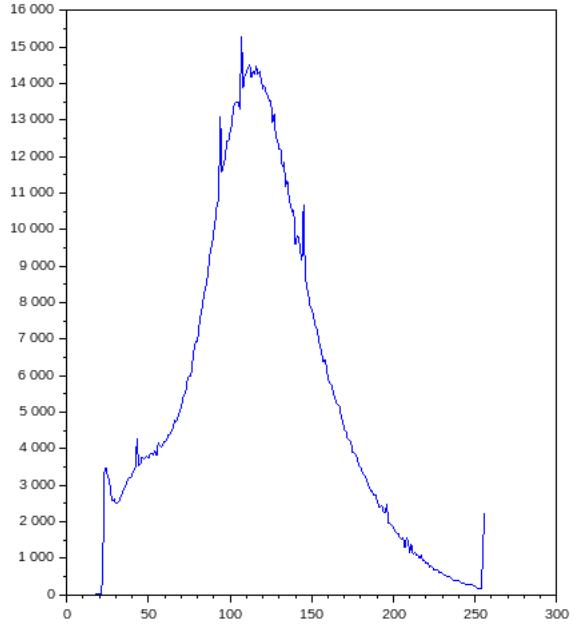
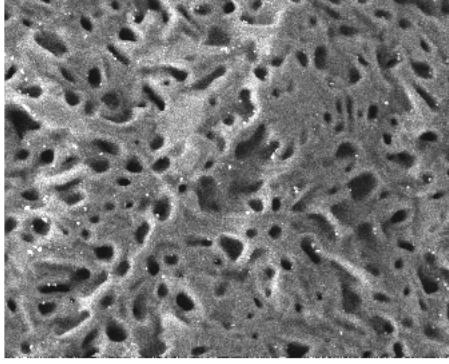


Figura 1: Plotagem de uma imagem e um histograma.

### 1.1.3 Vizinhanças de um pixel

Seja  $i$  a linha, e  $j$  a coluna, na matriz, de algum pixel. Então um pixel vizinho a  $(i, j)$  tem as coordenadas  $(i + a, j + b)$ , para algum  $a$  e  $b$ . Quais, e quantos  $a$  e  $b$ , dependem das regiões de interesse.

Geralmente, queremos explorar uma vizinhança em algum raio  $r$ , ou seja, para cada pixel numa posição  $(i, j)$ , queremos saber quais são os pixels nas posições  $i - r : i + r$  e  $j - r : j + r$ .

Podemos representar a vizinhança dos elementos de uma matriz por vários deslocamentos da matriz nos eixos  $i$  e  $j$ . Por exemplo, em uma vizinhança de raio 1, deslocamos a matriz 1 pixel nas direções leste, oeste, norte, sul, sudeste, sudoeste, nordeste e noroeste.

```

function V = vizinhos(X, raio)
    [linhas, colunas] = size(X);

    // cópia de X com bordas adicionais
    Y = zeros(linhas+raio*2, colunas+raio*2);
    Y(raio+1:linhas+raio, raio+1:colunas+raio) = X;

    // lista de matrizes, cada uma é o deslocamento de X
    // nos eixos i e j, de -raio até +raio
    V = list();
    for i = 1:raio*2+1
        for j = 1:raio*2+1
            k = size(V) + 1;
            // k-ésima matriz
            V(k) = Y(i:linhas+i-1, j:colunas+j-1);
        end
    end

```

```

end
endfunction

```

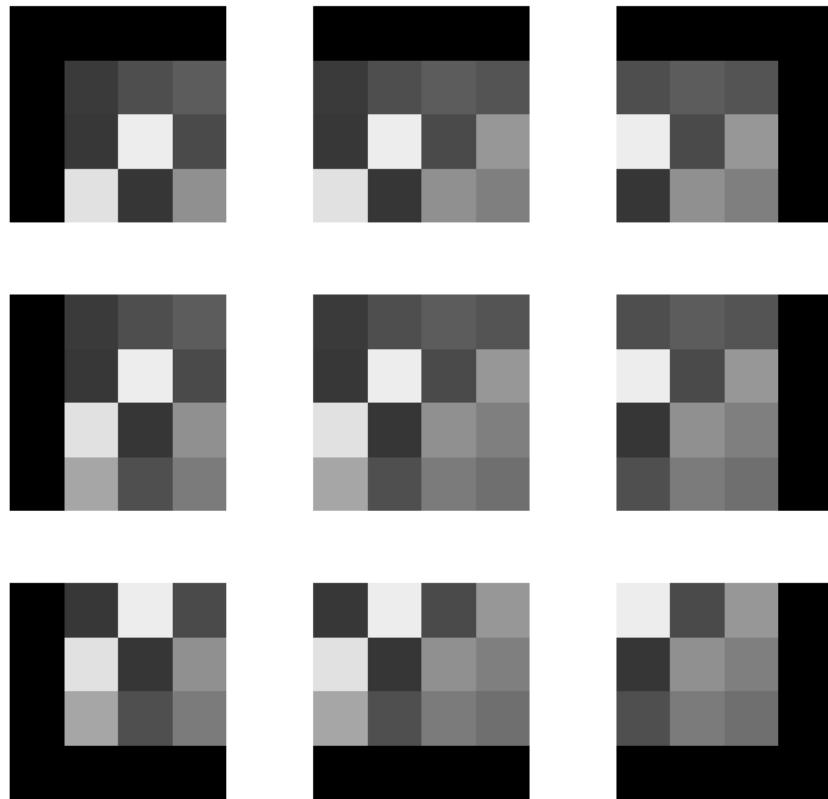


Figura 2: Matrizes de vizinhança de raio 1 de uma imagem gerada com o comando `X = rand(4,4) * 255.`

## 1.2 Pré-processamento de imagens

Cada imagem tem diferenças sutis que tornam o processamento difícil. O pré-processamento busca realçar as características comuns das fotos para que os procedimentos funcionem.

### 1.2.1 Filtro de média

O filtro de média tem o objetivo de eliminar ruídos das imagens.

```

function Y = filtro_media(X, raio)
    V = vizinhos(X, raio);
    Y = zeros(size(X, 1), size(X, 2))
    for k = 1:size(V)
        // somo todos os elementos da matriz
        // com seus elementos da k-ésima vizinhança
        Y = Y + V(k);
    end
    // divido pela quantidade de somas
    // para assim obter a média

```

```

Y = Y / size(V);
endfunction

```

### 1.2.2 Erosão

Cada pixel assume o valor mínimo dos vizinhos. O que acontece na prática é que regiões claras terão a área reduzida, ou “erodidas”. Se o raio escolhido for maior que o de alguma região clara, ela irá sumir.

```

function Y = erodir(X, raio)
    // cada elemento i,j em Y é o mínimo dentre os elementos i,j
    // correspondentes nas matrizes de vizinhança
    Y = min(vizinhos(X, raio));
endfunction

```

### 1.2.3 Dilatação

Cada pixel assume o valor máximo dos vizinhos. O que acontece na prática é que regiões claras terão a área aumentada, ou “dilatadas”. Se o raio escolhido for maior que o de alguma região escura, ela irá sumir.

```

function Y = dilatar(X, raio)
    // cada elemento i,j em Y é o máximo dentre os elementos i,j
    // correspondentes nas matrizes de vizinhança
    Y = max(vizinhos(X, raio));
endfunction

```

### 1.2.4 Abertura

É uma dilatação de uma erosão. A dilatação restaura a erosão, exceto pelas regiões que sumiram. Ou seja, a abertura elimina apenas regiões escuras menores que o raio, e o restante da imagem permanecerá o mesmo.

```

function Y = abertura(X, raio)
    Y = dilatar(erodir(X, raio), raio);
endfunction

```

### 1.2.5 Fechamento

É uma erosão de uma dilatação. A erosão restaura a dilatação, exceto pelas regiões que sumiram. Ou seja, o fechamento elimina apenas regiões claras menores que o raio, e o restante da imagem permanecerá o mesmo.

```

function Y = fechamento(X, raio)
    Y = erodir(dilatar(X, raio), raio);
endfunction

```

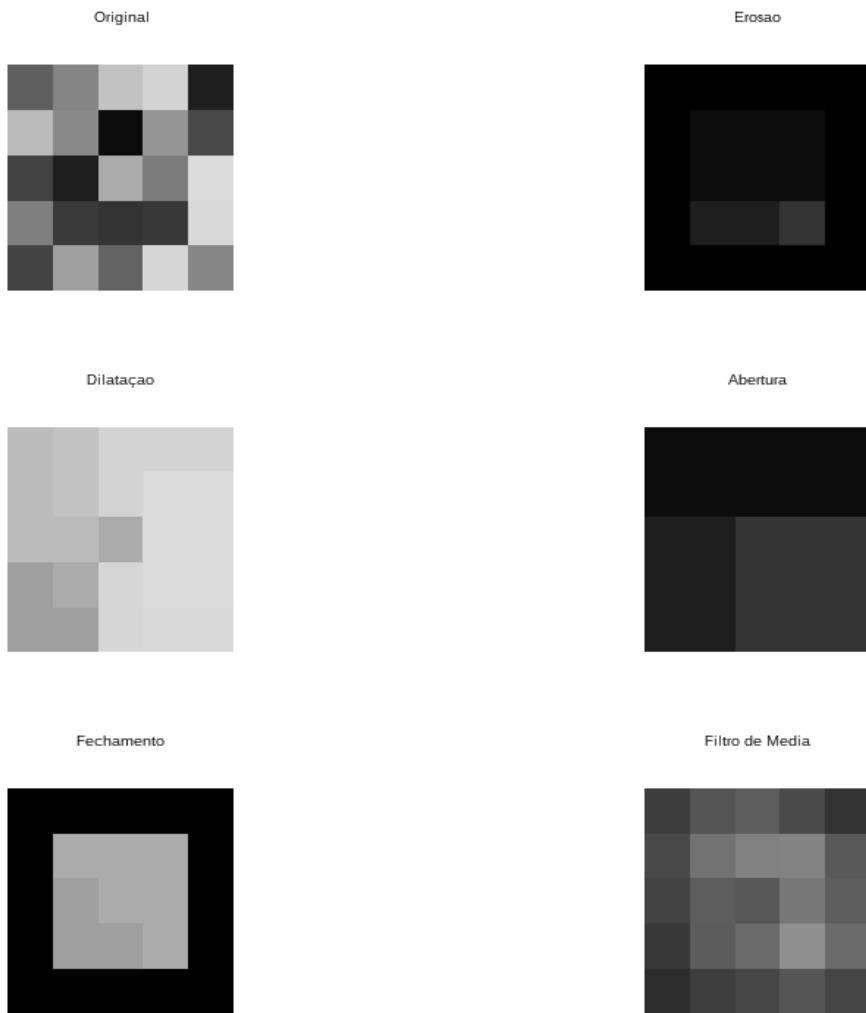


Figura 3: Exemplos de aplicações das funções para uma imagem aleatória 5 por 5.

### 1.3 Segmentação de imagem

Segmentar uma imagem é particioná-la baseado em mudanças bruscas nas cores. A seguir, apresentamos as técnicas utilizadas para segmentar as nanopartículas.

#### 1.3.1 Transformada *Top-Hat*

É uma técnica utilizada para a extração de objetos claros e pequenos em uma imagem. É a subtração da abertura.

Como a abertura elimina pequenos objetos claros, a subtração da imagem original por ela nos dará apenas os objetos pequenos, no caso, as nanopartículas.

```
function Y = top_hat(X, raio)
    Y = X - abertura(X, raio);
endfunction
```

### 1.3.2 Limiarização

Dada uma imagem (matriz)  $X$  em 256 níveis de cinza, um limiar  $L$ , A imagem limiarizada  $Y$  será tal que cada *pixel* de  $Y$  terá valor máximo se o *pixel* correspondente de  $X$  estiver acima de  $L$ , do contrário, terá valor mínimo.

```
function Y = limiarizar(X, L)
    Y = X > L;      // matriz de T ou F, se cada pixel > L
    Y = bool2s(Y); // convertemos para 1 e 0
    Y = Y * 255;   // valores 1 se tornam 255
endfunction
```

Uma forma de escolhermos limiares adequados para a segmentação de imagens é a inspeção do histograma da imagem.

### 1.3.3 Rotulamento de componentes conectados

Para identificarmos e contarmos as nanopartículas, percorremos os pixels brancos da imagem limiarizada e pintamos elementos conexos de cores únicas. No começo, pintamos cada pixel branco de uma cor única. Em seguida, cada pixel de maior valor é propagado para vizinhos de menor valor. Repetimos isto até que a matriz não mude mais.

```
function Y = rotular(X)
    [linhas, colunas] = size(X);
    // Y é cópia de X
    Y = X;

    // vetores de linhas e colunas em Y com pixels brancos
    [i_b, j_b] = find(Y > 0);
    // nº de pixels brancos
    n_b = size(i_b, 2);

    // para cada k pixel branco
    for k = 1:n_b
        // k-ésima linha e coluna do pixel branco
        i = i_b(k);
        j = j_b(k);
        // k é único, então usamos a cor "k" para "pintarmos" o pixel
        Y(i,j) = k;
    end

    // loop infinito
    while %t
        // valor máximo dos vizinhos
        Z = zeros(linhas, colunas);
        Z(find(Y > 0)) = dilatar(Y, 1)(find(Y > 0));
        // se nada mudar, paramos
        if isequal(Z, Y) then
            break;
        end
        Y = Z;
```

```

end
endfunction

```

Podemos plotar a imagem rotulada e cada objeto terá uma cor única.

```

--> Y = rotular(X);
--> matplot(Y);

```

Para sabermos quantas “cores” usamos, ou seja, quantos objetos foram segmentados:

```
--> size(unique(Y), 1)
```

## 2 Metodologia

O objetivo do projeto pode ser condensado na seguinte função:

```

function projeto(arquivo, raio_filtro_media, raio_top_hat, L)
    X = double(imread(arquivo, CV_LOAD_IMAGE_GRAYSCALE)(:,:));
    X = filtro_media(X, raio_filtro_media);
    X = top_hat(X, raio_top_hat);
    X = limiarizar(X, L);
    imwrite(arquivo + '_particulas.png', X);
    X = rotular(X);
    contagem = size(unique(X));
    disp('numero de particulas: ' + contagem);
endfunction

```

Por tentativa e erro, precisamos:

1. Determinar o raio do filtro de média;
2. Determinar o raio máximo das partículas para o *top hat*;
3. Determinar o limiar  $L$ ;
4. Apresentar os resultados.

### 3 Resultados

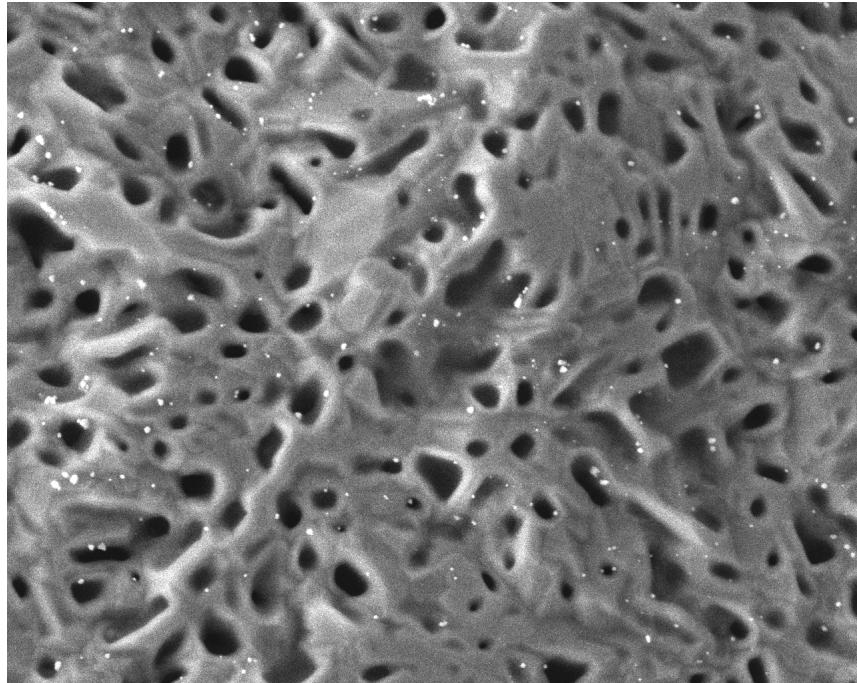


Figura 4: Imagem original.

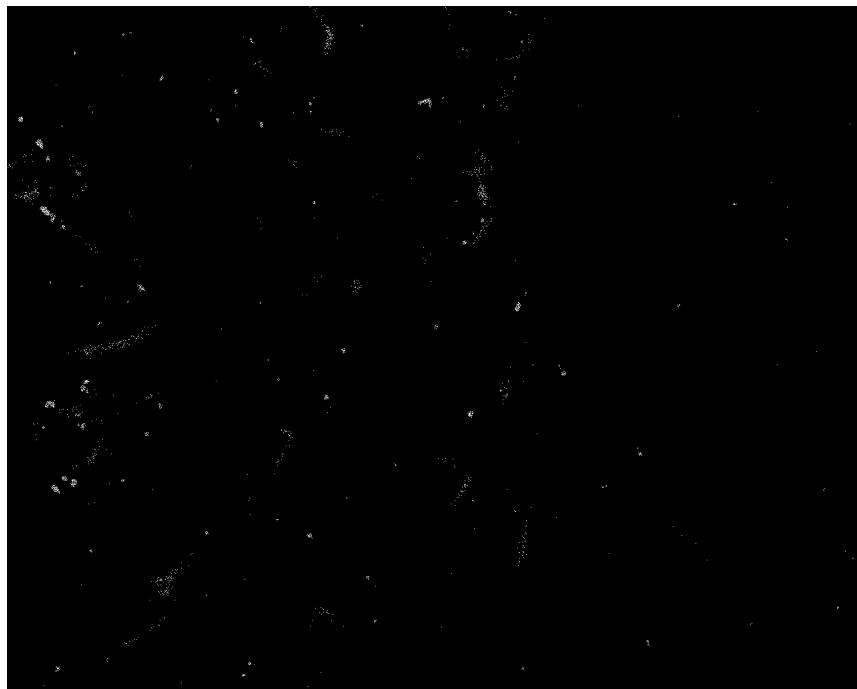


Figura 5: Imagem original limiarizada com  $L = 254$ . Além de não capturar todas as nanopartículas, ruídos (regiões que não são nanopartículas) estão inclusos.



Figura 6: Imagem passada pelo filtro de média com vizinhança 1, transformada *top-hat* de vizinhança 3, e limiarização com  $L = 50$ .

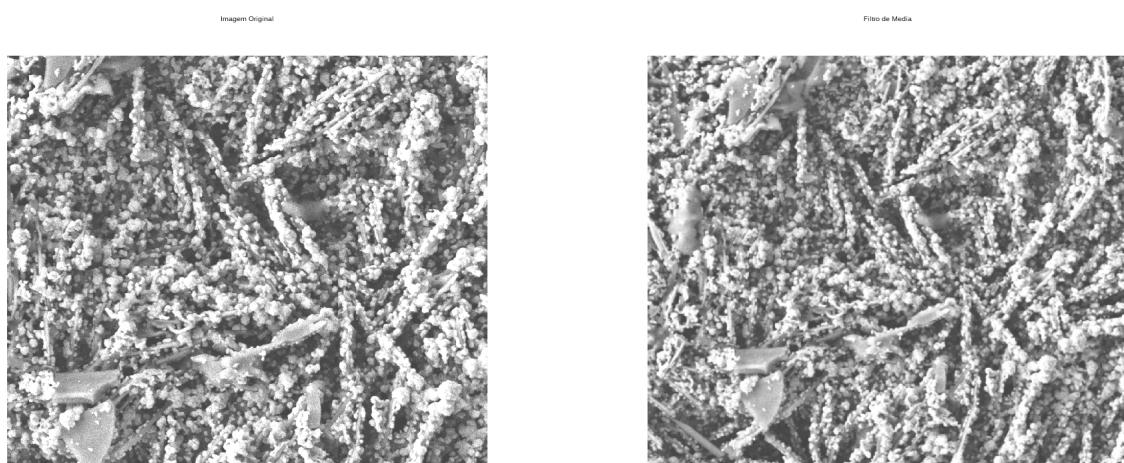


Figura 7: Filtro de média - redução de ruído

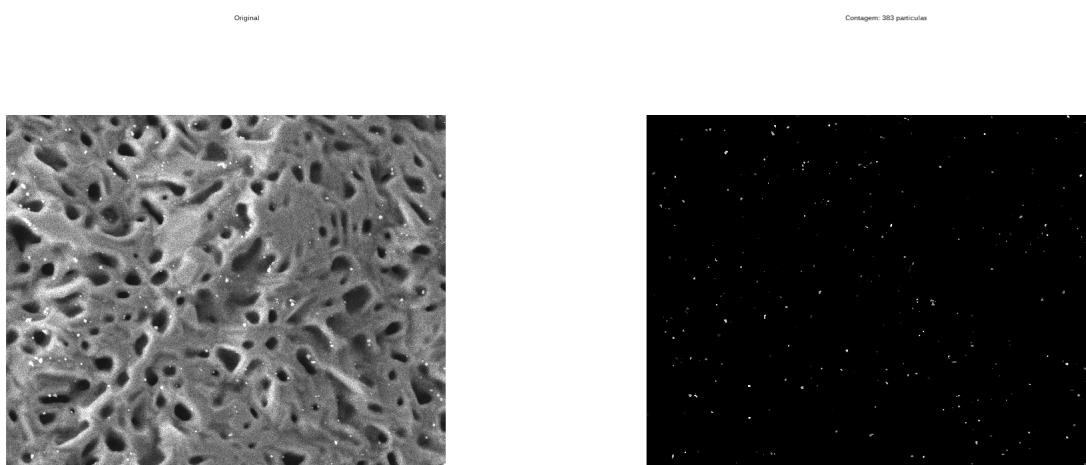


Figura 8: Contagem de Partículas 1 = 383

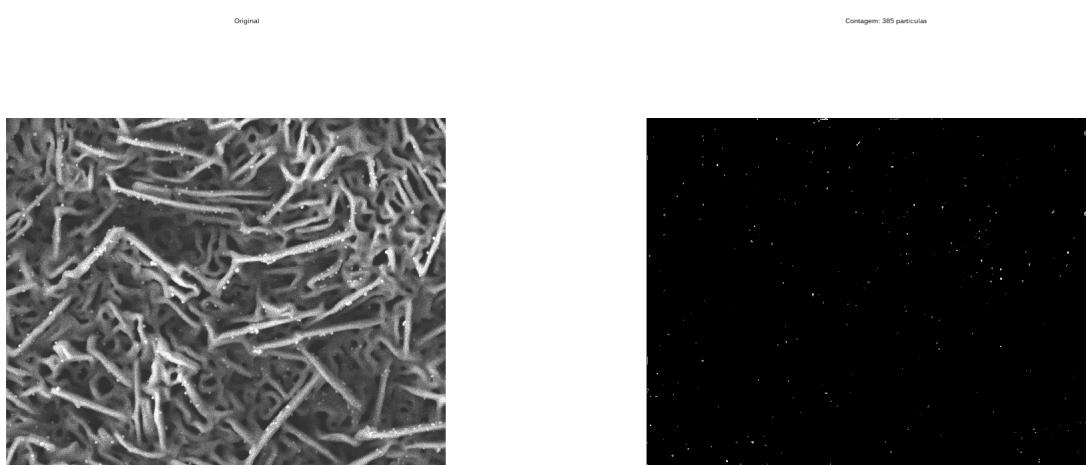


Figura 9: Contagem de Partículas 2 = 385

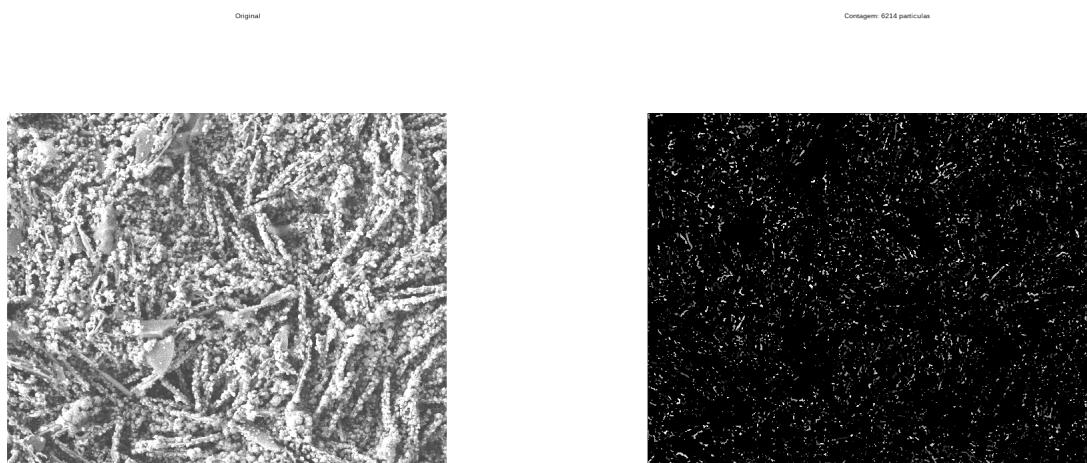


Figura 10: Contagem de Partículas  $3 = 6214$

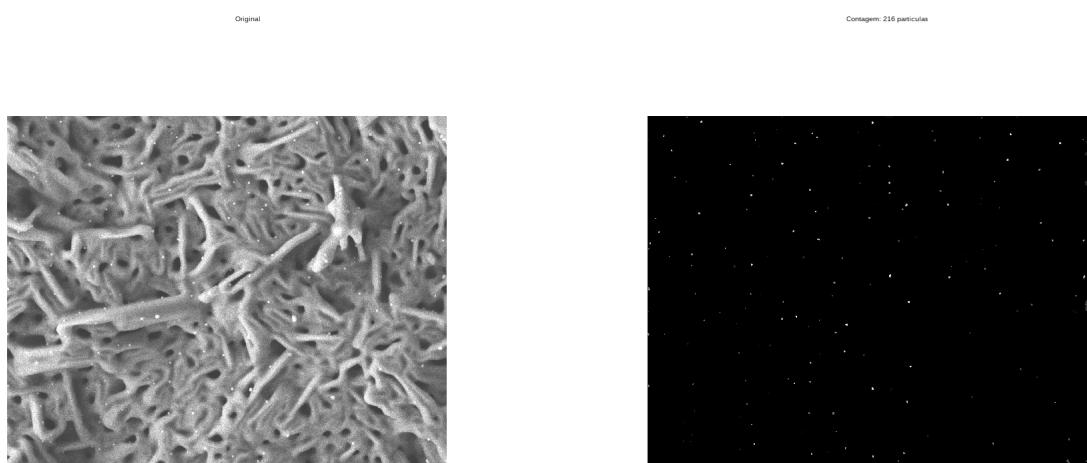


Figura 11: Contagem de Partículas  $4 = 216$

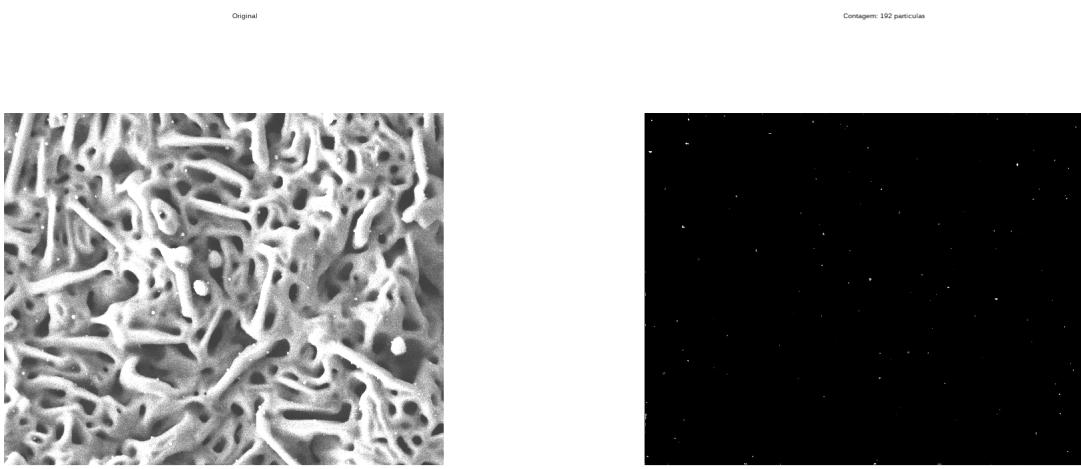


Figura 12: Contagem de Partículas  $5 = 192$

## 4 Conclusão

Nosso objetivo principal consistiu em auxiliar a contagem estimada de nanopartículas em uma imagem obtida por MEV (microscópio eletrônico de varredura), uma vez que originalmente essa contagem é realizada manualmente. Para esse fim, escrevemos um programa aplicando técnicas de processamento de imagens - nesse caso a modelagem foi a aplicação de filtros adequados para facilitar o reconhecimento das partículas e assim realizarmos uma contagem mais eficiente. A técnica de rotulamento - foi utilizada para contarmos o número de partículas das imagens.

Em resumo, as técnicas empregadas se mostraram eficientes para a redução de ruídos e processamento das imagens. O trabalho por tanto atingiu seu objetivo primordial. Contudo vale ressaltar que perdemos eficiência na contagem das partículas para aquelas imagens onde as regiões com nano-partículas, estão muito aglomeradas.

Para otimização da contagem de partículas, nos casos onde há região de muita aglomeração, poderíamos, por exemplo utilizar técnicas de aprendizagem de máquina, onde o programa reconheceria melhor onde estão os ruídos a partir de uma base de dados de milhares de imagens.

A medição do tamanho das partículas e sua distribuição de frequências representa a expansão desse trabalho, que precisa da orientação do departamento de CCNH que tem como objetivo final o auxílio na produção de artigos do departamento de Nanotecnologia.

## Referências

- [1] *Scilab Help*. [https://help.scilab.org/docs/6.0.1/en\\_US/index.html](https://help.scilab.org/docs/6.0.1/en_US/index.html) (Acesso em 10 de Agosto de 2018).
- [2] *Erosion (morphology)*. [https://en.wikipedia.org/wiki/Erosion\\_\(morphology\)](https://en.wikipedia.org/wiki/Erosion_(morphology)) (Acesso em 10 de Agosto de 2018).
- [3] *Dilation (morphology)*. [https://en.wikipedia.org/wiki/Dilation\\_\(morphology\)](https://en.wikipedia.org/wiki/Dilation_(morphology)) (Acesso em 10 de Agosto de 2018).

- [4] *Opening (morphology)*. [https://en.wikipedia.org/wiki/Opening\\_\(morphology\)](https://en.wikipedia.org/wiki/Opening_(morphology)) (Acesso em 10 de Agosto de 2018).
- [5] *Closing (morphology)*. [https://en.wikipedia.org/wiki/Closing\\_\(morphology\)](https://en.wikipedia.org/wiki/Closing_(morphology)) (Acesso em 10 de Agosto de 2018).
- [6] *Top-hat transform*. [https://en.wikipedia.org/wiki/Top-hat\\_transform](https://en.wikipedia.org/wiki/Top-hat_transform) (Acesso em 11 de Agosto de 2018).
- [7] *Connected-component labeling*. [https://en.wikipedia.org/wiki/Connected-component\\_labeling](https://en.wikipedia.org/wiki/Connected-component_labeling) (Acesso em 14 de Agosto de 2018).