

Worksheet5_pdfversion

Creation date: 2024-12-12 18:08
Modification date: Thursday 12th December 2024 18:08:01
PDF Ref:

Segmentação e etiquetagem

1. Construa uma função que aplique diferentes métodos de threshold à imagem “beans.png” para a separar em duas classes.
A função deverá conter os métodos (disponíveis no openCV):

- **a) Threshold manual;
- b) Threshold por média;
- c) Threshold por otsu;
- d) Threshold adaptativo;

Código:

```
thresholdFunction

1  Mat thresholdMethods(const string& imagePath, string method, string mode) {
2
3      Mat image = imread(imagePath, IMREAD_GRAYSCALE);
4
5      Mat binaryImage = Mat::zeros(image.size(), image.type()); // Initialize with zeros (all
black)
6
7      if (method == "manual" && mode == "normal") { // manual
8          threshold(image, binaryImage, 200, 255, THRESH_BINARY);
9      }
10     else if (method == "manual" && mode == "inverted") {
11         threshold(image, binaryImage, 200, 255, THRESH_BINARY_INV);
12     }
13     else if (method == "mean" && mode == "normal") { // mean
14         Scalar mean_value = cv::mean(image);
15         double mean = mean_value[0]; // Grayscale image has only one channel
16         // Apply thresholding using the mean value
17         threshold(image, binaryImage, mean, 255, THRESH_BINARY);
18     }
19     else if (method == "mean" && mode == "inverted") {
20         Scalar mean_value = cv::mean(image);
21         double mean = mean_value[0]; // Grayscale image has only one channel
22         // Apply thresholding using the mean value
23         threshold(image, binaryImage, mean, 255, THRESH_BINARY_INV);
24     }
25     else if (method == "otsu" && mode == "normal") { // otsu
26         threshold(image, binaryImage, 0, 255, THRESH_BINARY_INV + THRESH_OTSU);
27     }
28     else if (method == "otsu" && mode == "inverted") {
29         threshold(image, binaryImage, 0, 255, THRESH_BINARY + THRESH_OTSU);
30     }
31     // adaptative (blocksize is the kernel. AN ODD NUMBERS ONLY!!)
32     else if (method == "adaptative" && mode == "normal") {
33         adaptiveThreshold(image, binaryImage, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY,
201, 10);
34     }
35     else if (method == "adaptative" && mode == "inverted") {
36         (image, binaryImage, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY_INV, 201, 10);
37     }
38     return binaryImage;
39 }
```

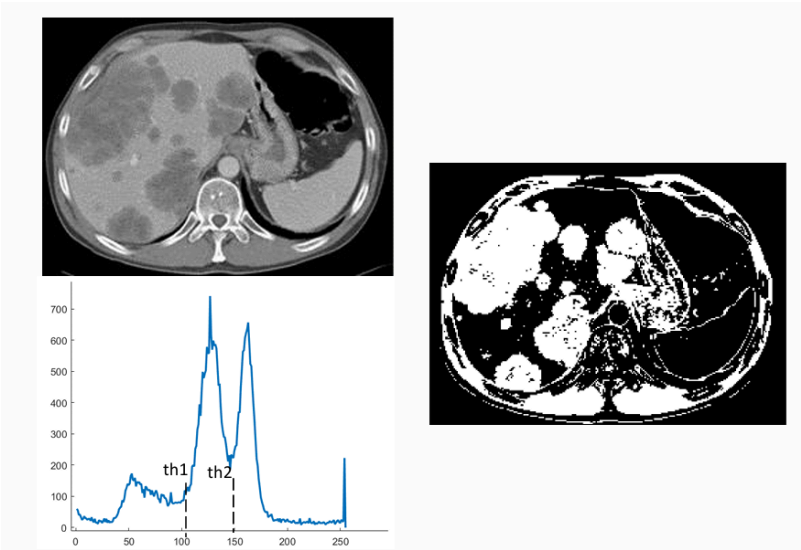
exercise1_ws5

```
1  void exercise1_ws5(const string& imagePath, const string& imageName, string method){
2
3      Mat image;
4      if (method != "otsu") {
5          image = thresholdMethods(imagePath, method, "inverted");
6      }else image = thresholdMethods(imagePath, method, "normal");
7
8      imshow("Thresholding", image);
9      waitKey(0);
10 }
```

Referências:

Segmentação por thresholding manual:

- Utilizada em imagens onde o objeto a segmentar possui intensidade distinta dos restantes elementos da imagem.
- O valor de threshold pode ser definido manualmente pelo utilizador ou calculado automaticamente.



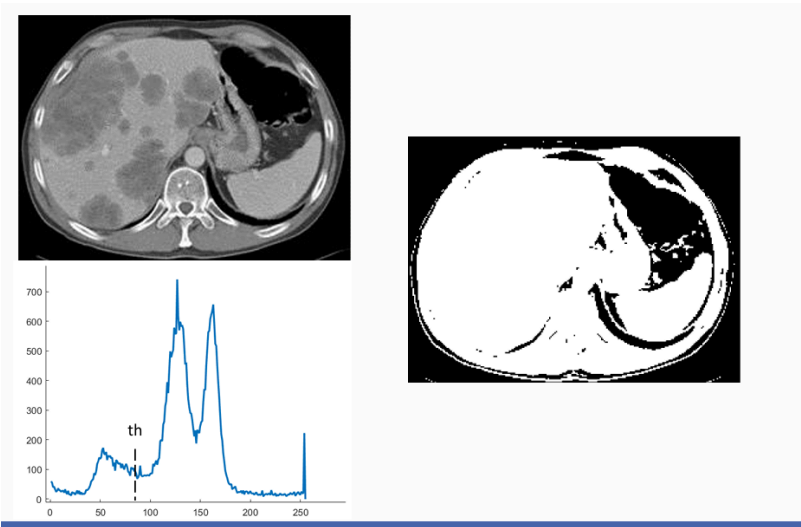
Pasted image 20241224120415.png#center

Segmentação por thresholding Média:

- O limiar (threshold) é definido pela média das intensidades de todos os pixels da imagem.

Segmentação por thresholding Otsu:

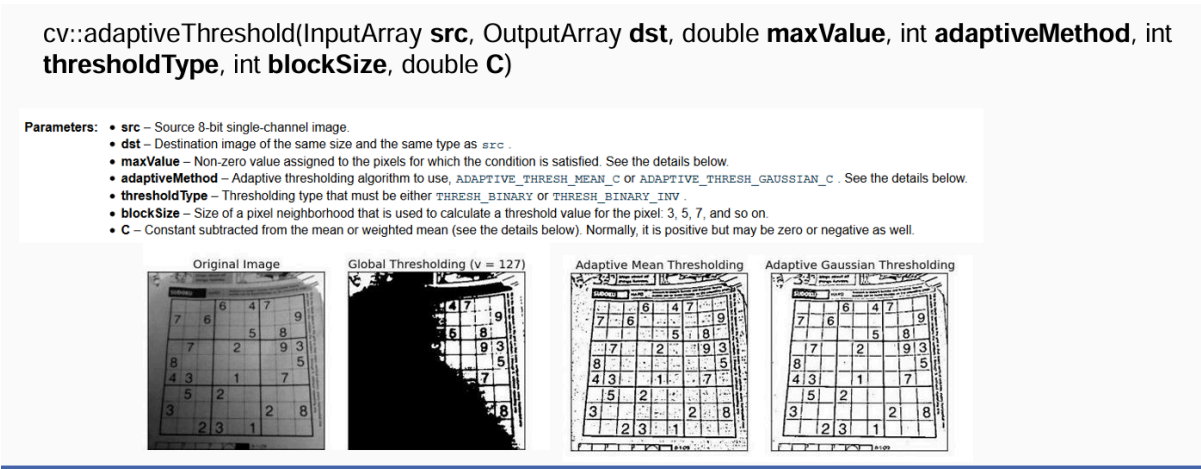
- Valor de threshold definido automaticamente.
- Otsu tenta **maximizar a diferença entre as intensidades** de fundo e objeto, enquanto minimiza a diferença dentro de cada parte (fundo ou objeto).
- Ele analisa a distribuição das intensidades na imagem e escolhe o limiar que dá a melhor separação possível entre essas duas regiões.
- O cálculo da variância, exige os valores das médias e pesos correspondentes para cada classe.



Pasted image 20241224120657.png#center

Segmentação por thresholding Adaptativo:


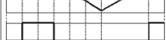
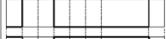

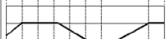
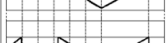
- Nos métodos adaptativos, a ideia consiste em utilizar, para cada pixel, um valor de limiar (threshold) calculado numa dada vizinhança desse pixel.




Pasted image 20241224121546.png#center

OpenCV:


`cv::threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type)`

• THRESH_BINARY	$dst(x,y) = \begin{cases} \text{maxval} & \text{if } src(x,y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$		Value and Threshold Level
• THRESH_BINARY_INV	$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$		Threshold Binary
• THRESH_TRUNC	$dst(x,y) = \begin{cases} \text{threshold} & \text{if } src(x,y) > \text{thresh} \\ src(x,y) & \text{otherwise} \end{cases}$		Threshold Binary, Inverted
• THRESH_TOZERO	$dst(x,y) = \begin{cases} src(x,y) & \text{if } src(x,y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$		Truncate
• THRESH_TOZERO_INV	$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > \text{thresh} \\ src(x,y) & \text{otherwise} \end{cases}$		Threshold to Zero, Inverted
			Threshold to Zero


Original Image




TRUNC




BINARY




TOZERO



BINARY_INV



TOZERO_INV



Pasted image 20241224120315.png#center

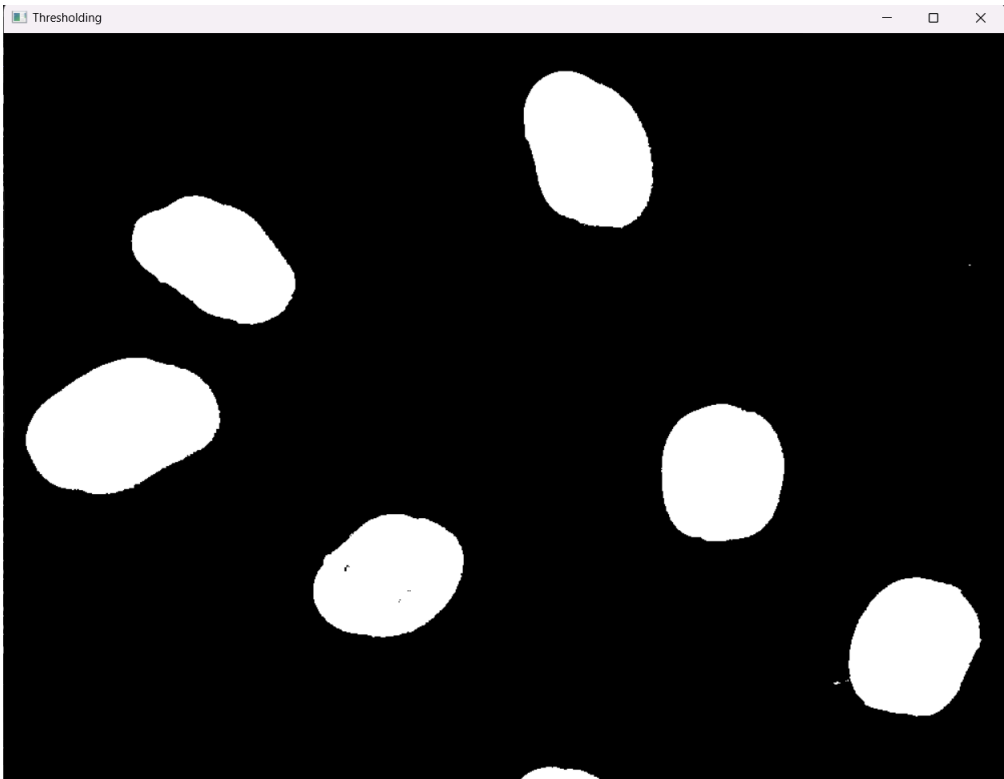
Resultados:

- Original:



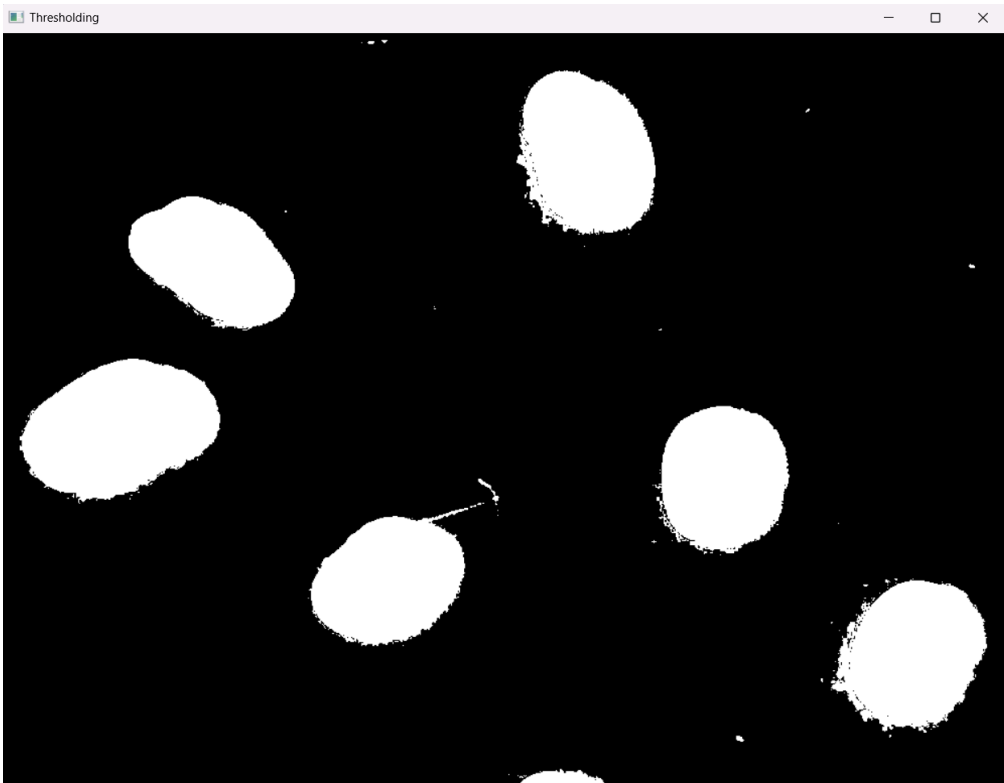
beans.png#center

- Manual:



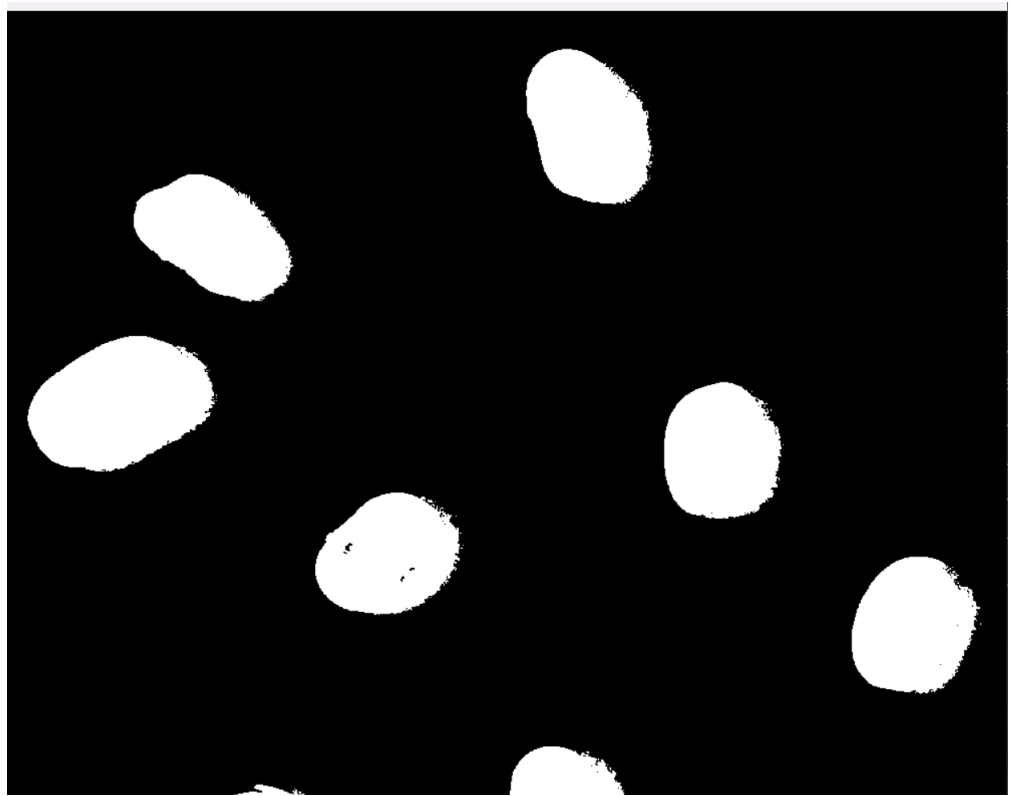
Pasted image 20241224111853.png#center

- Média:



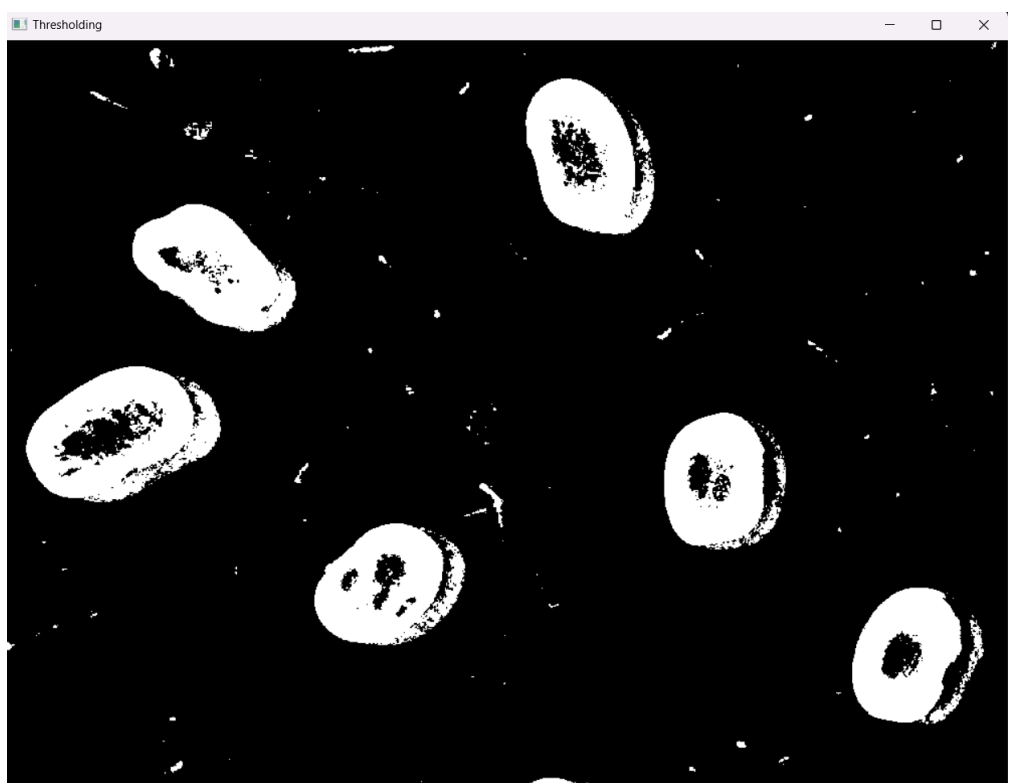
Pasted image 20241224111935.png#center

- Otsu:



Pasted image 20241226180743.png#center

- Adaptive (151, 10) (block size, constant):



Pasted image 20241224112049.png#center

Adaptive (201, 10) (block size, constant):



Pasted image 20241224112210.png#center

Código:

- Função que recebe o path da imagem, o tipo de método (manual, média...) e o modo (normal ou invertido), para realizar uma operação de threshold em qualquer imagem.
- A função usada é a threshold() do openCV que recebe como parametros a imagem de entrada, a de saída que é uma imagem vazia, os valores de threshold e o método.
- Para o método adaptativo usou-se a função adaptiveThreshold().
- No final retorna a imagem resultante da operação.
- Essa função é chamada através da função do exercício que recebe parâmetros conforme o input do utilizador.

Métodos:

- Um dos melhores resultados entre todos foi o método manual, visto que se trata de uma excelente abordagem para imagens onde o objeto a segmentar possui intensidade distinta dos restantes elementos da imagem.
- O outro foi o método Otsu que calcula a variância dos valores, separando a imagem em duas partes. O método tenta **maximizar a diferença entre as intensidades** de fundo e objeto, enquanto minimiza a diferença dentro de cada parte (fundo ou objeto), logo, nesta imagem o resultado vai ser excelente, visto que se trata de uma imagem onde o objeto a segmentar possui intensidade distinta dos restantes elementos da imagem.
- O método da média também tem um resultado aceitável, no entanto vai sofrer com o facto dos valores de intensidade serem muito dispares entre a zona de interesse e o background, logo o resultado é uma segmentação dos feijões com algum "ruído" de volta deles. Esses pequenos pontos de "ruído" são provavelmente provocados pelas sombras de volta dos feijões.
- O pior de todos foi o método adaptativo que sofreu com o facto do background não ser uma cor perfeita (mesa suja). Uma das possíveis razões pode estar ligada também ao método provocar problemas ao calcular limiares localmente para regiões com iluminação variável, levando a resultados inconsistentes.
- O block size maior ajudou a detetar melhor a parte interior dos feijões.

2. Construa uma função que faça o label de uma imagem (use o resultado da questão 1b) e que calcule as seguintes propriedades para cada uma das regiões (pode usar morfologia matemática para limpar pequenos pontos):

a) Área, largura, comprimento, centroide, orientação, excentricidade, perímetro

Código:


```

1  Mat imageLabelling(Mat& image, int areaValue, string ws4Flag) {
2
3      Mat labels, stats, centroids;
4
5      string side;
6
7      // for exercise 4 pixel spacing
8      double pixelSpacing = 0.8;
9      int colsSizePixels = image.cols * pixelSpacing;
10     int rowsSizePixels = image.rows * pixelSpacing;
11     int midColSizePixels = colsSizePixels / 2;
12
13     // Find connected components
14     int numLabels = connectedComponentsWithStats(image, labels, stats, centroids);
15     int objectNumber = 0;
16
17     // Create a copy of the image for output after blob removal
18     Mat resultImage = image.clone();
19
20     // Iterate through the components and remove small areas
21     for (int i = 1; i < numLabels; i++) {
22         int area = stats.at<int>(i, CC_STAT_AREA);
23         // Remove small objects (less than 10000 area)
24         if (area < areaValue) {
25             for (int y = 0; y < labels.rows; y++) {
26                 for (int x = 0; x < labels.cols; x++) {
27                     if (labels.at<int>(y, x) == i) {
28                         resultImage.at<uchar>(y, x) = 0; // Set to background (0)
29                     }
30                 }
31             }
32         }
33         else {
34             objectNumber++;
35
36             // Width and Height
37             int width = stats.at<int>(i, CC_STAT_WIDTH);
38             int height = stats.at<int>(i, CC_STAT_HEIGHT);
39
40             // Centroid x and y
41             Point2d centroid(centroids.at<double>(i, 0), centroids.at<double>(i, 1));
42             if (ws4Flag == "activated") {
43                 if ((centroid.x * pixelSpacing) < midColSizePixels)
44                     side = "left";
45                 else
46                     side = "right";
47             }
48
49             // Extract blob mask for this label
50             Mat blobMask = (labels == i);
51
52             // Perimeter
53             vector<vector<Point>> contours;
54             findContours(blobMask, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
55             double perimeter = 0.0;
56             if (!contours.empty()) {
57                 perimeter = arcLength(contours[0], true);
58             }
59
60             // Orientation and Eccentricity using Moments
61             Moments moments = cv::moments(blobMask, true);

```

```

62         double mu20 = moments.mu20 / moments.m00;
63         double mu02 = moments.mu02 / moments.m00;
64         double mu11 = moments.mu11 / moments.m00;
65
66         double theta = 0.5 * atan2(2 * mu11, mu20 - mu02); // Orientation in radians
67         double orientation = theta * (180.0 / CV_PI);      // Convert to degrees
68
69         // Eccentricity
70         double numerator = (mu20 - mu02) * (mu20 - mu02) + 4 * mu11 * mu11;
71         double denominator = (mu20 + mu02) * (mu20 + mu02);
72         double eccentricity = sqrt(1 - (numerator / denominator));
73
74         // Display properties
75         cout << "Blob " << objectNumber << ":" << endl;
76         cout << "Area: " << area << endl;
77         cout << "Width: " << width << endl;
78         cout << "Height: " << height << endl;
79         cout << "Centroid: (" << centroid.x << ", " << centroid.y << ")" << endl;
80         cout << "Orientation: " << orientation << " degrees" << endl;
81         cout << "Eccentricity: " << eccentricity << endl;
82         cout << "Perimeter: " << perimeter << endl;
83         if (side == "left") cout << "Left Side" << endl;
84         else if (side == "right") cout << "Right Side" << endl;
85         cout << "-----" << endl;
86     }
87 }
88
89 return resultImage;
90 }

```

exercise2_ws5

```

1  // Load and process the image
2  Mat image = thresholdMethods("beans.png", "mean", "inverted");
3  Mat structElementErosion = structuringElement(9, 1);
4  Mat structElementOpening = structuringElement(35, 1);
5
6  Mat openingImage = opening(image, structElementOpening);
7  Mat erodedImage = erosion(openingImage, structElementErosion);
8
9  Mat imageLabeled = imageLabelling(imageErosion, 8000);
10
11 // Resize
12 Mat resizedImage, resizedImageOutput, outputImage;
13 resize(image, resizedImage, Size(), 0.5, 0.5);
14 resize(erodedImage, resizedImageOutput, Size(), 0.5, 0.5);
15
16 hconcat(resizedImage, resizedImageOutput, outputImage);
17
18 // Show the result
19 imshow("Labelling", outputImage);
20 waitKey(0);

```

Referências:

Labelling:

- entra uma imagem binária e sai uma binária mas cada blob tem os seus pixels correspondentes ao valor atribuído. Por exemplo, objeto 1 = todos os pixels a 1, objeto 2 = todos os pixels a 2, etc.
- Depois de cada blob ter etiqueta, é possível analisar as suas características para descobrir qual o objeto de interesse:
 - Tamanho, propriedades da bounding box, circularidade, ...

Fecho:

- Dilatação seguida de uma erosão.
- Preenchimento de falhas.

```
cv::Mat labels, stats, centroids;
int numLabels = cv::connectedComponentsWithStats(binaryImage, labels, stats, centroids);

for (int i = 1; i < numLabels; ++i) { //exclude background
    int area = stats.at<int>(i, cv::CC_STAT_AREA);
    int left = stats.at<int>(i, cv::CC_STAT_LEFT);
    ....
    cv::Point2d centroid(centroids.at<double>(i, 0), centroids.at<double>(i, 1));
}
```

Pasted image 20241226184745.png#center

LABELLING

☐ Função que calcula várias características de objetos conectados (blobs):

- ☐ Outras funções
- ☐ `cv::Moments moments = cv::moments(Mask, true);`
- ☐ Os momentos são valores numéricos que captam propriedades geométricas e distributivas da região, como a área, o centróide e características de forma como a orientação e a excentricidade.

```
cv::Mat labels, stats, centroids;
int numLabels = cv::connectedComponentsWithStats(binaryImage,
labels, stats, centroids);

for (int i = 1; i < numLabels; ++i) { //exclude background

    cv::Mat Mask = (labels == i);
    cv::Moments moments = cv::moments(Mask, true); // Calculate
moments for additional properties
}
```

```
// spatial moments
double m00, m10, m01, m20, m11, m02, m30, m21, m12, m03;
// central moments
double mu20, mu11, mu02, mu30, mu21, mu12, mu03;
// central normalized moments
double nu20, nu11, nu02, nu30, nu21, nu12, nu03;
```

Pasted image 20241226184735.png#center

Medidas importantes:

- ☐ Eccentricity – Quão próximo o objeto está de um círculo;
- ☐ Círculo perfeito – 1
- ☐ Orientação.

$$a = \mu_{20} + \mu_{02}$$
$$b = \sqrt{(\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2}$$
$$\lambda_1 = (a + b)/2$$
$$\lambda_2 = (a - b)/2$$
$$ecc = \sqrt{1 - \lambda_2/\lambda_1}$$

$$\theta = 0.5 \arctan(\frac{2 \times \mu_{11}}{\mu_{20} - \mu_{02}})$$

https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga556a180f43cab22649c23ada36a8a139

Pasted image 20241226184824.png#center

Máscara de blob:

- É uma imagem binária onde os pixels que pertencem a um objeto específico são definidos como 1 (ou verdadeiro) e todos os outros pixels são 0 (ou falso).
- Facilita a análise de uma região específica da imagem.
- Usado para isolar um objeto específico permite analisar suas características sem interferências do restante da imagem.

Momentos:

- São medidas matemáticas utilizadas para capturar certas propriedades de formas na imagem, como área, centroide (centro de massa), e a distribuição da massa ao redor do centroide.
- Permitem calcular características como orientação, excentricidade, e outros parâmetros que descrevem a forma e a estrutura do objeto.
- Momentos fornecem uma forma compacta e eficiente de descrever a forma e a distribuição de um objeto, facilitando cálculos complexos de propriedades geométricas.

Orientação:

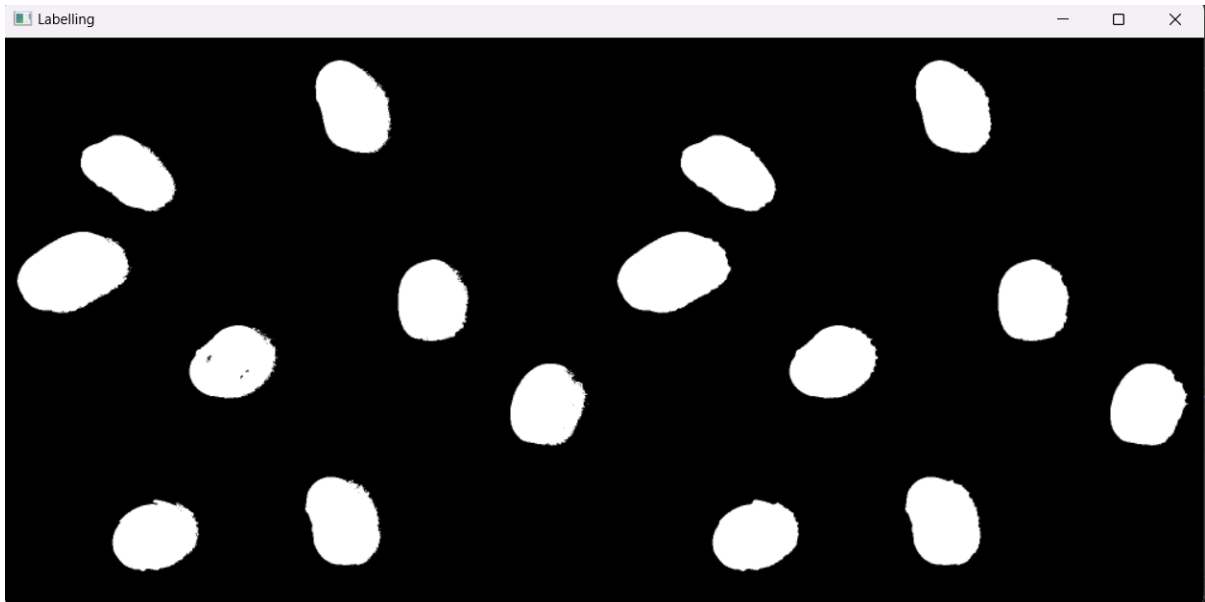
- Refere-se ao ângulo que descreve a direção principal de um objeto. É calculada usando os momentos centrais da forma.

Excentricidade:

- É uma medida de quão alongada ou circular uma forma é. Vai de 0 (um círculo perfeito) até próximo de 1 (uma elipse muito alongada).

Resultados:

- Threshold Otsu → Resultado final (closing):



Pasted image 20241226183136.png#center

- Informações dos blobs 1, 2 e 3:

```
Blob 1:
Area: 14756
Width: 127
Height: 155
Centroid: (581.39, 116.404)
Orientation: 63.3867 degrees
Eccentricity: 0.93336
Perimeter: 500.274
-----
Blob 2:
Area: 13235
Width: 159
Height: 127
Centroid: (208.161, 225.06)
Orientation: 30.087 degrees
Eccentricity: 0.884056
Perimeter: 474.416
-----
Blob 3:
Area: 18112
Width: 189
Height: 135
Centroid: (114.33, 391.266)
Orientation: -18.8476 degrees
Eccentricity: 0.907816
Perimeter: 548.6
-----
```

Pasted image 20241226183148.png#center

Código:

- Função que faz o labelling de qualquer imagem passada por parâmetro.
- O processo de labelling realiza-se através da função `connectedComponentsWithStats()` do openCV, que recebe a imagem e 3 outras variáveis tipo Mat (labels, stats, centroids), correspondentes às características que vão ser retiradas/calculadas.
- Um for loop percorre todos os objetos/blobs encontrados e analisa o parâmetro correspondente à área para eliminar todos aqueles abaixo de limiar definido manualmente (maior área = feijão).
- Se encontrar uma área inferior a 8000, dois for loops percorrem a image ponto a ponto e eliminam os pixels desses blobs (eliminar = colocar a preto).
- Para os feijões que tem uma área superior, são retiradas as informações relativas a área, largura, comprimento, centroide, orientação, excentricidade e perímetro.

Labeling:

- O labelling realizou-se através da função `connectedComponentsWithStats()`.
- Através dessa função, retirou-se diretamente as informações de área, largura, comprimento e centroide.
- Primeiramente criou-se uma blobmask que é uma imagem binária onde os pixels que pertencem a um objeto específico são definidos como 1 (ou verdadeiro) e todos os outros pixels são 0 (ou falso).
- Com essa blobmask calculou-se os contornos para permitir calcular o perímetro de cada blob.
- De seguida utilizou-se os momentos que são medidas matemáticas utilizadas para capturar certas propriedades de formas na imagem, como área, centroide, etc.

- Permitiu também calcular características como orientação, excentricidade, que descrevem a forma e a estrutura do objeto (baseado nas formulas apresentadas nas referências).
- Momentos fornecem uma forma compacta e eficiente de descrever a forma e a distribuição de um objeto, facilitando cálculos complexos de propriedades geométricas.

3. Construa uma pipeline para extrair os rins da imagem "kidney.png". Deve começar por binarizar a imagem usando o método de threshold que achar adequado. Posteriormente, deve aplicar pós-processamento por morfologia matemática e etiquetagem para obter uma máscara semelhante à imagem "kidney_segmented.png".

Código:

exercise3_ws5

```
1  void exercise3_ws5() {
2      Mat image = thresholdMethods("kidney.png", "manual", "normal");
3      Mat goal = thresholdMethods("kidney_segmented.png", "manual", "normal");
4
5      // Morphology
6      Mat strucElement1 = structuringElement(9, 1);
7      Mat strucElement2 = structuringElement(9, 4);
8      Mat imageClosing = closing(image, strucElement1);
9      Mat imageErosion = erosion(imageClosing, strucElement2);
10
11     // Labelling call
12     Mat imageLabeled = imageLabelling(imageErosion, 8000, "");
13
14     // Fill the labeled regions (after labelling)
15     Mat filledImage = imageLabeled.clone(); // Create a copy of the labeled image
16     vector<vector<Point>> contours;
17     vector<Vec4i> hierarchy;
18
19     // Find contours of the labeled regions
20     findContours(filledImage, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
21
22     // Fill each contour
23     for (size_t i = 0; i < contours.size(); i++) {
24         drawContours(filledImage, contours, (int)i, Scalar(255), FILLED); // Fill the regions
with white color
25     }
26
27     Mat strucElement3 = structuringElement(9, 1);
28     Mat imageDilation = dilation(filledImage, strucElement3);
29     Mat imageClosing2 = closing(imageDilation, strucElement3);
30
31     // Resize
32     Mat resizedImage, resizedGoalImage, resizedImageOutput, firstConcat, secondConcat;
33     resize(image, resizedImage, Size(), 0.5, 0.5);
34     resize(goal, resizedGoalImage, Size(), 0.5, 0.5);
35     resize(imageClosing2, resizedImageOutput, Size(), 0.5, 0.5);
36
37     hconcat(resizedImage, resizedGoalImage, firstConcat);
38     hconcat(firstConcat, resizedImageOutput, secondConcat);
39
40     // Show the result
41     imshow("Pipeline Kidney WS5", secondConcat);
42     waitKey(0);
43 }
```

Referências:

Labelling:

- entra uma imagem binária e sai uma binária mas cada blob tem os seus pixels correspondentes ao valor atribuído. Por exemplo, objeto 1 = todos os pixels a 1, objeto 2 = todos os pixels a 2, etc.
- Depois de cada blob ter etiqueta, é possível analisar as suas características para descobrir qual o objeto de interesse:
 - Tamanho, propriedades da bounding box, circularidade, ...

Erosão:

- Remover pixels aos limites de uma região segmentada, diminuindo a sua área e eliminando regiões de tamanho inferior ao elemento estruturante.

Dilatação:

- Adicionar pixels aos limites de uma região segmentada, aumentando a sua área e preenchendo algumas zonas no seu interior.

Fecho:

- Dilatação seguida de uma erosão.
- Preenchimento de falhas.

Resultados:

- Threshold manual + morfologia → Esperado → Resultado:



Pasted image 20241226191421.png#center

- Informações dos blobs:

```
Blob 1:
Area: 8781
Width: 133
Height: 127
Centroid: (631.097, 455.973)
Orientation: 42.1235 degrees
Eccentricity: 0.949114
Perimeter: 454.375
-----
Blob 2:
Area: 9694
Width: 134
Height: 136
Centroid: (281.062, 475.056)
Orientation: -75.2707 degrees
Eccentricity: 0.981613
Perimeter: 618.399
-----
```

Pasted image 20241226173936.png#center

Código:

- Realiza o mesmo processo que o exercício anterior de thresholding e labelling.
- Após terminado esse processo, realizou-se um fill dos blobs para eliminar as zonas pretas dentro do rim da direita.
- Aplicados métodos de morfologia matemática para melhorar o resultado final.

Pipeline:

- O pipeline consistiu na aplicação de um threshold manual normal da imagem Kidney.png.
- Para melhorar essa imagem aplicou-se operações morfológicas para remover as zonas sem interesse. As operações foram um fecho e uma erosão, que resultou na primeira imagem da imagem supracitada. Nessa imagem os rins estão praticamente isolados, faltando apenas limpar a imagem dos pequenos pontos sem interesse.
- Para remover foi aplicado o conceito de labelling com a avaliação da área. Superior a 8000 é um rim e inferior é eliminado (colocado a preto).
- Após esse processo, o resultado é uma imagem sem pequenos pontos mas com uma zona preta dentro do rim da direita. Resolveu-se através de um fill que é realizado através da obtenção dos contornos dos objetos para poder colocar todo o seu interior a um valor de intensidade de 255 (branco).
- Após esse processo, aplicou-se mais duas operações de morfologia, uma dilatação e um fecho, com o intuito de melhorar ainda mais o resultado final, reparando as pequenas falhas ("dentadas") nas bordas dos rins.
- O resultado final está muito próximo do resultado esperado.

4. Mostre os parâmetros de região de cada um dos rins, distinguindo entre o rim esquerdo e o rim direito. Assumindo que a imagem tem um pixel spacing de 0.8mm, mostre as medidas em mm.

Código:

O mesmo que o ws3 mas com o parâmetro para ativar a detecção dos lados:

- `Mat imageLabeled = imageLabelling(imageErosion, 8000, "activate");`

```
1  void exercise3_ws5() {
2      Mat image = thresholdMethods("kidney.png", "manual", "normal");
3      Mat goal = imread("kidney_segmented.png", IMREAD_GRAYSCALE);
4
5      // Morphology
6      Mat strucElement1 = structuringElement(9, 1);
7      Mat strucElement2 = structuringElement(9, 4);
8      Mat imageClosing = closing(image, strucElement1);
9      Mat imageErosion = erosion(imageClosing, strucElement2);
10
11     // Labelling call (image, area, activate ws4)
12     Mat imageLabeled = imageLabelling(imageErosion, 8000, "activate");
13
14     // Fill the labeled regions (after labelling)
15     Mat filledImage = imageLabeled.clone(); // Create a copy of the labeled image
16     vector<vector<Point>> contours;
17     vector<Vec4i> hierarchy;
18
19     // Find contours of the labeled regions
20     findContours(filledImage, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
21
22     // Fill each contour
23     for (size_t i = 0; i < contours.size(); i++) {
24         drawContours(filledImage, contours, (int)i, Scalar(255), FILLED); // Fill the regions
with white color
25     }
26
27     Mat strucElement3 = structuringElement(9, 1);
28     Mat imageDilation = dilation(filledImage, strucElement3);
29     Mat imageClosing2 = closing(imageDilation, strucElement3);
30
31     // Resize
32     Mat resizedImage, resizedGoalImage, resizedImageOutput, firstConcat, secondConcat;
33     resize(image, resizedImage, Size(), 0.5, 0.5);
34     resize(goal, resizedGoalImage, Size(), 0.5, 0.5);
35     resize(imageClosing2, resizedImageOutput, Size(), 0.5, 0.5);
36
37     hconcat(resizedImage, resizedGoalImage, firstConcat);
38     hconcat(firstConcat, resizedImageOutput, secondConcat);
39
40     // Show the result
41     imshow("Pipeline Kidney WS5", secondConcat);
42     waitKey(0);
43 }
```

Referências:

Cálculo dos valores das colunas da imagem com o pixel spacing:

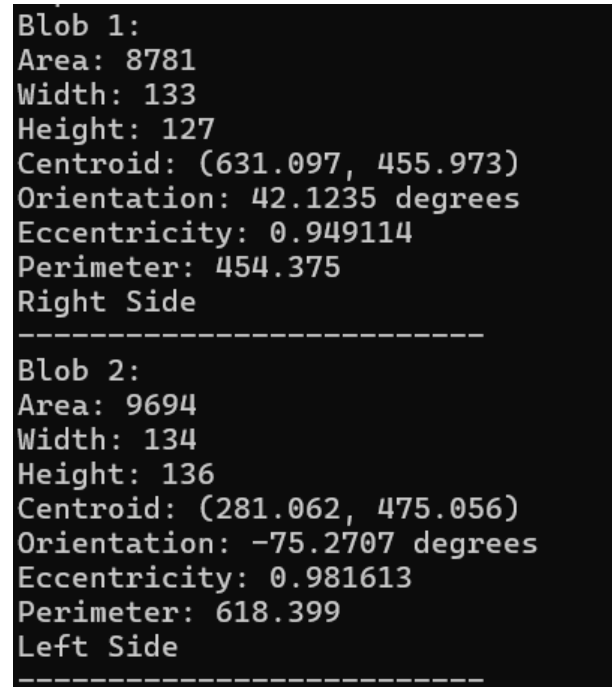
- `int colsSizePixels = image.cols * pixelSpacing;`

Separação da metade da imagem:

- `int midColSizePixels = colsSizePixels / 2;`

Resultados:

- Informações dos blobs com identificação dos lados:



Pasted image 20241226194959.png#center

Código:

- Cálculo dos valores das colunas da imagem com o pixel spacing:
 - `int colsSizePixels = image.cols * pixelSpacing;`
- Separação da metade da imagem:
 - `int midColSizePixels = colsSizePixels / 2;`
- Conta os pixels para verificar o lado
- Display do lado correspondente

Separação:

- Cálculo dos valores das colunas da imagem com o pixel spacing de 0.8mm, através da quantidade de colunas da imagem (ou seja, em x).
- De seguida separa a imagem pela metade para poder fazer a identificação dos rins através da distância entre o centroid e essa divisória.
- Se o valor do centroid.x multiplicado pelo spacing for menor que a metade da imagem, significa que o rim está à esquerda da imagem, senão está na direita.

5. Construa uma função para calcular as métricas de avaliação para a máscara de segmentação que gerou anteriormente na questão 3. Aplique a função também para a máscara que obteve no exercício 5 da worksheet 4 (deverá guardar a imagem binária obtida num png e abrir nesta worksheet). Compare os resultados.

Código:

confusionMatrix

```
1 void exercise5_ws5(const string& imageName, Mat& imageOutput, Mat& imageReference) {
2     int TP = 0, FP = 0, FN = 0, TN = 0;
3
4     // Loop through each pixel of the output and reference images
5     for (int r = 0; r < imageOutput.rows; r++) {
6         for (int c = 0; c < imageOutput.cols; c++) {
7             int ref = imageReference.at<uchar>(r, c) > 0 ? 1 : 0; // Binary value for reference
8             int pred = imageOutput.at<uchar>(r, c) > 0 ? 1 : 0; // Binary value for prediction
9
10            if (ref == 1 && pred == 1) ++TP; // True positive
11            else if (ref == 0 && pred == 1) ++FP; // False positive
12            else if (ref == 1 && pred == 0) ++FN; // False negative
13            else if (ref == 0 && pred == 0) ++TN; // True negative
14        }
15    }
16
17    // Calculate the metrics
18
19    // Accuracy
20    double accuracy = static_cast<double>(TP + TN) / (TP + FP + FN + TN);
21
22    // Precision
23    double precision = (TP + FP > 0) ? static_cast<double>(TP) / (TP + FP) : 0.0;
24
25    // Recall
26    double recall = (TP + FN > 0) ? static_cast<double>(TP) / (TP + FN) : 0.0;
27
28    // Dice Coefficient
29    double dice = (TP + TP > 0) ? (2.0 * TP) / (2.0 * TP + FP + FN) : 0.0;
30
31    // Intersection over Union (IoU)
32    double jaccard = (TP + FP + FN > 0) ? static_cast<double>(TP) / (TP + FP + FN) : 0.0;
33
34    // Output the confusion matrix and metrics
35    cout << std::fixed << std::setprecision(2);
36    cout << "Confusion Matrix:\n";
37    cout << "TP: " << TP << ", FP: " << FP << ", FN: " << FN << ", TN: " << TN << "\n";
38    cout << "Metrics:\n";
39    cout << "Accuracy: " << accuracy * 100 << "%\n";
40    cout << "Precision: " << precision * 100 << "%\n";
41    cout << "Recall: " << recall * 100 << "%\n";
42    cout << "Dice Coefficient: " << dice * 100 << "%\n";
43    cout << "Intersection over Union (IoU): " << jaccard * 100 << "%\n";
44
45    // Resize images for displaying side by side
46    Mat resizedImage, resizedGoalImage, concatOutput;
47    resize(imageOutput, resizedImage, Size(), 0.5, 0.5);
48    resize(imageReference, resizedGoalImage, Size(), 0.5, 0.5);
49
50    // Concatenate the images horizontally
51    hconcat(resizedImage, resizedGoalImage, concatOutput);
52
53    // Show the result
54    imshow(imageName, concatOutput);
55    waitKey(0);
56 }
```

AVALIAÇÃO DE UM MÉTODO DE SEGMENTAÇÃO

TP – True Positives;

TN – True Negatives;

FP – Falses Positives;

FN – Falses Negatives;

Dice

$$DSC = \frac{2TP}{2TP + FP + FN}$$

Accuracy

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

Precision

$$PPV = \frac{TP}{TP + FP}$$

IoU

$$TS = \frac{TP}{TP + FN + FP}$$

Ground truth pixels in class

Predicted pixels in class

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

A matriz de confusão é uma ferramenta utilizada para avaliar o desempenho de um modelo de classificação. Ele fornece uma análise detalhada das previsões do modelo, comparando-as com os valores reais. Para classificação binária (como sua tarefa de segmentação), ela categoriza as previsões em quatro resultados possíveis com base na comparação entre os valores previstos e reais (referência).

Confusion Matrix for Binary Classification

	Predicted: 1 (Positive)	Predicted: 0 (Negative)
Actual: 1 (Positive)	True Positive (TP)	False Negative (FN)
Actual: 0 (Negative)	False Positive (FP)	True Negative (TN)

True Positive (TP):

The model correctly predicted a positive result (the kidney was correctly segmented as part of the image).

False Positive (FP):

The model incorrectly predicted a positive result (the model incorrectly labeled a non-kidney pixel as part of the kidney).

False Negative (FN):

The model incorrectly predicted a negative result (the model failed to label a kidney pixel as part of the kidney).

True Negative (TN):

The model correctly predicted a negative result (the non-kidney pixels were correctly identified as background).

Pasted image 20241227151337.png#center

Summary of Metrics:

1. **Dice Coefficient:** A measure of overlap between the predicted and reference images.

$$Dice = \frac{2 \times TP}{2 \times TP + FP + FN}$$

2. **Intersection over Union (IoU):** A measure of the overlap between the predicted and reference images divided by their union.

$$IoU = \frac{TP}{TP + FP + FN}$$

3. **Accuracy:** The proportion of correctly predicted pixels (both true positives and true negatives) out of all pixels.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

4. **Precision:** The proportion of predicted positives that are actually positives.

$$Precision = \frac{TP}{TP + FP}$$

5. **Recall:** The proportion of actual positives that are correctly predicted.

$$Recall = \frac{TP}{TP + FN}$$

Pasted image 20241227151831.png#center

Metric	Higher Value Means	Lower Value Means	Use Case
Accuracy	More correct predictions overall	More incorrect predictions overall	General performance, but not ideal for imbalanced classes
Precision	Fewer false positives, high confidence in predicted positives	More false positives, less confidence in predictions	Use when false positives are costly (e.g., medical diagnoses)
Recall	Fewer false negatives, identifies more true positives	More false negatives, misses actual positives	Use when false negatives are costly (e.g., detecting diseases)
Dice Coefficient	Better overlap between predicted and reference positive regions	Worse overlap, model misses or wrongly predicts positive regions	Image segmentation, especially in medical imaging
IoU (Intersection over Union)	Better overlap with reference region relative to the union of regions	Worse overlap, meaning the predicted region doesn't cover the reference region well	Object detection, image segmentation

Pasted image 20241227152821.png#center

How to Use These Metrics:

- **Precision vs Recall:** There is often a trade-off between precision and recall. You can improve one at the expense of the other. For example, if you lower the threshold for predicting a positive, recall might increase, but precision might decrease.
- **Dice & IoU:** Both of these metrics focus on the overlap between predicted and reference regions, but the Dice coefficient gives a bit more weight to the overlap than IoU, making it more sensitive to smaller segmentations.

Pasted image 20241227160240.png#center

Resultados:

Código:

- Duas imagens são passadas como parâmetro da função onde uma é a resultante dos exercícios dos worksheets e outra a imagem de referencia de uma segmentação aos rins.
- Dois for loops percorrem a imagem pixel a pixel para calcular todas as métricas de avaliação do pipeline, em comparação entre os pixels da imagem de output e a de referência.
- Display das imagens resized.

Exercício 3 do ws5:

```
Confusion Matrix:
TP: 25401, FP: 1270, FN: 440, TN: 612281
Metrics:
Accuracy: 99.73%
Precision: 95.24%
Recall: 98.30%
Dice Coefficient: 96.74%
Intersection over Union (IoU): 93.69%
```

Pasted image 20241227151717.png#center



Pasted image 20241227151120.png#center

- Melhor exemplo de segmentação entre os 3 exemplos.
- Accuracy alta o que significa que o sistema foi capaz de obter mais previsões corretas.
- Precision alta, que significa que foram detetados menos False Negatives, o que aumenta a confiança nos Positives esperados.
- Recall alto, o que significa menor quantidade de False Negatives, ou seja, maior identificação de True Positives.
- Dice Coefficient alto, o que significa que existe melhor sobreposição entre regiões Positive previstas e de referência.
- IoU alto, o que significa melhor sobreposição com a região de referência em relação à união de regiões.

Exercício 5 do ws4:

```
Confusion Matrix:
TP: 23829, FP: 1562, FN: 2012, TN: 611989
Metrics:
Accuracy: 99.44%
Precision: 93.85%
Recall: 92.21%
Dice Coefficient: 93.02%
Intersection over Union (IoU): 86.96%
```

Pasted image 20241227151751.png#center



Pasted image 20241227151110.png#center

- Abordagem ligeiramente pior que a anterior, o que já era esperado visto que o método de segmentação apenas usa morfologia matemática.

Teste das métricas:

```
Confusion Matrix:  
TP: 19670, FP: 4926, FN: 6171, TN: 608625  
Metrics:  
Accuracy: 98.26%  
Precision: 79.97%  
Recall: 76.12%  
Dice Coefficient: 78.00%  
Intersection over Union (IoU): 63.93%
```

Pasted image 20241227152531.png#center



Pasted image 20241227152546.png#center

- Este teste foi feito para ter a certeza que as métricas estavam a ser corretamente avaliadas/calculadas.
- A imagem de input foi uma imagem pós threshold manual normal, com bastantes blobs da zona sem interesse.
- Os valores descem substancialmente em relação aos exemplos anteriores, como é esperado. Maior influência dos False Negatives e False Positives.