Advanced Prostate Tumor 3D Segmentation: A Hybrid Approach Using KMedROI and Clustering Techniques

Students: dr. Smaldini Raffaele, dr. Ardillo Michele

Professor: Prof. Andrea Guerriero

Contents

List of Figures

Listings

Abstract

This study presents an advanced segmentation pipeline for prostate tumor identification using a hybrid approach that integrates KMedROI and clustering techniques. The segmentation process leverages multiple preprocessing steps, including normalization, filtering, and dimensionality reduction, to enhance the accuracy of prostate MRI analysis. Traditional methods such as Otsu thresholding and Watershed segmentation are compared against the novel KMedROI approach, which refines the region of interest (ROI) by eliminating irrelevant anatomical structures. The effectiveness of the proposed method is validated on a dataset of multiparametric MRI scans, using metrics such as the Dice Similarity Coefficient and Intersection over Union (IoU) to assess segmentation accuracy. Experimental results demonstrate that the combination of KMedROI and KMeans outperforms conventional segmentation techniques, achieving superior precision and robustness in prostate tumor delineation. This approach has significant clinical implications for improving prostate cancer diagnosis, facilitating targeted biopsies, and enhancing treatment planning. Future work aims to integrate deep learning models and expand dataset validation to further refine segmentation performance.

1 Chapter 1: Problem Formulation and Models Presentation

This project aims to find a model that helps identify prostate tumors. The prostate is a gland that is part of the male reproductive system. The main role of this organ is to create a fluid that nourishes sperms and helps their mobility.
The main health issue affecting the prostate is **prostate cancer**. This kind of cancer is the most common in men and might lead to serious complications like dysfunction, difficulty urinating and even metastasis. Prevention and detection of prostate cancer are key methods to avoid such serious threats. This project aims to analyse how four main models works and compare their performances. The models are, some standard tecniques like Otsu and Watershed and a more refined tecnique like KMedRoi. At the end there is also a model that is made by combining KMedRoi and Otsu models.

- Otsu Model;
- WatherShed Model;
- kMeans and KMedROI models combined;
- Otsu and KMedROI models combined.

Nevertheless, the implementation of each model is similar.

1.1 Prostate Anatomy and Zonal Architecture

The prostate is a small, **walnut-shaped gland** located in the **pelvic cavity**, positioned just below the bladder and in front of the rectum. It surrounds the urethra and plays a crucial role in the male reproductive system, contributing to seminal fluid production. The gland's size can vary with age and medical conditions, particularly benign prostatic hyperplasia (BPH), which can alter its internal structure. The prostate is composed of **distinct anatomical zones**, each with different biological characteristics. The peripheral zone **(PZ)** constitutes the largest portion of the gland and is the most common site for prostate cancer, accounting for nearly **70–75%** of cases. On T2-weighted MRI, the PZ appears hyperintense (bright), and tumors typically manifest as hypointense (dark) regions. The transition zone **(TZ)** surrounds the urethra and is often affected by BPH, leading to heterogeneous MRI appearances. Though prostate cancer is less common in the TZ, about 20% of cases originate here, and distinguishing between malignant and benign nodules in this region can be challenging. Additionally, the central zone (CZ) and anterior fibromuscular stroma (AFS) contribute to the gland's structure, though they are less frequently involved in malignancies. Several key structures surrounding the prostate impact both its function and clinical considerations. The neurovascular bundles, located posterolaterally, are **critical for erectile function** and can be affected by **tumor invasion**. The seminal vesicles, positioned superior to the prostate, can be involved in more advanced cases of prostate cancer. The bladder neck and prostatic urethra are also important landmarks in tumor progression and surgical planning.

1.2 Prostate Cancer and MRI Imaging

Prostate cancer is one of the most prevalent malignancies in men, often detected through elevated prostate-specific antigen (PSA) levels or abnormalities on digital rectal examination. Early-stage prostate cancer is typically asymptomatic, while advanced disease may present with urinary dysfunction or metastatic symptoms such as bone pain. The **likelihood of malignancy increases with age**, and imaging plays a key role in diagnosis and management. **Multiparametric MRI (mpMRI) is the gold standard for prostate cancer imaging**, incorporating several complementary sequences. **T2-weighted imaging (T2WI)** provides detailed anatomical visualization, with tumors in the peripheral zone typically appearing as **hypointense areas** against the **normally hyperintense gland**. Diffusion-weighted imaging (DWI) and its corresponding apparent diffusion coefficient (ADC) maps highlight areas of restricted diffusion, which correlate with increased cellular density in malignant tissues. Tumors generally appear hypointense on ADC maps, helping differentiate them from normal prostate tissue. In some protocols, dynamic contrast-enhanced (DCE) imaging is used to assess vascularity, but in the dataset under consideration, only T2WI and ADC sequences are utilized.

1.3 Dataset and Segmentation Considerations

The dataset [5] used for segmentation consists of **48 prostate mpMRI** studies obtained from Radboud University Medical Center in Nijmegen, Netherlands [1]. The selected imaging modalities for analysis include transverse T2-weighted and ADC sequences, with a specific focus on segmenting the peripheral and transition zones. The segmentation task is particularly challenging due to the close anatomical proximity of these zones and significant inter-subject variability in prostate morphology. The **primary difficulty in segmentation arises from the heterogeneous appearance of the transition zone**, where BPH-related **nodules can confound the distinction between benign and malignant tissue**. Additionally, adjacency between the peripheral and transition zones can lead to challenges in delineating clear boundaries. Tumors in the peripheral zone tend to be more conspicuous on T2-weighted imaging due to their hypointense appearance against the normally bright peripheral gland, whereas tumors in the transition zone often require integration of ADC features for improved contrast.

1.4 Practical Considerations for MRI-Based Tumor Segmentation

Effective segmentation of prostate tumors on MRI requires careful preprocessing and interpretation of multi-sequence data. Registration of T2-weighted and ADC images is essential to ensure alignment, as slight motion artifacts or differences in acquisition parameters can misalign anatomical structures. Intensity normalization across scans improves the robustness of segmentation algorithms, particularly in deep learning-based approaches. A typical segmentation workflow includes three main steps. First, the entire prostate gland is segmented, providing a global region of interest. Next, the peripheral and transition zones are delineated, using T2-weighted signal intensity patterns and anatomical landmarks such as the surgical capsule boundary. Finally, tumor segmentation is performed, identifying hypointense areas on T2WI that correspond to regions of restricted diffusion on ADC maps.

While automated segmentation algorithms offer efficiency, expert radiologist annotations remain crucial for validating results. Radiologists rely on standardized criteria such as PI-RADS (Prostate Imaging–Reporting and Data System) to evaluate lesion suspicion, and incorporating such clinical guidelines into AI-based segmentation approaches can improve diagnostic performance.

1.5 Clinical Relevance of Accurate Segmentation

Accurate tumor segmentation in prostate MRI has significant clinical implications. Precise delineation of cancerous regions facilitates targeted biopsies, reducing the likelihood of false negatives. In surgical planning, particularly for nerve-sparing radical prostatectomies, mapping the tumor's relationship to critical structures such as the neurovascular bundles is essential. Similarly, in radiotherapy, precise segmentation ensures that therapeutic radiation doses are confined to malignant tissue, minimizing damage to adjacent organs like the rectum and bladder. Beyond immediate clinical applications, robust segmentation supports longitudinal monitoring of disease progression, treatment response assessment, and the development of AI-driven diagnostic tools. By enhancing the accuracy of prostate tumor segmentation, AI and image processing methodologies contribute to improved patient outcomes and more efficient clinical workflows.

2 Chapter 2: Data Analysis

The data set consists of 48 prostate multiparametric MRI (mp-MRI) **studies comprising T2-weighted, Diffusion-weighted and T1-weighted contrast-enhanced series**.
T1-weighted are images with contrastive fluid which enhances paramagnetic contrast by altering proton relaxation, making areas of accumulation appear brighter compared to standard T1 images. In this way blood vessels, infections and tumors are move visible. For instance areas that absorb contrast appear to be brighter.
T2-weighted are images that, are kind of the opposite of T1-weighted. In this case the aim is to highlight more relaxed areas. This areas are tissues full of water and are not infected by tumour or other lesions. In this kind of images tumour appears dark. This images cannot correctly differentiate between benign and malignant lesions.
DWI are images that highlight the presence of water molecules inside tissue and not water content in contrast with T2-weighted images. In this case tumour appear to be bright. Unfortunately this images might have some false positive.
The corresponding target ROIs were the prostate peripheral zone (PZ) and the transition zone (TZ). This data set was selected due to the challenge of segmenting two adjoined regions with very large inter-subject variability. *The data was acquired at Radboud University Medical Center, Nijmegen Medical Center, Nijmegen, The Netherlands.*
A subset of two series, transverse T2-weighted and the apparent diffusion coefficient (ADC) was selected.

All images are MRI scan of the prostate. MRI is a structure like such: (x, y, z, t) where the first three (x, y, z) are used to indicate the recorded level of radio waves, similar to the grey intensity in a picture. This values are later used to reconstruct the image. With t, is often referred to the type of protocol used for the MRI. Since there different and each one can highlight different aspects and illnesses.

2.1 Init

The `m_init_m.m` file is a simple file that helps to clean and organize the whole workspace. Extracts the directory path from the full file path and saves it using `pathstr`.
It also add paths to all the external functions such as, utils, trainers and other pre processing pipelines.

Listing 1: m_init_m.m

```
function [pathstr] = m_init_m()
    %% Load Modules
    clc;
    fprintf("Cleaning workspace..")
    clear all; close all force; clear global;
    pause(1);
    fprintf("Done.")
    %% Import of file, dataset and setting up the environment
    mainFile = mfilename('fullpath');
    [pathstr,~,~] = fileparts(mainFile);
    display(pathstr) %print the path of the dataset
    %% Adding functions paths
    addpath(fullfile(pathstr, 'utils')); % Set Path for utils functions
    addpath(fullfile(pathstr, 'Trainer')); % Set Path for trainer functions and
       models
    addpath(fullfile(pathstr, 'Trainer','Metrics')); % Set Path for Metrics
    addpath(fullfile(pathstr, 'Transformations')); % Set Path for
       transformations functions
    addpath(fullfile(pathstr, 'preProcessingPipelines')); % Set Path for
       preprocessing functions
end
```

2.2 Data Loader

Data are loaded thanks to the `DataLoader.m` file. The function takes as input a Json file that contains the MRI images. It loads and splits the imagaes in a training set and a test set. Data loaded are MRI images in a widely used file format for storing medical imaging data, especially in neuroimaging (MRI, fMRI, CT scans) as NIfTI images is mandatory to use `niftiread()` built-in Matlab function.

3 Chapter 3: Pre Process, Pipelines, Transformation & Utilities

3.1 Pre Processing: Normalization

Date are form diverse sources, so is helpful and mandatory to normalize them.
The normalization method chosen in this project is the *Min-Max Normalization*. The aim of the Min-Max is to shift data in a pre chosen range *[a,b]*. Where the original data, $x \in [min, max]$ and the normalized data are in a normalized range, as example $z \in [0, 1]$.

$$z = \frac{x - min(X)}{max(X) - min(X)} \tag{1}$$

Normalized data visualization is not different from the original data, data are just in a different scale. This normalization technique helps to increase the scale differences between pixels by reducing their differences and keeping the original data distribution.

To be more precise, in our work, since images are in a 3D shape, *[x,y,z]* a 3D version of the Min-Max is used. Input images are taken as function input of the Min-Max function and, since MRI images are in a `uint8, int16` formats they are modified in order to avoid integer division issues. Max and min values are later taken form each image and the normalization is done.

Listing 2: MinMaxNorm3D.m

```
function [mriImage] = MinMaxNorm3D(mriImage)
    % Min-Max Normalization for a 3D MRI image using [0,1] interval
    % INPUT: mriImage [X, Y, Z]
    % Convert to double precision
    mriImage = double(mriImage);
    %% Compute global minimum and maximum across the 3D volume
    mVal = min(mriImage(:));
    MVal = max(mriImage(:));
    mriImage = (mriImage - mVal) / (MVal - mVal);
end
```

3.2 Filters

To de-noise and to make images cleaner some filters are applied.

3.2.1 GaussianFilter3D

The application of **Gaussian Filter** has the only objective to leverage further more images thanks to a low-frequency filter by defining *dimension and standard deviation*. In this case, since we are using 3D images the Gaussian Filter might act accordingly, so:

$$G(x, y, z) = \frac{1}{\sqrt{(2\pi)^3}\sigma^3} e^{-\frac{x^2+y^2+z^2}{2\sigma^2}} \tag{2}$$

σ tunes the intensity of the blur and the window dimension. This is because the built-in Matlab function `imgaussfilt3(image,`σ`)` also determines the window dimension as σ multiplier.

So, a 3D MRI image, σ, metrics and a flag `use_filter` are taken as input, and there also is a checker that checks if those data are provided. If not they are assigned as default values. The `use_filter` if not specified applies the default 3D Gaussian filter with `imgaussfilt3(image,`σ`)`. Metrics are used to evaluate the obtained image.
The filter is applied by a custom filter.

Listing 3: Gaussian 3D Filter

```
1 if use_filter == true
2     % Compute kernel size: At least 6 * sigma + 1
3     kernel_val = ceil(sigma*6 + 1);
4     if mod(kernel_val,2) == 0
5         kernel_val = kernel_val - 1; % Ensure it's odd
6     end
7     filterSize = [kernel_val, kernel_val, kernel_val];
8 end
```

So, based on σ value the filter is applied. Since it has to be odd there is a check, with the if-statement in row 4.
The filter is accordingly applied to each dimension.

Listing 4: Gaussian 3D Filter Application

```
image3D_denoised = imgaussfilt3(image3D, sigma, 'FilterSize', filterSize);
```

3.2.2 AnistropicFilter3D

The **Anistropic Filter**, in this case 3D version, is a filtering technique used in image processing and computer vision that helps to enhance 3D data while preserving and keeping key structural features like edges and detail.

$$\frac{\partial I}{\partial t} = \nabla \cdot (c(x, y, z, t)\nabla I) \tag{3}$$

where:

- I(x,y,z,t) is the intensity of the 3D image at time t;
- c(x,y,z,t) is the conduction coefficient that varies based on local gradients (high near edges, low in uniform areas),
- ∇ is the gradient operator,
- $\nabla\cdot$ is the divergence operator

So, a 3D MRI image, number of iteration,gradient threshold, metrics and a flag `use_filter` are taken as input, and there also is a checker that checks if those data are provided. If not they are assigned as default values. The number of iteration determines how many times the filter is applied, more iterations means a smoother image but might blur some fine details. The gradient treshold meanwhile controls how filter reacts to edges, an higher values means that diffusion is stronger the picture has more smoothing. Metrics are used to evaluate the obtained image.
The filter is applied thanks to the `imdiffuserfilt()` Matlab function.

Listing 5: Anistropic 3D Filter Application

```
%% Apply Anisotropic Filtering to 3D volume using imdiffusefilt
image3D_denoised = imdiffusefilt(double(image3D), ...
'NumberOfIterations', numIterations, ...
'GradientThreshold', gradientThreshold);
```

3.2.3 HomomorphicFilter3D

The **Homomorphic Filter** applies a Fouriers transform and a Gaussian filter to an image. This kind of filter enhances contrast by balancing illumination and reflectance components.
This time the application is more difficult.
Fist the image is converted to a Log domain to separate illumination and reflectance, because illumination effects are multiplicative, and taking the log makes them additive, which helps in filtering.
And a (Fast) Fourier Transform is applied.

```
img_log = log1p(double(volume)); % log1p handles log(0)
img_fft = fftn(img_log);
```

The filter is later build based on the image dimension and volume:

```
dims = size(volume);
[u, v, w] = ndgrid(1:dims(1), 1:dims(2), 1:max(1, dims(3))); % 2D or 3D
```

and it is centred to have a better filtering.

```
u = u - floor(dims(1)/2);
v = v - floor(dims(2)/2);
if numel(dims) == 3
    w = w - floor(dims(3)/2);
    D = sqrt(u.^2 + v.^2 + w.^2);
else
    D = sqrt(u.^2 + v.^2); % 2D case
end
```

And in the end the filter is create from scratch (GammaL controls the frequency response. High GammaL a less detailed image) and applied, multiplies the Fourier-transformed image by the filter, suppressing illumination variations and enhancing details. GammaL controls the frequency response. High GammaL a less detailed image.

Listing 6: Homomorphic Filter Appliocation

```
H = (gammaH - gammaL) * (1 - exp(-(D.^2) / (2 * D0^2))) + gammaL;
img_hf_fft = img_fft .* H;
```

At the end the filtered image is converted back to a spatial domain and also a reverse log is applied.

3.3 PCA

Principal Component Analysis is a technique used to reduce dimensionality of data while preserving most of their variance (most of the important features). PCA identifies the directions (or ”principal components”) in which the data varies the most. In the case of an MRI image, the majority of the meaningful information (anatomical structures, tissues, etc.) often resides in a few dominant directions of variance. By retaining only these principal components (those with the highest variance), we can eliminate less significant variations, which are often attributed to noise.
So the function takes as input the 3D image, a τ the cumulative variance threshold used to decide how much of the total variance should be retained while performing PCA and the denoising metrics.
The first steps of the PCA are quite standard for this kind of problem. Data are flatten along the Z-dimension, and then centred.

Listing 7: PCA

```
[nx, ny, nz] = size(image3D);
data = reshape(image3D, [], nz); % Flatten along Z slices
data_mean = mean(data, 2);
data_centered = data - data_mean; % Centering data
```

Later a **Singular Value Decomposition** is used to perform PCA. SVD decomposes the data matrix into three matrices: U (left singular vectors), S (singular values), and V (right singular vectors). Thank to SVD we con know the values that are retained in the transformation.
The cumulative sum of squared singular values (lambda_cumsum) is computed and normalized by the total sum of squared singular values. This represents the explained variance for each principal component. And the number of reduced dimension **K** is determined by threshold.

```
singular_values = diag(S);
lambda_cumsum = cumsum(singular_values.^2) / sum(singular_values.^2);
k = find(lambda_cumsum >= tau, 1);
```

So now, data are reduced in a K dimension by keeping variance:

Listing 8: PCA application

```
V_k = V(:, 1:k);
data_reduced = data_centered * V_k;
```

In the end data are reconstructed to the original dimension.

3.4 Filter Metrics

For each filter and for the PCA transformation **PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index)** metrics are also calculated. This metrics helps to evaluate the denoised images. PSNR is particularly useful for comparing the overall noise levels in images, and a higher PSNR generally indicates better image quality. SSIM, on the other hand, focuses on perceptual aspects, considering not just pixel-wise differences but structural information like edges and textures. The typical range for PSNR is:

- 30–40 dB: High-quality image with minimal noise.
- 20–30 dB: Moderate quality with noticeable noise.
- $\leq$ 20 dB: Low-quality image with significant noise.

Meanwhile for SSIM values range from 0 (no similarity) to 1 (identical images). A higher SSIM indicates better preservation of the image's structural integrity.

3.5 Log Transform

The **Logarithmic Transformation** is a mathematical operation applied to images to enhance low-intensity (dark) regions while compressing high-intensity (bright) regions. It uses three parameters:

1. k: a multiplicative constant;
2. N: volume gain;
3. `logtype`: a way to chose the type of logarithm (natural or base-2).

There two different methods, with a natural based log or a 2-base log:

$$s = k \cdot \ln(1 + N \cdot r) \tag{4}$$

$$s = k \cdot \log_2(1 + N \cdot r) \tag{5}$$

So the function takes as input, the volume and all three parameters. If there are issues with one or more parameters these are assigned by some default values, for instance, **k=1.5**. However, if the volume is missing an error is raised. After that the real transforamtion is applied:

Listing 9: LogTransform.m

```
switch logtype
    case 'log2'
        volume = k * log2(1 + N * double(volume));
    case 'log'
        volume = k * log(1 + N * double(volume));
    otherwise
        error("The only available case are: ['log', 'log2']")
end
```

A switch is used to determine what kind of logarithm to apply if natural or base-2. In both cases `double(volume)` ensures the image is converted to floating-point format to avoid integer rounding issues.

3.6 Sharpen3D

The **Sharpen3D** transformation is an image enhancement technique that enhances edges and fine details in a 3D image (such as medical images like MRI or CT scans).
It works by highlighting edges while reducing regions. This helps to make structure more distinct. In our case, since we are using 3D images the filter operates across three dimensions.

The functions takes as input a 3D volume and the number of time the sharpening filter is applied. Those parameters are checked, if absent they are setted by hand however if volume is absent an error is raised. Converts volume to double precision to avoid integer rounding errors. This step is important

because `imsharpen` performs calculations that can lose precision in integer formats.
Number of rows, columns and slices,

$$(x, y, z)$$

are taken. Later the filter is applied:

Listing 10: Sharpen3D

```
sharp_volume = volume; % Copy input volume to preserve original data
for a=1:sharpenStacks  % Repeat sharpening process multiple times
    for i=1:nz
        sharp_volume(:,:,i) = imsharpen(sharp_volume(:,:,i));
    end
end
```

Input is copied to prevent problems to original data and for each time explained by the param *ShapenStacks* the filter is applied. `Sharp_volume(:,:,i)` extracts one 2D slice at a time (i-th slice in the Z direction). And `imsharpen()` is applied to enhance edges in that 2D slice.

3.7 FitOverLapMask

The *FitOverLapMask* is a function designed to identify the best-matching tumor label from a *PredictedMask*, given by a trainer, when compared to a *TrueMask*. The function returns a binary mask (*bestFit*) that highlights the tumor region with the most overlap and identifies the best tumor label (*bestLabel*).
The inputs of the functions are a 3D binary mask that represents the ground truth of tumor and a 3D Segmented mask generated by the model.
Now, for the `trueMask` all the dimensions are extracted and a zero matrix representing the best fit is initialized. This matrix has same dimensions as `trueMask`.

Listing 11: fitOverlapMask.m

```
function [bestFit, bestLabel] = fitOverlapMask(trueMask, predictedMask)

    [rows, cols, nz] = size(trueMask);
    bestFit = zeros(size(trueMask));
```

The background is identified by examining the values at each corner of the `PredictedMask` and the most common value is assumed to be the background.

```
    % Determine the background value from the corners of the 3D volume
    vortex = [
        predictedMask(1, 1, 1), predictedMask(1, 1, nz), ...
        predictedMask(1, cols, 1), predictedMask(1, cols, nz), ...
        predictedMask(rows, 1, 1), predictedMask(rows, 1, nz), ...
        predictedMask(rows, cols, 1), predictedMask(rows, cols, nz)
    ];
    background = mode(vortex); % Most common value among corners
```

To find the best tumor label, the function iterates along each slice of the z-axis and extracts some `uniqueLabels`. Where the `predictedMask` are stored. From both `trueMask` and `predictedMask` the the current i-th slices is extracted.

```

    for i = 1:nz
        % Extract the tumor region
        uniqueLabels = unique(predictedMask(:));
        trueSlice = trueMask(:,:,i);
        predictedSlice = predictedMask(:,:,i);

        % Exclude background
        uniqueLabels = uniqueLabels(uniqueLabels ~= background);

        bestLabel = 0;
        maxOverlap = 0;
```

Then the label with the best overlap is found. To do so a `bestLabel` and a `maxOverlap` are initialized, this helps to track the label with the best overlap and to store the maximum overlap pixels. Then for each predicted slice it mputes overlap as the number of pixels where `predictedSlice` matches the label and trueSlice is nonzero, if overlap is higher than maxOverlap, updates bestLabel. The best fit is finally saved.

```

        % Find the label with maximum overlap
        for label = uniqueLabels
            overlap = sum((predictedSlice == label) & trueSlice, 'all');
            if overlap > maxOverlap
                maxOverlap = overlap;
                bestLabel = label;
            end
        end
        % Save best fit
        bestFit(:, :, i) = (predictedSlice == bestLabel);
    end
end
```

Note that this algorithm will only be used in training to define good parameters for the models and pre-processing phase. During testing the `'sliceWiseTumourHeu'` algorithm will be used; performing testing phase with an heuristic approach to evaluate the actual performances without knowing the tumor `'trueMask'` and allows us to define the correct parameters for the heuristics. With a larger amount of data, a `Cross-validation` approach would have been possible by dividing the dataset into *train, validation and test*, allowing us to tune the heuristics on the validation set and evaluate the parameters on the test set.

3.8 sliceWiseTumourHeu

The testing phase follows a heuristic-based approach, helping us evaluate how well the models perform.
As always, the initial parameters and function inputs are setted up, these are: *segmentedMask* is the mask that labels different parts of the image (background, prostate, etc.); *original_volume* is the original 3D image, like an MRI or CT scan, *params* is a structure where some optional parameters are stored; if these aren't provided, defaults are set: weightSpatial = 0.3 and nbins = 256.
A tumorMask is created; `tumorMask = false(size(segmentedMask));`.
The center of the prostate is found, to help the heuristic, First, we create a logical mask (roiInd) where any non-zero pixel in *original_volume* is marked as true (indicating it's part of the prostate or tissue). Then, using *ind2sub*, we convert the linear indices of those non-zero pixels into 3D coordinates (r_all, c_all, s_all). The prostate center is simply the average of all those coordinates, effectively giving us the center of the prostate.

Listing 12: sliceWiseTumourHeu.m

```
roiInd = original_volume > 0;
[r_all, c_all, s_all] = ind2sub(size(original_volume), find(roiInd));
prostateCenter = [mean(r_all), mean(c_all), mean(s_all)];
```

We then iterate through each slice and for each of the we extract the i-th 2D slice from both the *original_volume* (to analyze the pixel values) and the *segmentedMask* (to see which regions are labeled

as parts of the prostate, tumor, etc.).
A background is identified and some cluster of non-background pixels are identified.

Listing 13: sliceWiseTumourHeu.m

```
vortex_slice = [ ...
    segmentedSlice(1, 1), ...
    segmentedSlice(1, end), ...
    segmentedSlice(end, 1), ...
    segmentedSlice(end, end) ...
];
background_slice = mode(vortex_slice);
clusters = unique(segmentedSlice);
clusters(clusters == background_slice) = [];
```

To understand how a cluster works against the other, metrics are intialized:

- **clusterMean**: The mean intensity value of the cluster. It represents the average brightness of the pixels in that cluster. Tumors typically have a lower intensity than surrounding tissues, so this value is useful for distinguishing between tumor and non-tumor regions.
- **clusterStd**: The standard deviation of the intensity values within the cluster. It measures the heterogeneity of the cluster's intensity. Tumors are often more heterogeneous (i.e., they vary more in intensity) than normal tissues, so this metric can help identify regions of interest.
- **clusterEntropy**: The entropy of the cluster's pixel intensity distribution. Entropy is a measure of texture complexity. A higher entropy indicates a more complex, less uniform texture, which is often found in tumors.
- **clusterCentroidDist**: The distance from the cluster's centroid (the center of mass of the pixels in the cluster) to the prostate center. Tumors that are closer to the prostate center are often more significant in the context of prostate cancer, so this metric helps prioritize potential tumor regions.

And for each cluster these are computed. The metrics for each cluster are normalized to the range [0, 1]. This ensures that all metrics are on the same scale, avoiding any one metric dominating the tumor score calculation.

Listing 14: sliceWiseTumourHeu.m

```
tumorScore = (1 - normMean) + normStd + normEntropy - params.weightSpatial *
    normCentroidDist;
```

This score is calculated basing on the normalized metrics, where:

- **(1 - normMean)**: Tumors are usually darker, so lower mean intensity is preferred. This term emphasizes the idea that tumors often have a lower average intensity compared to surrounding tissue, and thus the score benefits when the mean intensity is lower.
- **normStd**: Tumors tend to be more heterogeneous, so higher standard deviation is favored. Tumors generally show more variation in their intensity values, so this term rewards clusters with higher variability.
- **normEntropy**: Tumors often have more complex textures, so higher entropy is favored. The texture of tumor regions is typically more complex (i.e., less uniform), so this term gives a higher score to regions with more varied intensity distributions.

- **- params.weightSpatial * normCentroidDist**: Tumors that are closer to the prostate center are given a higher score, but we control how much this matters with `weightSpatial`. This term penalizes clusters that are farther from the prostate center, with the influence of this penalty being adjusted by `weightSpatial`.

We then select the tumor mask of the cluster with the highest score and update the tumor mask.

Listing 15: sliceWiseTumourHeu.m

```
[~, tumorClusterIdx] = max(tumorScore);
tumorClusterLabel = clusters(tumorClusterIdx);
tumorMask(:,:,i) = (segmentedSlice == tumorClusterLabel);
```

4 Chapter 4: Standard Approaches

4.1 Otsu

Otsu is a image processing technique based on **tresholding** that aims to separate a backgorund of an image to his foreground. It is a histogram-based approach that finds an optimal threshold value by minimizing intra-class variance between foreground and background.
So basically it compute the image histogram by simply counting the number of pixels for each class. Normalize the histogram so that those intensities become probabilities. Then, by iterating all possible tresholding *T*, we split the image in two classes: *Class1* (pixel density $\leq T$) and *Class2* (pixel density $> T$), and also we compute:

- $w_1(T)$ and $w_2(T)$ as probabilities for each class.
- $\mu_1(T)$ and $\mu_2(T)$ as mean intensity for each class.
- $\sigma_1^2(T)$ and $\sigma_2^2(T)$ as variance for each class.

We calculate the **intra-class variance** $\sigma_w^2(T)$:

$$\sigma_w^2(T) = w_1(T)\sigma_1^2(T) + w_2(T)\sigma_2^2(T) \tag{6}$$

and find the best Tresholding *T*.
In our case, we have choosen to split the image in three regions.

Listing 16: OtsuTreshold.m

```
function [map, Otsu_T] = OtsuThreshold(volume, ncluster)
    % OtsuThresholding function performs the otsu thresholding given the
    % volume and number of clusters (thresholds)
    dims = size(volume);
    volume = reshape(volume, [], 1);
    Otsu_T = multithresh(volume,ncluster);
    map = imquantize(volume, Otsu_T);
    map = reshape(map, dims);
end
```

The Otsu Function works as follow, image dimensions are taken and reshaped in a 1D vector. The function then calls `multithresh()`, which computes ncluster threshold values. These thresholds separate the intensity values into **ncluster + 1 distinct groups**. The result is a set of threshold values that define the boundaries between different regions. Once the thresholds are determined, the function uses `imquantize()`, which takes these threshold values and assigns each voxel in the volume to a

region based on which threshold range it falls into.
The segmented array is later reshaped into a 3D form.

The Function is called in different Matlab file, in order to have a more modular code. In this file `otsu_main.m` there are the pre process phase and data preparation that are constructed like a Pipeline, the input image loaded by the DataLoader(2.2) is rescaled by `mat2gray`, normalized by a 3D MinMax (3.1) and dimensionality reduced by a PCA 3.3 and all the filters discussed in sections (3.2.1),(3.2.2),(3.2.3) are applied. To the denoised image a Log Transformation (3.5) is applied. All the parameters for the pre processing phased are specified.

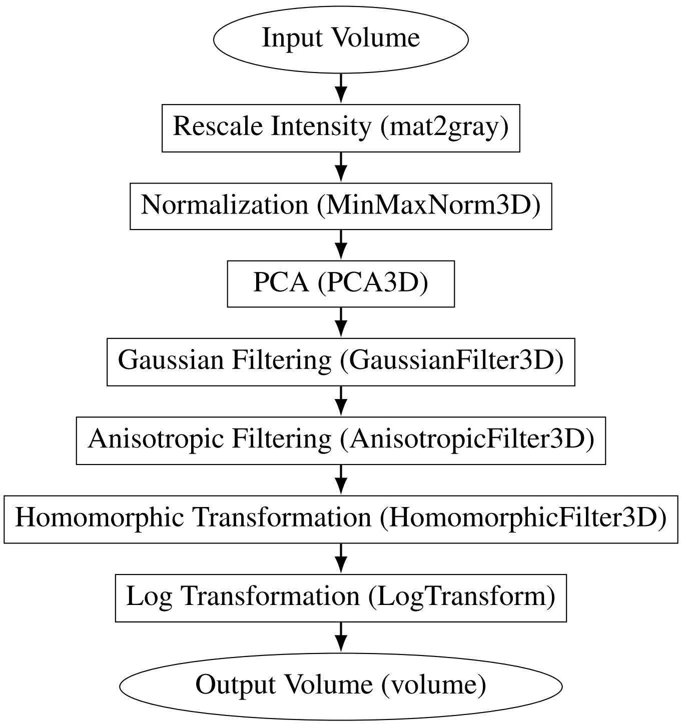
Input Volume
Rescale Intensity (mat2gray)
Normalization (MinMaxNorm3D)
PCA (PCA3D)
Gaussian Filtering (GaussianFilter3D)
Anisotropic Filtering (AnisotropicFilter3D)
Homomorphic Transformation (HomomorphicFilter3D)
Log Transformation (LogTransform)
Output Volume (volume)

Figure 1: Otsu Pipeline

At the end small and large gaps are filled in the segmented maks, thanks to `imclose()` and `imfill()` function and labels for each cluster are assigned.

The train phase starts with the `fitOverlapMask` (3.7). This helps to check and observe the initial model's performances.
The testing phase starts by choosing the heuristic values, follow the same preprocessing and then it is used to observe model performances using `sliceWiseTumourHeu` and without fit overlap mask.

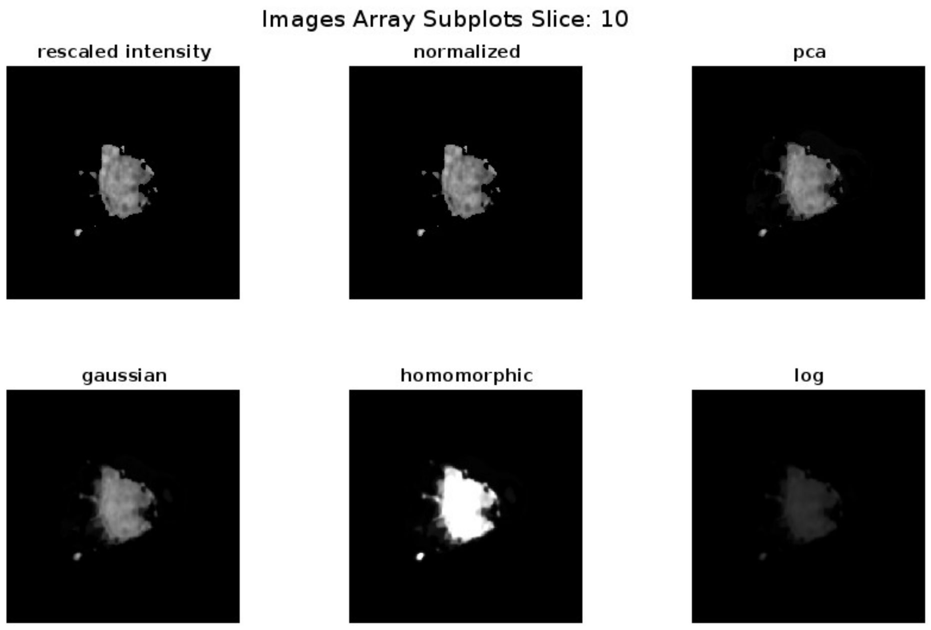

Figure 2: Otsu pre-processing steps [SUBJECT 15] - center slice

4.2 Watershed

Watershed segmentation is a region-based image segmentation that threats images as topograhpic surfaces. Watershed considers *high-intenisity regions as "mountains"* and *low-intensity regions as "valleys"*. It then floods the valleys with water, **simulating a rainfall or flooding process**[3]. As the water fills up the valleys, it forms "watershed lines" at the places where different water sources meet. These watershed lines serve as the segmentation boundaries.

It works by calculating the **gradient magnitude** of the image as:

$$G(x, y) = \sqrt{(\frac{\partial I}{\partial x})^2 + (\frac{\partial I}{\partial y})^2} \tag{7}$$

where $I(x, y)$ is the image density of the $p(x, y)$pixel. Then the watershed satisfy the function

$$\min\{W(x', y')|(x', y') \text{ is a neighbour of } (x, y)\} \tag{8}$$

ensuring that waters flows from High density areas to low density areas.
So it since is a region-based segmentation a Distance based transformation is applied to enchance object centers:

$$D(x, y) = \max_{(x',y')} ||(x, y) - (x', y')|| \tag{9}$$

a treshold is applied:

$$T(x, y) = 0.7 \times \max(D(x, y)) \tag{10}$$

Then the unbeknown regions is computed as $U = SureBackground - SureForeGround$.

The function workflows follow the same pattern as the OtsuFunction (4.1). The pre process phase works as a pipeline where the images are loaded by the DataLoader(2.2), and splitted in *train and test*. A train image is rescaled by mat2gray, normalized by a 3D MinMax (3.1). The image is also inverted and to the inverted image Anisotropic Filter [3](3.2.2) and Homomorphic filter(3.2.3) are applied. The denoised image a 3D shaper transofrmation is applied (3.6) and a Log Transormation (3.5). All the parameters for the pipeline are specified in `watershed_main.m`.

After all of that a post-processing step is also applied to removes small imperfections derived from small objects, and all the parameters for the trainer are assigned. Now the trainer is invoked.

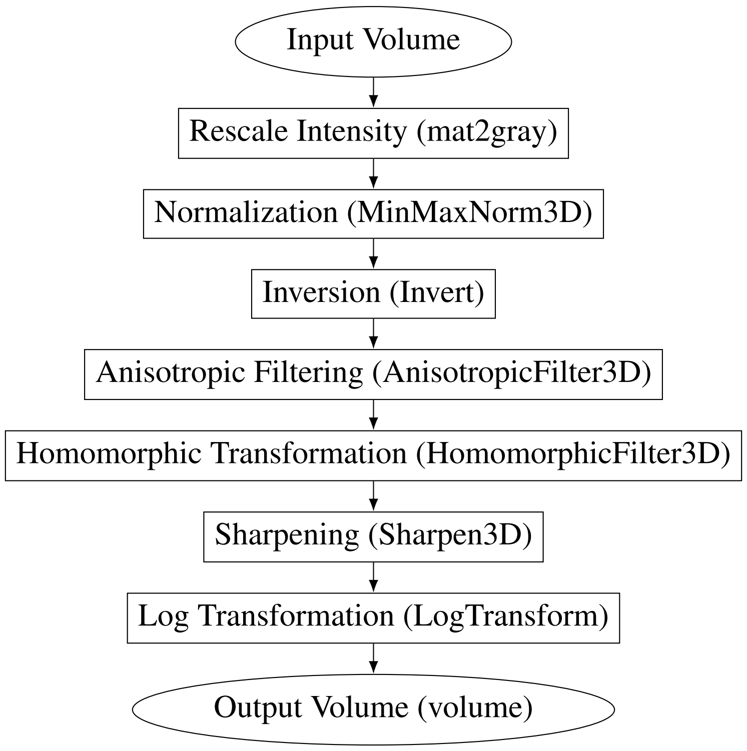
Input Volume
Rescale Intensity (mat2gray)
Normalization (MinMaxNorm3D)
Inversion (Invert)
Anisotropic Filtering (AnisotropicFilter3D)
Homomorphic Transformation (HomomorphicFilter3D)
Sharpening (Sharpen3D)
Log Transformation (LogTransform)
Output Volume (volume)

Figure 3: Watershed Pipeline

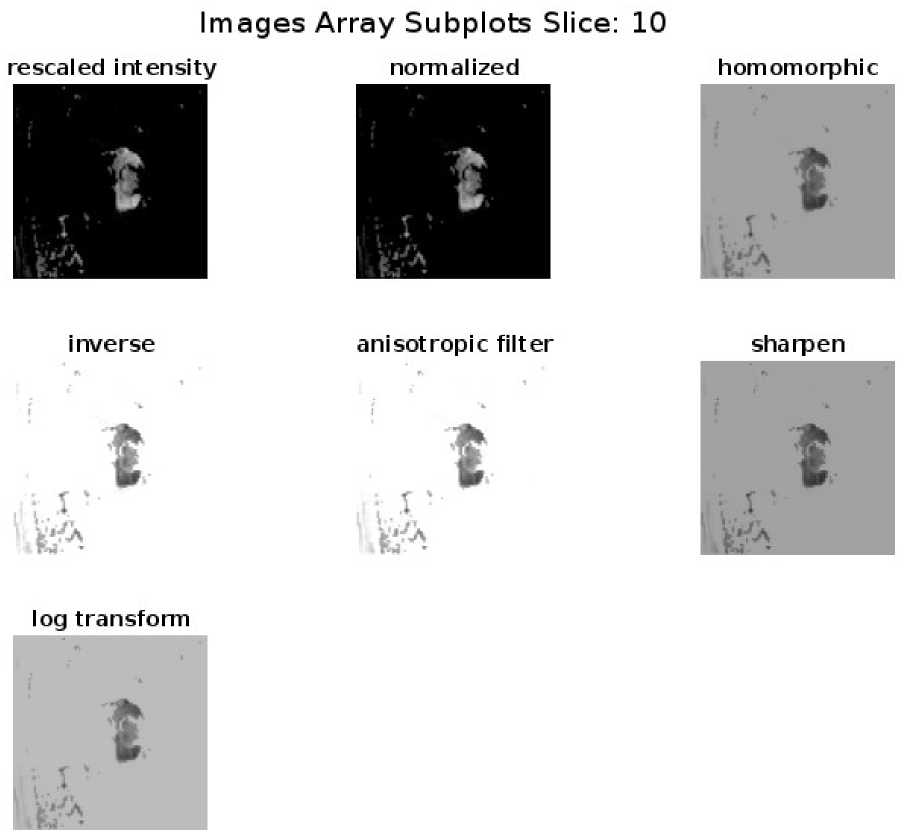

Figure 4: Watershed pre-processing steps [SUBJECT 12] - center slice

The trainer functions takes as input the 3D image now filtered, all the parameters for the segmentation and gives as output the 3D binary mask of the segmented regions, the best `minimaDepth` based on the training and the best score.
Before the train phase all the parameters are checked. Later the model variables are initialized these are: *minimaDepthValues* deriving from the min depth of the image, a way to track the store and the *segmentedVolume*. If any of the parameters is missing they're initialized by hand.

The trainer function, `waterShedTrainer` takes as input a `filterdVolume`, that is the image after all the preprocessing, a `trueMask` (A 3D binary ground-truth mask for segmentation), a `smallObjectsThreshold` (threshold) for removing small objects and some boolean flags that helps with visualization. It gives as outputs, the best segmented 3D mask, the minimal optimal depth for segmentation and the best Dice.
The process starts by determining the number of slices in the 3D volume. To track segmentation accuracy, a variable, `bestDice` is initialized to store the best result. A starting value for optimization is also set based on a predefined range. An empty binary mask is created to hold the final segmented output, while another structure is prepared to store visual representations of the segmented slices for easier analysis.

Listing 17: watershedTrainer.m

```
numSlices = size(filteredVolume, 3);
bestDice = 0;
optimalMinimaDepth = minimaDepthRange(1);
segmentedVolume = false(size(filteredVolume)); % Initialize the final segmented
    volume
mosaicWatershed = zeros(size(filteredVolume,1), size(filteredVolume,2),
    numSlices, 3, 'uint8');
```

The contrast is enhanced slide-by-slide. This helps to have a better contrast and improving feature detection. To do so, the `adapthisteq()` function is applied.
By iterating over different minimal depth values (this helps to have a better segmentation) a binary mask is created to store current segmentation. Then, each slide is processed by extracting the current slide and the corresponding ground truth and gradient magnitude. Then, the regional minimal is considered as **marker** and then applies the watershed for segmentation.

```
for i = 1:numSlices
        slice = enhancedVolume(:, :, i); % Current slice
        trueSlice = trueMask(:, :, i); % Ground truth for current slice

        % Compute gradient magnitude
        gradientSlice = imgradient(slice); % gradient magnitude

        % Marker-controlled watershed
        markers = imextendedmin(slice, depth); % Generate markers
        imposedGradient = imimposemin(gradientSlice, markers); % Impose markers
        watershedLabeled = watershed(imposedGradient); % Apply watershed
```

The tumor regions is extracted, this is the region with the **highest overlap**. And create a **Binary Mask** as the bestLabel as the segmented region.

```
uniqueLabels = unique(watershedLabeled(:));
bestLabel = 0;
maxOverlap = 0;

for label = uniqueLabels
    overlap = sum((watershedLabeled == label) & trueSlice, 'all');
    if overlap > maxOverlap
        maxOverlap = overlap;
        bestLabel = label;
    end
end
tempSegmentedVolume(:, :, i) = (watershedLabeled == bestLabel);
```

At the end there is a post process step, where gaps are filled and small objects are removed to enhance performances. As always, metrics are calculated and optimal parameters are updated.

After the train phase, as always there is the **test phase**.
To maintain the modularity of the project the test phase is a separate function. The input are the same as the train phase function explained before, there is just a new key input that is the `heuristic_params` that is a parameters for a heuristic tumor detection function.
The first step of the test phase works just as the train phase, the input image is taken, pre processed and sliced. For each slice a gradient magnitude is computed and a marker (region minima) is applied. Now, in order to have a better result, boundary artifacts are removed. First, we create a mask that includes everything except the boundary regions. Then, we use the Euclidean distance transform (bwdist) to figure out the nearest nonzero pixel for each boundary pixel. Finally, we fill in those boundary pixels by replacing them with their closest nonzero neighbor.

Listing 18: watershedTester.m

```
mask = (watershedLabeled > 0);
[~, idx] = bwdist(mask, 'euclidean');
watershedLabeled_filled = watershedLabeled;
watershedLabeled_filled(~mask) = watershedLabeled(idx(~mask));
```

Since the test phase is done thansk to an heuristic function, the image is convert to grayscale to enhance performances.

In the end, the `sliceWiseTumorHeu` (3.8) is applied and the result image is post processed, gaps are filled and small objects are removed to enhance performances.

5 Chapter 5: New Approaches

5.1 KMedROI and kMeans

The **KMedROI** is a technique that combines **K-means clustering** and **Region of Interest (ROI)** technique, often used in medical image analysis. While ROI refers to a specific area in an image that is of particular interest for further analysis, K-Means is a method to cluster data in a specified cluster [2]. In image processing K-Means is used to segmentate images by grouping pixels with similar intensity or colour values in different cluster.

5.1.1 KMeans

KMeans is a *clustering* algorithm that aims to split all data in *k* clusters in the best efficient way possible [4]. It worsk as follows:

- K cluster centres are choose, μ_k. Is not mandatory that the points is a data of the dataset.
- All point are associated to the nearest cluster center;
- The mean of the distance of each cluster point to its cluster center is calculated. This is the new centroids.
- Repeat until convergence.

Algorithm 1 K-Means Clustering Algorithm

```
Initialize $K$ cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$ randomly
repeat
    for $i = 1$ to $m$ do
        $c^{(i)} \leftarrow \arg\min_{k \in \{1,\ldots,K\}} d(x^{(i)}, \mu_k)$        ▷ Assign each data point to the nearest cluster
    end for
    for $k = 1$ to $K$ do
        $\mu_k \leftarrow \frac{1}{|C_k|} \sum_{x^{(i)} \in C_k} x^{(i)}$        ▷ Update centroids as mean of assigned points
    end for
until convergence
```

Despite all its wide use, Kmeans, is sensitive to the initial centroid placement and may converge to a local minimum.
KMedROI will also be used for the next pipeline training and testing; Follows the explanation of the `'kMedROI'` function and the `'sliceWiseROI'`, a function necessary to define the ROI of an image based on the density of the slice itself.

5.1.2 kMedROI function

`'kMedROI'` is the principal function that takes an input volume and a set of parameters, preparing the volume for segmentation through normalization, smoothing, clustering, morphological operations, and ROI masking. The pipeline concludes with active contouring to refine the region of interest. The function returns the masked volume that isolates the area of interest. [2]

- The volume is first rescaled to [0,1] and normalized to reduce intensity disparities.
- Gaussian and median filters are then applied to smooth the volume and reduce noise.
- The filtered data undergoes k-Means clustering, separating the volume into multiple regions.
- Morphological operations (opening and closing) eliminate small artifacts.
- A slice-wise masking step focuses on the central zone of interest in each 2D slice.
- Finally, a gradient-based edge map and Chan-Vese active contouring refine the ROI boundary, and the mask is applied to the original volume.

kMedROI checks and initializes parameters with **init_kMedROI**, then normalizes and filters the volume before applying k-Means clustering. After morphological smoothing of the clustered volume, **sliceWiseROI** imposes a 2D circular mask on each slice, potentially adjusting its radius based on density, as configured by **init_penalizer**. A **Sobel-based gradient map** then refines the edges, and a Chan-Vese active contour further sharpens the boundary. The final mask is applied to the original volume, yielding the maskedVolume ready for segmentation. This pipeline is designed to **highlight and isolate the region of interest in 3D data**, ensuring that subsequent segmentation techniques can work on a cleaner, more focused volume.

- Load and Initialize Parameters:

```
[param] = init_kMedROI(param);
```

- Preprocessing:

```
% Rescale intensity to [0,1]
volume = mat2gray(volume);
 % 3D normalization
volume = MinMaxNorm3D(volume);
volume = GaussianFilter3D(volume, param.gaussian_sigma, false, true);
% Further smoothing
volume = medfilt3(volume);
```

- k-Means Clustering [reduction clustering]:

```
[clustered_volume, ~] = KmeansThresholding(volume, param.kmeans_clusters);
```

- Morphological Operations (open/close):

```
clustered_volume = imopen(clustered_volume, strel('sphere', param.
    sphere_opening1));
clustered_volume = imclose(clustered_volume, strel('sphere', param.
    sphere_closing1));
```

- Slice-wise Masking + Sobel Gradient:

```
roi_masking = sliceWiseROI(clustered_volume, param.roi_param);
gradient_magnitude = imgradient3(roi_masking, 'sobel');
% ...
```

- Active Contour[3] + masking

```
contoured_mask = activecontour(volume, gradient_mask, param.ac_iters, 'Chan
    -Vese');
maskedVolume = original_volume .* contoured_mask;
```

5.1.3 kMedROI: 'init_kMedROI'

The function `'init_kMedROI'` ensures that all necessary parameters for kMedROI are present and sets defaults if they are missing. Typical parameters include sigma values for Gaussian filtering, the number of clusters for k-Means, the size of structuring elements for morphological operations, and the number of iterations for the active contouring process.

5.1.4 sliceWiseROI

`'sliceWiseROI'` applies a two-dimensional circular mask to each slice. The center and radius of this circle can be static or adjusted based on the density of non-background pixels in each slice, allowing the radius to shrink or expand when certain penalizations are in use. This function identifies background intensity by looking at corner voxels, assuming that these represent regions outside the object of interest.

```
for sliceIndex = 1:slices
    % Center: (centerX, centerY)
    % Calculate 'radius' based on base_scaler and possible penalty
    roi_mask = sqrt((X - centerX).^2 + (Y - centerY).^2) <= radius;
    masked_slice(~roi_mask) = background;
end
```

Density Calculation:

```
valid_pixels = current_slice(current_slice ~= background);
density = numel(valid_pixels) / (rows * cols);
```

A slice with a high number of non-background voxels has a higher density.

Penalization:
Depending on the user-selected method (log, sigmoid, or milder), a penalty factor is computed. This factor either weakens the radius growth for high densities or does so more gradually for lower densities, ensuring that if a slice has a lot of valid content, the mask doesn't become so large that it includes irrelevant regions. Different penalization methods are used to control the region of interest (ROI) expansion based on density. Each method modifies how aggressively the mask shrinks or expands.

- **Logarithmic Penalization:** The log-based penalty is designed to heavily penalize low-density slices while stabilizing for high-density regions. It ensures that regions with fewer valid pixels are significantly reduced, while for densely packed slices, the effect of the penalty gradually saturates, preventing overly small circles.

  $$\text{penalty} = \frac{1}{1 + \log(1 + \text{density})} \tag{11}$$

  This approach is particularly useful when the density varies widely across slices, as it prevents excessive shrinking of the ROI in denser areas.

- **Sigmoid Penalization:** The sigmoid function provides a smooth transition between low and high densities. Unlike the logarithmic approach, it does not strongly penalize low-density slices but instead ensures a gradual transition where penalties saturate at both ends.

  $$\text{penalty} = 1 - \frac{1}{1 + e^{-k(\text{density}-\text{threshold})}} \tag{12}$$

  The sigmoid threshold and scaling factor k control how sharply the transition occurs. This method is particularly useful when a balance is needed between penalization strength and adaptability.

- **Milder Penalization:** The milder penalty function uses a square root transformation, ensuring a slower reduction in penalty as density increases. It prevents drastic changes in the mask size while still maintaining control over how much the ROI grows.

$$\text{penalty} = \frac{1}{1+\sqrt{\text{density}}} \tag{13}$$

  Compared to the logarithmic and sigmoid approaches, this method provides a more conservative reduction in penalty, making it suitable for cases where gradual adaptation is preferred.

Each of these approaches serves a different purpose in controlling the ROI expansion, ensuring that segmentation remains focused while adapting dynamically to the anatomical structures present in the image.

5.1.5 sliceWiseROI: 'init_penalizer'

When the pipeline needs to adjust the radius of the circular mask according to density, init_penalizer configures the type of penalization. Options include a logarithmic, sigmoid, or milder approach, each modifying how the circular radius shrinks or grows as the slice density changes. Set Defaults for Gaussian sigma, k-means clusters, morphological structuring elements, and active contour iterations if missing.

5.2 KMedROI + kMeans

The sequence of operations begins by loading the data, initializing performance metrics, setting up parameters for key preprocessing functions (notably kMedROI), and then calling specialized training routines.
A custom data loader (`'DataLoader'`) is used to import volumetric images of the prostate. Only the training subset is extracted at first, so that ground-truth masks (if available) can be leveraged for metric computation and for methods that may require overlap-based calibration. Several metrics such as average Dice coefficients and Intersection over Union (IoU) are initialized to zero to track performance across the data. The pipeline parameters (pipeline_param) and model parameters (model_params) govern the type of transformation, clustering strategy, and post-processing methods applied to the data. For instance, a watershed, kmeans or otsu-based approach can be chosen, or it can be set to "None" if no additional pipeline steps are desired after the main kMedROI procedure. These parameters help ensure that each stage—be it anisotropic filtering, homomorphic filtering, or advanced morphological operations—can be flexibly toggled or configured according to experimental needs.

5.2.1 kMedROI parameters

A set of parameters for kMedROI are specified, including a base_scaler that influences how large the circular mask will be around each slice's center, a boolean toggle that allows the mask's radius to adapt based on density, and a penalization strategy (e.g., "log") dictating how the radius shrinks or expands when a slice is highly populated with non-background voxels.

5.2.2 Pipeline parameters

These govern additional transformations or filters applied before or after the main segmentation. The code shows an example of configuring an anisotropic filter (with a specified number of iterations and a gradient threshold) and a homomorphic filter to enhance contrast in different frequency bands. However, the pipeline can be switched to "None" if those steps should be bypassed.

5.2.3 Model and Morphological operations parameters

Subsequently, parameters are defined for the model necessary for a second clustering stage, in this case **k-Means** with a specified number of clusters, that can refine the masks generated by kMedROI [4]. This second pass helps to differentiate subregions in the volume that were not well separated in the first segmentation. A local function **(morpho_func)** demonstrates a typical morphological refinement procedure for the final segmentation mask. After holes in the binary mask are filled, minor openings and closings are performed with small spherical structuring elements to remove noise or fill small gaps. The **'morpho_func'** can be add to the pipeline or excluded in any moment. For data that has ground-truth labels available, the code can use a **"fitOverlapMask"** approach (use_fit_overlap = true) to select which cluster most likely represents the tumor. This approach checks the overlap between each cluster and the ground-truth mask, thereby helping the algorithm refine its classification of tumor versus non-tumor clusters and helps to find best parameters for the pre-processing phase.

5.2.4 Test set

The testing phase, uses again the TrainerMedROI function but on the test subset, with identical pipeline. This time, the use_fit_overlap flag is turned off, meaning that the ground-truth is not used to decide which cluster is tumor. Instead, a heuristic-based method (sliceWiseTumourHeu) is employed—guided by user-defined parameters such as the number of color bins for entropy calculation, a spatial weighting factor, or simply "skip" if no further heuristic is desired. By comparing these results to the known ground-truth, the system can quantitatively assess how well the model and the heuristic generalize in identifying tumor regions for previously unseen data.

5.3 KMedROI and Otsu

The code follows the same pipeline as kMedROI + kMeans, the main differences with the previous paragraph follow: The segmentation process is explicitly set to utilize an Otsu-based approach, as indicated by the assignment

```
pipeline_param.type = 'otsuPipeline'.
```

Additionally, several filtering parameters are specified to refine image quality, including the number of iterations for anisotropic filtering, the gradient threshold for edge preservation, and homomorphic filter settings that enhance contrast in different frequency bands. These preprocessing steps help to optimize the input data for subsequent segmentation stages. The secondary clustering stage refines the segmentation masks generated by the kMedROI method. In this implementation, the code employs Otsu clustering, with the number of clusters set to two, ensuring effective separation of distinct regions within the volumetric data.

```
model_params.type = 'otsu';
model_params.nclusters = 2; % clusters for otsu algorithm to segment the final
    mask
```

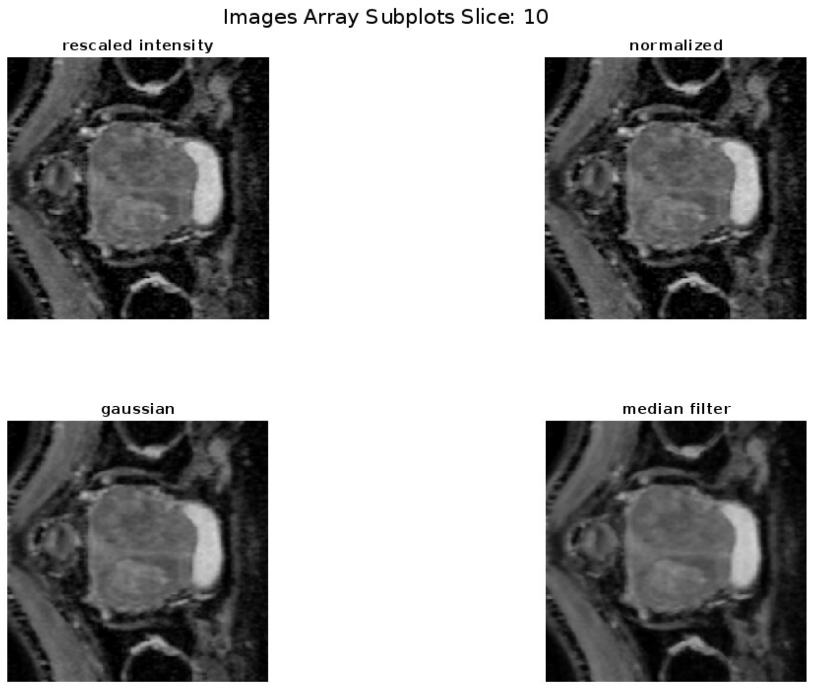

Figure 5: kMedROI pre-processing steps [SUBJECT 1] - center slice

Feature	**kMedROI + kMeans**	**kMedROI + Otsu**
Clustering	k-Means after kMedROI (`model_params.type = 'kmeans'`).	Otsu after kMedROI (`model_params.type = 'otsu'`).
Pipeline	`pipeline_param.type = 'watershedPipeline'`	`pipeline_param.type = 'otsuPipeline'.`
Morphological	uses `imfill, imopen and imclose` [strel('sphere', 3)]	uses `imclose` [strel('sphere', 2)] and `imfill`
Heuristic	heuristic_params.nbins=64; heuristic_params.weightSpatial=0.4;	heuristic_params.nbins=128; heuristic_params.weightSpatial=5;

Table 1: Differences between kMedROI+kMedROI and kMedROI+Otsu
NOTE: `'nbins' define entropy colors bins and 'weightSpatial' penalize clusters distant from center (since is a centered ROI image)`

5.4 Pipelines Table

Step / Transformation	kMedROI + kmeans	Watershed	Otsu	kMedROI + Otsu
kMedROI	✓	✗	✗	✓
mat2gray	✓	✓	✓	✓
MinMaxNorm3D.m	✓	✓	✓	✓
Invert.m	✓	✓	✗	✗
PCA3D.m	✗	✗	✓	✓
GaussianFilter3D.m	✗	✗	✓	✓
AnisotropicFilter3D.m	✓	✓	✓	✓
HomomorphicFilter3D.m	✓	✓	✓	✓
Sharpen3D.m	✓	✓	✗	✗
LogTransform.m	✓	✓	✓	✓
kMeans Clustering	✓	✗	✗	✗
Watershed	✗	✓	✗	✗
Otsu Threshold	✗	✗	✓	✓

Table 2: Comparison of preprocessing and model steps across four different pipelines. A check mark (✓) indicates inclusion of the step, while a cross (✗) indicates exclusion.

Listing 19: Paramters Initialization

```
% kMedROI params set for this dataset
kMedROI_param.roi_param.base_scaler = 0.4;
kMedROI_param.roi_param.use_density_adapt_radius = true;
kMedROI_param.roi_param.type = 'log';
```

```
% kMedROI + kMeans | same parameters of Watershed
pipeline_param.type = 'watershedPipeline';
pipeline_param.anisotropic_iterations = 5; % anisotropic iterations
pipeline_param.anisotropic_gthreshold = 0.06;
pipeline_param.D0 = 150; % cut-off for homomorphic filter
pipeline_param.gammaH = 1; % gain for high freq
pipeline_param.gammaL = 0.1; %gain for low freq
pipeline_param.smallObjThreshold = 200; % pixel threshold bwopen
pipeline_param.k = 1.5; pipeline_param.N = 1; % Log constants
model_params.type = 'kmeans';
model_params.nclusters = 3; % clusters for second kmeans
```

```
% kMedROI + Otsu parameters
pipeline_param.type = 'otsuPipeline';
pipeline_param.tau = 0.95; % CEV to keep
pipeline_param.sigma = 0.9; % gaussian sigma value
pipeline_param.anisotropic_iterations = 5; % anisotropic iterations
pipeline_param.anisotropic_gthreshold = 0.1;
pipeline_param.D0 = 1.5; % cut-off for homomorphic filter
pipeline_param.gammaH = 2; % gain for high freq
pipeline_param.gammaL = 1; %gain for low freq
pipeline_param.k = 0.1; % log transform
pipeline_param.N = 0.1; % log transform
pipeline_param.logtype = 'log'; % log transform
model_params.type = 'otsu';
model_params.nclusters = 2; % second segmentation with otsu
```

6 Chapter 6: Evaluation

In this section, we present the performance evaluation of our proposed segmentation approach. We compare different models and preprocessing strategies to assess their effectiveness in accurately identifying the region of interest. Quantitative results, including Dice and IoU metrics, are provided to measure segmentation accuracy, while qualitative visualizations offer further insights into model performance. Additionally, we analyze the impact of preprocessing steps, such as KMedROI masking, on the clustering process and overall segmentation quality. The findings highlight the advantages of our method in reducing the influence of anomalous structures and improving segmentation precision.

6.1 Metrics

To evaluate the performances of all the four models explained we have embraced two metrics, widely used in the image proccesing and segmentation task. These metrics are widely used for segmentation tasks, especially in prostate segmentation. These metrics are:

- Intersection over Union (IoU);
- Dice (similarity) Coefficient;

6.1.1 IoU

IoU, also called the *Jaccard Index*, measures the ratio of the intersection and the union of the predicted and ground truth segmentation:

$$IoU = \frac{|P \cap G|}{|P \cup G|} \tag{14}$$

where:

- $|P \cap G|$: is the intersection of the prediction and ground truth.
- $|P \cup G|$: is the union of the prediction and ground truth.

If *IoU* is equal to 1 we have a perfect segmentation, if is equal to 0 there is no overlap. So an higher *IoU* is preferable.

6.1.2 Dice

The Dice Coefficient is a measure of overlap between the predicted segmentation (*P*) and the ground truth (*G*). It is calculated as:

$$Dice = \frac{2|P \cap G|}{|P| + |G|} \tag{15}$$

where:

- $|P \cap G|$: is the number of pixels (or voxels in 3D) common between the prediction and ground truth.
- $|P|$: is the number of pixels in the predicted segmentation.
- $|G|$: is the number of pixels in the ground truth segmentation.

If *Dice* is equal to 1 we have a perfect overlap, if is equal to 0 there is no overlap. So an higher *IoU* is preferable.

6.2 Results

6.2.1 Train Results

In this section, we assess the model's performance on the training dataset to evaluate its ability to learn meaningful representations of the region of interest. The evaluation is conducted using Dice and IoU metrics to quantify how well the model fits the training data.

Model\Metrics	IoU	Most Dense slice avg IoU	Dice	Most Dense slice avg Dice
Otsu	0.2750	0.4774	0.4148	0.4678
WaterShed	0.3526	0.6935	0.5048	0.6726
KMedROI + KMeans	**0.5125**	**0.7169**	**0.6573**	**0.6964**
KMedRoi + Otsu	0.4113	0.5954	0.5699	0.5814

Table 3: Train Phase evalution

As observed in the table, the **KMedRoi + KMeans** model demonstrates significantly superior performance on the current sample compared to other models. This improvement stems from the KMedROI step, which effectively masks out irrelevant regions, ensuring that the subsequent KMeans clustering operates exclusively on the tumor region. By eliminating extraneous anatomical structures, the model can focus solely on meaningful segmentation, leading to more precise clustering and minimizing the impact of anomalies or noise.
Just to demonstrate how well KMedROI works, we have also trained it without all the pre process.

Model\Metrics	IoU	Most Dense slice avg IoU	Dice	Most Dense slice avg Dice
KMedROI + KMeans	**0.5125**	**0.7169**	**0.6573**	**0.6964**
KMedROI(w.o. Preprocess)	0.4948	0.7038	0.6375	0.6864

Table 4: Train Phase evaluation: KMedROI vs KMedROI w.o. Preprocess

Here, we demonstrate that even without the full preprocessing pipeline after KMedROI—by directly applying KMeans to the ROI-masked original volume—removing anomalous structures significantly enhances KMeans' performance. This confirms that isolating the region of interest (ROI) effectively eliminates irrelevant anatomical features, allowing the clustering process to focus solely on the tumor region, leading to more accurate and reliable segmentation.

6.3 Test Results

The test set evaluation aims to assess the generalization capability of our proposed approach on unseen data. We examine whether the model retains its segmentation accuracy when applied to new samples and analyze its robustness to variations in anatomical structures. The results provide insight into the model's ability to accurately differentiate tumor regions in previously unseen cases, demonstrating its potential applicability in real-world scenarios.

Model\Metrics	IoU	Most Dense slice avg IoU	Dice	Most Dense slice avg Dice
Otsu	0.0474	0.1879	0.0883	0.1847
WaterShed	0.3548	0.7221	0.5134	0.6202
KMedROI + KMeans	**0.4806**	**0.8207**	**0.6335**	**0.7205**
KMedROI + Otsu	0.3337	0.6629	0.4761	0.5639

Table 5: Test Phase evaluation

The evaluation on the test set reinforces the findings from the train set evaluation, confirming the superior performance of the **KMedROI + KMeans** model compared to other approaches. By effectively isolating the region of interest and eliminating irrelevant structures, this model achieves significantly higher segmentation accuracy. The results demonstrate that the benefits observed during training—such as improved clustering precision and reduced interference from anomalous regions—are consistently maintained on unseen data, highlighting the model's robustness and generalization capability.
We also want to show how the KMedROI worsk well on unseed data of the test set. The metrics are similar to the KMedROI with all the preprocess.

Model\Metrics	IoU	Most Dense slice avg IoU	Dice	Most Dense slice avg Dice
KMedROI	**0.4806**	**0.8207**	**0.6335**	**0.7205**
KMedROI(w.o. Preprocess)	0.4779	0.7397	0.6308	0.6367

Table 6: Test Phase evaluation: KMedROI vs KMedROI w.o. Preprocess

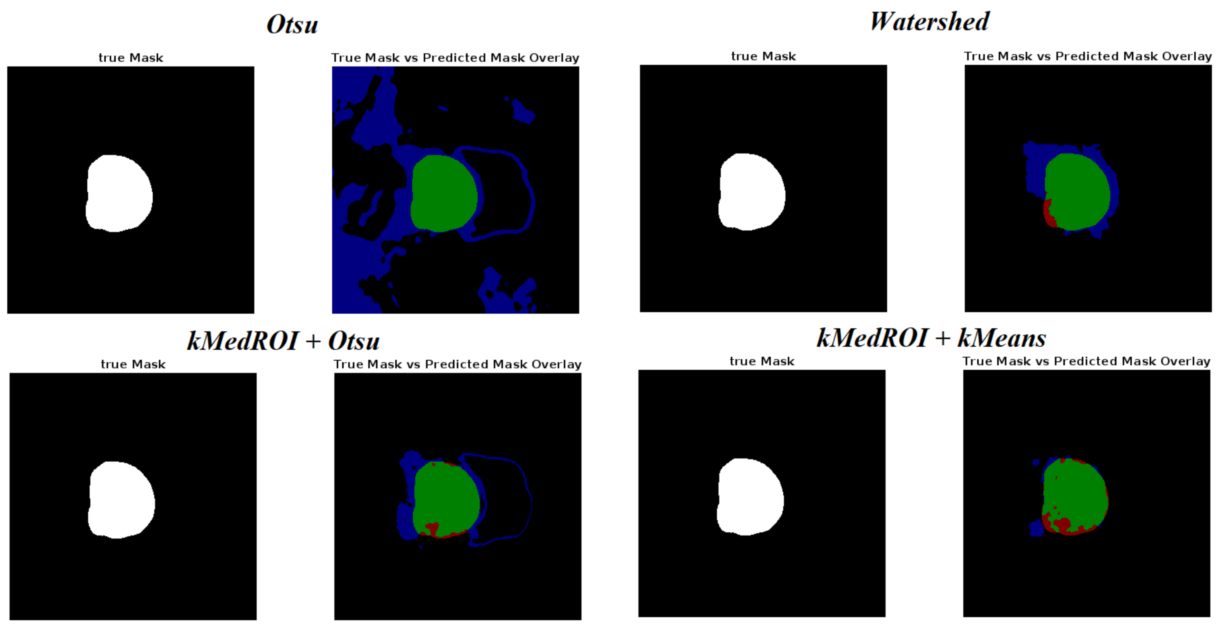
Otsu
true Mask
True Mask vs Predicted Mask Overlay
Watershed
true Mask
True Mask vs Predicted Mask Overlay
kMedROI + Otsu
true Mask
True Mask vs Predicted Mask Overlay
kMedROI + kMeans
true Mask
True Mask vs Predicted Mask Overlay

Figure 6: Example of overlap between ground truth and predicted mask [SUBJECT 2] - densest slice

6.3.1 Heart Tumor segmentation

We also ran some experiments on the 'Heart' dataset[5] (similar data structure but different organ), without any parameter tuning, simply leaving the parameters already used in previous experiments.

Model\Metrics	IoU	Most Dense slice avg IoU	Dice	Most Dense slice avg Dice
Otsu	0.0946	0.2502	0.1723	0.2433
KMedROI + Otsu	0.2978	0.6354	0.4548	0.6249
KMedROI + KMeans	**0.3148**	**0.6406**	**0.4732**	**0.6318**

Table 7: Train Phase of an Heart picture

These results further strengthen our assumptions regarding the effectiveness of kMedROI: therefore following an initial clustering and **masking of anomalous structures** the results of machine learning models such as kmeans, but also of simple thresholding models such as the otsu threshold, are significantly improved.

7 Chapter 7: Conclusion & Future Development

In this study, we proposed a segmentation pipeline leveraging KMedROI + KMeans, demonstrating its effectiveness in isolating tumor regions by masking out irrelevant structures. The experimental results, evaluated using Dice and IoU metrics, confirm that this approach significantly improves segmentation accuracy compared to other clustering models. The findings highlight the importance of preprocessing in enhancing clustering performance, particularly in medical imaging applications. Moreover, the consistency of results across both the training and test sets underscores the robustness of the proposed method. Future work will focus on extending this approach with more advanced learning techniques and broader dataset validation to further enhance its applicability in real-world clinical scenarios.

While the proposed approach demonstrates promising results in segmentation accuracy, several avenues for future improvement remain. One potential enhancement is the integration of deep learning-based segmentation methods to complement the clustering approach, potentially improving precision and robustness. Additionally, further refinement of the preprocessing pipeline, including adaptive region selection and advanced morphological filtering, could enhance the consistency of segmentation outcomes. Another important direction is the evaluation of the model on a larger and more diverse dataset to ensure its generalization across different anatomical variations and imaging conditions. Finally, real-time implementation and optimization for clinical applications could significantly improve the practical usability of this method in medical imaging workflows.

References

[1] Michela Antonelli and et al. *The Medical Segmentation Decathlon*. 2022. DOI: 10.1038/s41467-022-30695-9. URL: https://doi.org/10.1038/s41467-022-30695-9.

[2] Sayantan Bhattacharya; Apoorv Sharma; Rinki Gupta; Anupama Bhan. *Isolation of Prostate Gland in T1-Weighted Magnetic Resonance Images using Computer Vision*. DOI: 10.1109/SPIN48934.2020.9070912. URL: https://ieeexplore.ieee.org/document/9070912.

[3] Renuka Devi M N; Cauvery Raju; Rajesh T. M. *Detection of tumours from MRI scans using Segmentation techniques*. DOI: 10.1109/ICESC51422.2021.9532867. URL: https://ieeexplore.ieee.org/document/9532867.

[4] Kalyani C.S.; Mallikarjuna Swamy M.S. *Segmentation of rectum from CT images using K-means clustering for the EBRT of prostate cancer*. DOI: 10.1109/ICEECCOT.2016.7955181. URL: https://ieeexplore.ieee.org/document/7955181.

[5] *Medical Segmentation Decathlon Dataset*. https://drive.google.com/drive/folders/1HqEgzS8BV2c7xYNrZdEAnrHk7osJJ--2.