Medical image analysis

An overview of techniques and improvements in brain tumor segmentation using image processing algorithms

hands on Python & OpenCV

By
Poorya MohammadiNasab
Dr. Hossein Ebrahimpour-Komleh

University of Kashan
Department of Electrical & Computer Engineering

Medical image analysis

An overview of techniques and improvements in brain tumor segmentation using image processing algorithms

hands on python & OpenCV

Poorya MohammadiNasab
Dr. Hossein Ebrahimpour-Komleh

September, 2021

Acknowledgements

I would like to thank my teacher, mentor, and thesis supervisor Dr. Hossein Ebrahimpour for his continued guidance and support. Thanks to the electrical and computer engineering department of university of Kashan for providing me welfare and educational facilities during these years. And finally, thanks to my parents and numerous friends who endured this long process with me, always offering support and love.

Abstract

In recent years, the improvement of medical imaging techniques helps doctors to diagnose tumors. Early detection of tumors will increase the chance of treatment and plays an essential role in patient survival. In this thesis project we would like to find how to be an expert in medical image analysis by explaining fundamental concepts like medical imaging modalities and image processing techniques and study recent improvements of segmentation. Also, we provide a tutorial for python programming and OpenCV library to do some practical exercises. At the end of this project we implement a new brain tumor segmentation system based on fuzzy logic and image processing approaches.

Contents

Chapter 1

Introduction to Medical Image Analysis

1.1 Motivation

A **tumor** is an abnormal mass of tissue that forms when cells grow and divide unusually. Tumors are divided into two main categories: benign (not cancer) or malignant (cancer). Benign tumors may grow large but do not invade other parts of the body. On the other hand, malignant tumors can spread into nearby tissues.

There are different reasons for creating tumors; the first reason is the body tissue near the tumor area. But in some cases, it is brought from other parts of the body. The risk level of tumors depends on four factors:

1. Type of the tumor (benign or malignant)
2. Size of tumor
3. Location of tumor
4. Speed of tumor spread

In recent years, the improvement of medical imaging techniques helps doctors to diagnose tumors. Imaging tests are used to determine the size and location of cancer. Medical imaging is a technique and process to build a picture of the whole human body or a part of it for medical purposes.

Early detection of tumors will increase the chance of treatment and plays an essential role in patient survival. On the other hand, manual tumor segmentation (by an expert) among a large number of medical images could be hard and time-consuming. Also, it depends on the experience of the expert who does this segmentation. Therefore, we need a completely automated and precise approach to segment and detect tumors. Recently, using image processing and machine learning methods has become common due to their promising results.

A thorough analysis of the literature with the keywords 'brain tumor segmentation', 'medical image segmentation machine learning', and 'medical image analysis deep learning' on the Google Scholar search engine is performed in Fig. 1.1, queried on August 16,

2021. We can observe that the number of papers increases every year from 2013 to 2021, which means multi-modal medical image segmentation using machine learning, and deep learning are obtaining more and more attention in recent years.

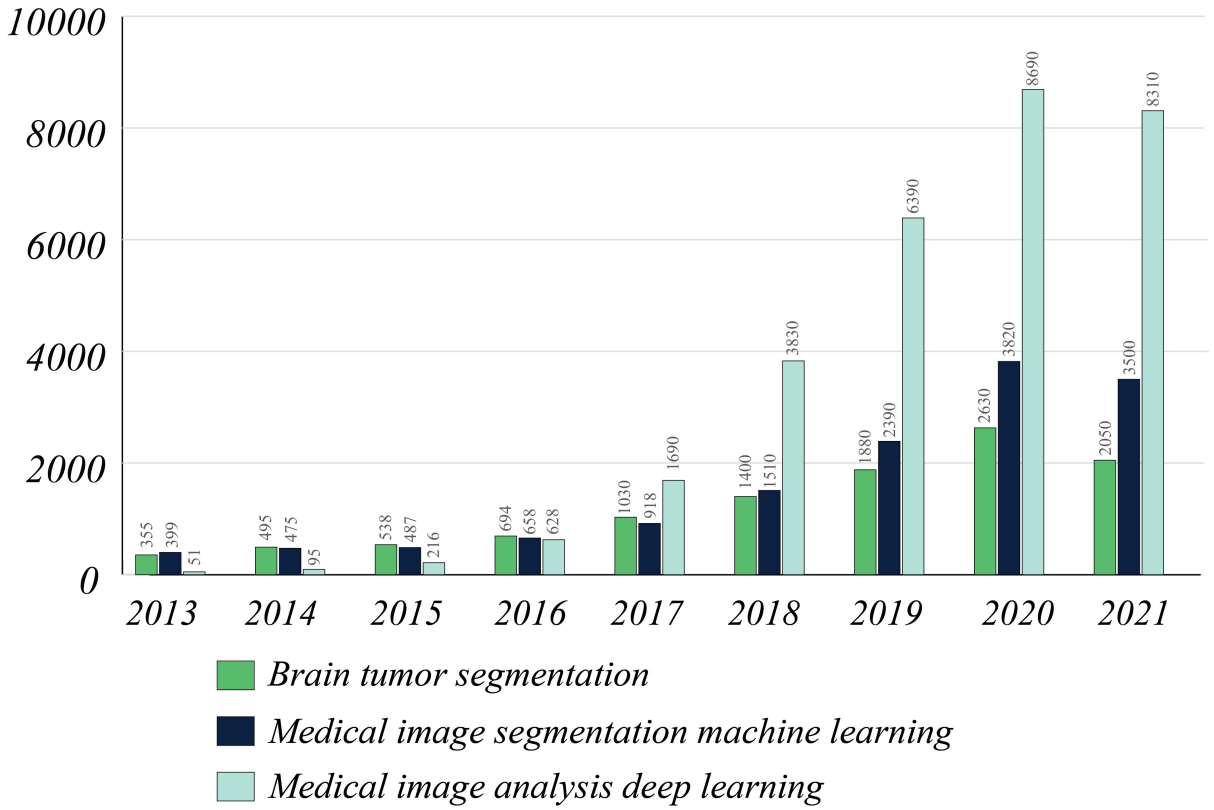


Figure 1.1: Keywords search results

1.2 what is medical image analysis

In 2020, the whole machine vision market was worth about 12.29 billion dollars, and out of this, 39% of the total share would be what will be taken down by medical image analysis itself. It will not be focused on hospitals or private health care centers or any particular modality like x-ray or ultrasound. So we can use it in different modalities, clinical indications, and different end-users. Figure 1.2 illustrates the concepts and applications of medical image analysis.

1.3 roadmap

If we look into different modalities such as CT, MRI, ultrasound, etc. The critical part is about the organ appearance module and what they mean. In other words, how different organs are going to appear in different modalities. For example, a bone will appear

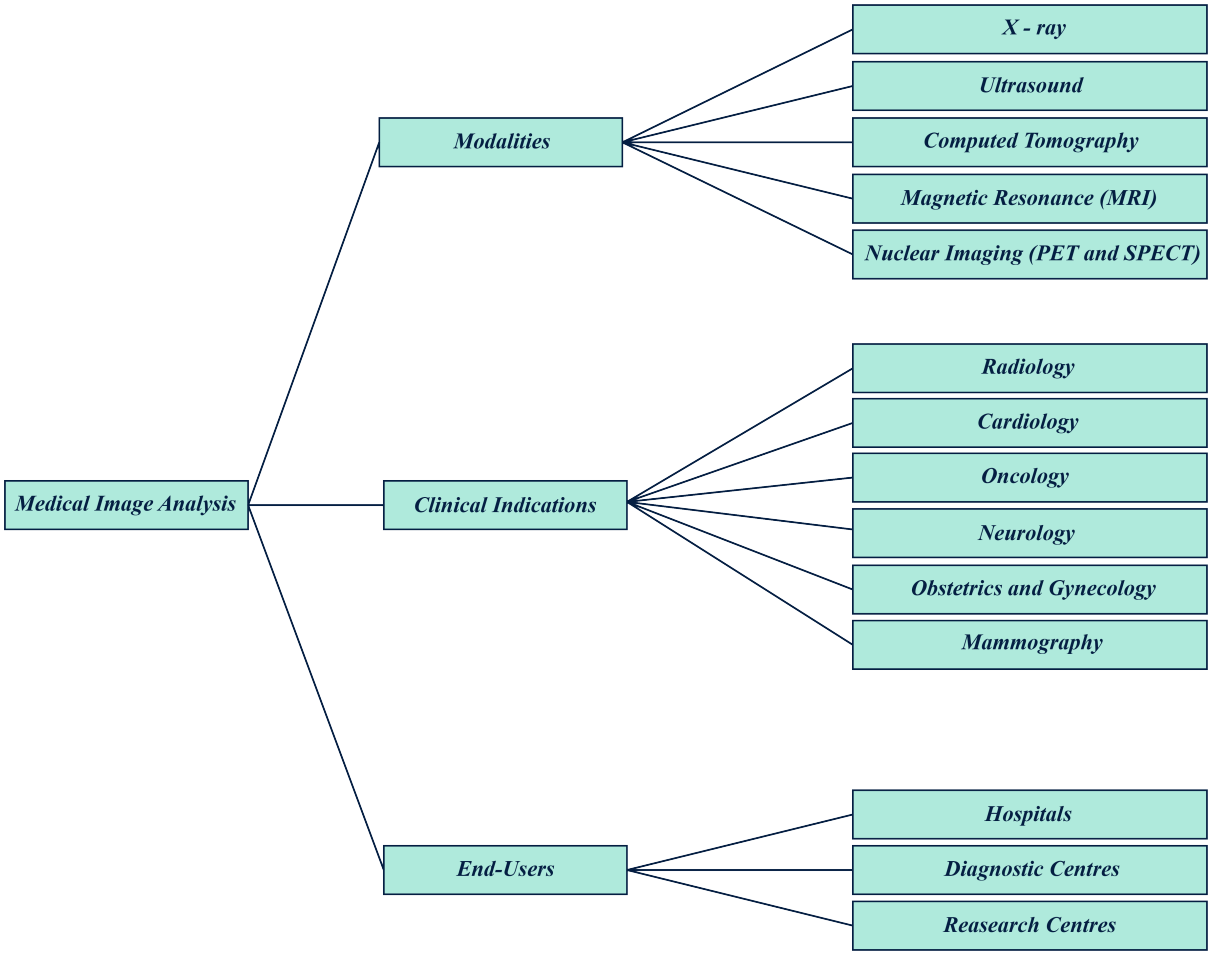


Figure 1.2: Medical image analysis (concepts and applications)

brighter on x-rays, and it would be darker on T1 MR and T2 MR. On the other hand, fatty regions appear brighter on MR and darker on x-rays; fatty areas will appear brighter on ultrasound. A water-filled part that would appear brighter on MR will appear darker on ultrasound.

The second part for understanding medical image processing is to understand the physics part of it. As we mentioned above, different parts of the body have different ways in which they are viewed, and it is where we need to understand tissue energy interactions together. And from there, image formation and statistics of image formation will get vital because we will see a lot of noise and uncertainty. For example, ultrasound is a specular modality that has a lot of jitters and noise around intensities. On the other hand, MRI will not have that much problem, but it suffers from lower resolution than ultrasound, and when we go into detail understanding of each of this physics and how operating conditions of an instrument affects the total resolution of image formation.

After medical imaging and physics, we would be moving on to image processing and graphics in the third step. Image processing is an essential part of medical image analysis because it contains Denoising, feature segmentation, image segmentation, and feature representation. After that, we got onto a very critical factor called visualization. Also, knowledge of graphics is fundamental.

Last but not least is machine learning which has grown in recent years. Machine learning

includes prediction models and a group of classifiers. We can use various methods to solve a particular problem, such as example-based learning methods or complex reasoning. So to conclude these four main steps of medical image processing, we bring them as Figure 1.3.

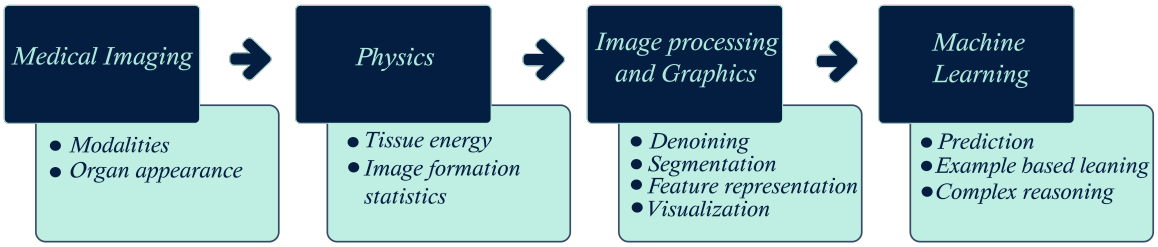


Figure 1.3: Medical image analysis roadmap

1.4 research and business areas

Let us look into the main areas of research and business that medical image analysis is concerned. The first one is the range of image modalities which are macro and micro imaging. Macro imaging is the method that the whole body gets imaged together, and obviously, the resolution is much lower, but the penetration depth is much higher. On the other hand, micro imaging has resolution at the microns' level and is used in some modalities like optical coherence, tomography, and histopathology for digital pathology. But, there is a tradeoff for scaling, and it is mesoscopic range imaging.

The second main area is image registration, a classical area for medical image analysis, and it says a patient goes for a CT scan. After six months, he or she does this again but at a different location and different CT machine. The resolutions are different because the patient might have a body change, such as losing some weight. The doctor wants actually to relate to what happened in the last six months.

Another exciting area is organ localization which is where we need to find out the organ itself. This area uses algorithms to make it much easier for clinicians than wasting a lot of time. Also, As you know, we imaging in 3D, but you see the output on your computer screen or x-ray films which present only 2D pictures. Organ localization tries to use algorithms in 3D and segment them. After organ localization has been over, you can segment out and get the volume information and surface information.

In recent years, the research on visualization using augmented reality has been expanded. This area is related to present how different kinds of fractures can be there on a bone and then try to find out what the bone would look like without any fractures.

Another interesting field is virtual anatomy, and it comes after we have a complete MR or CT scan of a patient body. The whole anatomy, such as every organ and every tissue or bone, is digitized and segmented using virtual anatomy. In other words, the body is equivalent to a computer-aided design (CAD) model. For example, if a doctor wants to find out internal injuries, specific legions, or diseases within the patient's body, they do not need to scan every x-ray.

A very critical application in this area is digital angiography which is also called digital subtraction angiography. It is a technique to find out where is a deposition of blood or regularity inflow of blood. Despeakling is related to speckle imaging modalities like optical coherence or ultrasonic tomography, where many speckles lead to tediousness. This situation cannot just be Denoising because it would get rid of edges and salient behaviors.

3D optical microscopy is a 3d model of a neuron image under fluorescence for different layers, segmented and resynthesized by alignment. We can look at the complete 3D model and view how neurons look and how the processes are going over there. This area has a significant role in a field of medicine called precision medicine.

Digital pathology is the next critical area. With the help of pathology, the pathologist doesn't need to be at the center where the slide comes down. So say there is a collection center that is far from the actual pathologist is located. Transporting a physical slide takes more time and money, but digital pathology solved this problem.

The last area in this section is computational imaging. Computational imaging is about synthesizing virtual equivalents of different organ types of different pathology types by looking at simple images and simple signals from image modalities. You can see all these areas in Figure 1.4 .

1.5 last 35 years of medical image analysis

This short review is based on a survey from Duncan and Ayache in January 2000. This era up to 1984 was basically when pattern recognition and analysis were carried on 2D images. Between 1985 and 1991, knowledge-based approaches started, and in that period, we had rule-based analysis and rule-based influencing. From 1992 to 1998, a lot of 3D imaging started into play, and contribution was about storing a more significant amount of data. Silicon fab processes improved process time. For example, nowadays, the whole human body CT could be processed in a couple of minutes, but it took about 24 hours in the past.

1999 to 2010 was the era of Shallow Reasoning with machine learning, and that is when again you have a lot of computing power. The final period, 2010 until now, is real booming in the field of how neural networks and deep learning was coming up and that is when complex reasoning started. A lot of approaches was developed in this period.

1.6 challenges

One of the most important challenges for medical image processing is automated detection and classification of breast cancer, and it also called as CAMELYON2017. The next challenge is about prostate lesions segmentation. Then you have the very famous sclerosis from brain MR. The next interesting problem called as M2CAI and this is where we look into interventional videos about surgeries. In other words, M2CAI is modeling and monitoring of computer assisted intervention. The final challenge is ultrasound nerve segmentation.

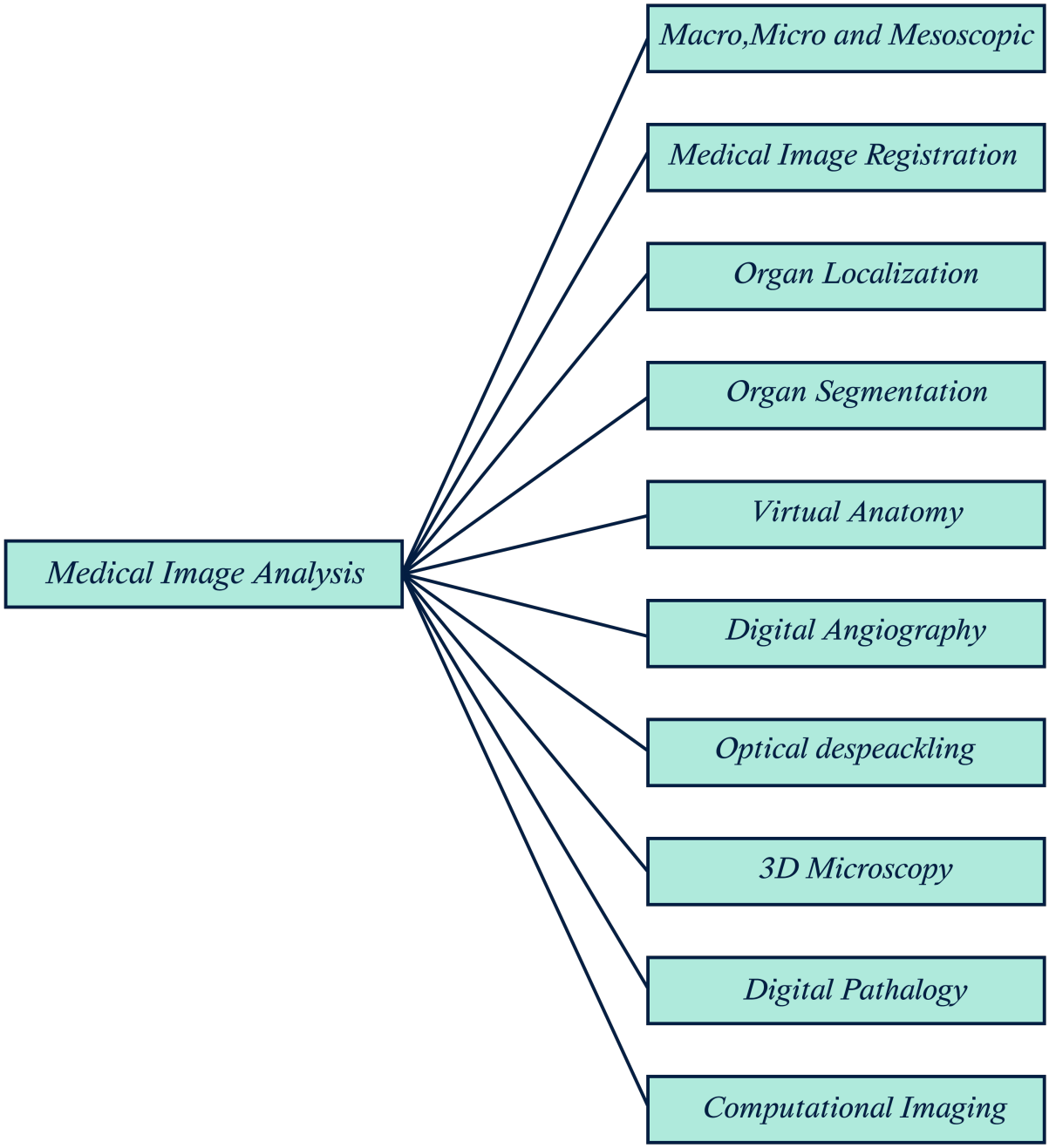


Figure 1.4: Medical image analysis key research(business) areas

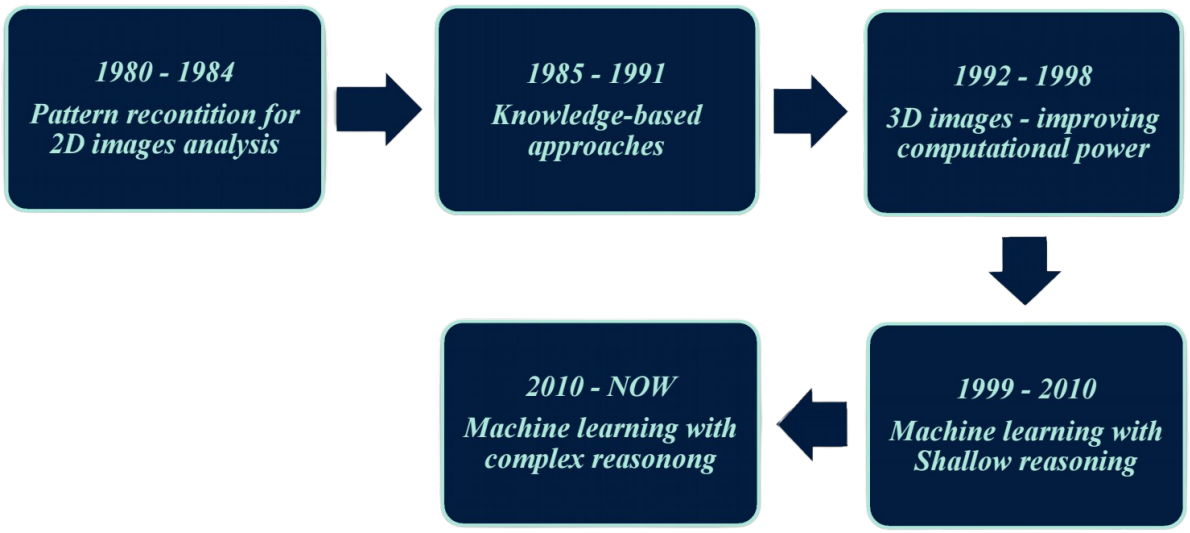


Figure 1.5: Medical image analysis key research(business) areas

Chapter 2

Medical imaging techniques

2.1 X-Ray & CT Imaging

X-ray is one of the early modules introduced by Wilhelm Rontgen. With this modality, we could look within the body without having to cut the body. Another modality that goes along with x-ray is computed tomography (CT). Computed tomography allows you to look into the 3D projection of every single part of the body and create visual anatomy of the body.

The first part is on x-ray generations and how it works. We have something called an **x-ray tube**, and this is the source of generated x-ray. It is not an optical photon, i.e., unlike LED lights or tungsten filament lamps where you pass electricity, and it is directly generated; in other words, it is an indirect process.

In this tube, there is a cathode that is heated up, and then it has many electrons around it. On the other side, an anode is a positively charged surface, and it would be a striking anode. In the next step, x-rays will be emitting. Figure 2.1 illustrates this process.

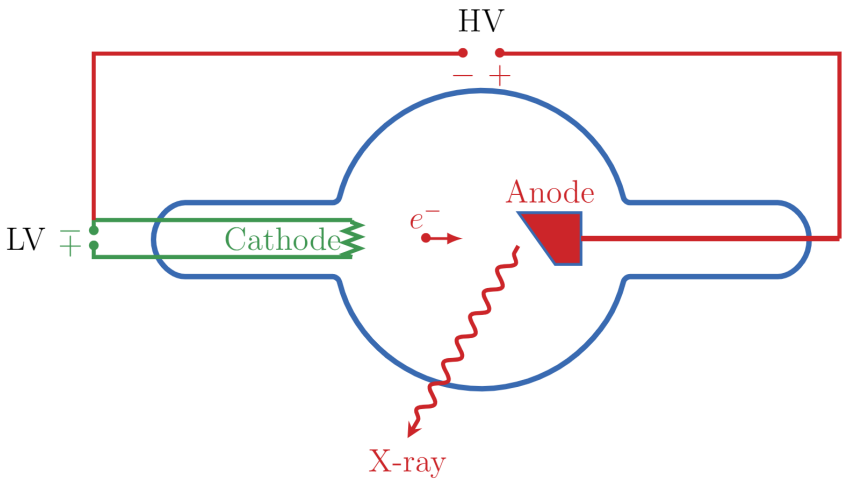


Figure 2.1: X-ray generation process

For more clarification, we focus on reflected x-rays. As you can see in figure 2.2, there is a 45 degrees inclination angle. For example, if you have an object at 45 degrees, you will

get a reflection, and the angle between this reflection and the incidence ray is 90 degrees. With this method, you put an object and get a magnified view of it on a film.

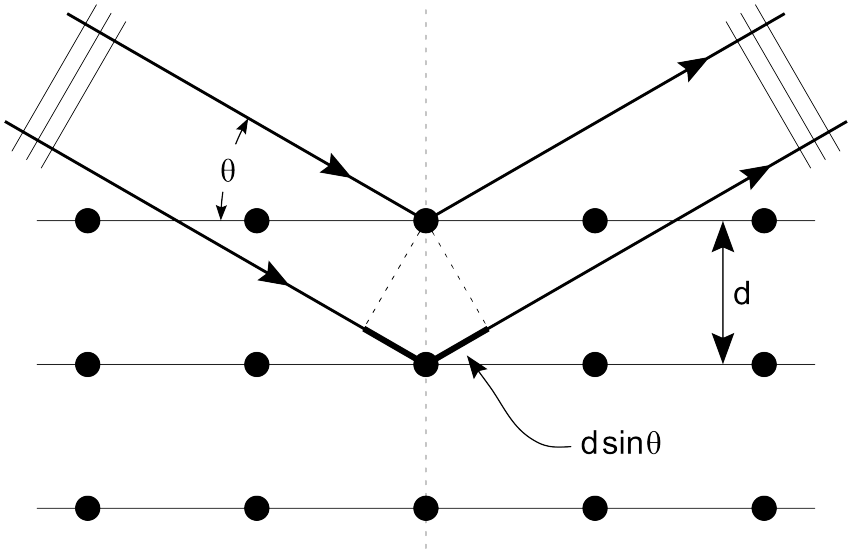


Figure 2.2: The process of generation x-ray in more details

The next modality is Computed Tomography, and we want to find how to acquire the images using this method. In this method, several detectors have been used, not a single one. The best way is to set an array of sensors, and each has a different line equation, to being solved out when you get the full projection. After that, you need to rotate this pair of the detector array. It keeps on rotating, and you will get multiple projections, so you can backtrack and reconstruct your whole object. Also, it is called CT reconstruction.

In the final step, we want to find out what is a CT machine looks like. The human body is divided into three planes(x, y, and z-axis); it is three orthogonal planes in medical images. The first plane is called Coronal, which crosses from your left-hand side to the right-hand side and moves the coronal plane from front to back and vice versa. The next one is the Sagittal plane that divides your body into left and right parts, and it can moves from left to right or reverse. Finally, the other plane is called as Axial plane. This plane divides your body into top and lower parts. Figure 2.3 shows a CT machine structure.

2.2 Magnetic Resonance Imaging (MRI)

The human body is made of multiple small magnets just because of the amount of water drank. This is the base of a concept known as nuclear magnetic resonance. Every single atom has a balanced ratio which is called the gyromagnetic ratio. The first part of MRI is the Spin-Echo concept. If we want to align something from a lower energy state to a higher energy state, we must provide extra energies. When we reduce the energy level, all the protons went down to a higher energy configuration. In other words, they are opposite to magnetic fields.

After some time, due to thermal motion, protons will lose some energy to other atoms. Once they keep on losing energy, they are eventually going to fall to a particular plane. Then some of them will go opposite each other, which means that the net magnetic field

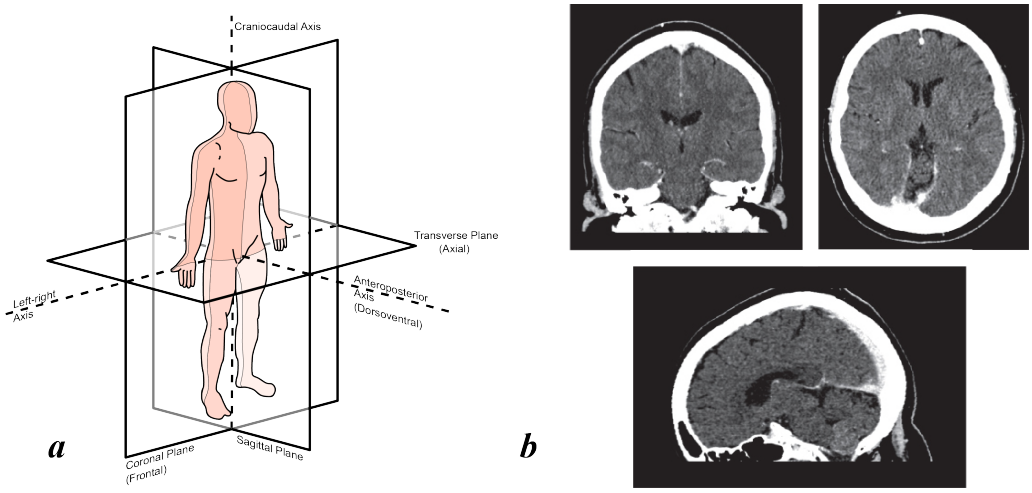


Figure 2.3: (a) Diagram of the axial, coronal and sagittal planes. (b) Corresponding CT images of a normal brain.

is 0. There are some different magnetic field directions. One of them is longitudinal; another one is traverse which is along the opposite direction. Figure 2.4 illustrates these directions.

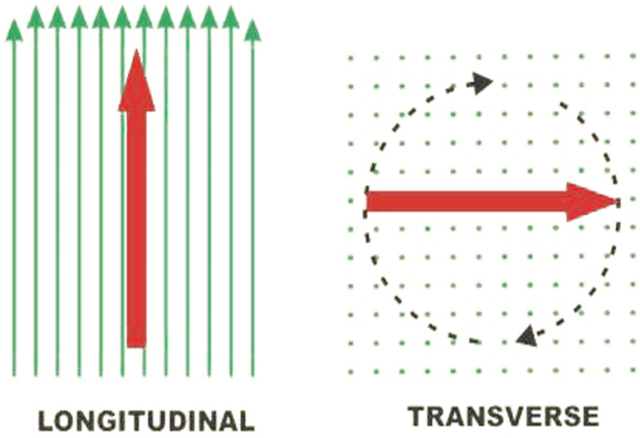


Figure 2.4: Different magnetic directions

Let's look at different materials and their T1 and T2 relaxation time and as it shown in table 2.1 the highest values are for Cerebrospinal fluids. Another interesting fact is if T1 is increasing, then T2 also is increasing. T1 is higher than fat, whereas T2 is lower than fat. Also, for muscles, T1 is higher than liver although T2 is higher than Liver but is it lower than fat. The first modality is called T1 weighed image and the second one is known as T2 weighed image.

In table 2.1, those regions with a much higher T1 relaxation time have much higher energy, respectively. If we have an MRI of a brain (Figure 2.5), it is made out of many fats. In the T1 modality, the fat regions in the brain are shown in white color, while in the T2 modality, it is a gray matter. Also, the correspondences between T1 and T2, white matter, and gray matter appear opposite to each other, and it is because of the relation between these two modalities.

The primary role of engineering comes into play when we want to apply a trick in MRI.

Table 2.1: Spin relaxation time

Tissue	**T1(ms)**	**T2(ms)**
Fat	260	80
Liver	550	40
Muscle	870	45
White matter	780	90
Gray matter	900	100
Cerebrospinal fluid	2400	160

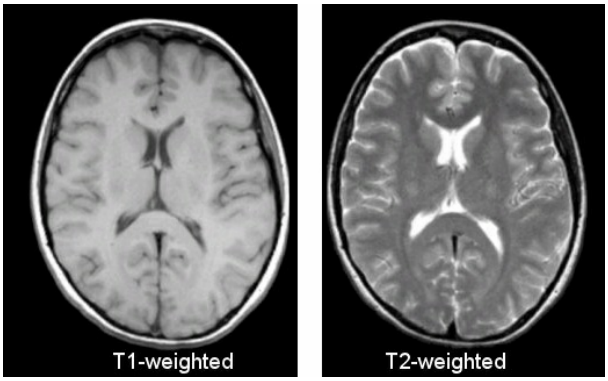


Figure 2.5: T1 and T2 modalities

Instead of exposing the whole body to a constant magnetic field, we create a gradient. For example, somewhere around the head, the magnetic field is about 1.5 tesla, and around the legs, it will become about 0.5 tesla. After applying this technique, we can divide the body into different slices, and each piece will have a different frequency.

Table 2.2 is a quick reference for different matters in the body in T1 and T2 modalities. For example, T1 and T2 weighted images for bones or bonny-like structures (including calcifications) have an external signal. It is contrary to CT images; i.e., in CT images, bones were shown as brighter structures.

Table 2.2: Comparison is made on the signal of muscles

Tissue	**T1 weighted**	**T2 weighted**
Bone cortex, calcification	Very low signal	Very low signal
Bone Marrow	High signal	High signal
Cartilage	Iso signal	Slightly low signal
Joint effusion	Iso signal	High signal
Acute hemorrhage	Low to iso signal	Low to iso signal
Subacute hemorrhage	High signal	High signal
Hemosiderin	Very low signal	Very low signal
Fat	High signal	High signal

2.3 Ultrasound Imaging

Ultrasound is an exciting imaging modality. The prefix ultra means it is above our standard audible limit. Human ears can hear with frequencies from 20 hertz to 20-kilo hertz. In this imaging method, you have a transducer placed in exact contact on the surface; it shouldn't have an air gap. The transducer emits a tiny bit pulse known as acoustic pulse. Actually, in ultrasound imaging, we probe waves in a fixed period and then draw the contour of all the points. It is like we throw a stone in water, and then we see circular waves around it. Figure 2.6 illustrates the ultrasound mechanism.

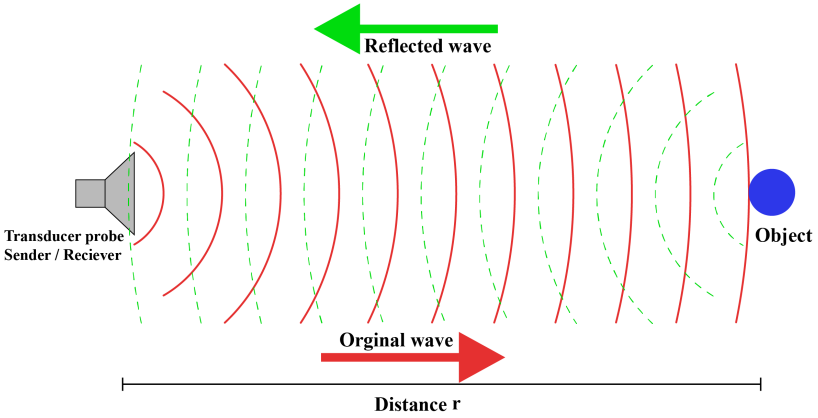


Figure 2.6: how ultrasound works

Interference between waves is the concept that helps to identify where the object is and what shape it is. In general, there is no control about where that object is located because, in 2d space, there is just 1d way of figuring out where the obstruction is located. The solution to this problem is to use more transducers. For example, consider we use two transducers, and they fire at the same time. The phase of the two waves is the same. Figure 2.7 show this scenario. Although they come from two different sources, there is constructive interference between these two waves. So these are some points that have the highest magnitude.

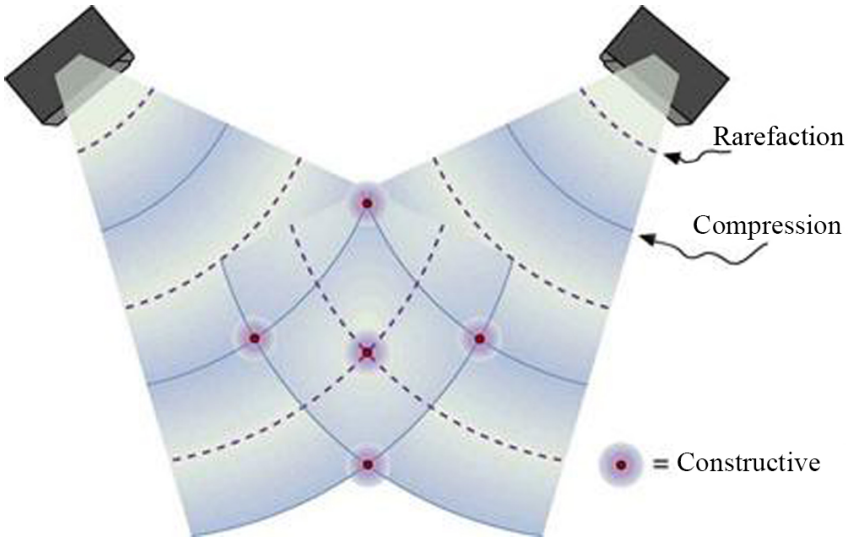


Figure 2.7: how ultrasound works

2.4 Optical Microscopy and Molecular Imaging

The last module which we cover is Optical Microscopy and Molecular imaging. The previous modules such as CT, MR, and Ultrasound are the majority of macro imaging modalities. In other words, in these modules, you look into larger parts of the body. In microscopy, you will look into a particular part of the body. This modality aims to study one single cell or a cluster of cells that was not possible with any other modalities.

A microscope usually does optical Microscopy and Molecular imaging. One of the first types of microscopy used in medical applications was the Single Photon Emission Computed Tomography (SPECT) system which was very useful to look into gamma photon generation. In this system, radio isotropy was put down along with glucose and give to the patient. If a particular organ has more glucose consumption, it means other organs are in there, and we would see more radionuclide coming down from there.

The resolution of SPECT was much lower and did not give a very high structural resolution. So, generally, structural scan systems are used either a structural CT or structural MR along with a SPECT. These multiple images are registered together to diagnose by knowing the structure and the function.

The other type of microscopy is Positron Emission Tomography, and when there is a positron emitted, you would see a pair of gamma photons created; this means that we always need to have a pair of detectors that are 180 degrees apart and if these detectors rotate we will be sensing a pair of incidences. This method helps to reduce noise, and the noises on each detector are independent. Figure 2.8 shows this method.

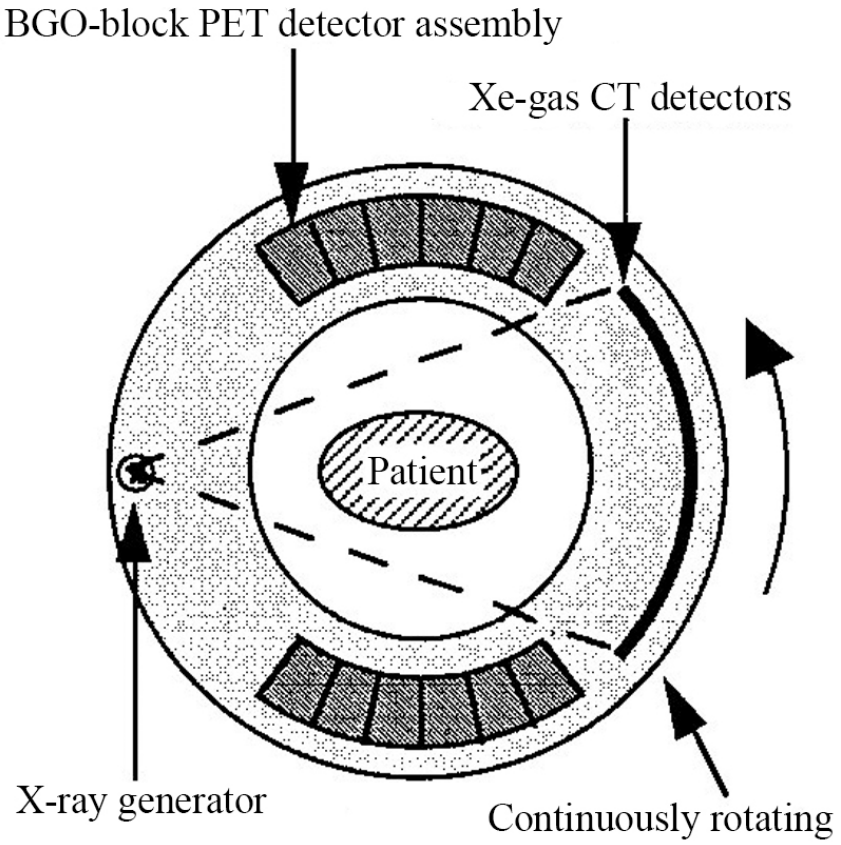


Figure 2.8: Optical PET imaging instrument

Chapter 3

Image processing concepts & techniques

3.1 Texture in medical images

The texture is a variation of data at scales smaller than the scales of interest. For example, if we take an MR image from the human brain, any variation in the gray values of the imaged brain may be thought of as texture. Sometimes variation due to noise may be considered as texture. The texture is an intrinsic property of an image, and it is a valuable cue to recognize an object in the image.

The most important characteristic of texture is that it is scale-dependent. In other words, different types of textures are visible at different scales. For example, if we look at a section of the brain through a microscope, we will see a different structure of the tissue than sulci monitored. So, to use texture for identifying different types of tissue, we should be able to measure image texture in a repeatable and reliable way. In other words, it is necessary to characterize texture in an objective way, independent of human perception and visual abilities.

Structural or statistical methods may characterize texture. A structural method identifies a texture primitive pattern, and this pattern is repeated to create the viewed appearance. This approach is also known as grammars for texture description. Usually, structure methods are less common in biomedical applications. The other types of methods are statistical approaches, and they are more attractive. Figure 3.1 shows how natural the transition from deterministic methods to statistical methods is. In this figure, we may easily infer from the first case in the left that a possible texture would be one surrounded by four 0s according to the other neighboring positions. But in the second case, we can see, in 7 out of 12 points, the arrangement of four 0s indicates a 1 in the middle, while in the remaining 5 cases, it indicates a 2. So the missing value has a probability of 58% to be one and a chance of 42% to be 2. This method is known as Markov Random Field (MRF) modeling.

In MRF, we model the probability with which the neighbors' values are used to characterize the texture. As the statistics have to be performed on neighbor value combinations, and as pixels, particularly in medical images, take up many gray values, one needs an

1	0	1	0	1	0	1
0	1	0	1	0	1	0
1	0	1	0		0	1
0	1	0	1	0	1	0
1	0	1	0	1	0	1
0	1	0	1	0	1	0
1	0	1	0	1	0	1

1	0	1	0	1	0	1
0	1	0	1	0	1	0
2	0	1	0		0	1
0	2	0	2	0	1	0
2	0	1	0	2	0	1
0	1	0	2	0	2	0
2	0	1	0	1	0	1

Figure 3.1: Optical PET imaging instrument

extensive sample of the texture to work out these parameters.

One of the issues in MRF texture approaches is the definition of the size of the neighborhood. In the previous example, we considered the smallest possible neighborhood. However, the neighborhood may be much larger to include pixels further away from the focal pixel. In the extreme case, all other pixels in the image are thought of as neighbors. In other words, the value of a pixel is directly influenced by all of the other pixels, and all scales of influence are essential.

In MRF texture descriptors, each pixel's value depends on the values of the neighbors, which have values that depend on their neighbors, and so on. The fractal models model this global dependence **explicitly**, while MRF models this global dependence **implicitly**. So, the MRF formulation leads to a formulation that models the **joint** probability density function of a particular combination of values of the whole image. Indeed, under certain conditions, an MRF modeling is equivalent to a Gibbs modeling.

Gibbs model is called clique potentials. A clique is a set of grid positions that are neighbors of each other according to the Markov model, therefore, directly influence each other's value according to this model. In figure 10, the only cliques one can have are pairs of pixels next to each other, either vertically or horizontally. In Gibbs formalism, some parameters multiply the difference of the pairs of values of neighboring pixels to express their dependence. These parameters characterize the texture.

Gibbs distributions are not very useful in texture descriptions. However, they are beneficial in other problems of medical image processing, such as image restoration or image matching. What is important to us is how to move from local description (neighborhood-dependent modeling) to a global (joint probability).

Another noticeable transform before proceeding to quantify texture is to transform the image to the spatial frequency domain. The Discrete Fourier Transform (DFT) identifies the content of an image. DFT is complex but usually adopts its magnitude, which is known as the power spectrum. The power spectrum helps us to identify the periodicities are present in the texture.

If we want to analyze the frequency content of the image in terms of different frequency ranges (bands), it is necessary to isolate bands in the frequency domain. Band isolation can be done with the help of some window centered at a particular frequency, and it will separate the required band. The most commonly used window is the **Gaussian** because it does not create artifacts in the image. Figure 3.2 illustrates the process of band isolation.

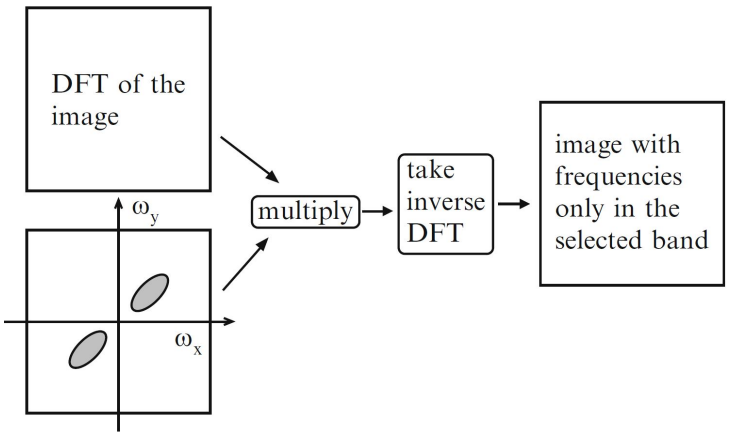


Figure 3.2: Optical PET imaging instrument

After extracting selected bands, we can compute features from them to characterize texture. The most helpful feature is the sum of the squares of the values of the output image. This feature is to find its energy in the particular frequency band.

3.2 Segmentation

The segmentation purpose is to split the image into parts, which are specific applications. The quality of the image segmentation scheme is estimated according to factors like Similarity Index, Dice Score, Sensitivity or Overlap fraction, Extra Fraction, Specificity, and Accuracy. For example, the Similarity Index (SI) tells about the mutual value among input MRI brain images and segmented output images. Accuracy is cast off to estimate the accomplishment of the tumor detection and segmentation system.

In medical imaging, segmentation often refers to the explanation of specific structures. So, it includes details of classification methods. Segmentation strategies in medical imaging combine data knowledge with domain knowledge to arrive at the result. Data knowledge refers to assumptions about continuity, homogeneity, and local smoothness of image features within segments. Domain knowledge represents information about the objects to be delineated.

Quantitative analysis of medical images requires objects or object features in the image to be identified. First, the image is segmented into regions that are the possible candidates of objects. This is followed by assigning meaning to these regions. For analyzing a digital photograph, **the segmentation** task would group pixels to areas that may belong to parts of objects based on the attributes of these regions. Hence, segmentation of images is similar to creating phonemes in speech or detecting syllables in written text. It means that the segmentation of images creates basic semantic entities from images.

However, it isn't easy to apply domain knowledge about objects in an image to segmentation. The purpose of segmentation is to create semantic entities in the first place. After segmentation, every pixel is assigned to exactly one segment since every location in an image carries just one meeting. The problem is appearances of objects in some images may be very different within and between objects classes. For example, Figure 3.3 is a set

of pictures contain a landscape with a house in the foreground, some trees and mountains in the background.

Figure 3.3: Finding a general solution (find houses) for a segmentation task

For the segmentation of medical images, the situation is even worse, as a medical image represents the measurement of a diagnostically relevant entity measured in the same way everywhere in the body. For instance, consider a CT where X-ray attention is calculated on the normalized Hounsfield scale. Attention should not vary with location but only with density and atomic number.

There are several ways to deal with missing information without missing the assumption that a low-level segmentation criterion is valid everywhere in the image.

Foreground segmentation focuses on a single object in the image. Segmentation criteria create an excellent partitioning of foreground objects, whereas the quality of partitioning the background is irrelevant. The strategy requires some model knowledge to be applied after segmentation for separating foreground segments from the background. A simple way to introduce the model would be let the user point out foreground segments. Figure 3.4 is an example of segmenting liver in a CT image.

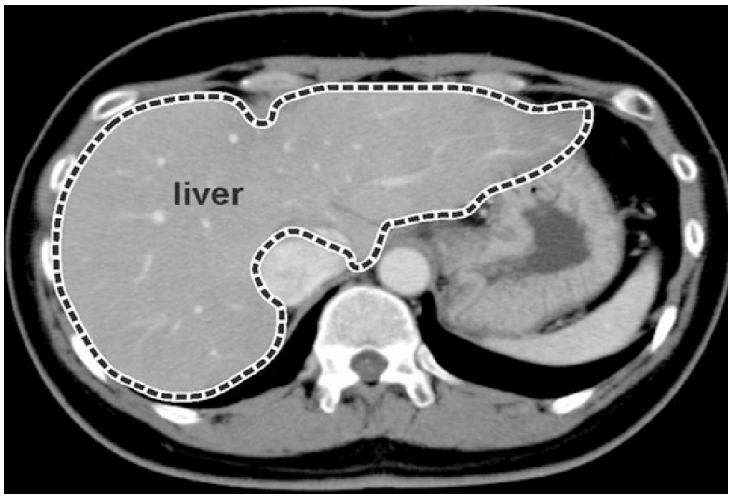


Figure 3.4: Segmentation of liver from all other tissues

Hierarchical segmentation (Figure 3.5) applies a multi-resolution concept for gradual refinement. A first segmentation creates segments that are smaller than the smallest object. It is assumed that a common criterion can be found at this scale. The result is sometimes called over-segment. Some of these segments are merged into larger segments at the next level according to domain knowledge about object appearance. Successful application of this strategy requires that common criteria define meaningful segments at a single but unknown scale. This scale is found by analyzing the levels of the segmentation hierarchy.

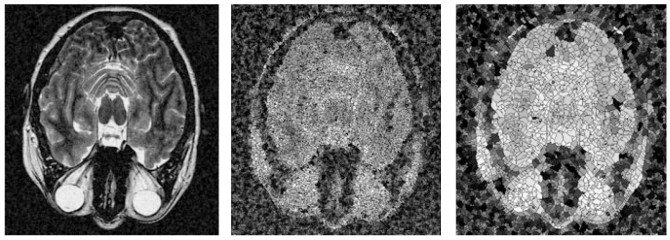


Figure 3.5: hierarchical segmentation different stages

Multilayer segmentation is another multi-resolution technique. It is assumed that a common segmentation criterion exists but that its scale may vary throughout the image (like the structured texture of an object in which the scale varies with the distance of the object to the camera). Segmentation is carried out at different scales producing layers of segments. Multilayer segmentation is more general than the previous strategy. The criterion scale often varies because an appropriate scale for every segment has to be established independently from other segments. Figure 3.6 illustrates steps of multilayer segmentation.

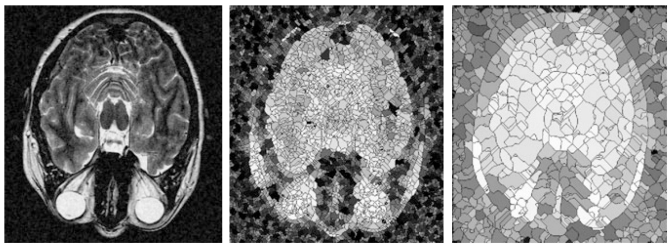


Figure 3.6: segmentation at different levels of resolution are evaluated in parallel

The search for a specific structure by segmentation causes foreground segmentation to be more frequent in medical image analysis than in other image analysis tasks. Since it may involve a detection task, a model-driven approach then discriminates the structure from the background. Model knowledge may be integrated into the algorithm or supplemented interactively. Popular segmentation techniques, such as the various region growing, active contours and surfaces, and active shape models, use a model-driven approach.

3.3 Thresholding

Thresholding is the simplest method of image segmentation. Various thresholding techniques are:

1. **Global Thresholding**: In this method, we use a bimodal image. A bimodal image is an image with 2 peaks of intensities in the intensity distribution plot. One for the object and one for the background. Then we deduce the threshold value for the entire image and use that global threshold for the whole image. A disadvantage of this type of threshold is that it performs really poorly during poor illumination in the image(Figure 3.7).

2. **Multimodal Thresholding**: A general approach to thresholding is based on assumption that images are multimodal, that is, different objects of interest relate

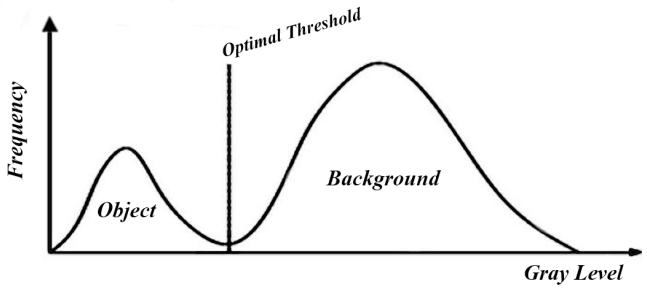


Figure 3.7: Global Thresholding

to distinct peaks (or modes) of the 1D signal histogram. The thresholds have to optimally separate these peaks in spite of typical overlaps between the signal ranges corresponding to individual peaks(Figure 3.8).

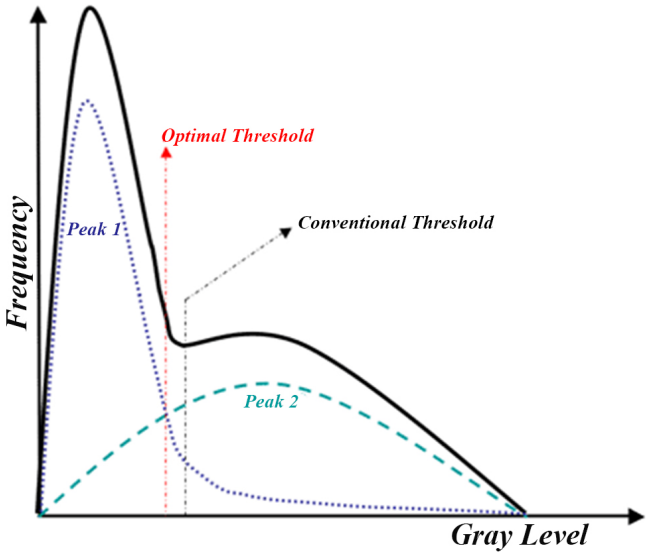


Figure 3.8: Multimodal Thresholding

3. **Adaptive Thresholding**: The image is divided into various subregions, and all these regions are segmented using the threshold value calculated for all these regions. Then these subregions are combined to image the complete segmented image. This helps in reducing the effect of illumination to a certain extent.

3.4 Artificial Neural Network

ANN stands for Artificial Neural Network. Just like humans have BNN i.e. the Biological Neural Network, the researchers tried to replicate the working of the human brain and came up with the great invention of an Artificial Neural Network. ANNs have been developed for a wide range of applications such as function approximation, feature extraction, optimization, and classification. In particular, they have been developed for image enhancement, segmentation, registration, feature extraction, and object recognition.

There are three types of learning in ANN. They are supervised learning, unsupervised learning and reinforced learning. **Supervised learning** is based on error between output

and input data. A teacher or supervisor is offered during learning process and offered projected output. In **Unsupervised learning** no teacher is presented to teach to the network. Unsupervised learning is very complex and difficult to implement. The system learns of its own by discovering and adapting to the structural features in the input sample of data. **Reinforced learning** is output based learning. Although a teacher existed during the procedure but, the teacher is not going to present the actual output. The teacher only indicated whether the output is correct or incorrect.

3.5 Clustering

Clustering is a type of unsupervised machine learning algorithm. It is highly used for the segmentation of images. One of the most dominant clustering-based algorithms used for segmentation is KMeans Clustering. This type of clustering can be used to make segments in a colored image. Three main clustering methods are as follows:

1. **Traditional Clustering**

2. **Hierarchical Clustering**: Also called Hierarchical cluster analysis or **HCA** is an unsupervised clustering algorithm. The algorithm groups similar objects into groups called clusters. The endpoint is a set of clusters that have ordering from top to bottom.

3. **Partitional Clustering**: These algorithms minimize a given clustering criterion by iteratively relocating data points between clusters until a (locally) optimal partition is attained. Famous **K-means clustering** comes under partitioning method.

3.6 Edge Detection

Edge-based segmentation relies on edges found in an image using various edge detection operators. These edges mark image locations of discontinuity in gray levels, color, texture, etc. When we move from one region to another, the gray level may change. So if we can find that discontinuity, we can find that edge. This process detects outlines of an object and boundaries between objects and the background in the image. An edge-detection filter can also be used to improve the appearance of blurred image. A variety of edge detection operators are available but the resulting image is an intermediate segmentation result and should not be confused with the final segmented image. We have to perform further processing on the image to the segment it. Additional steps include combining edges segments obtained into one segment in order to reduce the number of segments.

Edges are usually associated with “Magnitude” and “Direction”. Some edge detectors give both directions and magnitude. We can use various edge detectors like Sobel edge operator, canny edge detector, Kirsch edge operator, Prewitt edge operator, Robert’s edge operator, etc.

3.7 Region-Based

A region can be classified as a group of connected pixels exhibiting similar properties. The similarity between pixels can be in terms of intensity, color, etc. In this type of segmentation, some predefined rules are present which have to be obeyed by a pixel in order to be classified into similar pixel regions. Region-based segmentation methods are preferred over edge-based segmentation methods in case of a noisy image. Region-Based techniques are further classified into 3 types based on the approaches they follow.

1. **Region Growing**: We start with some pixel as the seed pixel and then check the adjacent pixels. If the adjacent pixels abide by the predefined rules, then that pixel is added to the region of the seed pixel and the following process continues till there is no similarity left.

2. **Region Splitting**: The whole image is first taken as a single region. If the region does not follow the predefined rules, then it is further divided into multiple regions (usually 4 quadrants) and then the predefined rules are carried out on those regions in order to decide whether to further subdivide or to classify that as a region. This process continues till there is no further division of regions required

3. **Splitting & Merging**: In these methods, after splitting regions we consider every pixel as an individual region. We select a region as the seed region to check if adjacent regions are similar based on predefined rules. If they are similar, we merge them into a single region and move ahead in order to build the segmented regions of the whole image.

3.8 Graph-Based

graph-based methods present different graph representations, where the nodes may be pixels, pixel vertices, regions, or user-drawn markers. They also differ in the graph algorithm used to solve the problem: graph matching, random walker, the min-cut/max-flow algorithm, Dijkstra's algorithm, Kruskal's or Prim's algorithm, etc. Figure 3.9 illustrates how graph-based segmentation algorithms work.

3.9 Derformable Models

Deformable models have been extensively studied and widely used in medical image segmentation, with promising results. Deformable models are curves or surfaces defined within an image domain that can move under the influence of internal forces, which are defined within the curve or surface itself, and external forces, which are computed from the image data. The internal forces are designed to keep the model smooth during deformation. The external forces are defined to move the model toward an object boundary or other desired features within an image.

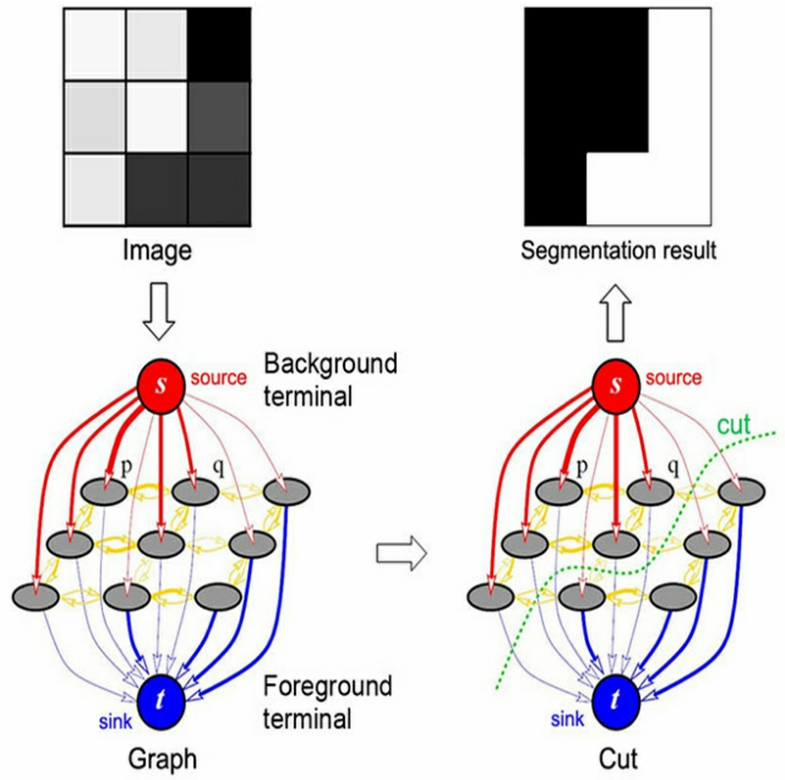


Figure 3.9: Graph-based segmentation

3.10 Watershed

It is often used when we are dealing with one of the most difficult operations in image processing – separating similar objects in the image that are touching each other. To understand the "philosophy" behind the watershed algorithm we need to think of a grayscale image as a topographic surface. In such an image high-intensity pixel values represent peaks (white areas), whereas low-intensity values represent valleys – local minima (black areas). we start filling every isolated valley with water. the rising water from different valleys will start to merge. To avoid that, we need to build barriers in the locations where the water would merge. These barriers we call watershed lines and they are used to determine segment boundaries.

We described some of the most common strategies for segmenting medical images. Figure 3.10 shows various approaches for segmentation.

3.11 Denoising

Image denoising is required to use in digital image processing. From the researchers' point of view, image denoising is still challenging task in medical images. Image denoising stays a normal process in digital image processing. It remains a pre-processing period to eradicate specific unidentified such as Additive White Gaussian Noise (AWGN) from an image to get a clean image used in favor of image processing, such as image segmentation. A representative instance of the image processor is image denoising prototypical. This

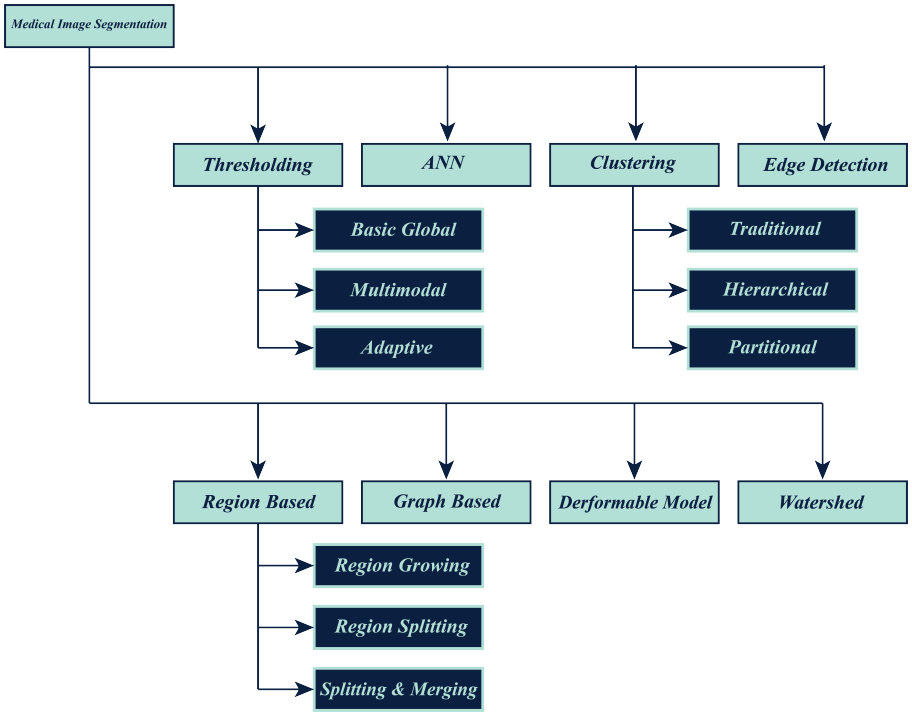


Figure 3.10: segmentation techniques

method incorporates a disintegrated prototypical giving to some prior information of a corrupted image. To reestablish the distorted image in favor of obtaining a promising result.

In this instance, n_0 tells about noisy medical image and w is an original, nevertheless usually known or unknown image i.e.,

$$n_0 = Dw + n_1 \tag{3.1}$$

.

In which D is a linear or non-linear operator representing the distortion and n_1 is additive noise.

The performance of the image denoising method is measured by using the essential factors such as Mean or Average pixel intensity, Standard Deviation, Mean Square Error, Root Mean Square Error, Mean Absolute Error, Peak Signal to Noise Ratio, Structural Similarity, Universal Image Quality Index, and Entropy. For example, the mean parameter determines the dissimilarity of the image by adding all the pixels and dividing the total pixels of the image. The average gradient gives the sharpness and clarity of the image.

There are other algorithms and general technologies designed intended for image denoising. These are listed below:

3.12 Statistical modeling-based image denoising

In 2018, Wang presented the minimax-concave TV (MCTV) to reconstruct the MR brain image. MCTV is non-convex, mainly to represent the nonconvexity parameter correctly and at the same time maintain a convexity for every iteration step.

In 2016, Park suggested a method of learning-based denoising. This method significantly reduces PI noise, and the start-up time is much faster. It creates high-quality images that do not have degrading information to increase speed. In the same year, Zhang demonstrated an MRI denoising framework that structurally correlates between users to evaluate the models efficiently to reduce noise in the signal.

3.13 Derivative-based image denoising

In 2017, Magudeeswaran developed a novel level-set technique originates from the active contour prototypical. This technique diminishes the costly initialization of a level set approach further efficiently. Primarily, images are pre-processed with the Contrast Limited Fuzzy Adaptive Histogram Equalization (CLFAHE) improvement technique to enhance the dissimilarity of the image.

In 2016, Sedaaghi solved PM flux analysis problems. In this method, the image remains bisected into three slices, depending on gradient level: the area where the slope is inferior to the smoothness of the area, where the slope is between the smoothness and the flux point, and the slope area is higher than the flux point. Also, in this year, Yang proposed an image denoising algorithm using PDE and GCV. In this method, the blocking effect is reduced by tetrolet transformation and is considered to be good.

To eradicate the staircase consequence aimed at TV filter and evading boundaries distorting in support of fourth-order PDE filter, Tan suggested the anisotropic diffusion in favor of image denoising. In this method, if the pixels are part of the flat area, a fourth-order filter is accepted that can remove the staircase artifacts. Figure 3.11 illustrates these denoising methods in an abstract view.

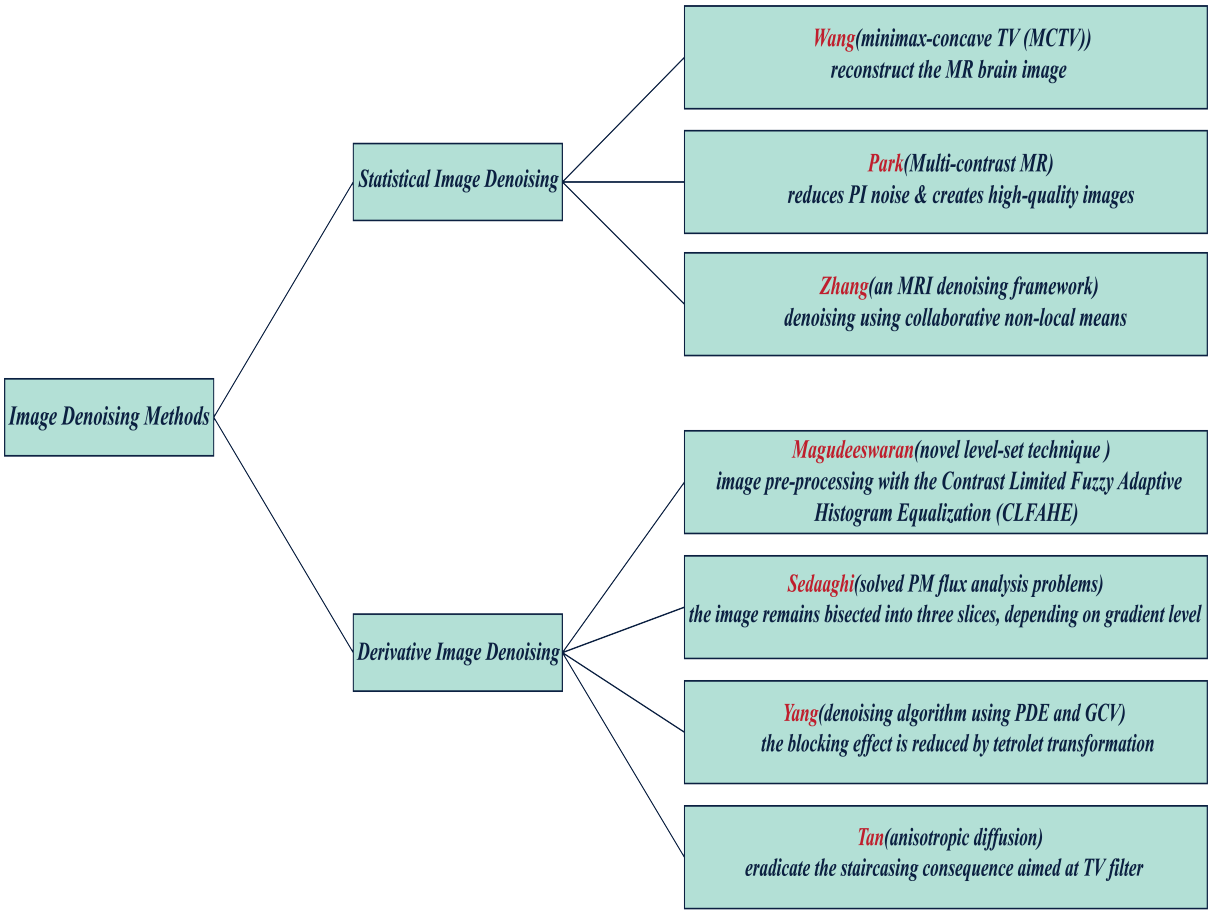


Figure 3.11: Image Denoising Methods

Chapter 4

Hands on Python & OpenCV

Magnetic Resonance Imaging (MRI) helps the healthcare field detect very minor abnormal growth in any part of the human being. While deep neural networks (NNs) and machine learning techniques have good achievement in 2D image segmentation, but it's a challenging task for NNs to segment critical organs from 3D medical MR images. Segmentation relating to tumor detection includes several processing techniques categorized into Pre-Processing, Segmentation, Optimization, and Feature Extraction.

4.1 Introduction to OpenCV

4.1.1 Installing OpenCV for Python

Windows

In order to get OpenCV-Python up and running, we need to perform the following steps:

1. **Install Python**:Make sure you have Python 3.8.x installed on your machine. If you don't have it, you can install it from https://www.python.org/downloads/windows/

2. **Install NumPy**: NumPy is a great package to do numerical computing in Python. It is very powerful and has a wide variety of functions. OpenCV-Python plays nicely with NumPy, and we will be using this package a lot, during the course of this book. You can install the latest version from http://sourceforge.net/projects/numpy/files/NumPy/. We need to install all these packages in their default locations. Once we install Python and NumPy, we need to ensure that they're working fine. Open up the Python shell and type the following:

   ```
   >>> import numpy as np
   ```

   If the installation has gone well, this shouldn't throw any error

3. **Install Matplotlib**: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. We use pip package to install this library. Type the folloing script in your command prompt.

   pip install matplotlib

4. **Install OpenCV**:download the latest OpenCV version from http://opencv.org/downloads.html. Once you finish downloading it, double-click to install it. You're all set! Let's make sure that OpenCV is working. Open up the Python shell and type the following:

   >>> import cv2

Linux (for Ubuntu)

Let's go ahead and build OpenCV with Python support. The commands below will install all packages necessary to run OpenCV.

$ sudo apt update

$ sudo apt install python3-opencv

To verify the installation, import the cv2 module and print the OpenCV version:

>>> import cv2

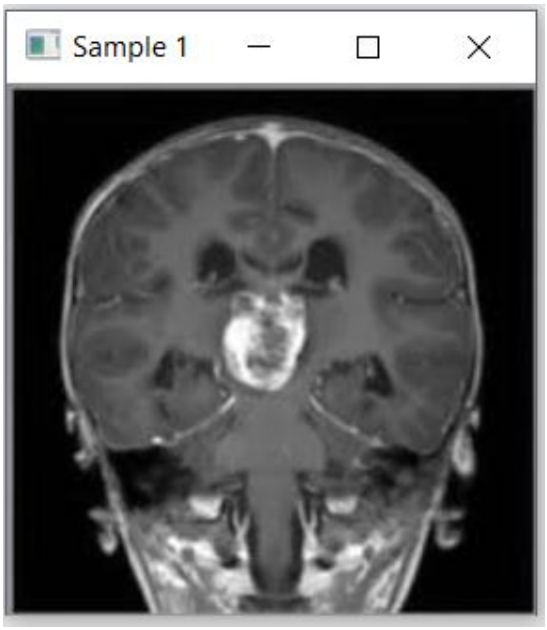
>>> print (cv2.___version___)

4.1.2 Reading, Displaying, and Saving images

Let's see how we can load an image in OpenCV-Python. Create a new jupyter notebook named **program 1**. Create a folder named **images** in the current folder, and make sure that you have at least one image with .jpg format in that folder. Once you do that, add the following lines of code to one of the Python cells. If you run the preceding program, you will see an image being displayed in a new window.

[ ]:
```
import cv2
```

[ ]:
```
img = cv2.imread('images/Sample1.jpg')
cv2.imshow('Sample 1',img)
cv2.waitKey()
```



Let's understand the previous piece of code, line by line. In the first line, we are importing the OpenCV library. We need this for all the functions we will be using in the code. In the second line, we are reading the image and storing it in a variable. OpenCV uses NumPy data structures to store the images. In the next line, we display the image in a new window. The first argument in **cv2.imshow** is the name of the window. The second argument is the image you want to display.

The function, **cv2.waitKey()**, is used in OpenCV for keyboard binding. It takes a number as an argument, and that number indicates the time in milliseconds. Basically, we use this function to wait for a specified duration, until we encounter a keyboard event. The program stops at this point, and waits for you to press any key to continue. If we don't pass any argument or if we pass 0 as the argument, this function will wait for a keyboard event indefinitely.

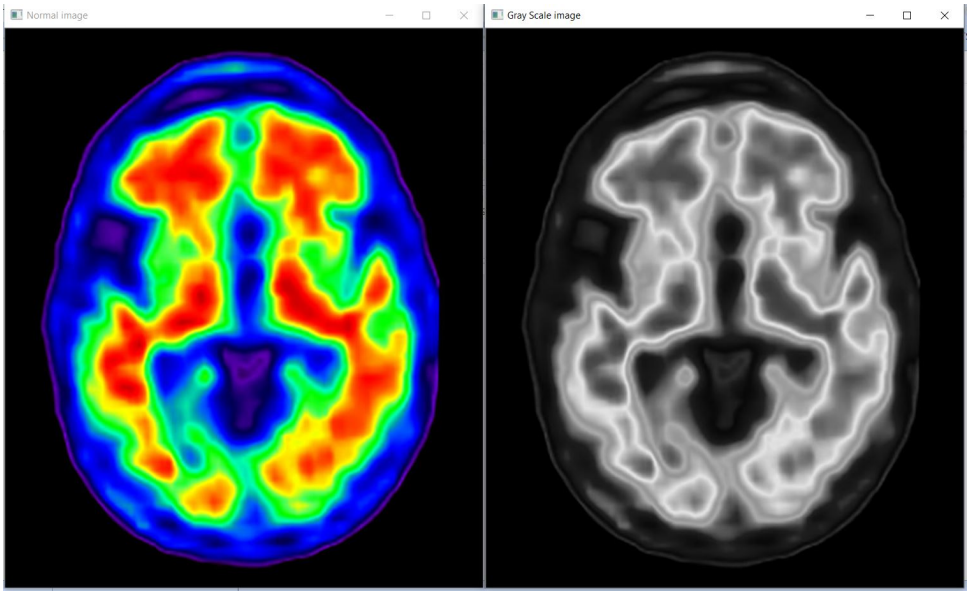
4.1.3 Loading and Saving images

OpenCV provides multiple ways of loading an image. Let's say we want to load a color image in grayscale mode. We can do that using the following piece of code:

[ ]:
```
import cv2
```

[ ]:
```
normal_img = cv2.imread('images/Sample2.jpg')
gray_img = cv2.imread('images/Sample2.jpg',cv2.IMREAD_GRAYSCALE)
cv2.imshow('Normal image',normal_img)
cv2.imshow('Gray Scale image',gray_img)
cv2.waitKey()
```

Here, we are using the flag **cv2.IMREAD_GRAYSCALE** to load the image in grayscale mode. You can see that from the image being displayed in the new window.



We can save this image into a file as well:

```
[ ]: cv2.imwrite('images/output.jpg',gray_img)
```

This will save the grayscale image into an output file named **output.jpg**.

4.1.4 Image color spaces

In computer vision and image processing, color space refers to a specific way of organizing colors. A color space is actually a combination of two things: a color model and a mapping function. The reason we want color models is because it helps us in representing pixel values using tuples. The mapping function maps the color model to the set of all possible colors that can be represented.

There are many different color spaces that are useful. Some of the more popular color spaces are RGB, YUV, HSV, Lab, and so on. Different color spaces provide different advantages. We just need to pick the color space that's right for the given problem. Let's take a couple of color spaces and see what information they provide:

- **RGB**: It's probably the most popular color space. It stands for Red, Green, and Blue. In this color space, each color is represented as a weighted combination of red, green, and blue. So every pixel value is represented as a tuple of three numbers corresponding to red, green, and blue. Each value ranges between 0 and 255.
- **YUV**: Even though RGB is good for many purposes, it tends to be very limited for many real life applications. People started thinking about different methods to separate the intensity information from the color information. Hence, they came up with the YUV color space. Y refers to the luminance or intensity, and U/V channels represent color information. This works well in many applications because the human visual system perceives intensity information very differently from color information.

- **HSV**: : As it turned out, even YUV was still not good enough for some of the applications. So people started thinking about how humans perceive color and they came up with the HSV color space. HSV stands for Hue, Saturation, and Value. This is a cylindrical system where we separate three of the most primary properties of colors and represent them using different channels. This is closely related to how the human visual system understands color. This gives us a lot of flexibility as to how we can handle images.

4.1.5 Converting between color spaces

Considering all the color spaces, there are around 190 conversion options available in OpenCV. If you want to see a list of first 30 available flags, type the following codes in your Jupyter notebook:

[ ]:
```
import cv2
```

[ ]:
```
i = 0
for x in dir(cv2):
    if(x.startswith('COLOR_') and i<50):
        print(x,end=' , ')
        i += 1
```

```
COLOR_BAYER_BG2BGR , COLOR_BAYER_BG2BGRA , COLOR_BAYER_BG2BGR_EA ,
COLOR_BAYER_BG2BGR_VNG , COLOR_BAYER_BG2GRAY , COLOR_BAYER_BG2RGB ,
COLOR_BAYER_BG2RGBA , COLOR_BAYER_BG2RGB_EA , COLOR_BAYER_BG2RGB_VNG ,
COLOR_BAYER_GB2BGR , COLOR_BAYER_GB2BGRA , COLOR_BAYER_GB2BGR_EA ,
COLOR_BAYER_GB2BGR_VNG , COLOR_BAYER_GB2GRAY , COLOR_BAYER_GB2RGB ,
COLOR_BAYER_GB2RGBA , COLOR_BAYER_GB2RGB_EA , COLOR_BAYER_GB2RGB_VNG ,
COLOR_BAYER_GR2BGR , COLOR_BAYER_GR2BGRA , COLOR_BAYER_GR2BGR_EA ,
COLOR_BAYER_GR2BGR_VNG , COLOR_BAYER_GR2GRAY , COLOR_BAYER_GR2RGB ,
COLOR_BAYER_GR2RGBA , COLOR_BAYER_GR2RGB_EA , COLOR_BAYER_GR2RGB_VNG ,
COLOR_BAYER_RG2BGR , COLOR_BAYER_RG2BGRA , COLOR_BAYER_RG2BGR_EA ,
COLOR_BAYER_RG2BGR_VNG , COLOR_BAYER_RG2GRAY , COLOR_BAYER_RG2RGB ,
COLOR_BAYER_RG2RGBA , COLOR_BAYER_RG2RGB_EA , COLOR_BAYER_RG2RGB_VNG ,
COLOR_BGR2BGR555 , COLOR_BGR2BGR565 , COLOR_BGR2BGRA , COLOR_BGR2GRAY ,
COLOR_BGR2HLS , COLOR_BGR2HLS_FULL , COLOR_BGR2HSV , COLOR_BGR2HSV_FULL ,
COLOR_BGR2LAB , COLOR_BGR2LUV , COLOR_BGR2Lab , COLOR_BGR2Luv ,
COLOR_BGR2RGB , COLOR_BGR2RGBA
```

You will see a list of options available in OpenCV for converting from one color space to another. We can pretty much convert any color space into any other color space.

4.2 Geometric Transformations

4.2.1 Image Translation

In this section, we will discuss about shifting an image. Let's say we want to move the image within our frame of reference. In computer vision terminology, this is referred to as translation. Let's go ahead and see how we can do that:

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

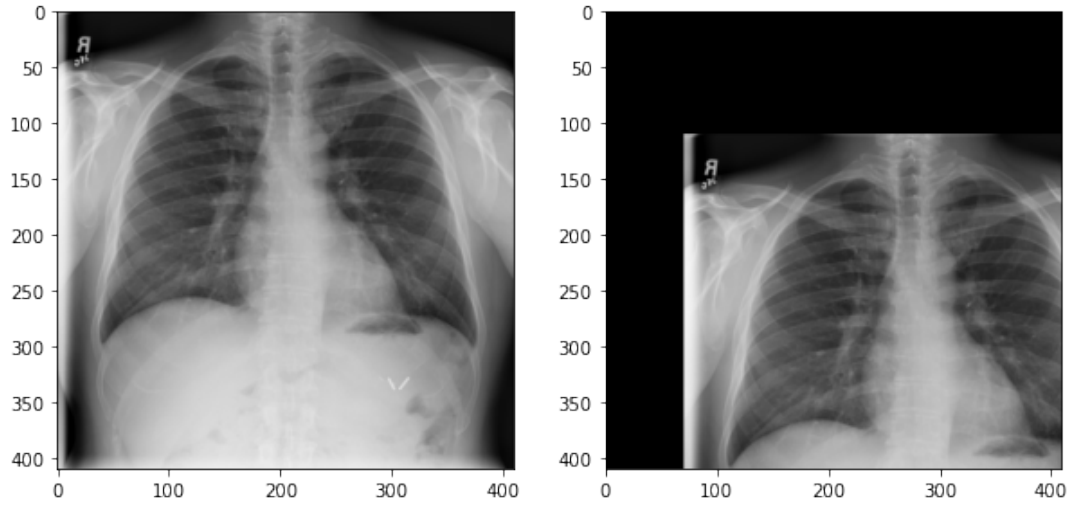
[ ]:
```
img = cv2.imread('images/Sample3.png')
num_rows, num_cols = img.shape[:2]

print('rows:{0}'.format(num_rows))
print('columns:{0}'.format(num_cols))
```

```
rows:410
columns:410
```

[ ]:
```
translation_matrix = np.float32([ [1,0,70], [0,1,110] ])
img_translation = cv2.warpAffine(img, translation_matrix, (num_cols,
num_rows))
```

[ ]:
```
fig, axs = plt.subplots(1,2,figsize=(10,5))
axs[0].imshow(img)
axs[1].imshow(img_translation)
```

[ ]:
```
<matplotlib.image.AxesImage at 0x273d77da430>
```



To understand the preceding code, we need to understand how warping works. Translation basically means that we are shifting the image by adding/subtracting the X and Y coordinates. In order to do this, we need to create a transformation matrix, as shown as follows:

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

Here, the t_x and t_y values are the X and Y translation values, that is, the image will be moved by X units towards the right, and by Y units downwards. So once we create a matrix like this, we can use the function, warpAffine, to apply to our image. The third argument in **warpAffine** refers to the number of rows and columns in the resulting image. Since the number of rows and columns is the same as the original image, the resultant image is going to get cropped. The reason for this is because we didn't have enough space in the output when we applied the translation matrix. To avoid cropping, we can pass a bigger row size and column size in the third argument.

4.2.2 Image Rotation

Now, we will see how to rotate a given image by a certain angle. We can do it using the following piece of code:

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

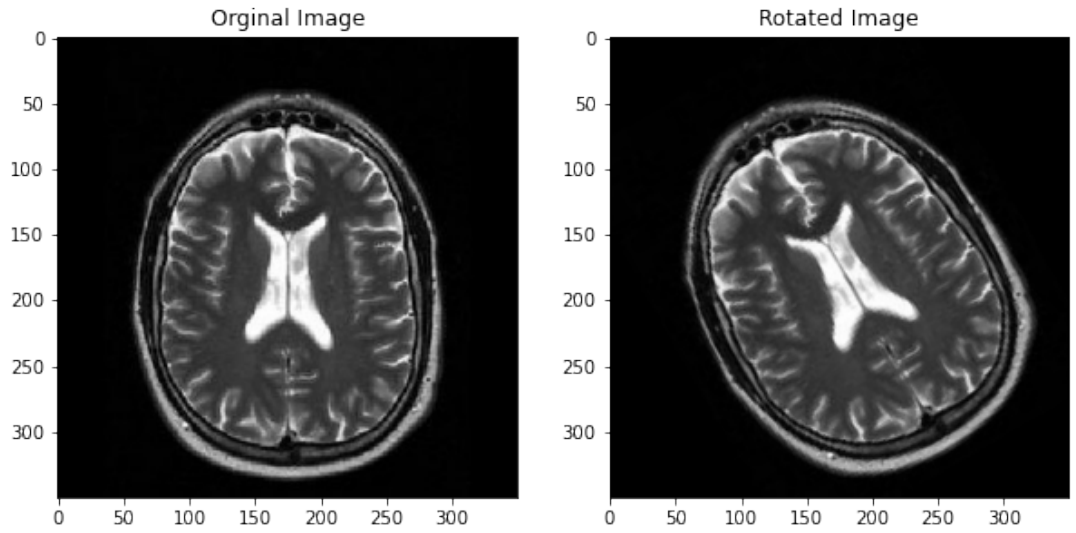
[ ]:
```
img = cv2.imread('images/Sample4.jpg')
num_rows, num_cols = img.shape[:2]

print('Rows : {}'.format(num_rows))
print('Columns : {}'.format(num_cols))
```

```
Rows : 350
Columns : 350
```

[ ]:
```
rotation_matrix=cv2.getRotationMatrix2D((num_cols/2, num_rows/2), 30, 1)
img_rotation=cv2.warpAffine(img,rotation_matrix,(num_cols,num_rows))
```

[ ]:
```
fig, axs = plt.subplots(1,2,figsize=(10,5))

axs[0].imshow(img)
axs[0].set_title('Orginal Image')

axs[1].imshow(img_rotation)
axs[1].set_title('Rotated Image')
```

```
[ ]: Text(0.5, 1.0, 'Rotated Image')
```



In order to understand this, let's see how we handle rotation mathematically. Rotation is also a form of transformation, and we can achieve it by using the following transformation matrix:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Here, θ is the angle of rotation in the counterclockwise direction. OpenCV provides closer control over the creation of this matrix through the function, **getRotationMatrix2D**. We can specify the point around which the image would be rotated, the angle of rotation in degrees, and a scaling factor for the image. Once we have the transformation matrix, we can use the **warpAffine** function to apply this matrix to any image.

4.2.3 Image Scaling

In this section, we will discuss about resizing an image. This is one of the most common operations in computer vision. We can resize an image using a scaling factor, or we can resize it to a particular size. Let's see how to do that:

```
[ ]: import cv2
     import matplotlib.pyplot as plt
```

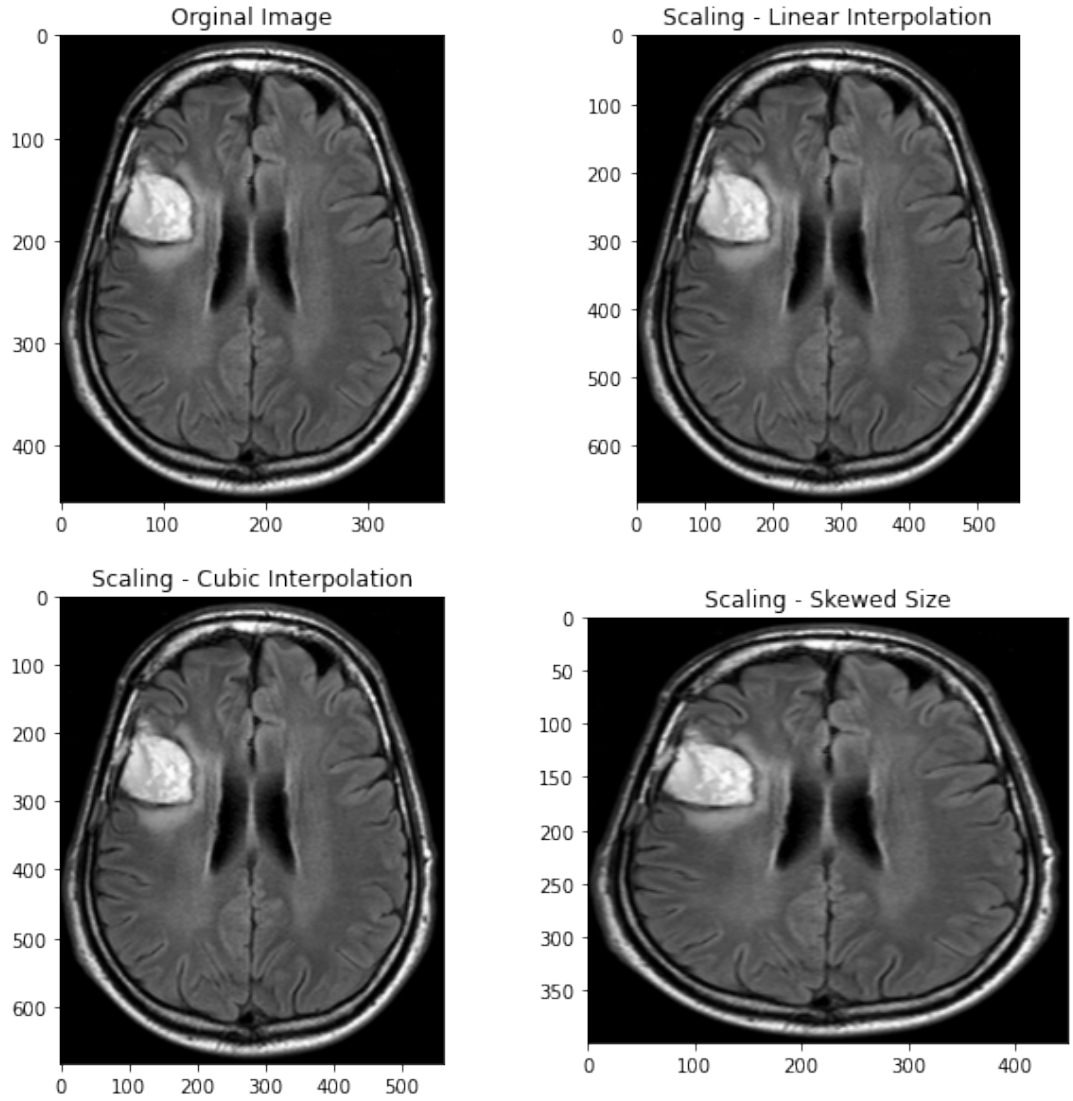
```
[ ]: img = cv2.imread('images/Sample5.jpg')
     print('Orginal image: {}'.format(img.shape))
```

```
img_scaled_Linear = cv2.resize(img,None,fx=1.5, fy=1.5, interpolation =
cv2.INTER_LINEAR)
print('Linear scaled image: {}'.format(img_scaled_Linear.shape))

img_scaled_Cubic = cv2.resize(img,None,fx=1.5, fy=1.5, interpolation =
cv2.INTER_CUBIC)
print('Cubic scaled image: {}'.format(img_scaled_Cubic.shape))

img_scaled_Skewed = cv2.resize(img,(450, 400), interpolation =
cv2.INTER_AREA)
print('Skewed scaled image: {}'.format(img_scaled_Skewed.shape))
```

```
Orginal image: (456, 374, 3)
Linear scaled image: (684, 561, 3)
Cubic scaled image: (684, 561, 3)
Skewed scaled image: (400, 450, 3)
```

[ ]:
```
fig, axs = plt.subplots(2,2,figsize=(10,10))

axs[0,0].imshow(img)
axs[0,0].set_title('Orginal Image')

axs[0,1].imshow(img_scaled_Linear)
axs[0,1].set_title('Scaling - Linear Interpolation')

axs[1,0].imshow(img_scaled_Cubic)
axs[1,0].set_title('Scaling - Cubic Interpolation')

axs[1,1].imshow(img_scaled_Skewed)
axs[1,1].set_title('Scaling - Skewed Size')
```

[ ]:
```
Text(0.5, 1.0, 'Scaling - Skewed Size')
```



Whenever we resize an image, there are multiple ways to fill in the pixel values. When we are enlarging an image, we need to fill up the pixel values in between pixel locations. When we are shrinking an image, we need to take the best representative value. When we are scaling by a non-integer value, we need to interpolate values appropriately, so that the quality of the image is maintained. There are multiple ways to do interpolation. If we are enlarging an image, it's preferable to use linear or cubic interpolation. If we are shrinking an image, it's preferable to use the area-based interpolation. Cubic interpolation is computationally more complex, and hence slower than linear interpolation. But the quality of the resulting image will be higher.

OpenCV provides a function called **resize** to achieve image scaling. If you don't specify a size (by using **None**), then it expects the X and Y scaling factors. In our example, the image will be enlarged by a factor of 1.5. If we want to resize it to a particular size, we can use the format shown in the last resize instance. We can basically skew the image and resize it to whatever size we want.

4.2.4 Affine Transformations

Before talking about affine transformations, let's see what Euclidean transformations are. Euclidean transformations are a type of geometric transformations that preserve length and angle measure. As in, if we take a geometric shape and apply Euclidean transformation to it, the shape will remain unchanged. It might look rotated, shifted, and so on, but the basic structure will not change. So technically, lines will remain lines, planes will remain planes, squares will remain squares, and circles will remain circles.

Coming back to affine transformations, we can say that they are generalizations of Euclidean transformations. Under the realm of affine transformations, lines will remain lines but squares might become rectangles or parallelograms. Basically, affine transformations don't preserve lengths and angles. In order to build a general affine transformation matrix, we need to define the control points. Once we have these control points, we need to decide where we want them to be mapped. In this particular situation, all we need are three points in the source image, and three points in the output image. Let's see how we can convert an image into a parallelogram-like image:

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
img = cv2.imread('images/Sample6.jpg')
rows, cols = img.shape[:2]

print('rows : {}'.format(rows))
print('columns : {}'.format(cols))
```

```
rows : 278
columns : 236
```

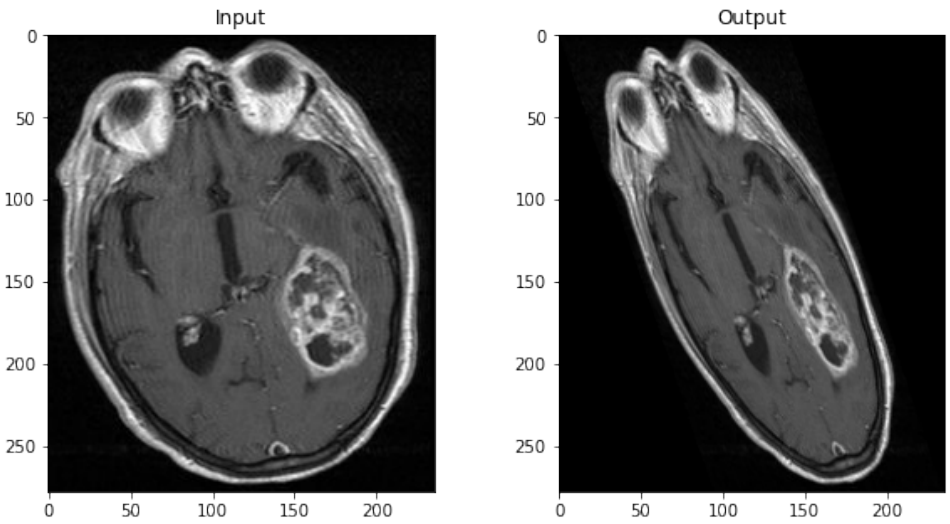
[ ]:
```
src_points = np.float32([[0,0], [cols-1,0], [0,rows-1]])
dst_points = np.float32([[0,0], [int(0.6*(cols-1)),0], [int(0.
 ↪4*(cols-1)),rows-1]])

print('Source points:\n {}'.format(src_points))
print('Destination Points:\n {}'.format(dst_points))
```

```
Source points:
[  0.   0.]
[235.   0.]
[  0. 277.]
Destination Points:
[  0.   0.]
[141.   0.]
[ 94. 277.]
```

```
[ ]: affine_matrix = cv2.getAffineTransform(src_points, dst_points)
     img_output = cv2.warpAffine(img, affine_matrix, (cols,rows))
```

```
[ ]: fig, axs = plt.subplots(1,2,figsize=(10,5))

     axs[0].imshow(img)
     axs[0].set_title('Input')
     axs[1].imshow(img_output)
     axs[1].set_title('Output')
```

```
[ ]: Text(0.5, 1.0, 'Output')
```



As we discussed earlier, we are defining control points. We just need three points to get the affine transformation matrix. We want the three points in **src_points** to be mapped to the corresponding points in **dst_points**. are mapping the points as shown in Figure 4.1.

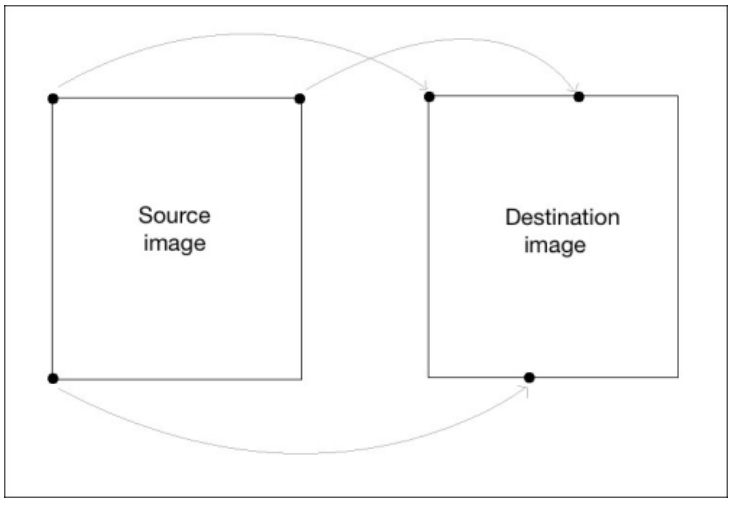


Figure 4.1: Affine Transformation

To get the transformation matrix, we have a function called **getAffineTransform** in OpenCV. Once we have the affine transformation matrix, we use the **warpAffine** function to apply this matrix to the input image. We can also get the mirror image of the input image. We just need to change the control points in the following way:

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
img = cv2.imread('images/Sample7.jpg')
rows, cols = img.shape[:2]

print('rows : {}'.format(rows))
print('columns : {}'.format(cols))
```

```
rows : 299
columns : 299
```

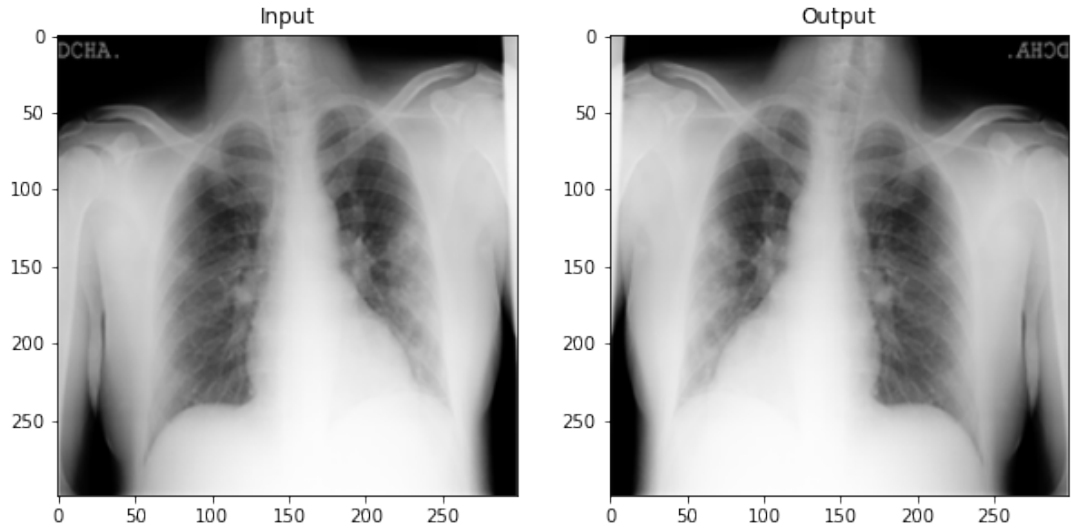
[ ]:
```
src_points = np.float32([[0,0], [cols-1,0], [0,rows-1]])
dst_points = np.float32([[cols-1,0], [0,0], [cols-1,rows-1]])

print('Source points:\n {}'.format(src_points))
print('Destination Points:\n {}'.format(dst_points))
```

```
Source points:
[  0.   0.]
[298.   0.]
[  0. 298.]
Destination Points:
[298.   0.]
[  0.   0.]
[298. 298.]
```

[ ]:
```
affine_matrix = cv2.getAffineTransform(src_points, dst_points)
img_output = cv2.warpAffine(img, affine_matrix, (cols,rows))
```

[ ]:
```
fig, axs = plt.subplots(1,2,figsize=(10,5))

axs[0].imshow(img)
axs[0].set_title('Input')
axs[1].imshow(img_output)
axs[1].set_title('Output')
```

[ ]:
```
Text(0.5, 1.0, 'Output')
```



Here, the mapping looks something like this(Figure 4.2):

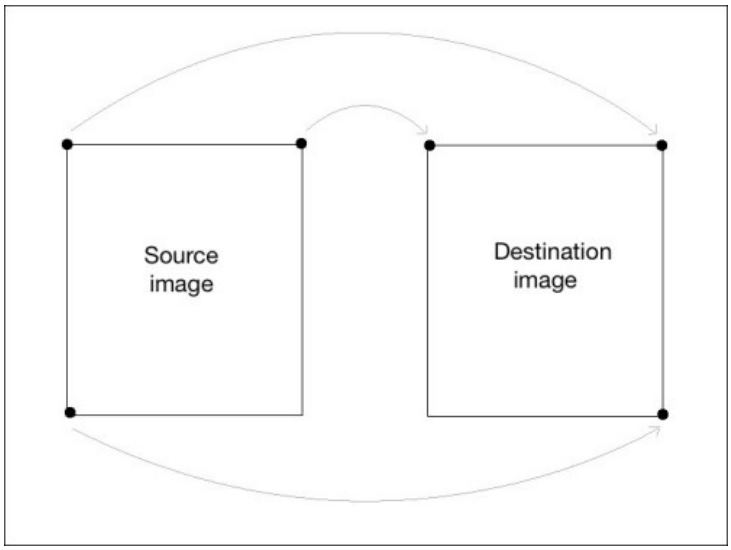


Figure 4.2: Affine Transformation

4.2.5 Projective Transformations

Affine transformations are useful, but they impose certain restrictions. A projective transformation, on the other hand, gives us more freedom. It is also referred to as **homography**. In order to understand projective transformations, we need to understand how projective geometry works. We basically describe what happens to an image when the point of view is changed. For example, if you are standing right in front of a sheet of paper with a square drawn on it, it will look like a square. Now, if you start tilting that sheet of paper, the square will start looking more and more like a trapezoid. Projective transformations allow us to capture this dynamic in a nice mathematical way. These

transformations preserve neither sizes nor angles, but they do preserve incidence and cross-ratio.

We can say that any two images on a given plane are related by a homography. As long as they are in the same plane, we can transform anything into anything else. This has many practical applications such as augmented reality, image rectification, image registration, or the computation of camera motion between two images. Once the camera rotation and translation have been extracted from an estimated homography matrix, this information may be used for navigation, or to insert models of 3D objects into an image or video. This way, they are rendered with the correct perspective and it will look like they were part of the original scene.

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
img = cv2.imread('images/Sample8.jpg')
rows, cols = img.shape[:2]

print('Rows: {}'.format(rows))
print('Columns: {}'.format(cols))
```

```
Rows: 638
Columns: 750
```

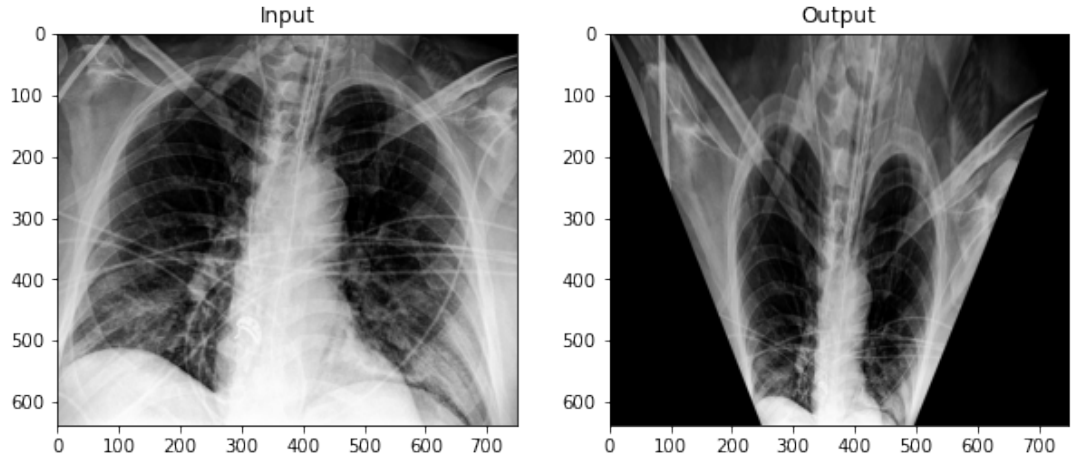
[ ]:
```
src_points = np.float32([[0,0], [cols-1,0], [0,rows-1],
[cols-1,rows-1]])
dst_points = np.float32([[0,0], [cols-1,0], [int(0.33*cols),rows-1],
[int(0.66*cols),rows-1]])

print('Source points:\n {}'.format(src_points))
print('Destination points:\n {}'.format(dst_points))
```

```
Source points:
[  0.   0.]
[749.   0.]
[  0. 637.]
[749. 637.]
Destination points:
[  0.   0.]
[749.   0.]
[247. 637.]
[495. 637.]
```

[ ]:
```
projective_matrix = cv2.getPerspectiveTransform(src_points, dst_points)
img_output = cv2.warpPerspective(img, projective_matrix, (cols,rows))
```

```
[ ]: fig, axs = plt.subplots(1,2,figsize=(10,5))

     axs[0].imshow(img)
     axs[0].set_title('Input')
     axs[1].imshow(img_output)
     axs[1].set_title('Output')
```

```
[ ]: Text(0.5, 1.0, 'Output')
```



We can choose four control points in the source image and map them to the destination image. Parallel lines will not remain parallel lines after the transformation. We use a function called **getPerspectiveTransform** to get the transformation matrix.

4.3 Image Filters

In this section, we will learn how to use fundamental image processing operators. We are going to discuss edge detection and how we can use image filters to apply various effects on photos.

4.3.1 2D Convolution

Convolution is a fundamental operation in image processing. We basically apply a mathematical operator to each pixel and change its value in some way. To apply this mathematical operator, we use another matrix called a kernel. The kernel is usually much smaller in size than the input image. For each pixel in the image, we take the kernel and place it on top such that the center of the kernel coincides with the pixel under consideration. We then multiply each value in the kernel matrix with the corresponding values in the image, and then sum it up. This is the new value that will be substituted in this position in the output image.

Here, the kernel is called the **image filter** and the process of applying this kernel to the given image is called **image filtering**. The output obtained after applying the kernel to the image is called the filtered image. Depending on the values in the kernel, it performs different functions like blurring, detecting edges, and so on. Figure 4.3 should help you visualize the image filtering operation:

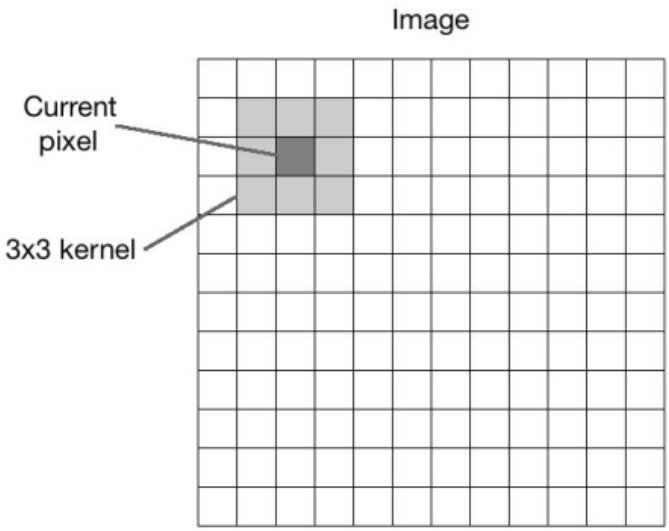


Figure 4.3: Image Kernel

Let's start with the simplest case which is identity kernel. This kernel doesn't really change the input image. If we consider a 3×3 identity kernel, it looks something like the following:

$$I = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

4.3.2 Blurring

Blurring refers to averaging the pixel values within a neighborhood. This is also called a **low pass filter**. A low pass filter is a filter that allows low frequencies and blocks higher frequencies. Frequency refers to the rate of change of pixel values. So we can say that the sharp edges would be high frequency content because the pixel values change rapidly in that region. Going by that logic, plain areas would be low frequency content. Going by this definition, a low pass filter would try to smoothen the edges.

A simple way to build a low pass filter is by uniformly averaging the values in the neighborhood of a pixel. We can choose the size of the kernel depending on how much we want to smoothen the image, and it will correspondingly have different effects. If you choose a bigger size, then you will be averaging over a larger area. This tends to increase the smoothening effect. A 3×3 low pass filter kernel looks like the following:

$$I = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

We are dividing the matrix by 9 because we want the values to sum up to 1. This is called **normalization**, and it's important because we don't want to artificially increase the intensity value at that pixel's location. So you should normalize the kernel before applying it to an image. Here is the code to apply this low pass filter to an image:

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
img = cv2.imread('images/Sample9.jpg')
rows, cols = img.shape[:2]

print('Rows: {}'.format(rows))
print('Columns: {}'.format(cols))
```

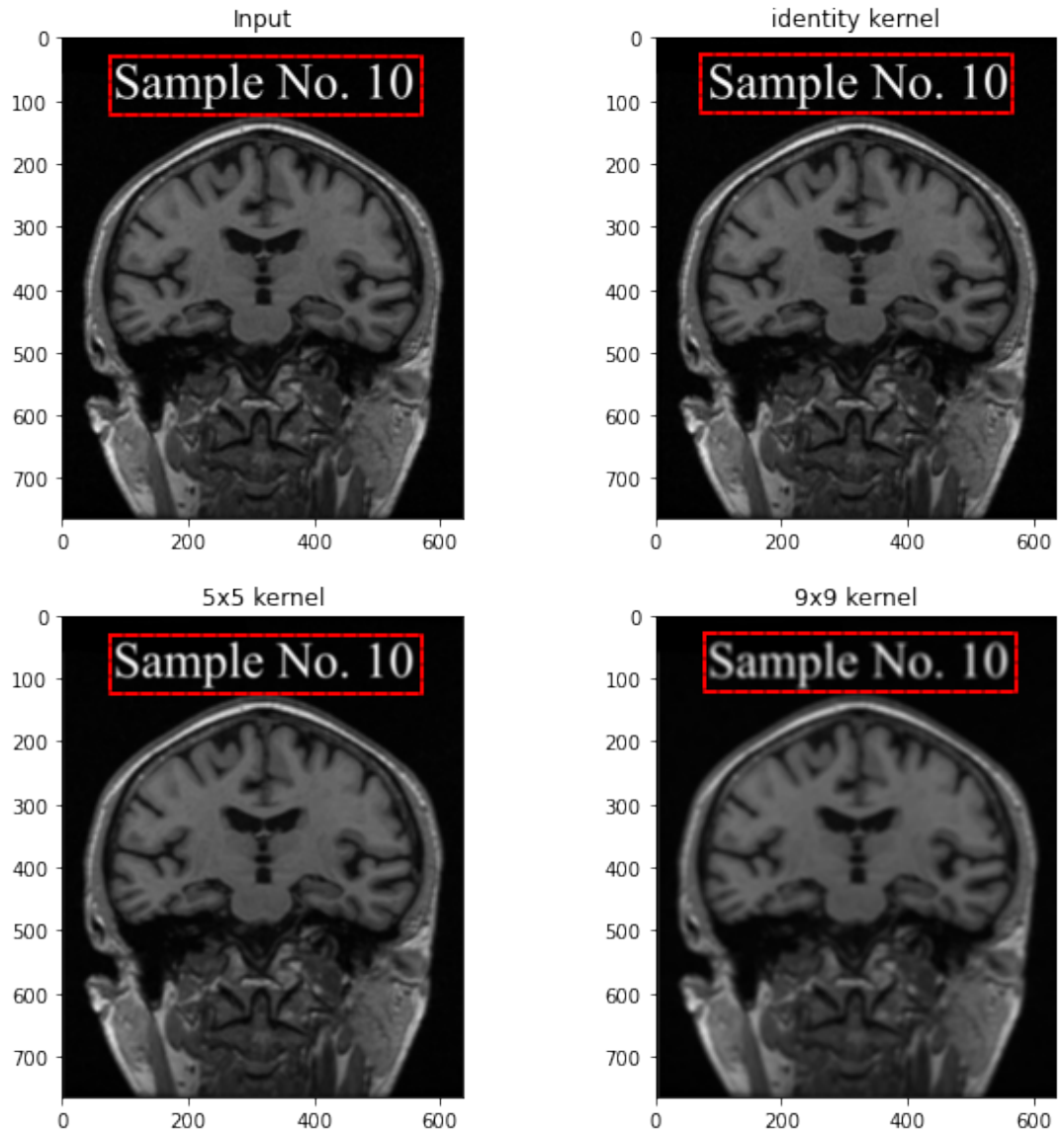
```
Rows: 765
Columns: 637
```

[ ]:
```
# Create Kernels
kernel_identity = np.array([[0,0,0], [0,1,0], [0,0,0]])
kernel_5x5 = np.ones((5,5), np.float32) / 25.0
kernel_9x9 = np.ones((9,9), np.float32) / 81.0
```

[ ]:
```
# Apply Kernels
identity_img = cv2.filter2D(img, -1, kernel_identity)
kernel_5x5_img = cv2.filter2D(img, -1, kernel_5x5)
kernel_9x9_img = cv2.filter2D(img, -1, kernel_9x9)
```

[ ]:
```
fig, axs = plt.subplots(2,2,figsize=(10,10))

axs[0,0].imshow(img)
axs[0,0].set_title('Input')

axs[0,1].imshow(identity_img)
axs[0,1].set_title('identity kernel')

axs[1,0].imshow(kernel_5x5_img)
axs[1,0].set_title('5x5 kernel')

axs[1,1].imshow(kernel_9x9_img)
axs[1,1].set_title('9x9 kernel')
```

```
[ ]: Text(0.5, 1.0, '9x9 kernel')
```



In the preceding code, we are generating different kernels in the code which are kernel_identity, kernel_5×5, and kernel_9×9. We use the function, **filter2D**, to apply these kernels to the input image. If you look at the images carefully, you can see that they keep getting blurrier as we increase the kernel size. The reason for this is because when we increase the kernel size, we are averaging over a larger area. This tends to have a larger blurring effect. An alternative way of doing this would be by using the OpenCV function, **cv2.blur**. If you don't want to generate the kernels yourself, you can just use this function directly.

4.4 Edge Detection

The process of edge detection involves detecting sharp edges in the image and producing a binary image as the output. Typically, we draw white lines on a black background to indicate those edges. We can think of edge detection as a high pass filtering operation. A high pass filter allows high frequency content to pass through and blocks the low frequency content. As we discussed earlier, edges are high frequency content. In edge detection, we want to retain these edges and discard everything else. Hence, we should build a kernel that is the equivalent of a high pass filter.

Let's start with a simple edge detection filter known as the Sobel filter. Since edges can occur in both horizontal and vertical directions, the **Sobel filter** is composed of the following two kernels:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 1 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The kernel on the left detects horizontal edges and the kernel on the right detects vertical edges. OpenCV provides a function to directly apply the Sobel filter to a given image.

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[10]:
```
img = cv2.imread('images/Sample10.jpg')
rows, cols, _ = img.shape

print('Rows: {}'.format(rows))
print('Columns: {}'.format(cols))
```

```
Rows: 212
Columns: 220
```

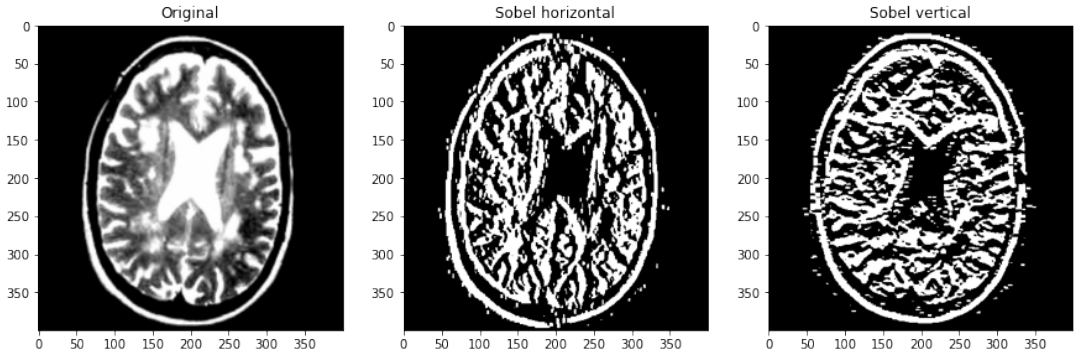
[ ]:
```
# iameg resizing
img = cv2.resize(img,(400,400))
rows, cols,_ = img.shape

print('Rows: {} \nColumns: {}'.format(rows,cols))
```

```
Rows: 400
Columns: 400
```

[ ]:
```
# create Sobel Kernels
sobel_horizontal = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
sobel_vertical = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)
```

[ ]:
```
fig, axs = plt.subplots(1,3,figsize=(15,5))

axs[0].imshow(img)
axs[0].set_title('Original')

axs[1].imshow(sobel_horizontal)
axs[1].set_title('Sobel horizontal')

axs[2].imshow(sobel_vertical)
axs[2].set_title('Sobel vertical')
```

[ ]:
```
Text(0.5, 1.0, 'Sobel vertical')
```



In the preceding figure, the image in the middle is the output of horizontal edge detector, and the image on the right is the vertical edge detector. As we can see here, the Sobel filter detects edges in either a horizontal or vertical direction and it doesn't give us a holistic view of all the edges. To overcome this, we can use the **Laplacian filter**. The advantage of using this filter is that it uses double derivative in both directions.

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

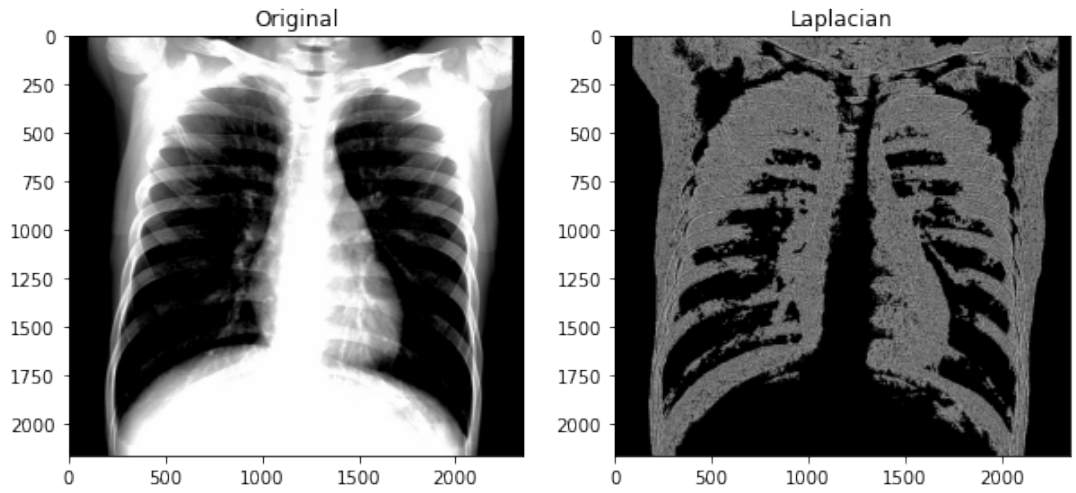
[ ]:
```
img = cv2.imread('images/Sample11.jpg')
rows, cols, _ = img.shape

print('Rows: {} \nColumns: {}'.format(rows,cols))
```

```
Rows: 2166
Columns: 2352
```

[ ]:
```
# apply Laplacian Kernel
laplacian = cv2.Laplacian(img, cv2.CV_64F)
```

[ ]:
```
fig, axs = plt.subplots(1,2,figsize=(10,5))

axs[0].imshow(img)
axs[0].set_title('Original')

axs[1].imshow(laplacian)
axs[1].set_title('Laplacian')
```

[ ]:
```
Text(0.5, 1.0, 'Laplacian')
```



Even though the Laplacian kernel worked in this case, it doesn't always work well. It gives rise to a lot of noise in the output. To overcome this problem, we use the **Canny edge detector**. To use the Canny edge detector, we can use the following code:

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
img = cv2.imread('images/Sample11.jpg')
rows, cols, _ = img.shape

print('Rows: {} \nColumns: {}'.format(rows,cols))
```

```
Rows: 2166
Columns: 2352
```

[ ]:
```
# iameg resizing
img = cv2.resize(img,(588,541))
```

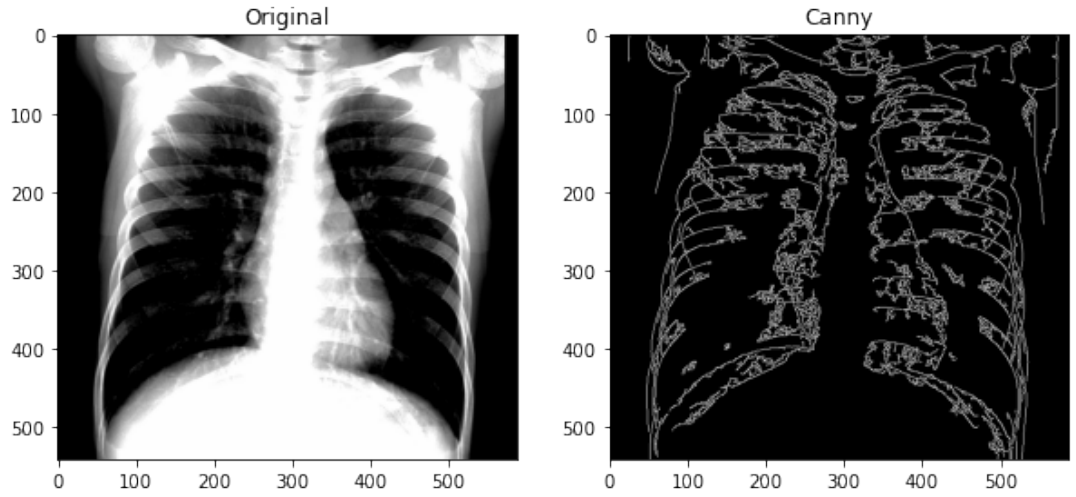
```
rows, cols,_ = img.shape

print('Rows: {} \nColumns: {}'.format(rows,cols))
```

```
Rows: 541
Columns: 588
```

[ ]:
```
# apply Canny Kernel
canny = cv2.Canny(img, 50, 240)
```

[ ]:
```
fig, axs = plt.subplots(1,2,figsize=(10,5))

axs[0].imshow(img)
axs[0].set_title('Original')

axs[1].imshow(canny,cmap='gray')
axs[1].set_title('Canny')
```

[ ]:
```
Text(0.5, 1.0, 'Canny')
```



As we can see, the quality of the **Canny edge detector** is much better. It takes two numbers as arguments to indicate the thresholds. The second argument is called the low threshold value, and the third argument is called the high threshold value. If the gradient value is above the high threshold value, it is marked as a strong edge. The Canny Edge Detector starts tracking the edge from this point and continues the process until the gradient value falls below the low threshold value. As you increase these thresholds, the weaker edges will be ignored. The output image will be cleaner and sparser.

4.5 Sharpening

Applying the **sharpening filter** will sharpen the edges in the image. This filter is very useful when we want to enhance the edges in an image that's not crisp. The level of sharpening depends on the type of kernel we use. We have a lot of freedom to customize the kernel here, and each kernel will give you a different kind of sharpening. To just sharpen an image we would use a kernel like this:

$$M_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Also, if we want to do **excessive sharpening** we would use the following kernel:

$$M_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -7 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

But the problem with these two kernels is that the output image looks artificially enhanced. If we want our images to look more natural, we would use an **Edge Enhancement filter**. The underlying concept remains the same, but we use an approximate Gaussian kernel to build this filter. It will help us smoothen the image when we enhance the edges, thus making the image look more natural.

$$M_3 = \frac{1}{8} \begin{bmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & 2 & 8 & 2 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Here is the code to achieve the effects applied to the image:

[ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
img = cv2.imread('images/Sample12.jpeg')
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
rows, cols, dimensions = img.shape


print('Rows: {} \nColumns: {}'.format(rows,cols))
```

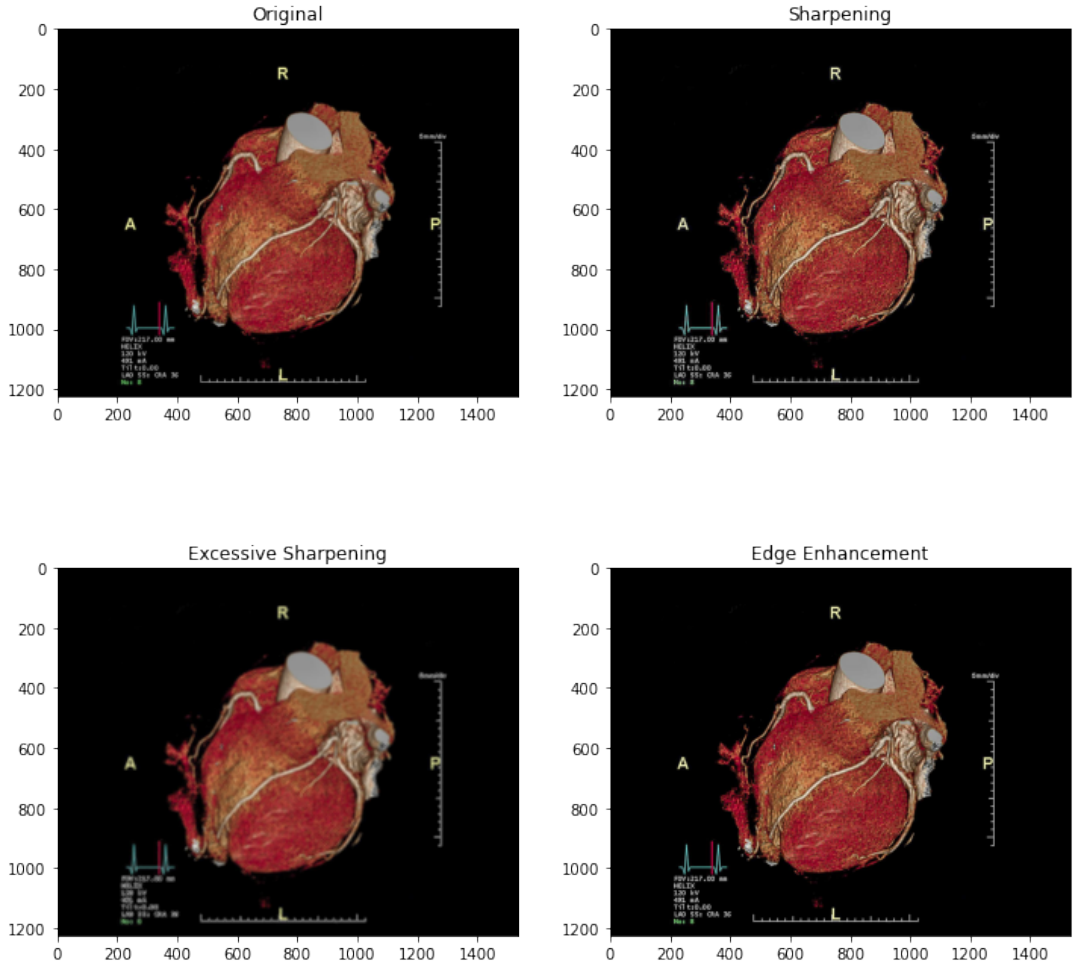
```
Rows: 1225
Columns: 1536
```

[ ]:
```
# generating the kernels

# sharpening
kernel_sharpening = np.array([[-1,-1,-1], [-1,9,-1], [-1,-1,-1]])
#excessive sharpening
kernel_excessive_sharpening = np.array([[1,1,1], [1,-7,1], [1,1,1]])
# Edge Enhancement
kernel_edge_enhancement = np.array([[-1,-1,-1,-1,-1],
                                    [-1,2,2,2,-1],
                                    [-1,2,8,2,-1],
                                    [-1,2,2,2,-1],
                                    [-1,-1,-1,-1,-1]]) / 8.0
```

[ ]:
```
# applying different kernels to the input image
output_1 = cv2.filter2D(img, -1, kernel_sharpening)
output_2 = cv2.filter2D(img, -1, kernel_excessive_sharpening)
output_3 = cv2.filter2D(img, -1, kernel_edge_enhancement)
```

[ ]:
```
fig, axs = plt.subplots(2,2,figsize=(12,12))

axs[0,0].imshow(img)
axs[0,0].set_title('Original')

axs[0,1].imshow(output_1)
axs[0,1].set_title('Sharpening')

axs[1,0].imshow(output_2)
axs[1,0].set_title('Excessive Sharpening')

axs[1,1].imshow(output_3)
axs[1,1].set_title('Edge Enhancement')
```

[ ]:
```
Text(0.5, 1.0, 'Edge Enhancement')
```



If you noticed, in the preceding code, we didn't divide the first two kernels by a normalizing factor. The reason is because the values inside the kernel already sum up to 1, so we are implicitly dividing the matrices by 1.

4.6 Enhancing the Contrast

Whenever we capture images in low-light conditions, the images turn out to be dark. The reason this happens is because the pixel values tend to concentrate near 0 when we capture the images. When this happens, a lot of details in the image are not clearly visible to the human eye. The human eye likes contrast, and so we need to adjust the contrast to make the image look nice and pleasant. We use a process called Histogram Equalization to achieve the higher contrast. We need to adjust the pixel values so that they are spread across the entire spectrum of values, that is, between 0 and 255. Following is the code for adjusting the pixel values:

[ ]:
```
import cv2
import numpy as np
```

```
import matplotlib.pyplot as plt
```

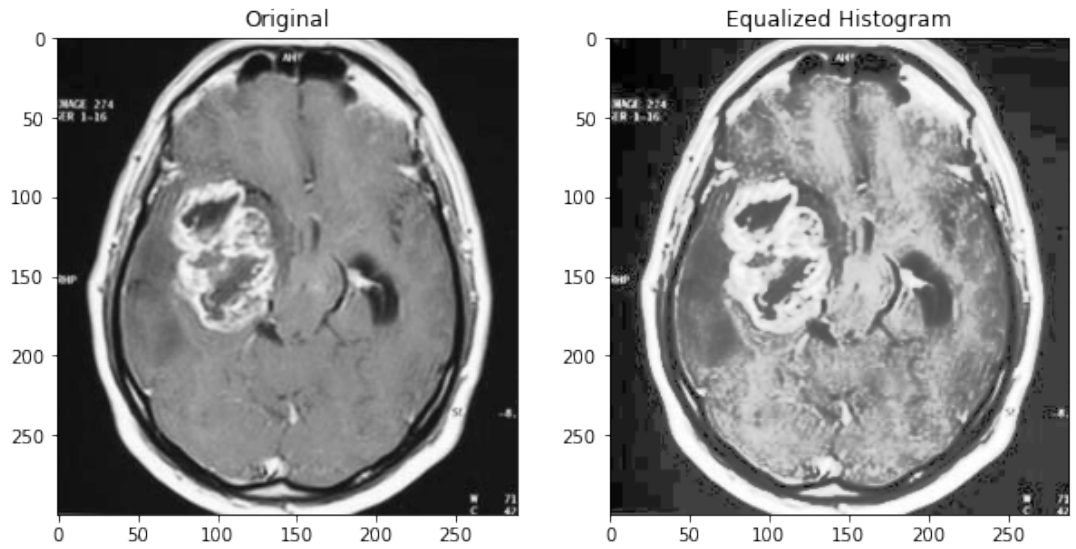
[ ]:
```
img = cv2.imread('images/Sample14.jpg',0)
(rows,cols) = img.shape

(rows,cols)
```

[ ]:
```
(300, 289)
```

[ ]:
```
# equalize the histogram of the input image
histeq = cv2.equalizeHist(img)
```

[ ]:
```
fig, axs = plt.subplots(1,2,figsize=(10,5))

axs[0].imshow(img,cmap='gray')
axs[0].set_title('Original')

axs[1].imshow(histeq,cmap='gray')
axs[1].set_title('Equalized Histogram')
```

[ ]:
```
Text(0.5, 1.0, 'Equalized Histogram')
```



Chapter 5

Detecting Objects and Image Segmentation

In this chapter n, we are going to learn about shape analysis and medical image segmentation. We will learn how to recognize objects and estimate the exact boundaries. We will discuss how to segment an image into its constituent parts using various methods. We will learn how to separate the foreground from the background as well.

5.1 segmentation using Fuzzy C-Means and Thresholding

This article focusses on identification of brain tumor in MR images, It involves in removing noise using noise removal technique AMF followed by enhancing the images using Balance Enhancement Contrast technique (BCET).Further, image segmentation is performed using fuzzy c-means and finally the segmented images are produced as an input to a canny edge detection resulting with the tumor image.

5.1.1 Fuzzy C-Means (FCM)

Fuzzy c-means (FCM) is a method of clustering which allows one piece of data to belong to two or more clusters. This method is frequently used in pattern recognition. This algorithm works by assigning membership to each data point corresponding to each cluster center based on distance between the cluster and the data point. The nearer the data is to the cluster center the better is membership to the cluster center. Algoritm 1 is FCM procedure.

Algorithm 1 Fuzzy C-Means (FCM) Algorithm

Input: data x = x_1,x_2,...,x_k , Size of data: N

Local: Fuzzification parameter: m , Threshold: ϵ , Number of clusters: c

Initialize partition matrix randomly(U^0)

t = 0

repeat

  for i = 1 : c **do**

    $V_i(\mathrm{t}) = \frac{\sum_{k=1}^{N} \mu_{ik}^{m}(t) x_k}{\sum_{k=1}^{N} \mu_{ik}^{m}(t)}$

  for i = 1 : c **do**

    for k = 1 : N **do**

      $\mu(\mathrm{t}+1) = \frac{1}{\sum_{j=1}^{c} (\frac{||x_k - v_i(t)||}{||x_k - v_j(t)||})^{\frac{2}{m-1}}}$

  t = t + 1

until $||\mu(t+1) - \mu(t)|| \leq \epsilon$

Return U,V

5.1.2 Balance Enhancement Contrast Technique (BECT)

Color bias is one major cause of poor color composite images. To eliminate this, the three bands used for color composition must have an equal value range and mean. The balance contrast enhancement technique (BCET) is a simple solution for this problem. Using a parabolic or cubic function defined by three coefficients, BCET can stretch (or compress) images exactly to a value range and mean given by a user without changing the basic shapes of the image histograms. As color bias is completely avoided and the full value range of the display system is properly used, high-quality color composites as well as black and white singleband images are produced by BCET.

We need some parameters for BCET such as:

- **l**: represents the minimum value of the input image
- **h**: denotes the maximum value of the input image
- **e**: denotes the mean value of the input image
- **L**: represents the minimum value of the output image
- **H**: denotes the maximum value of the output image
- **E**: denotes the mean value of the output image
- **s**: denotes the mean square sum of the input image

The general form of the parabolic function is defined as:

$$y = a(x - b)^2 + c \tag{5.1}$$

The three coefficients a,b and c are derived from the following formulas:

$$a = \frac{H - L}{(h - l)(h + l - 2b)} \tag{5.2}$$

$$b = \frac{h^2(E - L) - s(H - L) + l^2(H - E)}{2(h(E - L) - e(H - L) + l(H - E))} \tag{5.3}$$

$$c = L - a(l - b)^2 \tag{5.4}$$

$$s = \frac{1}{N}\sum_{i=1}^{N} x_i^2 \tag{5.5}$$

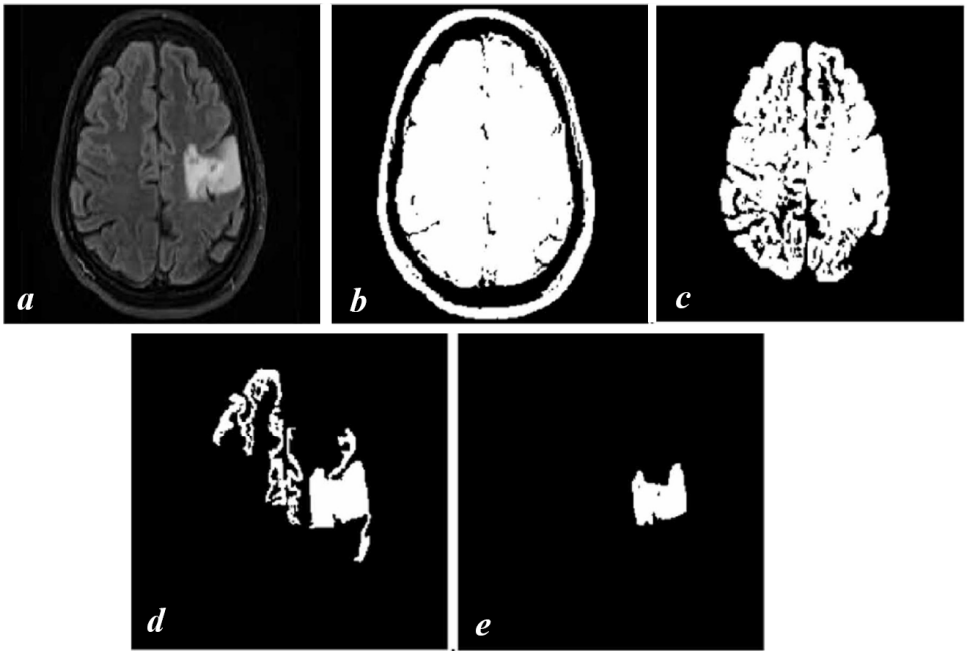


Figure 5.1: Example of the segmentation for different mean of BCET: a) original image, b) BCET 120, c) BCET 100, d) BCET 80, e) BCET 60

5.1.3 Adaptive Median Filtering (AMF)

Image noise can be briefly defined as random variations in some of the pixel values of an image. We know filters are used to reduce the amount of noise present in an image. There are some outlier pixel values in images, due to these outlier cause disturbance (salt & pepper) or also called as noise in images. Median filtering is excellent at reducing this type of noise. The filtering algorithm will scan the entire image, using a small matrix

and recalculate the value of the center pixel by simply taking the median of all the values inside the matrix.

the median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the mean of neighboring pixel values, it replaces it with the median of those values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value. The following example should clear the concept.

$$current\ value = \begin{bmatrix} 19 & 12 & 38 \\ 42 & 11 & 9 \\ 27 & 40 & 33 \end{bmatrix}$$

Sort: 9 , 11 , 12 , 19 , 27 , 33 , 38 , 40 , 42

Median: 27

$$after\ apply\ filter = \begin{bmatrix} 19 & 12 & 38 \\ 42 & 27 & 9 \\ 27 & 40 & 33 \end{bmatrix}$$

The Adaptive Median Filter performs spatial processing to determine which pixels in an image have been affected by impulse noise. The Adaptive Median Filter classifies pixels as noise by comparing each pixel in the image to its surrounding neighbor pixels. The size of the neighborhood is adjustable, as well as the threshold for the comparison. A pixel that is different from most of its neighbors, as well as being not structurally aligned with those pixels to which it is similar, is labeled as impulse noise. These noise pixels are then replaced by the median pixel value of the pixels in the neighborhood that have passed the noise labeling test.

The Adaptive Median Filter needs the following parameters:

- S_{xy}: the support of the filter centered at (x, y)
- Z_{min}: minimum gray level value in S_{xy}
- Z_{max}: maximum gray level value in S_{xy}
- Z_{med}: median of gray levels in S_{xy}
- Z_{xy}: gray level at coordinates (x, y)
- S_{max}: maximum allowed size of S_{xy}

AMF uses the following algorithm (Algorithm 2):

Algorithm 2 Adaptive Median Filter

Level A:
A1 = Z_{med} - Z_{min}
A2 = Z_{med} - Z_{max}
if A1 $>$ 0 AND A2 $<$ 0 **then**
    go to **level B**
else
    increase the window size
    if window size $<$ S_{max} **then**
        repeat level A
    else
        output Z_{xy}
Level B:
B1 = Z_{xy} - Z_{min}
B2 = Z_{xy} - Z_{max}
if B1 $>$ 0 AND B2 $<$ 0 **then**
    output Z_{xy}
else
    output Z_{med}

Level A determines if the output of the filter Z_{med} is an impulse or not(black or white). If it is not an impulse we go to **level B**. On the other hand, if it is an impulse, the window size is increased until it reaches the maximum window size or Z_{med} is not an impulse.

level B determines if the pixel value at (x, y), that is Z_{xy}, is an impulse or not. If it is not an impulse, the algorithm output wont change current Z_{xy}. But, if it is an impulse the algorithm output will be Z_{med}.

5.1.4 Otsu's Thresholding

Converting a grayscale image to monochrome is a common image processing task. Otsu's method, named after its inventor Nobuyuki Otsu, is one of many binarization algorithms, it uses data-driven approach which can adaptively find the optimal threshold to distinguish two-class data, by going through all possible threshold values (from 0 to 255), it can find the optimal threshold value of input image. This method can be applied in image segmentation and image binarization, in the current project the use was for image binarization.

Threshold is a value between 0 to 255. For example, if we set threshold value T = 128, then the image is separated into two classes, which are Class 1 (pixel value<= 128) and Class 2 (pixel value> 128) We can say that these two classes represent background and foreground of the input image, respectively.

5.1.5 Canny Edge Detection

In the final step of the algorithm, the 2 segmented images resulted from the segmentation process are combined using opencv **addWeighted** method by providing desired weights, the tumor being weighted more. This combined image is provided as an argument for the canny edge detection, the method highlights the tumor region and results with detected tumor output image. There are many other methods which can be used for contour extraction such as Roberts, Prewitt, Sobel, and more complex ones like LoG but we used as a reference shown Canny method to be effective for contour extraction.

5.1.6 Architecture of the system

Figure 5.2 illustrates the architecture of the system.

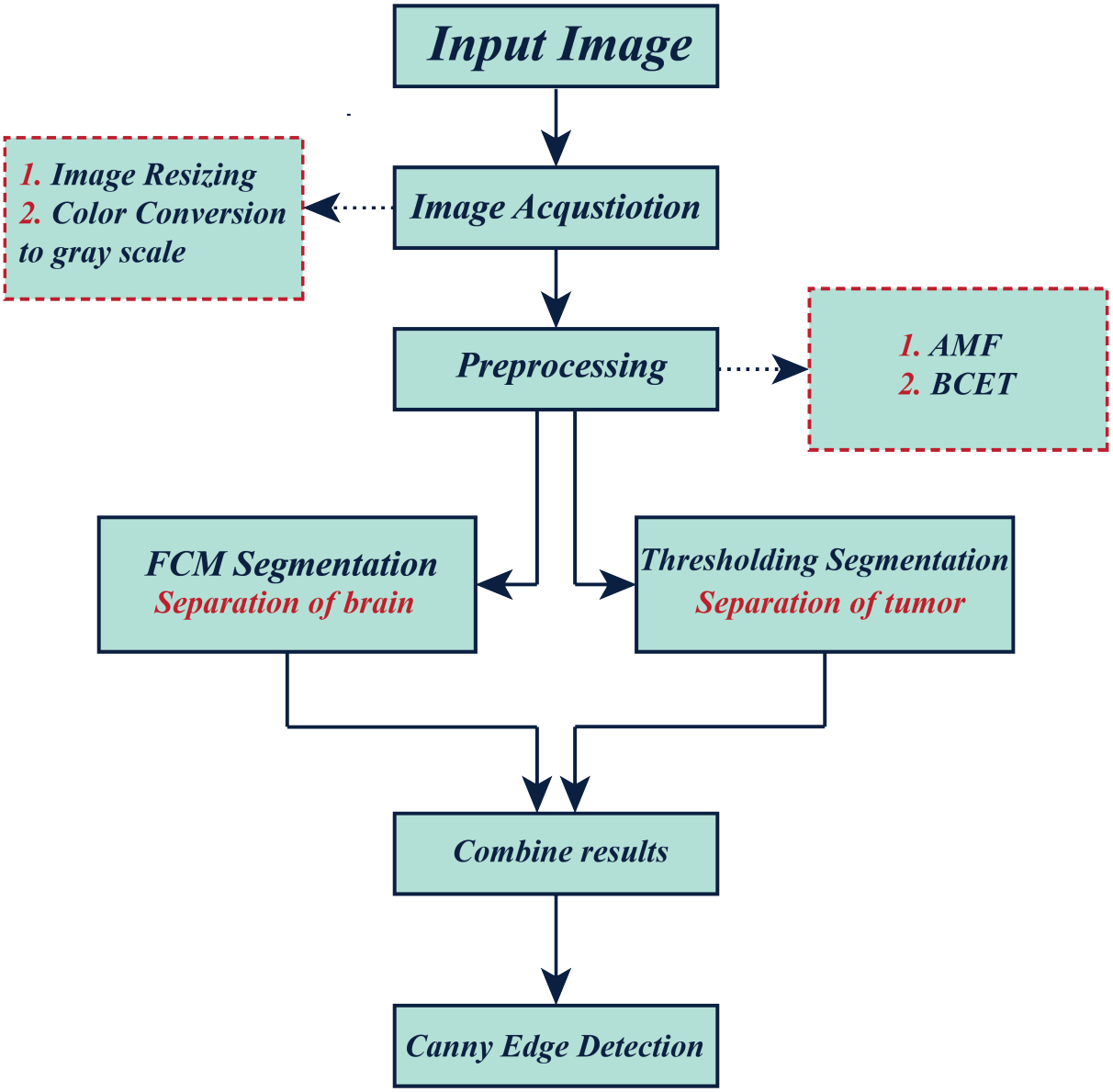


Figure 5.2: Architecture of the system

5.1.7 Implementation

[ ]:
```
import sys
import os
from time import time
import cv2
import numpy as np
import matplotlib.pyplot as plt
import skfuzzy as fuzz
```

[ ]:
```
def AdaptiveMedianFilter(grayimage):
    try:
        img_out = grayimage.copy()
```

```
        height = grayimage.shape[0]
        width = grayimage.shape[1]
        for i in np.arange(6, height - 5):
            for j in np.arange(6, width - 5):
                neighbors = []

                for k in np.arange(-6, 6):
                    for l in np.arange(-6, 6):
                        a = grayimage.item(i + k, j + l)
                        neighbors.append(a)
        neighbors.sort()
        median = neighbors[30]
        b = median
        img_out.itemset((i, j), b)

    except Exception as e:
        print("Error=" + e.args[0])
        tb = sys.exc_info()[2]
                print(tb.tb_lineno)

    return img_out.astype(np.uint8)
```

[ ]:
```
def BalanceContrastEnhancementTechnique(gray_image):
    x = im2double(gray_image) # INPUT IMAGE
    Lmin = np.min(x.ravel()) # MINIMUM OF INPUT IMAGE
    Lmax =np.max(x.ravel()) # MAXIMUM OF INPUT IMAGE
    Lmean = np.mean(x) # MEAN OF INPUT IMAGE
    LMssum = np.mean(pow(x,2)) # MEAN SQUARE SUM OF INPUT IMAGE

    Gmin = 0 # MINIMUM OF OUTPUT IMAGE
    Gmax = 255 # MAXIMUM OF OUTPUT IMAGE
    Gmean =85 # MEAN OF OUTPUT IMAGE 80 (Recomended)

    bnum = pow(Lmax,2) * (Gmean - Gmin) - LMssum * (Gmax - Gmin) +
    pow(Lmin,2) * (Gmax - Gmean)
    bden = 2 * (Lmax * (Gmean - Gmin) - Lmean * (Gmax - Gmin) + Lmin *
    (Gmax - Gmean))

    b = bnum / bden
    a = (Gmax - Gmin) / ((Lmax - Lmin) * (Lmax + Lmin - 2 * b))
    c = Gmin - a * pow((Lmin - b), 2)
    y = a *pow((x - b),2)+ c # PARABOLIC FUNCTION
    y = y.astype(np.uint8)

    return y
```

[ ]:
```
def im2double(im):
    min_val = np.min(im.ravel())
    max_val = np.max(im.ravel())
    out = (im.astype('float') - min_val) / (max_val - min_val)
    return out
```

NumPy python package is used to calculate the min,max,mean of the input image by converting the given image pixel array to double precession and flattening out the array using NumPy ravel function and applying desired min,max functions existing in NumPy package. The resultant 'y' is the new enhanced/stretched contrast image, this image is used in further processing for thresholding and contour extraction.

[ ]:
```
def FCM(image_bcet):
    list_img = []
    img =cv2.imread("C:/Users/Poorya/Desktop/"+str(image_bcet))

    rgb_img = img.reshape((img.shape[0] * img.shape[1], 3))
    list_img.append(rgb_img)
    n_data = len(list_img)
    clusters = [2]

    for index, rgb_img in enumerate(list_img):
        img = np.reshape(rgb_img, (256, 256, 3)).astype(np.uint8)
        shape = np.shape(img)
        # looping every cluster
        for i, cluster in enumerate(clusters):
            # Fuzzy C Means
            new_time = time()

            cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(rgb_img.T,
            cluster, 2, error=0.005, maxiter=1000, init=None, seed=42)
            new_img = change_color_fuzzycmeans(u, cntr)
            fuzzy_img = np.reshape(new_img, shape).astype(np.uint8)
            ret, seg_img = cv2.threshold(fuzzy_img,np.max(fuzzy_img) -
            1,255,cv2.THRESH_BINARY)

            seg_img_1d = seg_img[:, :, 1]
            bwfim1 = bwareaopen(seg_img_1d, 500)
            bwfim2 = imclearborder(bwfim1)

            cv2.imwrite('C:/Users/Poorya/Desktop/border.jpg', bwfim2)


            bwfim3 = imfill(bwfim2)

            cv2.imwrite('C:/Users/Poorya/Desktop/FCM.jpg', bwfim3)
    return bwfim3
```

[ ]:
```
def change_color_fuzzycmeans(cluster_membership, clusters):
    img = []
    for pix in cluster_membership.T:
        img.append(clusters[np.argmax(pix)])
    return img
```

[ ]:
```
def imfill(im_th):
    im_floodfill = im_th.copy()
    # Mask used to flood filling.
    # Notice the size needs to be 2 pixels than the image.
    h, w = im_th.shape[:2]
    mask = np.zeros((h + 2, w + 2), np.uint8)
    # Floodfill from point (0, 0)
    cv2.floodFill(im_floodfill, mask, (0, 0), 255);
    # Invert floodfilled image
    im_floodfill_inv = cv2.bitwise_not(im_floodfill)
    # Combine the two images to get the foreground.
    im_out = im_th | im_floodfill_inv
    return im_out
```

After clustering, the cluster centers are passed to custom function and a new image is constructed according to the cluster membership. Thresholding is performed on this image using opencv threshold method, this is usually done for optimal contour extraction.

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.Here they suggest performing thresholding before contour extraction. The contour extraction is applied at 2 different stages:

[ ]:
```
def bwareaopen(imgBW, areaPixels):
    # Given a black and white image, first find all of its contours
    imgBWcopy = imgBW.copy()
    contours, hierarchy = cv2.findContours(imgBWcopy.copy(), cv2.
 ↪RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
    # For each contour, determine its total occupying area
    for idx in np.arange(len(contours)):
        area = cv2.contourArea(contours[idx])
        if (area >= 0 and area <= areaPixels):
            cv2.drawContours(imgBWcopy, contours, idx, (0, 0, 0), -1)

    return imgBWcopy
```

The above function determines the total occupying area of each contour and on comparing with the fixed value of area pixels- set to 500, this is value is used to preserve and draw the contours which ranges in the specific area. All the extracted contours which fall in the range are redrawn using opencv drawcontours method.

The next step is a selection the boundary pixels (edge pixels). A pixel is considered as boundary pixel if the magnitude of the gradient of this pixel is greater than that of two

neighbors in the direction of the gradient In the below method, each contour row and column pixel is evaluated, the row and column check conditions are used to determine the border of the brain and hence appending to a new list of contours, this list is redrawn using drawContours method. This image is the final FCM segmented image.

[ ]:
```
def imclearborder(imgBW):
    # Given a black and white image, first find all of its contours
    radius = 2
    imgBWcopy = imgBW.copy()

    contours, hierarchy = cv2.findContours(imgBWcopy.copy(), cv2.
 ↪RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

    # Get dimensions of image
    imgRows = imgBW.shape[0]
    imgCols = imgBW.shape[1]
    contourList = [] # ID list of contours that touch the border

    # For each contour...
    for idx in np.arange(len(contours)):
        # Get the i'th contour
        cnt = contours[idx]
        # Look at each point in the contour
        for pt in cnt:
            rowCnt = pt[0][1]
            colCnt = pt[0][0]
            # If this is within the radius of the border
            # this contour goes bye bye!
            check1 = (rowCnt >= 0 and rowCnt < radius) or (rowCnt
            >= imgRows - 1 - radius and rowCnt < imgRows)
            check2 = (colCnt >= 0 and colCnt < radius) or (colCnt >=
            imgCols - 1 - radius and colCnt < imgCols)

            if check1 or check2:
                contourList.append(idx)
                break

        for idx in contourList:
            cv2.drawContours(imgBWcopy, contours, idx, (0, 0, 0), -1)
    return imgBWcopy
```

[ ]:
```
def Thresholding(image_bcet):
    blur = cv2.GaussianBlur(image_bcet,(5,5),0)
    T, thresh_f = cv2.threshold(blur,200,255,   cv2.THRESH_BINARY)
    return thresh_f
```

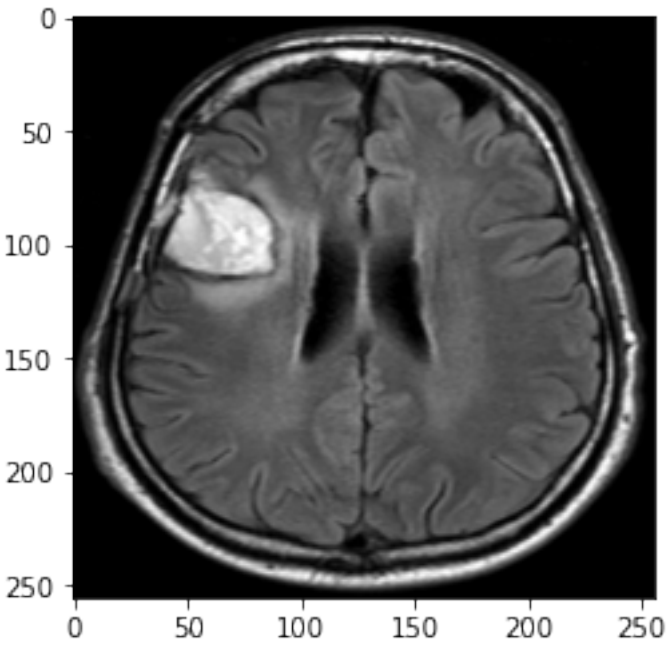
[ ]:
```
def segmentation(fcm_image,ths_image):
    brain = fcm_image
```

```
    tumor = ths_image
    segimg = cv2.addWeighted(brain, 0.5, tumor, 0.7, 0)
    return segimg
```

[ ]:
```
def dataanalysis(seg_img):
    try:
        detected_edges = cv2.Canny(seg_img, 10, 10 * 3, 5)
        colour = cv2.applyColorMap(seg_img, cv2.COLORMAP_JET)
        return detected_edges
    except Exception as e:
        print("Error=" + e.args[0])
        tb = sys.exc_info()[2]
        print(tb.tb_lineno)
```

[ ]:
```
# Main
# Image Acquisition
image = cv2.imread('images/Sample16.jpg')
image = cv2.resize(image,(256,256),cv2.INTER_AREA)
image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
plt.imshow(image,cmap='gray')
```

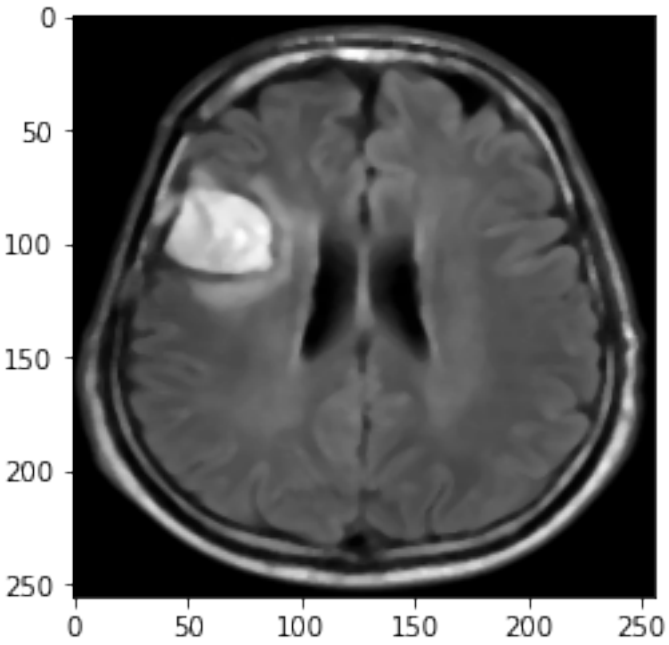
[ ]:
```
<matplotlib.image.AxesImage at 0x213442859a0>
```



[ ]:
```
# AMF image preprocessing
img_median_sigmoid = AdaptiveMmedianFilter(image)
img_median = cv2.medianBlur(image, 5)
cv2.imwrite('C:/Users/Poorya/Desktop/AMF.jpg', img_median_sigmoid)
```

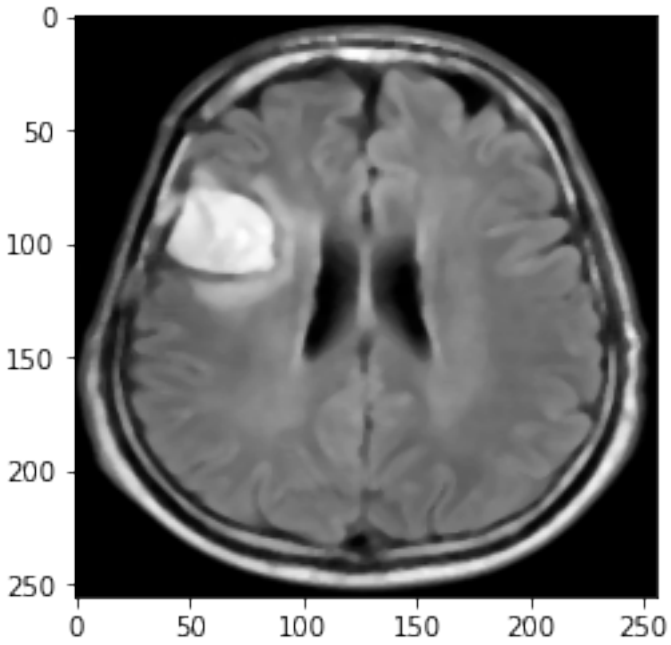
```
plt.imshow(img_median,cmap='gray')
```

[ ]: <matplotlib.image.AxesImage at 0x21344ddedc0>



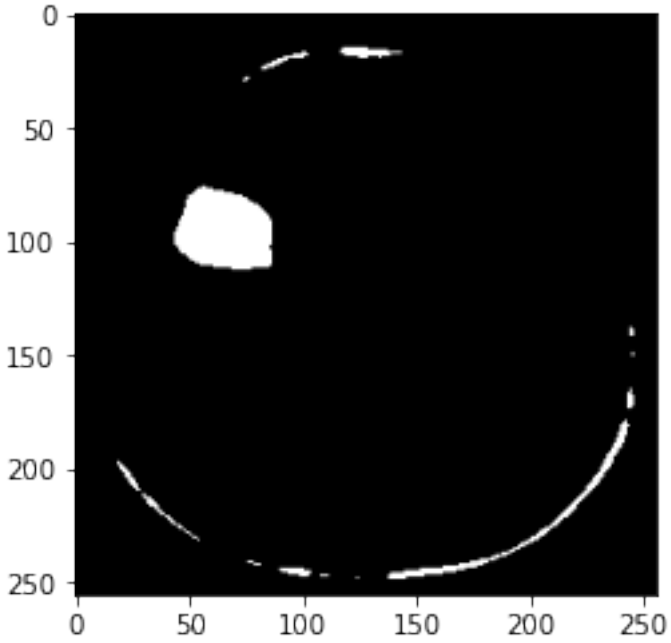
[ ]:
```
# BCET image preprocessing
image_bcet = BalanceContrastEnhancementTechnique(img_median)
cv2.imwrite('C:/Users/Poorya/Desktop/img_bcet.jpg', image_bcet)
plt.imshow(image_bcet,cmap='gray')
```

[ ]: <matplotlib.image.AxesImage at 0x21344ddad60>



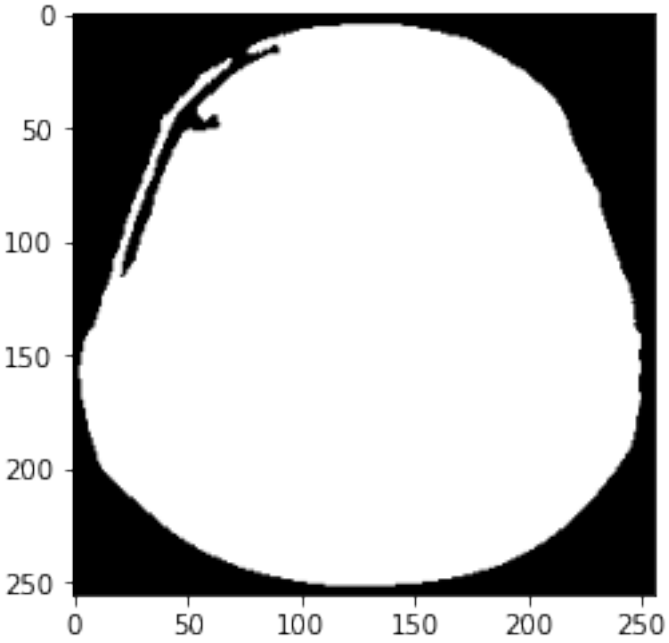
[ ]:
```
# Segmentation thresholding for tumor region of brain
thresh_f = Thresholding(image_bcet)
plt.imshow(thresh_f,cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x213453f6d30>
```



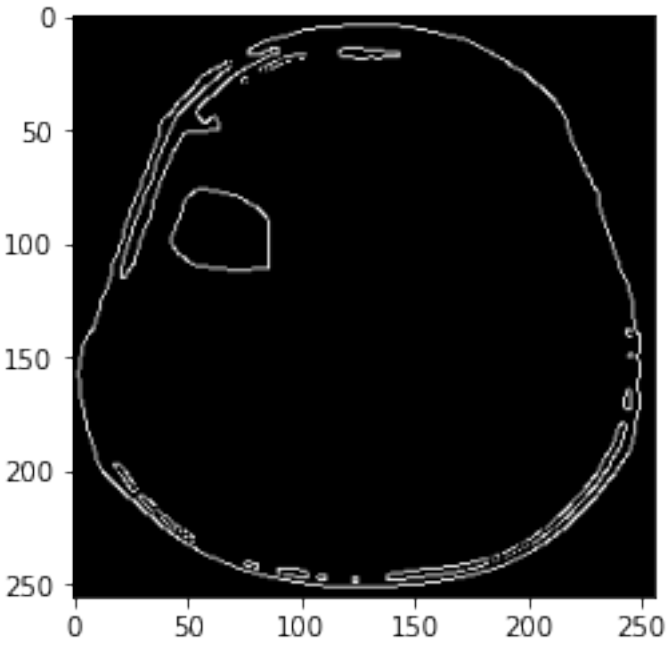
```
[ ]: # FCM segmentation for normal region of brain
     image_FCM = FCM('img_bcet.jpg')
     plt.imshow(image_FCM,cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x213440da850>
```



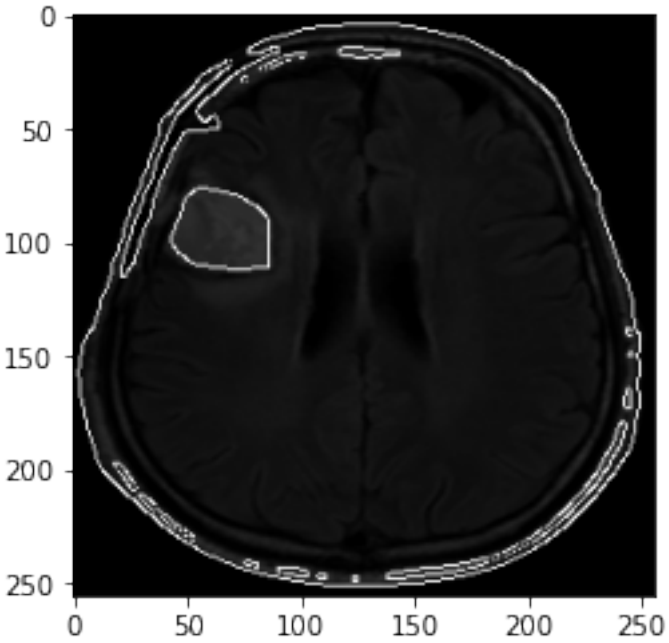

```
[ ]: seg_img=segmentation(image_FCM,thresh_f)
     edges=dataanalysis(seg_img)
     plt.imshow(edges,cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x21344fc6b80>
```



[ ]:
```
segimg2 = cv2.addWeighted(edges, 1, image, 0.2, 0)
plt.imshow(segimg2,cmap='gray')
```

[ ]:
```
<matplotlib.image.AxesImage at 0x213444330a0>
```



5.1.8 Discussion

To identify the real process of this system Figure 5.3 shows an example for segmenting a tumor in the brain.

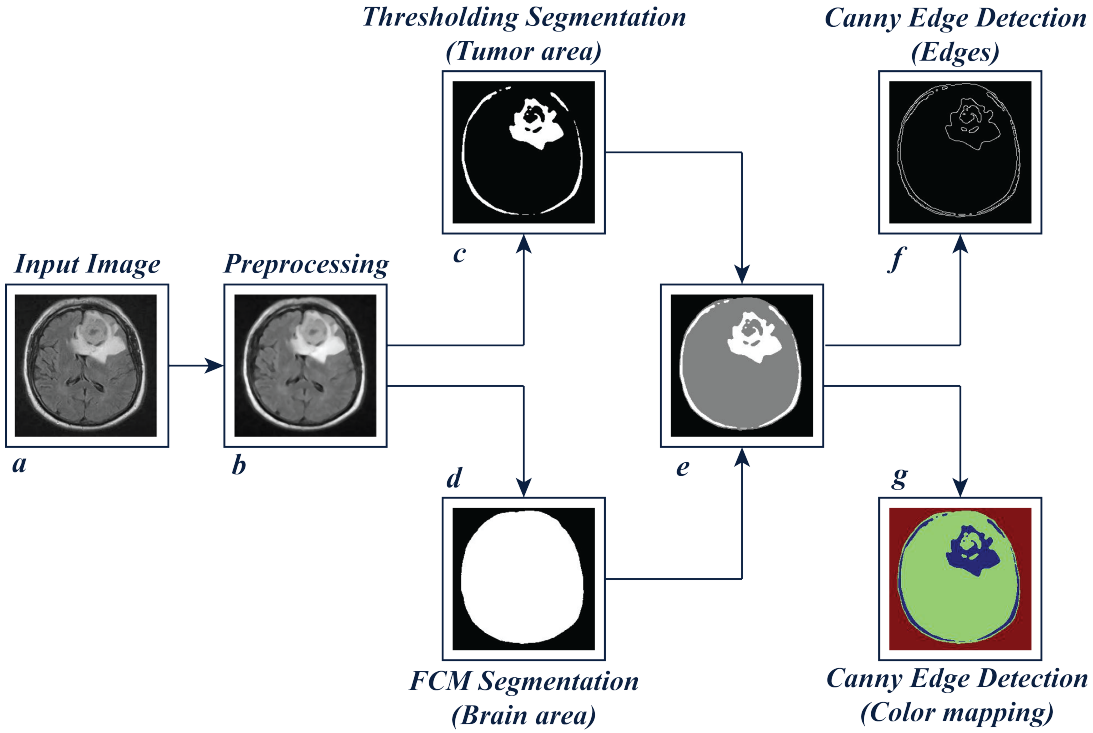


Figure 5.3: example of the system. a)input image, b)preprocessing image, c)result after thresholding, d)result after FCM clustering, e)combine both segmented images, f)final edge map, g)combined image with the color map

This tumor segmentation system has its advantages and disadvantages. First, in cases with tumors, it has a very accurate segmentation. According to the source article, it has 97% accuracy in detecting tumors. Also, it is simple to understand and implement due to using basic image processing and segmentation approaches. But this simplicity causes some drawbacks for this system. In MRI, tumor tissues have a similar appearance with some organs like eyes. So, usually, basic thresholding segments these normal organs as a tumor. Also, in no tumor cases, the system detects a very large brain area as the tumor.

Chapter 6

Conclusion

This research project based on learning medical image analysis with some theoretical and practical researches and activities. This project divided to 3 main stages, the first stage was introducing concepts and approaches; we covered medical imaging modalities, image processing concepts and recent improvements in segmentation and Denoising. In the next step, we examined these concepts in practice; we provide a handy tutorial for python and OpenCV library which contains various examples for processing medical images. And, in the final step a real segmentation system was implemented. This system uses two different segmentation techniques (Fuzzy C-means and Thresholding) and then combine results. We discussed about its benefits and drawbacks. As a future work, we try to use a more complicated segmentation technique instead of simple thresholding to enhance the segmentation performance.

Chapter 7

References

[1] Lina Sun, Rajiv Kumar Gupta, and Amit Sharma. Review and potential for artificial intelligence in healthcare. *International Journal of System Assurance Engineering and Management*, pages 1–9, 2021.

[2] Mark A Haidekker. Medical imaging technology, 2013.

[3] Avinash C Kak and Malcolm Slaney. *Principles of computerized tomographic imaging*. SIAM, 2001.

[4] Murtaza Ali, Dave Magee, and Udayan Dasgupta. Signal processing overview of ultrasound systems for medical imaging. *SPRAB12, Texas Instruments, Texas*, 2008.

[5] Thomas M Deserno. Fundamentals of medical image processing. In *Springer Handbook of Medical Technology*, pages 1139–1165. Springer, 2011.

[6] Isaac Bankman. *Handbook of medical image processing and analysis*. Elsevier, 2008.

[7] Karol Miller. *Biomechanics of the Brain*. Springer, 2011.

[8] Brian H Brown, Rod H Smallwood, David C Barber, PV Lawford, and DR Hose. *Medical Physics and Biomedical Engineering: Medical Science Series*. CRC Press, 2017.

[9] Klaus D Toennies. *Guide to medical image analysis*. Springer, 2017.

[10] Leo Grady. Random walks for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 28(11):1768–1783, 2006.

[11] Sreedhar Kollem, K Rama Linga Reddy, and D Sreenivasa Rao. Image denoising by using modified sghp algorithm. *International Journal of Electrical and Computer Engineering*, 8(2):971, 2018.

[12] Jin Liu, Min Li, Jianxin Wang, Fangxiang Wu, Tianming Liu, and Yi Pan. A survey of mri-based brain tumor segmentation methods. *Tsinghua science and technology*, 19(6):578–595, 2014.

[13] Muhammad Aksam Iftikhar, Abdul Jalil, Saima Rathore, and Mutawarra Hussain. Robust brain mri denoising and segmentation using enhanced non-local means algorithm. *International journal of imaging systems and technology*, 24(1):52–66, 2014.

[14] Saritha Saladi and N Amutha Prabha. Analysis of denoising filters on mri brain images. *International Journal of Imaging Systems and Technology*, 27(3):201–208, 2017.

[15] Yousif Hamad, Konstantin Simonov, and Mohammad Naeem. Detection of brain tumor in mri images, using a combination of fuzzy c-means and thresholding. *International Journal of Advanced Pervasive and Ubiquitous Computing*, 11:45–60, 01 2019.

[16] Durga Prasad Bavirisetti, Vijayakumar Kollu, Xiao Gang, and Ravindra Dhuli. Fusion of mri and ct images using guided image filter and image statistics. *International journal of Imaging systems and Technology*, 27(3):227–237, 2017.

[17] Yilin Liu, Huiqian Du, Zexian Wang, and Wenbo Mei. Convex mr brain image reconstruction via non-convex total variation minimization. *International Journal of Imaging Systems and Technology*, 28(4):246–253, 2018.

[18] Kinam Kwon, Dongchan Kim, and HyunWook Park. Multi-contrast mr image denoising for parallel imaging using multilayer perceptron. *International Journal of Imaging Systems and Technology*, 26(1):65–75, 2016.

[19] Geng Chen, Pei Zhang, Yafeng Wu, Dinggang Shen, and Pew-Thian Yap. Denoising magnetic resonance images using collaborative non-local means. *Neurocomputing*, 177:215–227, 2016.

[20] J Fenshia Singh and V Magudeeswaran. A machine learning approach for brain image enhancement and segmentation. *International Journal of Imaging Systems and Technology*, 27(4):311–316, 2017.

[21] Hossein Khodabakhshi Rafsanjani, Mohammad Hossein Sedaaghi, and Saeid Saryazdi. Efficient diffusion coefficient for image denoising. *Computers & Mathematics with Applications*, 72(4):893–903, 2016.

[22] Changjiang Zhang, Yuan Chen, Chunjiang Duanmu, and Yinhuan Yang. Image denoising by using pde and gcv in tetrolet transform domain. *Engineering Applications of Artificial Intelligence*, 48:204–229, 2016.

[23] Kui Liu, Jieqing Tan, and Liefu Ai. Hybrid regularizers-based adaptive anisotropic diffusion for image denoising. *SpringerPlus*, 5(1):1–24, 2016.

[24] Saravanan Alagarsamy, Kartheeban Kamatchi, Vishnuvarthanan Govindaraj, and Arunprasath Thiyagarajan. A fully automated hybrid methodology using c uckoo-based fuzzy clustering technique for magnetic resonance brain image segmentation. *International journal of Imaging systems and technology*, 27(4):317–332, 2017.

[25] Yuichiro Anzai. *Pattern recognition and machine learning*. Elsevier, 2012.

[26] Geng-Cheng Lin, Wen-June Wang, Chung-Chia Kang, and Chuin-Mu Wang. Multispectral mr images segmentation based on fuzzy knowledge and modified seeded region growing. *Magnetic resonance imaging*, 30(2):230–246, 2012.

[27] Koon-Pong Wong. Medical image segmentation: methods and applications in functional imaging. In *Handbook of biomedical image analysis*, pages 111–182. Springer, 2005.

[28] Tom Brosch, Youngjin Yoo, Lisa YW Tang, David KB Li, Anthony Traboulsee, and Roger Tam. Deep convolutional encoder networks for multiple sclerosis lesion segmentation. In *International conference on medical image computing and computer-assisted intervention*, pages 3–11. Springer, 2015.

[29] Hao Chen, Yefeng Zheng, Jin-Hyeong Park, Pheng-Ann Heng, and S Kevin Zhou. Iterative multi-domain regularized deep learning for anatomical structure detection and segmentation from ultrasound images. In *International Conference on Medical image computing and computer-assisted intervention*, pages 487–495. Springer, 2016.

[30] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31, 2017.

[31] Heba Mohsen, El-Sayed A El-Dahshan, El-Sayed M El-Horbaty, and Abdel-Badeeh M Salem. Classification using deep learning neural networks for brain tumors. *Future Computing and Informatics Journal*, 3(1):68–71, 2018.

[32] Liu Jian Guo. Balance contrast enhancement technique and its application in image colour composition. *Remote Sensing*, 12(10):2133–2151, 1991.

[33] Coursera Online Certificates: Computer Vision Basics.

[34] Coursera Online Certificates: Image and Video Processing: From Mars to Hollywood with a Stop at the Hospital.

[35] Coursera Online Certificates: Introduction to Machine Learning.

[36] NPTEL: Medical Image Analysis Online course.

[37] Carl Zeiss Microscopy Online Campus.

[38] University of Tartu: Digital Image Processing Course.

[39] Open Source Computer Vision (OpenCV) Tutorials.

[40] Udemy Online Courses: Python for Computer Vision with OpenCV and Deep Learning.

[41] Emerj Artificial Intelligence Research: What is Machine Learning?

[42] Expert.ai Platform: machine learning definition.

[43] Research HUB: Fuzzy c-Means.

[44] Image Processing documents and codes: Balance Contrast Enhancement Technique.

[45] Medium articles: Median Filtering with Python and OpenCV.

[46] Medium articles: Otsu thresholding(image binarization).

[47] SciKit-Fuzzy Python Library Documentation and Tutorials.

[48] Kaggle Datasets: Brain MRI Images for Brain Tumor Detection.

[49] Kaggle Datasets: Alzheimers Brain MRI.

[50] Kaggle Datasets: Mosmed COVID-19 CT Scans (1000 CT scans of healthy and COVID-19 confirmed patients.).

[51] Mathworks Answers: Why an image is converted to double using I=im2double(I); in matlab?

[52] Towards Data Science: Image Segmentation.

[53] Machine Vision Market Size, Share Trends Analysis Report.