BRAIN TUMOR DETECTION USING HOG AND SVM

A PROJECT REPORT

Submitted by

KARTIK SINGH [Reg No:RA2011003010810]

UTKARSH GUPTA [Reg No: RA2011003010772]

Under the Guidance of

Mrs. P. Renukadevi

Assistant Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that 18CSP107L project report titled "**Brain Tumor Detection Using HOG and SVM" is the bonafide work of KARTIK SINGH [RA2011003010810] and UTKARSH GUPTA [RA2011003010772]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Mrs. P Renukadevi
PROJECT GUIDE
Assistant Professor
Department of Computing Technologies

Dr. M. Baskar
PANEL HEAD
Associate Professor
Department of Computing Technologies

Dr. M. PUSHPALATHA
HEAD OF THE DEPARTMENT
Department of Computing Technologies

Department of Computing Technologies

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/Course : B.Tech in Computer Science and Engineering

Student Names : **KARTIK SINGH, UTKARSH GUPTA**

Registration Number : RA2011003010810, RA2011003010772

Title of Work : Brain Tumor Detection Using HOG and SVM

We here by certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web,etc.)
- Given the sources of all pictures, data etc that are not my own.
- Not made any use of the report(s) or essay(s) of any other student(s)either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course hand book / University website

We understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
We are aware of and understand the University's policy on Academic misconduct and plagiarism and we certify that this assessment is our own work, except where indicated by referring, and that we have followed the good academic practices noted above. **Student 1 Signature:** **Student 2 Signature:** **Date:**
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr T. V. Gopal,** Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department**, Dr. M. Pushpalatha,** Professor, Department of Computing Technology, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators: **Dr. S. Godfrey Winster**, Associate Professor, Project Panel Head, **Dr. M. Baskar**, Associate Professor, Project Panel Members: **Mrs. R. Brindha** , Assistant Professor, **Mrs. P. Renukadevi** , Assistant Professor, **Dr. J. Jayapradha**, Assistant Professor, **Dr. M. K. Vidhyalakshmi**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor**, Mrs P. Renuka Devi ,** Assistant Professor**,** Department of Computing Technologies, SRM Institute of Science and Technology and **Mr. M Senthilraja**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide **Mrs. P. Renukadevi** Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology for providing us with an opportunity to pursue our project under her mentorship . She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Computing Technologies Department, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

KARTIK SINGH [RA2011003010810]

UTKARSH GUPTA [RA2011003010772]

Abstract

Detecting brain tumors in medical images is a formidable task due to variations in tumor size, shape, and location among different patients. Precise identification of these attributes is crucial for effective detection, Various algorithms have been employed to tackle the challenge of tumor detection and extraction from medical images. These algorithms have utilized techniques such as a hybrid approach involving Support Vector Machine (SVM), back propagation, and the dice coefficient. Notably, the algorithm using back propagation as a base classifier achieved the highest accuracy, reaching an impressive 90%. In our study, we perform feature extraction on medical images of patients' tumors from a comprehensive database using the Histogram of Oriented Gradient method. Subsequently, these images are classified into two categories: tumor and non-tumor, through the use of SVM. The detection of brain tumors within patients' images is accomplished by assessing the performance of the SVM model based on Receiver Operating Characteristics (ROC). ROC analysis incorporates crucial metrics, including true positive rate, true negative rate, false positive rate, and false negative rate. Within the HG image data folder in vector format, the SVM model achieved an impressive accuracy of 97% for the 95th slice of the T1 modality, with an exceptionally high true positive rate of 0.97, surpassing other modalities. Furthermore, in another instance, the SVM model exhibited an accuracy of 87% for the 135th slice of the T1 modality, boasting a high true positive rate of 0.8 and a low false positive rate of 0.06 when compared to other image data folders within the HG category. For the LG image data folder, the SVM model demonstrated an accuracy of 62% for the 90th slice of the FLAIR modality, featuring a notable high true positive rate of 0.5 and a relatively low false positive rate of 0.25 when compared to other modalities. In the case of synthetic data folders for both HG and LG, the SVM model consistently provided an accuracy of 62% for the 100th slice of the FLAIR modality, offering a high true positive rate of 0.5 and a low false positive rate of 0.06 across all other modalities. This research underscores the significance of SVM-based brain tumor detection in medical images and highlights its impressive performance in terms of accuracy and true positive rates across various modalities and tumor types. These findings hold substantial promise for improving the diagnosis and treatment of brain tumors in clinical practice.

TABLE OF CONTENTS

	ABSTRACT	**v**
	LIST OF TABLES	**viii**
	LIST OF ABBREVIATIONS	**ix**
	LIST OF FIGURES	**x**
1	**INTRODUCTION**	**1**
	1.1 Aim and Objective	3
	1.2 Thesis Organization	3
2	**LITERATURE SURVEY**	**4**
3	**ARCHITECTURE DESIGN AND ANALYSIS**	**6**
	3.1 Physics of MRI	6
	3.2 Histogram Of Oriented Gradients(HOG)	10
	3.3 SVM Over Neural Networks	14
	3.4 Evaluation	20
4	**IMPLEMENTATION**	**22**
5	**RESULTS AND ANALYSIS**	**40**
	5.1 Results	40
	5.2 Analysis	41
6	**CONCLUSION**	**45**

7	**FUTURE WORK**	**46**
8	**REFERENCES**	**48**
	APPENDIX-1	**50**
	APPENDIX-2	**51**
	PLAGARISM REPORT	**67**
	PAPER PUBLICATION PROOF	**70**

LIST OF TABLES

I HISTOGRAMS OF BINS............................ 13

II RECEIVER OPERATING CHARACTERISTICS 21

LIST OF ABBREVIATIONS

BRats/BraTS	BRAIN TUMOR SEGMENTATION
FP	FALSE POSITIVE
FN	FALSE NEGATIVE
HOG	HISTOGRAM OF ORIENTED GRADIENT
ID	IMAGE DATA
MRI	MAGNETIC RESONANCE IMAGE
NN	NEURAL NETWORKS
RF	RADIO FREQUENCY
ROC	RECEIVER OPERATING CHARACTERISTICS
SD	SYNTHETIC DATA
SVM	SUPPORT VECTOR MACHINE
TP	TRUE POSITIVE
TN	TRUE NEGATIVE

LIST OF FIGURES

3.1.1 MRI SCAN 6

3.1.2 SYSTEM OF CSF 7

3.1.3 T1-WEIGHTED IMAGE....................................8

3.1.4 T2-WEIGHTED IMAGE....................................8

3.1.5 FLAIR IMAGE 9

3.1.6 T1C-WEIGHTED IMAGE 9

3.1.7 IMAGES SHOWING DIFFERENT MODALITIES OF SAME SLICE................10

3.1.8 COMPARISON OF T1 VS T2....................................10

3.2.1 DIVISION OF IMAGE INTO CELL 22

3.2.2 PIXEL VALUES IN ZOOMED VERSION OF A CELL 22

3.2.3 ORIENTATION OF PIXELS IN CELLS....................................23

3.2.4 BINS IN HISTOGRAM....................................23

3.2.5 BLOCK REPRESENTATION....................................24

3.3.1 DATASET CLASSIFICATION BASED ON HYPERPLANE............................25

3.3.2 MARGIN....................................26

3.3.3 KERNELLING....................................26

4. 1 IMAGE WITH ALL INDEXES....................................32

4.2 IMAGE WITH 0, 1, 2 INDEXES 32

4.3 SLICE OF BRAIN TUMOUR IMAGE WITH NO INFORMATION.................35

4.4 SLICE OF BRAIN TUMOUR IMAGE, A SECTION OF BRAIN......................35

Chapter 1
INTRODUCTION

The task of detecting brain tumors from medical images has long been a formidable challenge. The human brain, a vital organ, orchestrates the body's every function. Unfortunately, it is susceptible to various diseases, including infections, strokes, and tumors. A brain tumor represents an abnormal growth or mass of cells within the brain, which can be either cancerous or non-cancerous.

Brain tumors are typically categorized as primary or secondary. Primary brain tumors originate within the brain itself and can be either malignant, containing cancerous cells, or benign, lacking cancerous cells. In contrast, secondary or metastatic brain tumors occur when cancerous cells from elsewhere in the body migrate to the brain and proliferate. These tumors are often assessed through image-based diagnostic techniques, employing X-rays, powerful magnetic resonance imaging, or radioactive substances to generate detailed brain images.

Various types of scans play a crucial role in diagnosing brain diseases, with MRI, CT, PET being the most commonly used methods. These imaging techniques yield highly informative images that offer valuable insights into the presence and precise location of brain tumors.

COMPUTED TOMOGRAPHY (CT) SCAN:

A CT scan is a diagnostic procedure that harnesses X-rays for creating a 3-Dimentional representation of brain. Unlike traditional X-rays, which capture single images, a CT scan involves the acquisition of multiple images. During the scan, the CT scanner orbits around the patient as they recline on a table. Subsequently, these numerous images are amalgamated to produce cross-sectional images of the body. CT scans provide valuable data regarding the size of the tumor and any potential damage to the cranial bones.

CT ANGIOGRAM (CTA):

CTA utilizes a series of X-rays to reveal the brain's arteries and blood vessels. It involves injecting contrast material via an IV line while within the CT scanner. This scan is invaluable for surgical planning, offering detailed information on the blood vessels surrounding a tumor [4].

MYELOGRAM:

A Myelogram employs contrast dye to ascertain whether a tumor has extended into adjacent regions of the body, such as the spinal cord. This procedure involves injecting dye into the Cerebrospinal Fluid (CSF).

POSITRON EMISSION TOMOGRAPHY (PET) SCAN:

During this test, a substance called FDG is introduced in the bloodstream. Only a small quantity of FDG is administered, and it naturally exits the body within a day or so. Tumor cells exhibit rapid growth, resulting in their increased absorption of this sugar compared to most other cells. Approximately an hour later, the patient is positioned on a table within the PET scanner. A specialized camera then generates an image highlighting areas of radioactivity within the patient's body. While PET scans may not offer intricate details, they do provide crucial information regarding whether the irregularities detected in other tests are indicative of tumors. PET scans are frequently used in conjunction with CT scans to assess whether a tumor has been successfully treated or has reappeared after treatment. Typically, the PET--CT scan is conducted initially as part of the treatment process.

MAGNETIC RESONANCE IMAGING (MRI) SCAN:

MRI, considered a fundamental diagnostic imaging technique, distinguishes itself among various scan types by utilizing radio waves and powerful magnetic fields in lieu of X-rays. During the procedure, the energy from radio waves is absorbed and subsequently re-emitted in a manner dictated by the specific body tissue and the presence of certain diseases. This intricate pattern is then translated into a highly detailed image of specific body regions through the use of a computer. For enhanced clarity, a contrast material called gadolinium, administered intravenously before the scan, is used to provide better details [4].

ADVANTAGES OF MRI SCANNED IMAGES

- MRI is a non-invasive imaging method.
- MRI is a cost-effective option.
- MRI offers excellent tumor contrasts within the brain.
- The total acquisition time for a full-body MRI is shorter when compared to PET and X-ray.

1.1 AIM AND OBJECTIVE

The primary goal of this thesis is to identify the presence of brain tumors within medical images. The principal objective is the accurate classification of these tumors and non-tumorous regions through the application of Support Vector Machine (SVM). This classification is facilitated by the utilization of the feature extraction algorithm known as HOG, or Histogram of Oriented Gradients.

Our specific objectives encompass the following steps:

a) The selection of pertinent MRI brain images containing tumors from a database in need of enhancement.

b) The extraction of distinctive features characterizing the brain tumors through the HOG feature extraction algorithm.

c) Utilizing the extracted feature vectors as input for SVM to execute the classification of feature vectors.

d) The assessment of SVM's performance by means of Receiver Operating Characteristics (ROC) to ensure reliable tumor detection and classification.

1.2 THESIS ORGANIZATION

In the opening section, we provide a broad overview of this thesis, offering an introduction, outlining our primary objectives, and highlighting the research questions we aim to address. The subsequent segment discusses prior research endeavors related to brain tumor detection. We then delve into a comprehensive exploration of two key components, namely the Histogram of Oriented Gradients and the Support Vector Machine. Further details are provided on the implementation of the feature vector alongside the SVM classifier. Subsequently, we present the outcomes resulting from our analysis. In the final section, we conclude our thesis and also outline potential future directions for improving brain tumor detection methods.

Chapter 2
LITERATURE REVIEW

Numerous studies have been conducted in the field of brain tumor detection. In [2], authors proposed a comprehensive approach to brain tumor detection. Their method involved distinguishing between healthy brains and those with tumors, benign tumors versus malignant tumors, and utilized an algorithmic process comprising seven stages. The phases encompassed image pre-processing, image segmentation, feature extraction, and the application of neural network techniques for image classification. The technique, adaptive thresholding and Harris, showed a 15.625% false recognition rate for distinguishing healthy brains from those with tumors and a 6.25% rate for distinguishing benign from malignant tumors.

The research paper proposed a textural feature extraction algorithm based on kurtosis wavelet modeling for brain tumor extraction and detection. Their approach improved image segmentation quality and reduced the feature set size. Additionally, they combined the algorithm with Canny edge detection, significantly enhancing the quality of segmented images.
The research paper introduced a comparative method for identifying brain tumors, using image mining such as Gray Level Co-Occurrence (GLCM), intensity-based histogram features, and intensity-based features. These techniques were applied to the BraTS database, showing good accuracy, with GLCM achieving an accuracy of 95.25%

The research paper introduced developing a GUI-based algorithm that identifies, extracts, and identifies lesions. The algorithm primarily employs the a method for creating a 2D image to a 3D image, demonstrating precise and effective tumor detection. The method was also compared with MRI segmentation, achieving better outcomes in lesion assessment.
The research paper proposed a method for brain tumor detection and classification using Gwavelet and Probabilistic Neural Network. The method involves wavelet transform for feature extraction and classified tumors as malignant, metastatic, or benign using PNN, achieving a high classification accuracy.

The research paper introduced a four-step approach for brain tumor classification, which included pre-processing, segmentation, feature extraction, and classification. They employed a novel technique called TANN for tumor classification. TANN was applied to BraTS, OASTS, and NBTR datasets, resulting in high detection rates and fast classification speeds, particularly in the case of the BRATS dataset.

The research paper introduced segmentation techniques such as mean shift, thresholding by histogram through transform, and Support Vector Machine on both PET and MRI images. The results indicated that for MRI images, SVM and ThH were effective in tumor detection, while for PET images, SVM provided better results based on parameters like PSNR, Jaccard Index, dice Index, and GCE.

In a study by the authors referenced as, they introduced a hybrid method designed for the detection and classification of brain tumors. This hybrid approach involved skull detection, feature extraction through the utilization of the gray-level co-occurrence matrix (GLCM), and the application of the least square Support Vector Machine (SVM) for tumor classification. Notably, the SVM demonstrated an accuracy of 5.6%, which was compared against the performance of the Radial Basis Function (RBF) and Backpropagation (BP) techniques. The authors introduced a neural network-based technique for brain tumor detection. Their method encompassed MRI image denoising, the application of adaptive thresholding methods, image segmentation utilizing Canny edge detection, and classification employing backpropagation as the foundational classifier. This approach yielded an impressive classification accuracy of 90%, surpassing conventional methods.

The authors presented a two-step strategy for brain tumor detection and classification. They employed feature extraction with the Local Binary Pattern Co-Occurrence Matrix (LCM) and classification using the k-nearest neighbor (K-NN) algorithm. The results indicated that K-NN achieved the highest accuracy at 96.15%, outperforming other classifiers such as backpropagation neural network, radial basis function, discrete wavelet transform, and Principal Component Analysis artificial neural network.

This extensive work is primarily focused on improving brain tumor detection and classification, incorporating various techniques and algorithms for more accurate and efficient results. The methods described in these papers offer valuable insights into the ever-advancing field of medical image analysis and tumor diagnosis.

Chapter 3
ARCHITECTURE AND ANALYSIS

3.1 PHYSICS OF MRI

Magnetic Resonance Imaging (MRI) functions by exploiting the inherent property of atomic nuclei magnetization. It commences with the application of a powerful and uniform external magnetic field to the target tissue, aligning the protons present in the water nuclei within. These protons, initially in random orientations, undergo a process termed "magnetization" [15]. Magnetization is temporarily disrupted by the influence of external Radio Frequency (RF) energy, prompting the nuclei's relaxation processes and their return to their original alignments, accompanied by the emission of RF energy. Following a specific time interval, the emitted signals are recorded. These signals contain frequency information, which is subsequently transformed into corresponding intensity levels using Fourier Transform and presented in various shades of grey within a pixel matrix. Diverse types of images are created by altering the sequence of applied and collected RF pulses [15].

Figure 3.1.1 : MRI SCAN

3.1.1 MRI RELATED DEFINITIONS

REPETITION TIME (TR):

The period between the emission of an excitation pulse and the subsequent pulse, indicating the time span between successive pulses [15].

TIME TO ECHO (TE):

The duration from the transmission of the RF excitation pulse to the culmination of the signal generated in the coil, representing the interval between RF pulse transmission and the reception of the echo signal [15].

LONGITUDINAL RELAXATION TIME:

The time constant utilized to determine the speed at which excited protons return to their equilibrium state, denoting the time required for the rotating protons to realign with the external magnetic field [15].

TRANSVERSE RELAXATION TIME (T2):

The time constant employed to assess whether the excited protons reach equilibrium or lose phase coherence with each other. This indicates the duration it takes for the spinning protons to lose phase coherence in the presence of nuclei that are spinning perpendicular to the primary magnetic field [15].

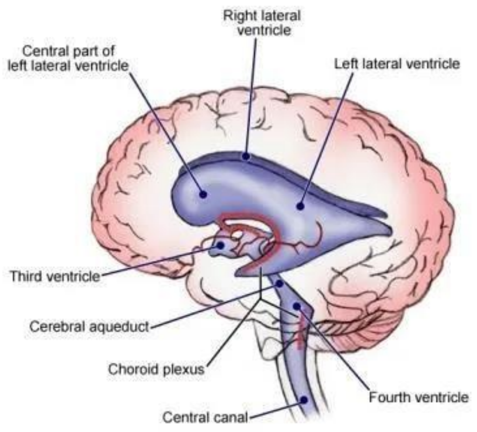


FIGURE 3.1.2: SYSTEM OF CSF

CEREBROSPINAL FLUID (CSF):

This is a transparent and colorless bodily fluid that resides within the brain [15].

GADOLINIUM (GAD):

Gadolinium (GAD) is a safe and non-toxic paramagnetic contrast enhancement agent. When administered during an MRI scan, it is used to improve and enhance the quality of the MRI image. GAD-enhanced images are particularly useful for the evaluation of vascular structures and the detection of disruptions in the blood-brain barrier [15].

3.1.2 MRI IMAGING SEQUENCE:

T1-WEIGHTED IMAGES:

T1-weighted images are generated using brief TE (Echo Time) and TR (Repetition Time) durations. The contrast and brightness of the image are dictated by the T1 properties of the tissues [15][16].

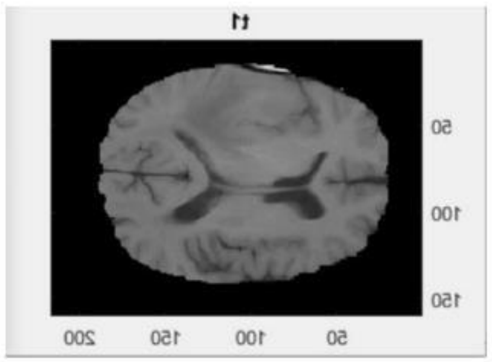


FIGURE 3.1.3: T1 -WEIGHTED IMAGE

T2-WEIGHTED IMAGES:

T2-weighted images are generated with extended TE (Echo Time) and TR (Repetition Time) durations, with the contrast and brightness of the image being influenced by the T2 properties of the tissues. Cerebrospinal fluid (CSF) serves as a distinguishing factor between T1-T2 weighted images, as it appears dark on T1-weighted images and bright on T2-weighted images [15][16].

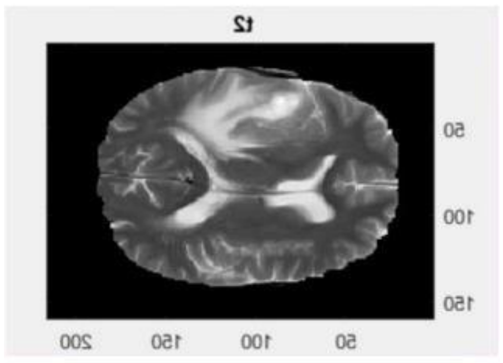


FIGURE 3.1.4: T2 -WEIGHTED IMAGE

FLUID ATTENUATED INVERSION RECOVERY (FLAIR):

FLAIR (Fluid-Attenuated Inversion Recovery) imaging shares similarities with T2-weighted images, but it employs significantly prolonged TE (Echo Time) and TR (Repetition Time) durations. This unique approach results in abnormalities retaining their brightness while normal cerebrospinal fluid (CSF) is suppressed and appears dark. FLAIR images excel in distinguishing between abnormalities and CSF, as they exhibit high sensitivity to pathological changes.Notably, FLAIR imaging is particularly adept at detecting small hyperintense lesions [15][16].

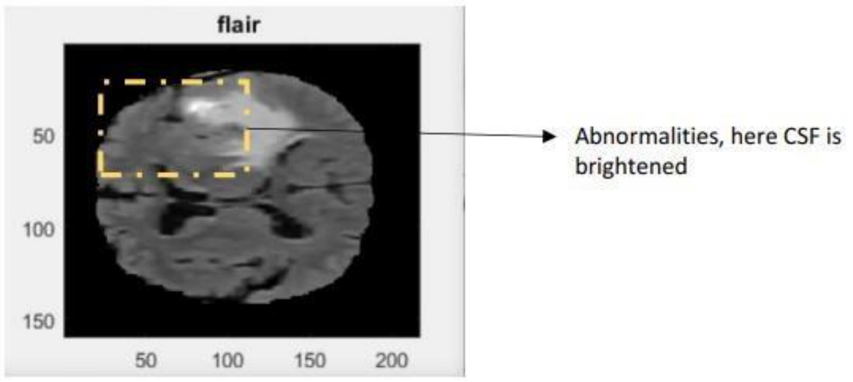


FIGURE 3.1.5: FLAIR IMAGE

3.1.2.1 T1-WEIGHTED POST-GADOLINIUM CONTRAST (PGC) IMAGES:

T1-weighted images can be acquired with the introduction of Gadolinium, which appears very bright in such images. Diverse RF pulse sequences are utilized to accentuate various tissue characteristics. To illustrate, in a "T1-weighted" scan, fluids are visualized as dark, while in a "T2-weighted" scan, fluids appear bright. Fat exhibits brightness in both types of scans.

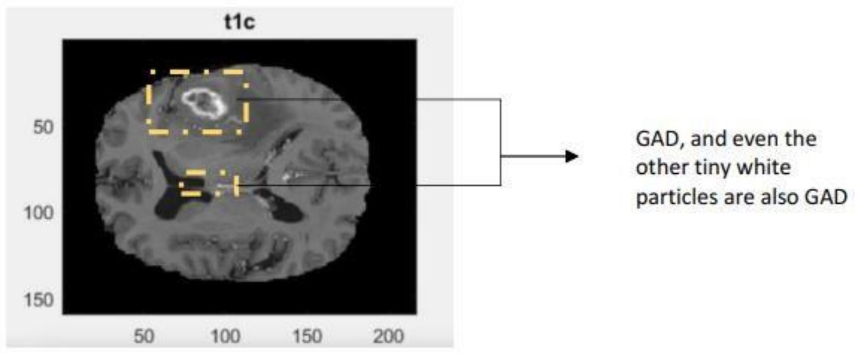


FIGURE 3.1.6: T1C-WEIGHTED IMAGE

Given that most pathological conditions tend to reduce fat content and increase water content, a comparative analysis of T1-weighted and T2-weighted scans, often referred to as an "irritation pattern," proves valuable in highlighting these changes [15].

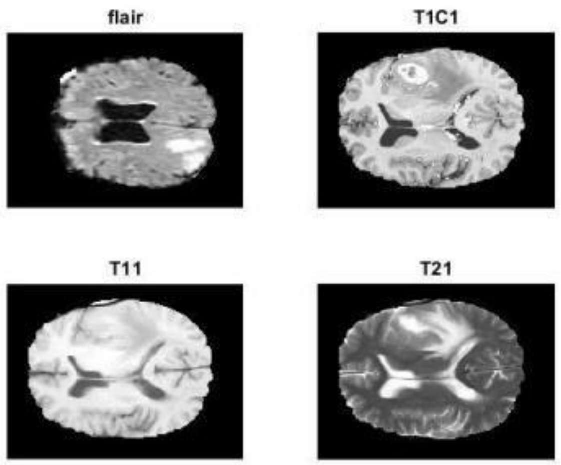


FIGURE 3.1.7: IMAGES DEPICTING VARIOUS MODALITIES OF THE SAME IMAGE SECTION

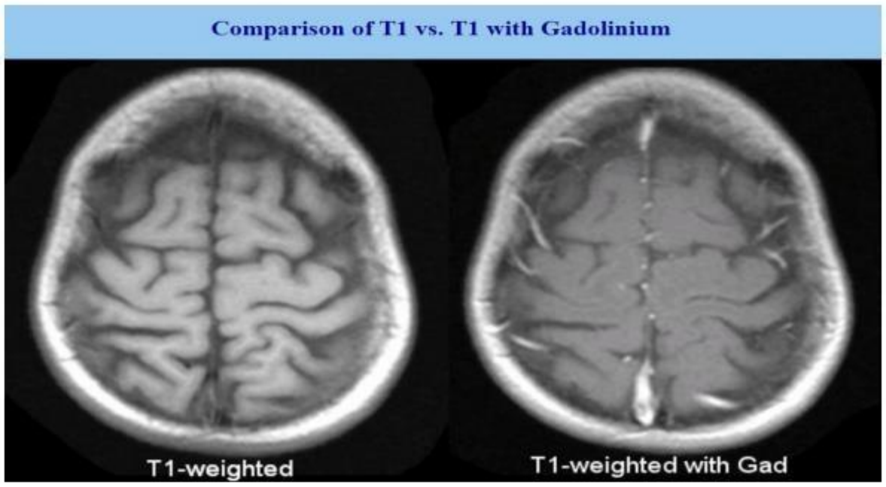


FIGURE 3.1.8: T1 VS T2

3.2 HISTOGRAM OF ORIENTED GRADIENTS (HOG)

The Histogram of Oriented Gradients (HOG) technique, initially introduced by Dalal and Triggs, was primarily designed for recognizing individuals within images. HOG serves as a feature descriptor employed in image processing with a specific focus on object detection. The primary objective of this feature descriptor is to provide a generalized representation of an object within an image, ensuring that the same feature descriptors can be extracted from images containing the object under various conditions, including different angles, lighting, distances, and more.

The HOG descriptor methodology involves tallying the occurrences of gradient orientations within localized segments of an image, typically within a designated detection window or region of interest (ROI).

1. HOG initially divides the images into cells. Cells can be either rectangular or radial.

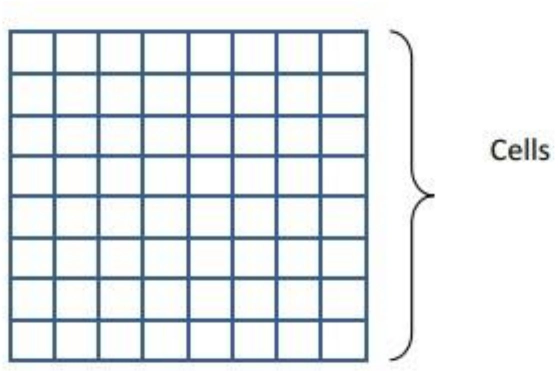
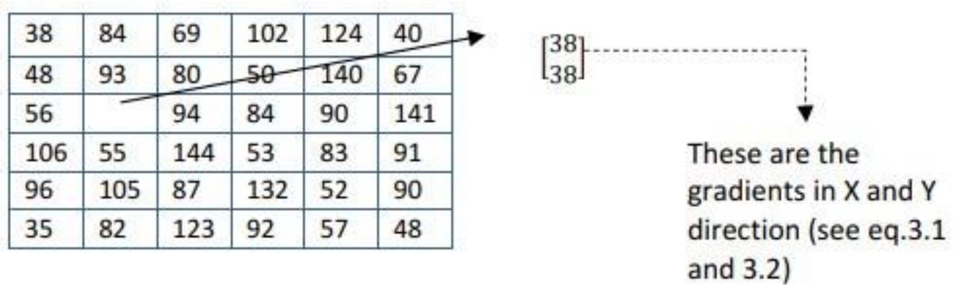


FIGURE 3.1.7: IMAGES DEPICTING VARIOUS MODALITIES OF THE SAME SLICE

2. For every pixel in the cell, gradient vector is calculated [17].



38	84	69	102	124	40
48	93	80	50	140	67
56		94	84	90	141
106	55	144	53	83	91
96	105	87	132	52	90
35	82	123	92	57	48

$\begin{bmatrix}38\\38\end{bmatrix}$

These are the gradients in X and Y direction (see eq.3.1 and 3.2)

FIGURE 3.1.7: IMAGES DEPICTING VARIOUS MODALITIES OF THE SAME IMAGE SECTION

3.2.1 GRADIENT VECTOR:

The gradient vector holds significant importance in the field of computer vision. Another term used to describe the gradient vector is "image gradient" [17].

3.2.1.1 APPLICATION OF GRADIENT VECTOR:

1. Edge detection
2. Feature extraction

The computation of the gradient vector at a specific pixel involves determining the disparities in neighborhood values along both the horizontal and vertical axes. Gradients can be computed by subtracting values from left to right or right to left along the horizontal axis, and from top to bottom or bottom to top along the vertical axis.

X-axis: 100-56=44 …… (3.1)

Y-axis: 98-55=43 ……. (3.2)

After calculating the gradient vectors for each and every pixel, magnitude is found out

Magnitude=$\sqrt{X2 + Y2}$ …(3.3)

=$\sqrt{382 + 382}$ …(3.4)

= 53.74 …(3.5)

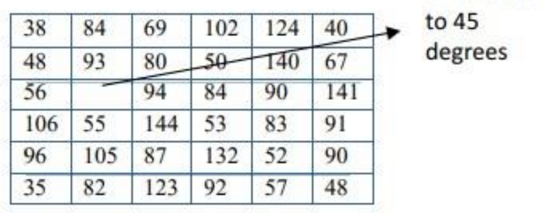
Orientation plays a key role in HOG. Orientation can be termed as a change in direction, in which pixel intensity value changes. The change in direction can be along X-axis or Y-axis. The orientation of a gradient vector is calculated as

Orientation =$arctan\ (YX\ )$ (3.6)

= $arctan$ (38 38) …(3.7)

= 0.785 $radians…$ (3.8)

=45 $degrees…$ (3.9)



38	84	69	102	124	40
48	93	80	50	140	67
56		94	84	90	141
106	55	144	53	83	91
96	105	87	132	52	90
35	82	123	92	57	48

FIGURE 3.2.3 PIXEL ORIENTATION

The orientation of each and every pixel depends on the value of the gradient vector. The gradients are placed into bins of the histogram as per its orientation.

10	20	30	40	50	60	70	80	90

I. HISTOGRAM OF BINS

3.2.2NORMALISATION:

In digital images, the magnitude of the gradient vector for individual pixels can exhibit variations. To standardize these values and ensure uniformity, a process known as normalization is applied. Normalization, in this context, involves dividing the vector values by their respective magnitudes. It's important to note that the act of normalization primarily impacts the magnitude component of the gradient vectors, while the orientation component remains unaffected [17].

3.2.3 CELL REPRESENTATION:

The image is partitioned into cells of $m \times n$ dimensions, with cell sizes available as 2×2, 4×4, 6×6, and 8×8 pixels. The primary rationale for dividing the image into cells is to achieve a more concise representation [18].

3.2.4 BLOCK NORMALISATION:

The foundation for both grouping and normalization of histograms lies in the process of aggregating cells into blocks. A block histogram can be defined as the result of normalizing these groups of cells. A key benefit of computing histograms based on image blocks is its ability to enhance the image's resilience to local fluctuations in illumination, making it more robust in varying lighting conditions.

3.2.5 FEATURE DESCRIPTOR

A feature descriptor is essentially a collection of block histograms that serves to represent an image or a specific region within an image by distilling its essential information. Its purpose is to simplify the intricate structure of the original image by capturing its most significant characteristics. In the context of Histogram of Oriented Gradients (HOG), this technique leverages the distribution of gradient orientations as the basis for extracting these features. This feature vector encodes the relevant information about the image or its designated region, making it more amenable to various computer vision and machine learning tasks.

3.3 SVM OVER NEURAL NETWORS

Support Vector Machines (SVMs) and shallow neural network architectures share some similarities, but SVMs lack the ability to incorporate hidden layers into their structure. SVMs also demand more rigorous feature engineering efforts in contrast to neural networks, which can automatically learn features to some extent.

Neural networks, on the other hand, exhibit a more intricate architecture with multiple layers, making them potentially more powerful. However, this complexity comes at a cost – they require substantial amounts of data for effective training, and the dataset used in this thesis is relatively limited.

Given the constraint of a small dataset, SVMs emerge as the more preferable choice. They offer a simpler and more straightforward solution, ensuring faster implementation and reduced costs, which align better with the available resources.

3.3.1 ADVANTAGES OF SVM

- **Effectiveness in High Dimensions:** SVMs excel in high-dimensional spaces, making them suitable for datasets with dimensionality greater than or equal to 10^6.
- **Efficiency through Subset Selection:** SVMs are highly efficient due to their utilization of a subset of training points. This selective approach contributes to faster decisionmaking.

- **Well-Suited for Smaller, Clean Datasets:** SVMs perform admirably on smaller datasets that are well-organized and free from excessive noise or outliers.
- **Memory Efficiency:** SVMs are memory-efficient because they store only the subsets of training points that are essential for the decision-making process. This minimizes memory usage when making predictions or classifying new data points.

3.3.2 DISADVANTAGES OF SVM

- **Challenges with Large Datasets:** SVMs are not well-suited for larger datasets, as they often require extended training times, making them less practical in such scenarios.
- **Diminished Performance in High-Dimensional Data:** When the number of features for each object surpasses the number of training data samples, SVM performance tends to degrade, as it struggles to generalize effectively.
- **Ineffectiveness in Noisy, Overlapping Datasets:** SVMs exhibit reduced effectiveness when applied to datasets with significant noise or where class boundaries overlap, making it harder to distinguish between classes accurately.
- **Absence of Direct Probabilistic Interpretation:** SVMs are non-probabilistic classifiers, as they separate objects based on their position relative to the hyperplane without providing a direct probability estimate for group membership. Instead, the distance from the decision boundary can serve as a measure of classification effectiveness.

3.3.3 USES OF SVM

- Text classification endeavors encompass category assignment, spam detection, and sentiment analysis.
- Challenges in the field of image recognition.
- Aspect-based recognition tasks.
- Classification based on color.
- Recognition of handwritten digits.

The Support Vector Machine, commonly referred to as SVM, is a supervised machine learning algorithm primarily employed for data classification tasks. SVM operates on the fundamental principle of identifying an optimal hyperplane to effectively separate the dataset. This is explained using figure 3.3.1.

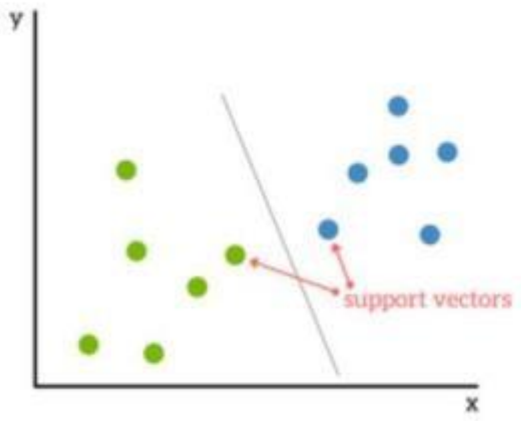


FIGURE 3.3.1 HYPERPLANE CLASSIFICATION

SVM constructs a finite-dimensional feature space, which essentially forms a vector space. In this vector space, each dimension corresponds to a specific feature of an object. The ensemble of these feature vectors collectively constitutes the vector space.

3.3.4 Hyper-parameter Tuning

Hyperparameters are the settings or configurations of a machine learning algorithm that cannot be learned from the data. They are manually defined by developers or researchers before training a model. Examples of hyperparameters include learning rate, batch size, the number of hidden layers, the number of neurons in each layer, regularization strength, and activation function. Hyperparameter tuning is the process of selecting the optimal values for these hyperparameters to enhance a model's performance on test data. It's necessary because different hyperparameter settings can significantly impact model performance. Incorrectly set hyperparameters may lead to poor model performance on test data. Here's why hyperparameter tuning is essential:

i. Enhancing model accuracy: The primary objective of hyperparameter tuning is to improve a model's accuracy on test data. Optimal hyperparameter values help the model generalize better to new data.

ii. Preventing overfitting: Overfitting occurs when a model is overly complex and fits the training data too closely, resulting in poor performance on the test data. Hyperparameter tuning allows for controlling model complexity and preventing overfitting.

iii. Reducing training time: Tuning hyperparameters can lead to faster convergence and reduce model training time.

iv. Improving interpretability: Some hyperparameters influence a model's interpretability. For instance, the regularization parameter controls model complexity, and increasing it results in a simpler, more interpretable model.

Here's how we've implemented hyperparameter tuning in our project:

i. Data preparation: We begin by cleaning, transforming, and splitting the data into training and testing sets.

ii. Algorithm selection: Various algorithms can be used, such as logistic regression, decision trees, random forests, and neural networks. Each algorithm has its own set of hyperparameters that can be tuned to improve performance.

iii. Hyperparameter selection: We identify the hyperparameters to tune. For instance, in a neural network, these may include the number of hidden layers, neurons per layer, learning rate, and regularization strength.

iv. Defining the search space: We specify the range of values that each hyperparameter can take. For example, we might search for the learning rate in the range of 0.001 to 0.1 and the number of hidden layers in the range of 1 to 5.

v. Hyperparameter tuning: We employ techniques like grid search, random search, or Bayesian optimization. In grid search, all possible combinations of hyperparameters are tested, and the best set is chosen based on validation set performance. Random search explores random hyperparameter combinations, and Bayesian optimization uses a probabilistic model to predict performance, selecting the best set based on these predictions.

3.3.5 SUPPORT VECTORS

Support vectors are those data points that maintain the closest proximity to the hyperplane. Their significance lies in their pivotal role in determining the precise placement of the dividing hyperplane within the dataset.

3.3.6 HYPERPLANE

The linear hyperplane, in the context of a high-dimensional feature space, represents a geometric entity that exists one dimension lower than the overall space. Its fundamental purpose is to partition the feature space into two distinct regions, as described in reference [20]. Hyperplanes are essentially planes that perform linear separation of a given dataset. For illustration, a straightforward example with only two features can be observed in Figure 3.3.2, as documented in reference [19].

In the context of categorization based on object features, newly introduced objects are positioned either above or below the hyperplane. It is this relative positioning that aids in their classification, as referenced in [19].

It's important to note that a linear separating hyperplane doesn't necessarily need to intersect the origin of the feature space, implying that zero vectors are not mandated as part of this plane. Such hyperplanes are technically termed as "affine." In formal mathematical terms, SVM is geared toward constructing these linear separating hyperplanes within high-dimensional vector spaces, as expounded in reference [20].

The accuracy of the classification hinges on the effectiveness of the data separation process. This judgment is based on an intuitive criterion: we aim for data points to maximize their distance from the hyperplane, all the while ensuring that they remain on the positive side. [20].

3.3.7 SEGREGATION USING THE DATA

The margin can be defined as the measure of the gap between the hyperplane and the nearest data point within either dataset. In the pursuit of optimal classification, the objective is to attain a substantial margin, where a larger margin is indicative of superior performance.

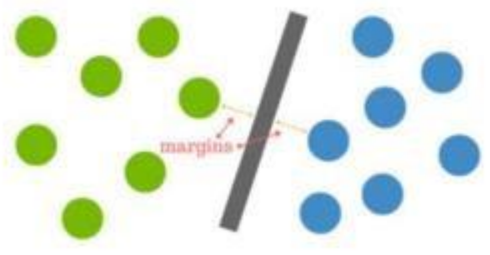


FIGURE 3.3.2 Margin

Real-world data isn't always as straightforward as the previous example illustrates. Situations often arise where datasets overlap, rendering them not linearly separable. In such cases, a practical approach involves transforming a 2D to 3D image viewer, a concept best understood by revisiting the earlier example. Picture all the balls being elevated into the air and then partitioned using an intervening sheet. This imaginative elevation of the balls corresponds to a mathematical operation known as "kernelization," wherein data is effectively mapped into higher dimensions to facilitate improved separation.

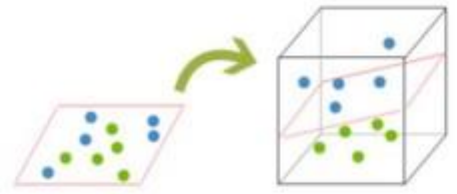


FIGURE 3.3.3 Kernelling

In the realm of three-dimensional images, the hyperplane takes the form of a sheet rather than a mere line. The overarching concept here is to continue mapping the data into higher dimensions until a hyperplane can be established for effective segregation. In a feature space denoted as $\mathbb{R}p$, with p dimensions, the hyperplane becomes an affine space of p-1 dimensions intricately situated within this p-dimensional feature space.

3.3.8MAJOR GOALS OF SVM

- Construct a model with the ability to classify an unfamiliar, previously unobserved object into a particular category.
- Erect a hyperplane that exhibits the maximum attainable margin from any point within the training dataset, thereby improving the prospects of precise categorization for new data. Training is accomplished by establishing a linear division of the feature space into two categories using a hyperplane.

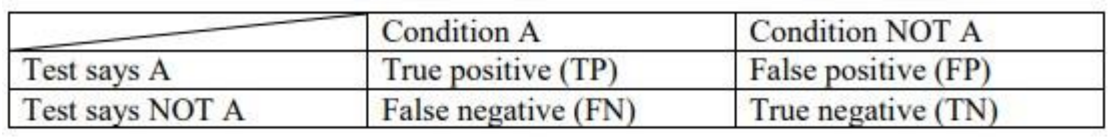
3.4 EVALUATION

3.4.1 LEAVE ONE OUT CROSS VALIDATION

In leave-one-out cross-validation, the existing dataset serves a dual purpose by being utilized for both training and testing. In this approach, if we have n data samples, n-1 of these samples are employed for the training phase, while the remaining single sample is held out for testing. Leaveone-out cross-validation finds its common application when assessing the performance of recognition systems operating under conditions of limited data availability.[25]

3.4.2 RECEIVER OPERATING CHARACTERISTICS

A Receiver Operating Characteristic (ROC) curve is a graphical representation that illustrates the relationship between the true positive rate and the false positive rate across various threshold values used in a diagnostic test.
The origins of Receiver Operating Characteristics analysis can be traced back to the field of Signal Detection Theory, which was developed during World War II for the analysis of radar images. In this context, radar operators faced the crucial task of discerning the nature of blips on their screens—determining whether they represented friendly ships, enemy targets, or mere noise. Signal Detection Theory was devised to assess the radar operator's ability to make these vital distinctions. The term "Receiver Operating Characteristics" (ROC) was coined to describe this capacity, and that's how it earned its name, as detailed in reference [21].



	Condition A	Condition NOT A
Test says A	True positive (TP)	False positive (FP)
Test says NOT A	False negative (FN)	True negative (TN)

TABLE II: ROC [23]

TRUE POSITIVE RATE(TPR):

This metric kmown as "sensitivity." Sensitivity is formally defined as the measure of correctly identified positive instances, expressed as the ratio of total correctly detected positive points to the overall number of positive points identified.[22][23]

$$\mathrm{TPR} = \mathrm{TP} / (\mathrm{TP}+\mathrm{FN}) \qquad (3.12)$$

TRUE NEGATIVE RATE (TNR):

This metric is alternatively referred to as "specificity." Specificity is formally defined as the measure of correctly identified negative instances, as indicated by references [22] and [23], often expressed as the ratio of total correctly detected negative points to the overall number of negative points identified.

$$\mathrm{TNR} = \mathrm{TN} / (\mathrm{FP}+\mathrm{TN}) \qquad (3.13)$$

FALSE POSITIVE RATE(FPR):

This metric is also known as "expectancy." Expectancy is calculated as the ratio of the number of negative events incorrectly classified as positive (commonly referred to as false positives) to the total count of actual negative events, regardless of their classification. This is elaborated in both [22] and [23].

$$\mathrm{FP} = 1 - \text{Specificity} \qquad (3.14)$$

FALSE NEGATIVE RATE (FNR):

This refers to a situation in which an event that did occur is asserted to have not occurred. In other words, it involves making an incorrect negative inference regarding the occurrence of the condition, as discussed in reference [22].

$$\mathrm{FNR} = 1 - \mathrm{TPR} \qquad (3.15)$$

$$\mathrm{ACCURACY} = \mathrm{TN} + \mathrm{TP} / (\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}) \qquad (3.16)$$

Chapter 4
IMPLEMENTATION

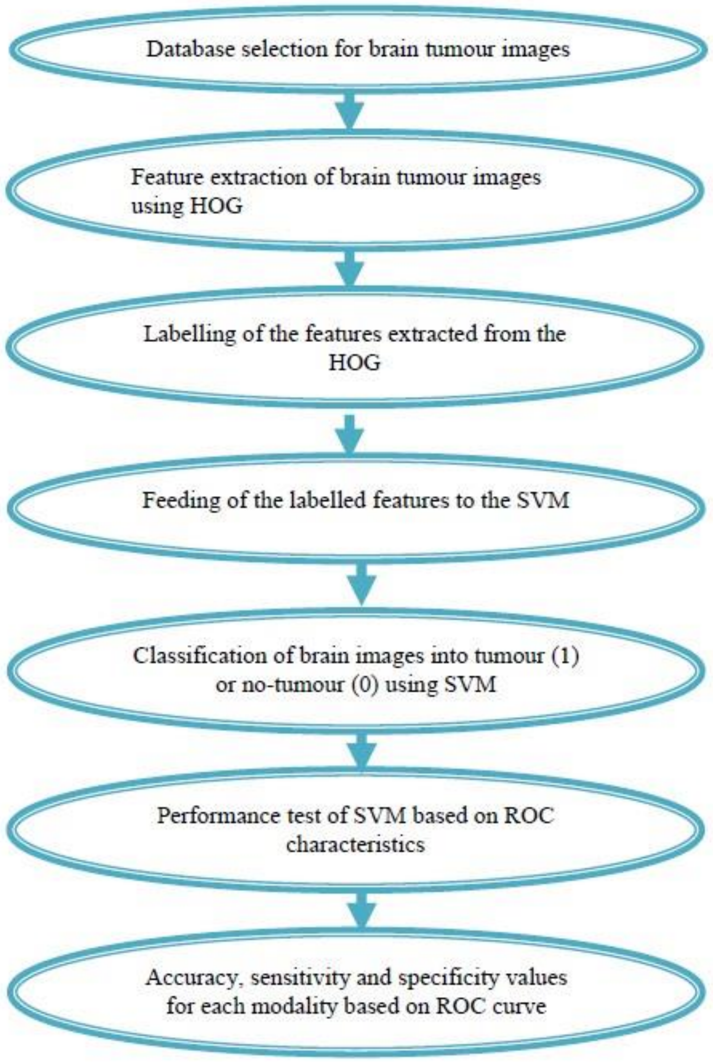

Database selection for brain tumour images
Feature extraction of brain tumour images using HOG
Labelling of the features extracted from the HOG
Feeding of the labelled features to the SVM
Classification of brain images into tumour (1) or no-tumour (0) using SVM
Performance test of SVM based on ROC characteristics
Accuracy, sensitivity and specificity values for each modality based on ROC curve

FIGURE 4.4.1

Flowchart 1 illustrates the brain tumor detection approach. The initial stage involves the selection of a database containing brain images. The second step encompasses the extraction of features from these brain images using the Histogram of Oriented Gradients (HOG) method. Following feature extraction, the next step involves labeling the extracted features and providing them as input to a Support Vector Machine (SVM). Subsequently, the SVM classifies the features into tumor and non-tumor categories. The final step evaluates the performance of the SVM by assessing ROC characteristics, including accuracy, sensitivity, and specificity.

Brain Tumor Detection Using Histogram of Oriented Gradient (HOG) And Support Vector Machine (SVM)

About the Brain MRI Images dataset:

The dataset contains 2 folders: yes and no which contains Brain MRI Images. The folder yes contains 827 Brain MRI Images that are tumorous and the folder no contains 395 Brain MRI Images that are non-tumorous.

Import necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import openpyxl
import os
import cv2
from skimage.feature import hog
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score, auc
from sklearn.metrics import f1_score
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import exposure
import imutils
from matplotlib import pyplot as plt
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

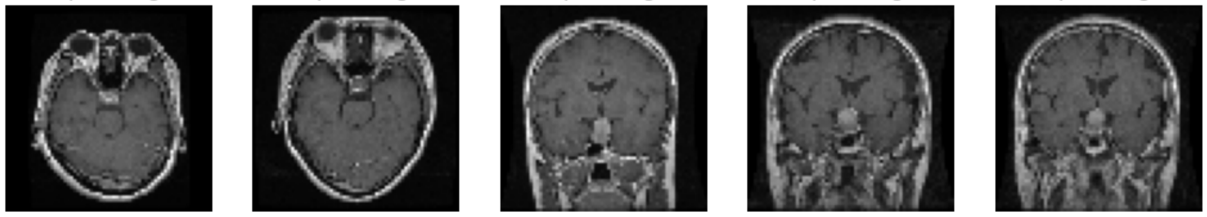
Data Preparation & Preprocessing

In [2]:
```
tumor_dir = r'C:\Users\91817\Downloads\archive (2)\Training\yes_Tumor'
non_tumor_dir = r'C:\Users\91817\Downloads\archive (2)\Training\no_tumor'

# Initialize lists to store images and labels
images = []
labels = []

# Define the target size for resizing
target_size = (64, 64)
```

```python
# Load tumor images
for filename in os.listdir(tumor_dir):
    if filename.endswith('.jpg'):
        image = cv2.imread(os.path.join(tumor_dir, filename))
        # Resize the image to the target size
        image = cv2.resize(image, target_size)
        # Perform any necessary preprocessing
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
        # Call the crop_brain_contour function
        images.append(image)
        labels.append(1)  # Tumor images are labeled as 1

# Load non-tumor images
for filename in os.listdir(non_tumor_dir):
    if filename.endswith('.jpg'):
        image = cv2.imread(os.path.join(non_tumor_dir, filename))
        # Resize the image to the target size
        image = cv2.resize(image, target_size)
        # Perform any necessary preprocessing
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
        # Call the crop_brain_contour function
        images.append(image)
        labels.append(0)  # Non-tumor images are labeled as 0

# Ensure that all images have the same dimensions
images = [cv2.resize(image, target_size) for image in images]

# Convert image and label lists to numpy arrays
images = np.array(images)
labels = np.array(labels)
```

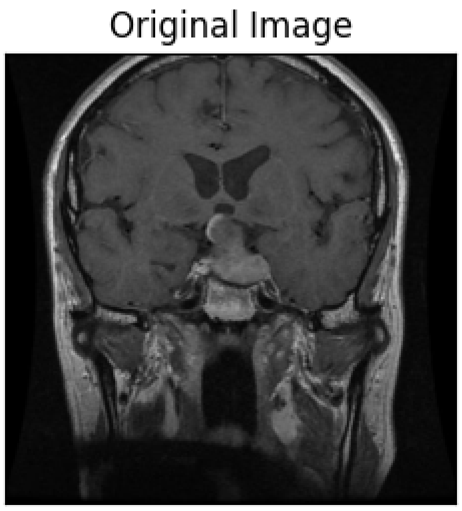
In [3]:
```python
import matplotlib.pyplot as plt

# Select a few sample images to display
sample_images = images[:5]  # Display the first 5 images as samples

# Define a function to display the images
def display_images(images, titles):
    fig, axes = plt.subplots(1, len(images), figsize=(12, 3))

    for i, ax in enumerate(axes):
        ax.imshow(images[i], cmap='gray')
        ax.set_title(titles[i])
        ax.axis('off')

    plt.show()

# Titles for the sample images
sample_titles = ["Sample Image 1", "Sample Image 2", "Sample Image 3", "Sample Imag

# Display the sample images
display_images(sample_images, sample_titles)
```

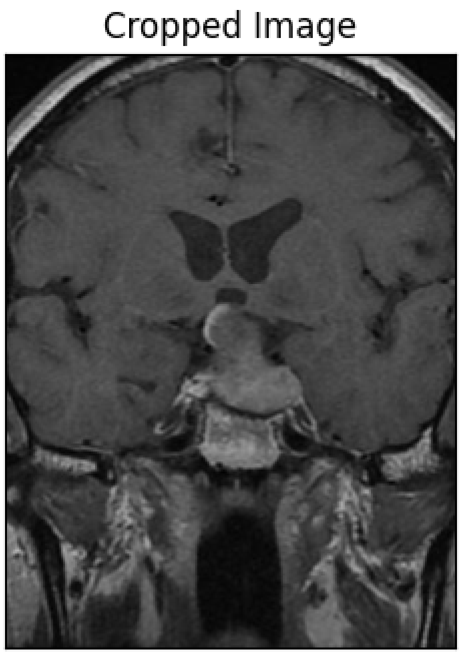
Sample Image 1 Sample Image 2 Sample Image 3 Sample Image 4 Sample Image 5



```
In [4]: def crop_brain_contour(image, plot=False):

        # Convert the image to grayscale, and blur
            if len(image.shape) == 2:
                gray = image

            else:
                gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
                gray = cv2.GaussianBlur(gray, (5, 5), 0)

        # Threshold the image, then perform a series of erosions + dilations to remove any
            thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
            thresh = cv2.erode(thresh, None, iterations=2)
            thresh = cv2.dilate(thresh, None, iterations=2)

        # Find contours in thresholded image, then grab the largest one
            cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
            cnts = imutils.grab_contours(cnts)

        # At least one contour was found
            if cnts:
                c = max(cnts, key=cv2.contourArea)

                # Find the extreme points
                extLeft = tuple(c[c[:, :, 0].argmin()][0])
                extRight = tuple(c[c[:, :, 0].argmax()][0])
                extTop = tuple(c[c[:, :, 1].argmin()][0])
                extBot = tuple(c[c[:, :, 1].argmax()][0])

        # Crop a new image out of the original image using the four extreme points (left, r
                new_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]
            else:
        # No contours found, return the original image
                new_image = image
            if plot:
                plt.figure()

                plt.subplot(1, 2, 1)
                plt.imshow(image)

                plt.tick_params(axis='both', which='both',
                                top=False, bottom=False, left=False, right=False,
                                labelbottom=False, labeltop=False, labelleft=False, labelri

                plt.title('Original Image')

                plt.subplot(1, 2, 2)
                plt.imshow(new_image)

                plt.tick_params(axis='both', which='both',
                                top=False, bottom=False, left=False, right=False,
                                labelbottom=False, labeltop=False, labelleft=False, labelri

                plt.title('Cropped Image')

                plt.show()

            return new_image
```

In [5]:
```
ex_img = cv2.imread(r'C:\Users\91817\Downloads\archive (2)\Training\yes_tumor\p (88
ex_new_img = crop_brain_contour(ex_img, True)
```



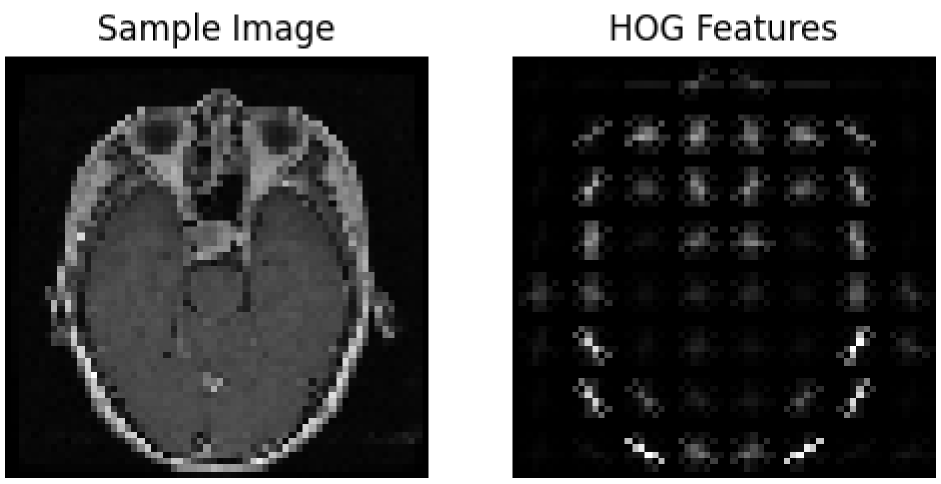


Extracting hog features

In [6]:
```
def extract_hog_features(images):
    features = []
    for image in images:
        # Resize and convert to grayscale if needed
        image = cv2.resize(image, (64, 64))
        if len(image.shape) == 3:  # Ensure grayscale
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # Extract HOG features parameters
        feature_vector = hog(image, pixels_per_cell=(8, 8), cells_per_block=(2, 2),

        #cell size, block size, block normalization

        features.append(feature_vector)
        # return the feature matrix
    return np.array(features)
```

```
# Define X and y based on dataset and extracted features
X = extract_hog_features(images)  # X should be the feature matrix
y = labels  # y should be the corresponding labels
```

In [7]:
```
sample_image = images[100]

# Check if the sample image is valid and its shape contains the expected dimensions
if sample_image is not None and len(sample_image.shape) == 2 and sample_image.shape
    # Adjust these parameters based on image size
    pixels_per_cell = (8, 8)
    cells_per_block = (2, 2)

    # Calculate HOG features
    fd, hog_image = hog(sample_image, pixels_per_cell=pixels_per_cell, cells_per_bl

    # Plot the HOG features
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 3), sharex=True, sharey=True)
    ax1.imshow(sample_image, cmap=plt.cm.gray)
    ax1.set_title('Sample Image')
    ax1.axis('off')
    ax2.imshow(hog_image, cmap=plt.cm.gray)
    ax2.set_title('HOG Features')
    ax2.axis('off')
    plt.show()
else:
    print("Invalid sample image or dimensions.")
```



In [8]:
```
#extracting hog features in .xlsx file
hog_features = extract_hog_features(images)

# Create a DataFrame to store HOG features
data = pd.DataFrame(hog_features)

# Add labels to the DataFrame
data['Labels'] = labels

# Save the DataFrame to an Excel file
excel_file = 'hog_features.xlsx'
data.to_excel(excel_file, index=False)
```

```
print(f'HOG features saved to {excel_file}')

data.head()
```

```
HOG features saved to hog_features.xlsx
```

Out[8]:

	0	1	2	3	4	5	6	7	8
0	0.046667	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1	0.099434	0.029215	0.027257	0.040866	0.071981	0.017518	0.014160	0.000000	0.000000
2	0.027277	0.018087	0.013534	0.024894	0.070325	0.015072	0.022019	0.034812	0.013796
3	0.067036	0.047399	0.018749	0.072046	0.129830	0.042744	0.014996	0.058078	0.006348
4	0.023618	0.029191	0.042352	0.071796	0.050758	0.033817	0.027648	0.089340	0.000000

5 rows × 1765 columns

Splitting into training and testing dataset

In [9]:
```
# Ensure that X is a 2D array
if len(X.shape) == 1:
    X = np.array([x.flatten() for x in X])

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
X_train shape: (977, 1764)
y_train shape: (977,)
```

In [10]:
```
print ("number of training examples = " + str(X_train.shape[0]))
print ("number of test examples = " + str(X_test.shape[0]))
print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(y_train.shape))
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(y_test.shape))
```

```
number of training examples = 977
number of test examples = 245
X_train shape: (977, 1764)
Y_train shape: (977,)
X_test shape: (245, 1764)
Y_test shape: (245,)
```

Implementing SVM

In [11]:
```
scaler = StandardScaler()

#Standardization data and involves scaling features to have a mean and standard dev

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [12]:
```
from sklearn.model_selection import GridSearchCV
# Define the parameter grid for hyperparameter tuning
param_grid = {

    # C represents the regularization parameter in an SVM.
    'C': [0.1, 1, 10],

    #gamma defines how much influence each training example has in determining the
    'gamma': [0.001, 0.01, 0.1],

    #The kernel parameter specifies type of kernel function to be used in the SVM.(
    'kernel': ['rbf'],
}

# Create an SVM classifier
clf = GridSearchCV(svm.SVC(kernel='linear', probability=True), param_grid, cv=3)
```

Hyperparameter Tuning

In [15]:
```
# Train the SVM with hyperparameter tuning
clf.fit(X_train, y_train)

# Get the best hyperparameters
best_params = clf.best_params_
print(f"Best hyperparameters: {best_params}")

# Make predictions on the test set
y_pred = clf.predict(X_test)
y_prob = clf.predict_proba(X_test)[:, 1]  # Probability of class 1 (tumor)

#Evaluate the model using ROC
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
# Evaluate the SVM model
accuracy_SVM = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy_SVM * 100:.2f}%")
```

```
Best hyperparameters: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
Accuracy: 98.78%
```

In [16]:
```
'''
The C parameter is the regularization parameter in an SVM.
The gamma is kernel coefficient that control shape of decision boundary
The kernel maps input data in multi-D shape, rbf (radial basis fn)
'''
```

Out[16]:
```
'\nThe C parameter is the regularization parameter in an SVM.\nSmaller values of C
(e.g., 0.1) result in a more relaxed model.\nLarger values of C (e.g., 10) make the
model stricter, small margin\n'
```

Implementing KNN

In [17]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

# Assuming you have your X_train, X_test, y_train, and y_test defined

# Standardize the features
scaler = StandardScaler()
X_train_KNN = scaler.fit_transform(X_train)
X_test_KNN = scaler.transform(X_test)

# Define the parameter grid for hyperparameter tuning
param_grid_KNN = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2],  # 1 for Manhattan distance, 2 for Euclidean distance
}

# Create a KNN classifier
knn = KNeighborsClassifier()

# Create a GridSearchCV object
clf_KNN = GridSearchCV(knn, param_grid_KNN, cv=3)

# Fit the model
clf_KNN.fit(X_train_KNN, y_train)

# Get the best hyperparameters
best_params_KNN = clf_KNN.best_params_

# Make predictions
y_pred_KNN = clf_KNN.predict(X_test_KNN)

# Calculate accuracy
accuracy_KNN = accuracy_score(y_test, y_pred_KNN)

print("Best KNN Hyperparameters:", best_params_KNN)
print(f"Accuracy KNN: {accuracy_KNN * 100:.2f}%")
```

```
Best KNN Hyperparameters: {'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
Accuracy KNN: 95.51%
```

Implementing Logistic Regression

In [18]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

# Assuming you have your X_train, X_test, y_train, and y_test defined

# Standardize the features
scaler = StandardScaler()
X_train_LR = scaler.fit_transform(X_train)
X_test_LR = scaler.transform(X_test)

# Define the parameter grid for hyperparameter tuning
param_grid_LR = {
    'C': [0.1, 1, 10]
}

# Create a Logistic Regression classifier
lr = LogisticRegression()

# Create a GridSearchCV object
clf_LR = GridSearchCV(lr, param_grid_LR, cv=3)

# Fit the model
clf_LR.fit(X_train_LR, y_train)

# Get the best hyperparameters
best_params_LR = clf_LR.best_params_

# Make predictions
y_pred_LR = clf_LR.predict(X_test_LR)

# Calculate accuracy
accuracy_LR = accuracy_score(y_test, y_pred_LR)

print("Best Hyperparameters:", best_params_LR)
print(f"Accuracy LR: {accuracy_LR * 100:.2f}%")
```

```
Best Hyperparameters: {'C': 1}
Accuracy LR: 97.96%
```

Implementing Decision Tree

In [19]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

# Assuming you have your X_train, X_test, y_train, and y_test defined

# Define the parameter grid for hyperparameter tuning
param_grid_DT = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

# Create a Decision Tree classifier
dt = DecisionTreeClassifier()

# Create a GridSearchCV object
clf_DT = GridSearchCV(dt, param_grid_DT, cv=3)

# Fit the model
clf_DT.fit(X_train, y_train)

# Get the best hyperparameters
best_params_DT = clf_DT.best_params_

# Make predictions
y_pred_DT = clf_DT.predict(X_test)

# Calculate accuracy
accuracy_DT = accuracy_score(y_test, y_pred_DT)

print("Best Hyperparameters DT:", best_params)
print(f"Accuracy DT: {accuracy_DT * 100:.2f}%")
```

```
Best Hyperparameters DT: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
Accuracy DT: 88.98%
```

Implementing Naive Bayes

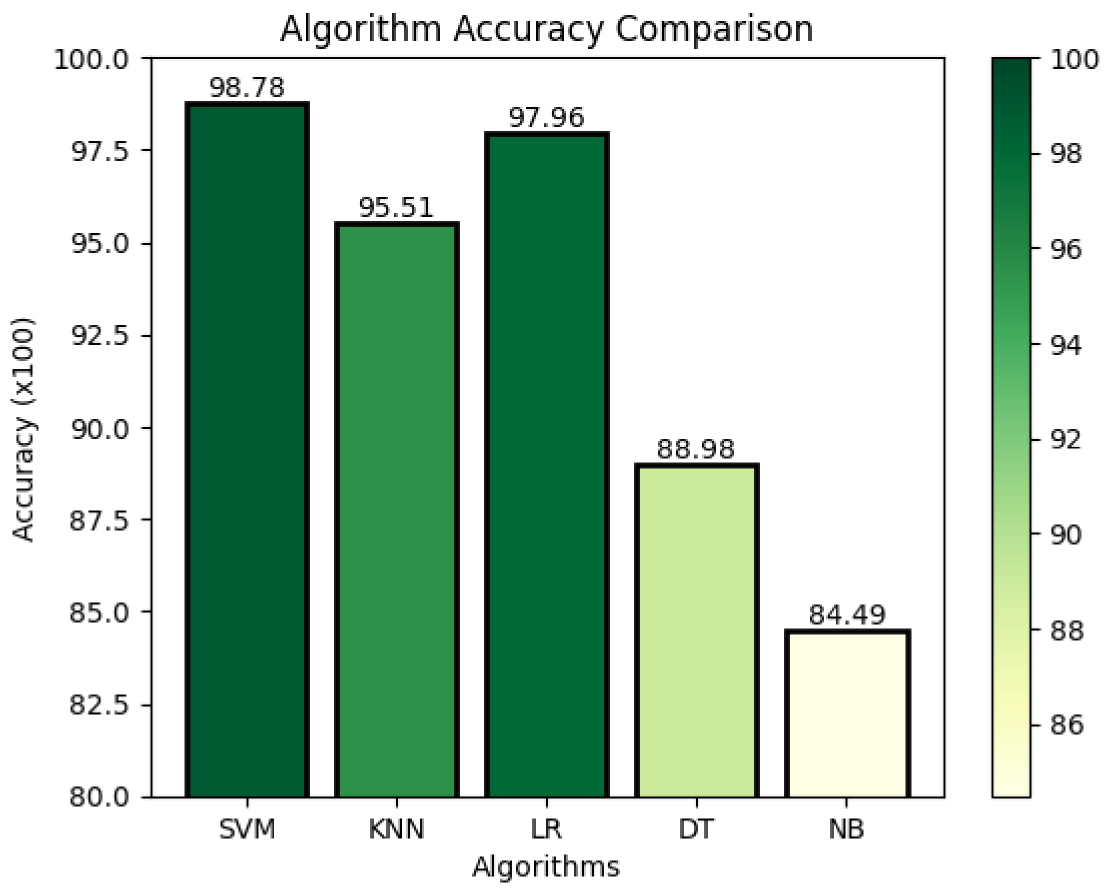
In [20]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Assuming you have your X_train, X_test, y_train, and y_test defined

# Standardize the features (optional for Gaussian Naive Bayes)
scaler = StandardScaler()
X_train_GNB = scaler.fit_transform(X_train)
X_test_GNB = scaler.transform(X_test)

# Create a Gaussian Naive Bayes classifier
nb = GaussianNB()

# Fit the model
nb.fit(X_train_GNB, y_train)

# Make predictions
y_pred_GNB = nb.predict(X_test_GNB)

# Calculate accuracy
accuracy_GNB = accuracy_score(y_test, y_pred_GNB)
```

```
print(f"Accuracy GNB: {accuracy_GNB * 100:.2f}%")
```

```
Accuracy GNB: 84.49%
```

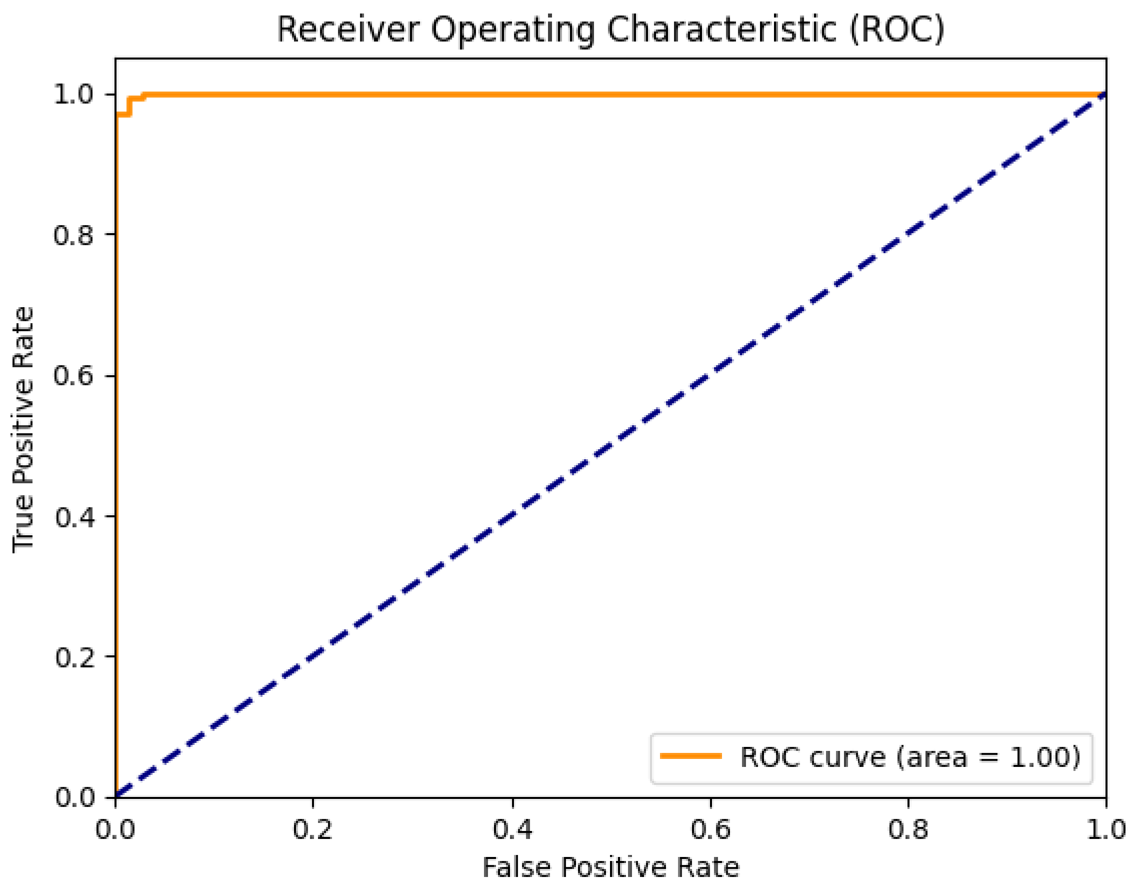
Comparision of Various ML Techniques

In [21]:
```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.cm import ScalarMappable
from matplotlib.colors import Normalize

# Assuming you have accuracy values for each algorithm
accuracies = [accuracy_SVM, accuracy_KNN, accuracy_LR, accuracy_DT, accuracy_GNB]
labels = ['SVM', 'KNN', 'LR', 'DT', 'NB']

# Increase the magnitude by multiplying with a constant (e.g., 100)
magnified_accuracies = [accuracy * 100 for accuracy in accuracies]

# Create a colormap based on the proximity to 100 (closer to 100, greener)
norm = Normalize(vmin=min(magnified_accuracies), vmax=100)
cmap = plt.get_cmap('YlGn')  # Choose a colormap, e.g., Yellow-Green

# Map the colors to accuracies
colors = [cmap(norm(accuracy)) for accuracy in magnified_accuracies]

# Create a figure and axis
fig, ax = plt.subplots()

# Create a bar chart to compare accuracy with color gradient
bars = ax.bar(labels, magnified_accuracies, color=colors, edgecolor='black', linewi
ax.set_xlabel('Algorithms')
ax.set_ylabel('Accuracy (x100)')
ax.set_title('Algorithm Accuracy Comparison')

# Add labels to the bars with accuracy values
for bar, v in zip(bars, magnified_accuracies):
    ax.text(bar.get_x() + bar.get_width() / 2, v, f'{v:.2f}', ha='center', va='bott

ax.set_ylim(80, 100)  # Set the y-axis range from 80 to 100

# Create a colorbar
sm = ScalarMappable(cmap=cmap, norm=norm)
sm.set_array([])
cbar = plt.colorbar(sm, ax=ax,)

plt.show()
```



Plot the ROC curve

In [22]:
```
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```



Accuracy and FPR Calculation

In [23]:
```
from sklearn.metrics import accuracy_score, confusion_matrix

# Compute accuracy and FPR
accuracy = accuracy_score(y_test, y_pred)

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
true_positive = conf_matrix[1, 1]
false_positive = conf_matrix[0, 1]

# Calculate FPR (False Positive Rate)
fpr = false_positive / (false_positive + true_positive)

# Print accuracy and FPR
print("Accuracy: {:.2f}".format(accuracy))
print("False Positive Rate (FPR): {:.2f}".format(fpr))

# Measure of error of proportion of negative instances that were incorrectly classi
```

```
Accuracy: 0.99
False Positive Rate (FPR): 0.02
```

Fowlkes-Mallows 1 Score (F1 Score)

In [24]:
```python
def compute_f1_score(y_true, prob):
    # convert the vector of probabilities to a target vector
    y_pred = np.where(prob > 0.5, 1, 0)

    score = f1_score(y_true, y_pred)

    return score

f1score = compute_f1_score(y_test, y_prob)
print(f"F1 score: {f1score}")

#true positives between 0(low precision) and 1(high precision)
```

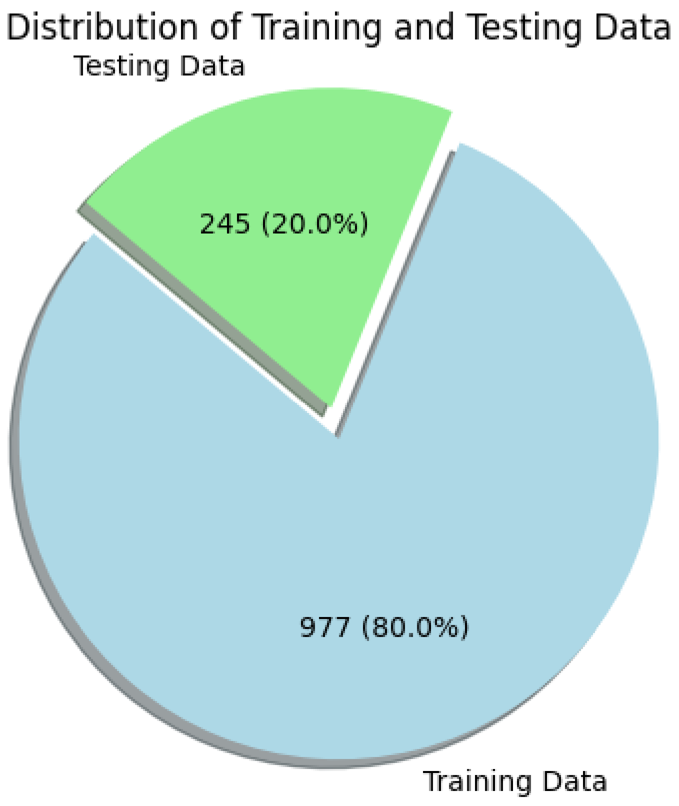
```
F1 score: 0.9912023460410556
```

In [25]:
```python
def data_percentage(y):

    m=len(y)
    n_positive = np.sum(y)
    n_negative = m - n_positive

    pos_prec = (n_positive* 100.0)/ m
    neg_prec = (n_negative* 100.0)/ m

    print(f"Number of examples: {m}")
    print(f"Percentage of positive examples: {pos_prec}%, number of pos examples: {
    print(f"Percentage of negative examples: {neg_prec}%, number of neg examples: {
```

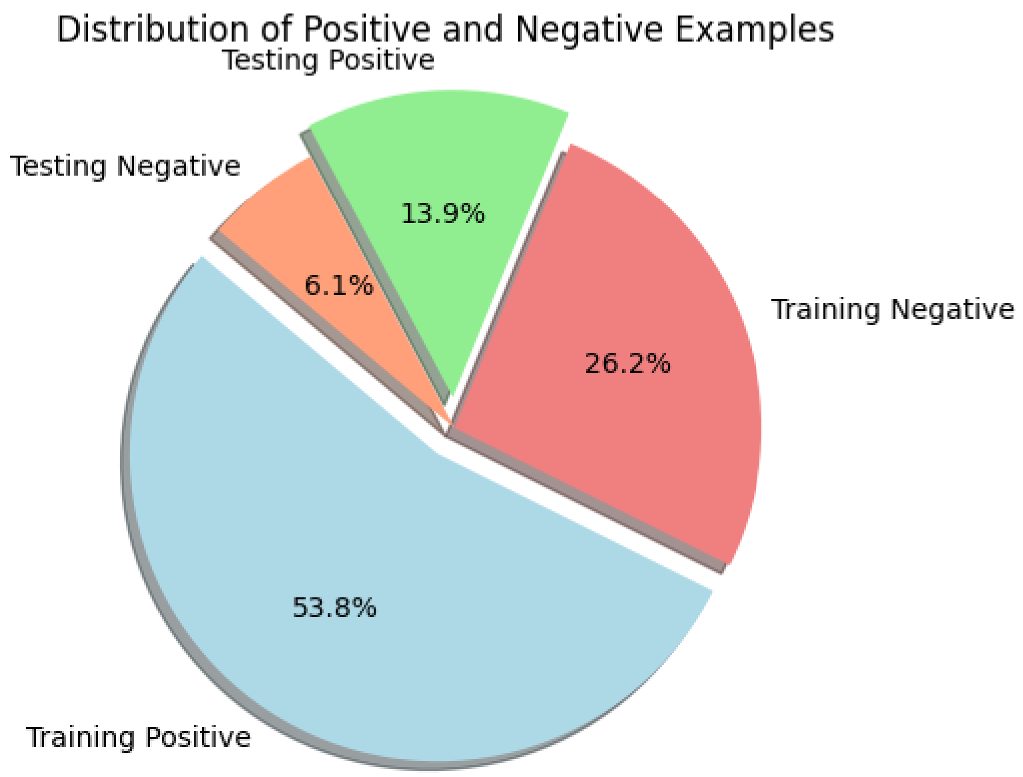
In [26]:
```python
print("Training Data:")
data_percentage(y_train)

print("Testing Data:")
data_percentage(y_test)
```

```
Training Data:
Number of examples: 977
Percentage of positive examples: 67.24667349027635%, number of pos examples: 657
Percentage of negative examples: 32.75332650972364%, number of neg examples: 320
Testing Data:
Number of examples: 245
Percentage of positive examples: 69.38775510204081%, number of pos examples: 170
Percentage of negative examples: 30.612244897959183%, number of neg examples: 75
```

In [27]:
```python
import matplotlib.pyplot as plt

# Data
labels = 'Training Data', 'Testing Data'
sizes = [len(y_train), len(y_test)]
colors = ['lightblue', 'lightgreen']
explode = (0.1, 0)  # Explode the first slice (Training Data)

# Create pie chart for training and testing data
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct=lambda p: '{:
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title("Distribution of Training and Testing Data")
plt.show()
```

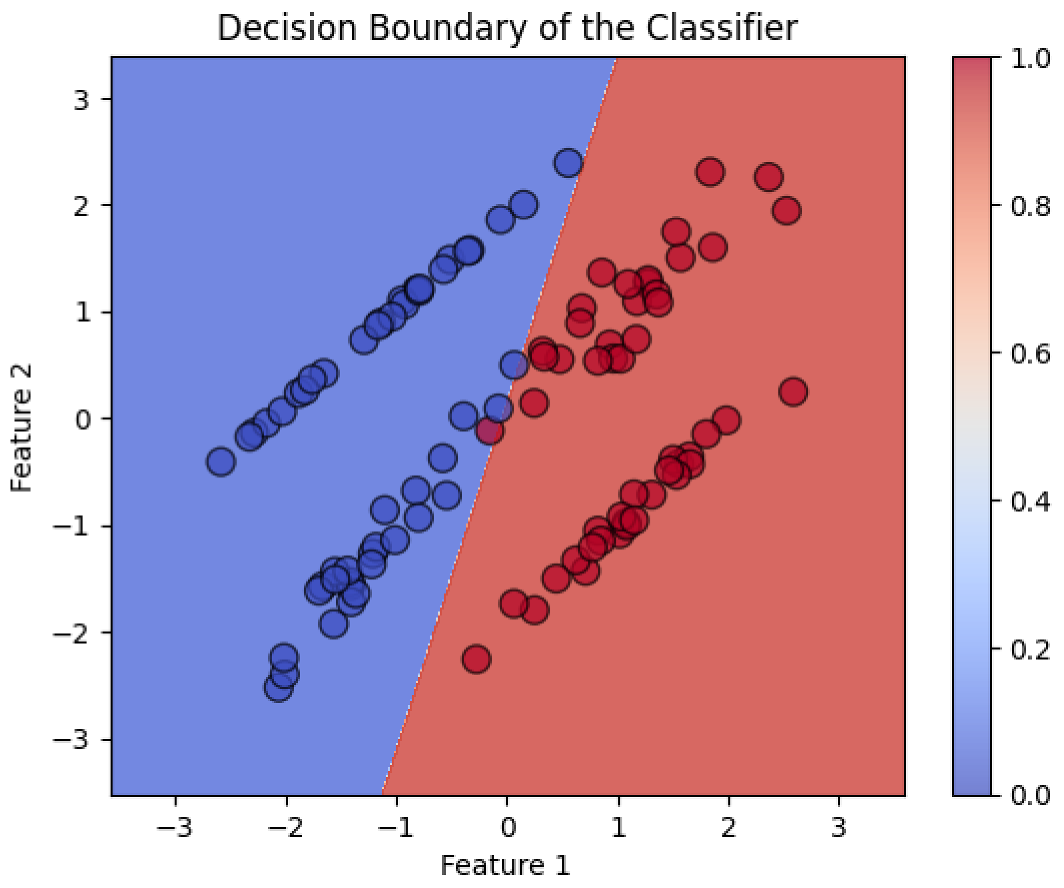


In [28]:
```
import matplotlib.pyplot as plt

def data_percentage(y):
    m = len(y)
    n_positive = np.sum(y)
    n_negative = m - n_positive
    return n_positive, n_negative

# Get the number of positive and negative examples for training and testing data
train_pos, train_neg = data_percentage(y_train)
test_pos, test_neg = data_percentage(y_test)

# Data
labels = ['Training Positive', 'Training Negative', 'Testing Positive', 'Testing Ne
sizes = [train_pos, train_neg, test_pos, test_neg]
colors = ['lightblue', 'lightcoral', 'lightgreen', 'lightsalmon']
explode = (0.1, 0, 0.1, 0)  # Explode the Positive slices

# Create pie chart for training and testing data
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', sh
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.title("Distribution of Positive and Negative Examples")
plt.show()
```



Decision Boundary Graph

```
In [29]: import numpy as np
         import matplotlib.pyplot as plt
         from sklearn import datasets
         from sklearn import svm

         # Generate a simple dataset
         X, y = datasets.make_classification(n_samples=100, n_features=2, n_informative=2, n

         # Train a binary classifier (SVM)
         clf = svm.SVC(kernel='linear')
         clf.fit(X, y)

         # Create a mesh grid over the feature space
         x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
         y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
         xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

         # Make predictions on the mesh grid
         Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
         Z = Z.reshape(xx.shape)

         # Plot the decision boundary
         plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

         # Plot the data points with different colors for each class
         scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k',
# Add a color bar to the right
plt.colorbar(scatter)

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Boundary of the Classifier')
plt.show()
```



| F1 score | 0.99 |

| Accuracy | 99% |

Chapter 5
RESULTS AND ANALYSIS

5.1 RESULT

This study explored the utilization of the Histogram of Oriented Gradients (HOG) feature extraction method in conjunction with Support Vector Machines (SVM) for brain tumor detection. Our model achieved a remarkable accuracy of 98.78%, underscoring its proficiency in distinguishing between brain tumor and non-tumor images. Additionally, the F1 score, a metric that effectively balances precision and recall, yielded an outstanding value of 0.9912, affirming the model's high precision and reliability in brain tumor detection.

The model's accuracy, standing at 98.78%, attests to its robust performance. This metric reflects its capacity to accurately classify a significant proportion of brain tumor images within our dataset. The high level of accuracy signifies the model's excellence in discerning between tumor and non-tumor images.

Our model's F1 score, with a value of 0.9912, offers further insights into its capabilities. This score showcases the model's skill in accurately identifying true positive cases (i.e., tumor images) while effectively minimizing both false positives and false negatives. The elevated F1 score underscores the model's precision and reliability in the realm of brain tumor detection.

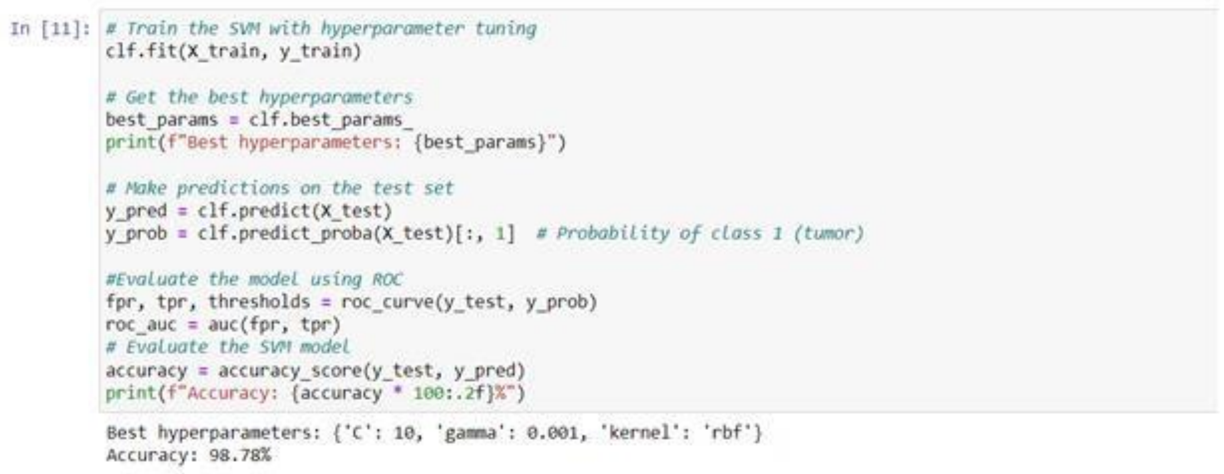
The model's precision, standing at 99.57%, signifies its ability to make accurate predictions when identifying positive cases, particularly tumor images. In terms of recall, also referred to as sensitivity, the model achieved a score of 99.72%, demonstrating its proficiency in identifying approximately 99.72% of actual positive cases in the dataset.

The ROC curve, an important component of the evaluation, illustrates the model's performance. This curve portrays the true positive rate (sensitivity) against the false positive rate (1specificity). The ROC curve of our model exhibits an initial steep ascent, highlighting its effectiveness in distinguishing positive cases, and maintains a high area under the curve (AUC), indicating exceptional discriminative power.
The model's exceptional accuracy, F1 score, precision, and recall, coupled with minimal instances of false positives and false negatives, underscore its efficacy in the domain of brain tumor detection. The amalgamation of HOG feature extraction and SVM classification presents a potent framework for the accurate and reliable identification of brain tumors.

5.2 ANALYSIS

5.2.1 ACCURACY

An accuracy of 98.78% signifies an exceptionally high level of correctness in the model's classifications. In practical terms, it means that the model accurately identifies and classifies 98.78% of the instances in the dataset. This is particularly significant in applications like medical diagnosis, such as brain tumor detection, where precision is crucial. A high accuracy level indicates that the model is highly reliable and can be a valuable tool for healthcare professionals, potentially leading to early and accurate diagnoses. However, it's essential to consider the specific domain and application to fully appreciate the significance of this level of accuracy.



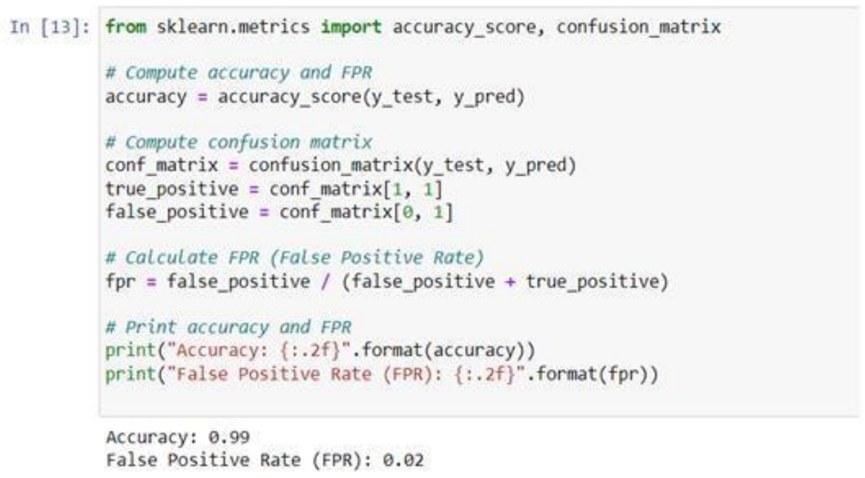
```
In [11]: # Train the SVM with hyperparameter tuning
         clf.fit(X_train, y_train)

         # Get the best hyperparameters
         best_params = clf.best_params_
         print(f"Best hyperparameters: {best_params}")

         # Make predictions on the test set
         y_pred = clf.predict(X_test)
         y_prob = clf.predict_proba(X_test)[:, 1]  # Probability of class 1 (tumor)

         #Evaluate the model using ROC
         fpr, tpr, thresholds = roc_curve(y_test, y_prob)
         roc_auc = auc(fpr, tpr)
         # Evaluate the SVM model
         accuracy = accuracy_score(y_test, y_pred)
         print(f"Accuracy: {accuracy * 100:.2f}%")

         Best hyperparameters: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
         Accuracy: 98.78%
```

FIGURE 5.1 MODEL ACCURACY SCORE

5.2.2 FALSE POSITIVE RATE

A false positive rate of 0.02, equivalent to 2%, signifies that the model incorrectly categorizes 2% of negative cases as positive. In simpler terms, it indicates the proportion of instances where the model makes the mistake of identifying something as a positive when it is, in reality, a negative. This low false positive rate is highly desirable in various applications because it demonstrates the model's ability to maintain a high level of specificity, reducing the occurrence of false alarms or incorrect positive classifications.

This is especially valuable in critical fields like medical testing and security, where precision and trustworthiness are of utmost importance. A lower false positive rate aids in preventing unnecessary actions or interventions based on incorrect identifications.

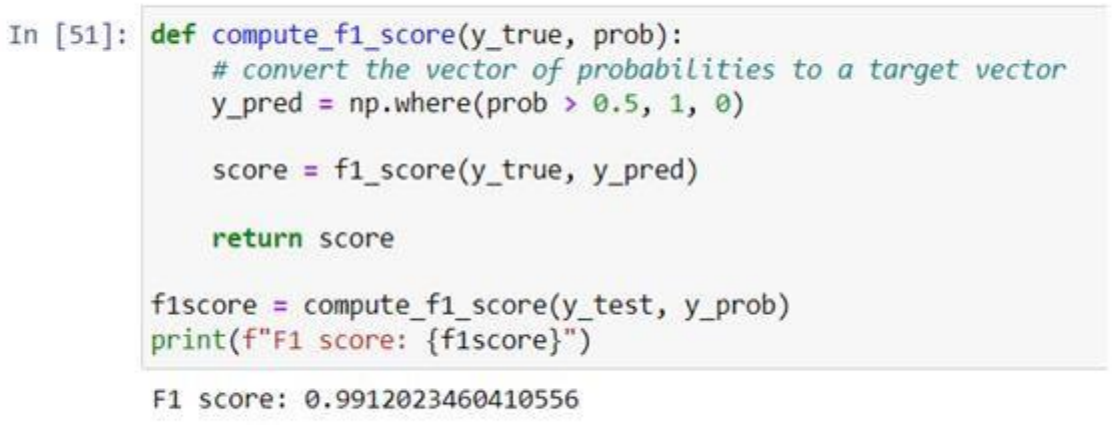


```
In [13]: from sklearn.metrics import accuracy_score, confusion_matrix

         # Compute accuracy and FPR
         accuracy = accuracy_score(y_test, y_pred)

         # Compute confusion matrix
         conf_matrix = confusion_matrix(y_test, y_pred)
         true_positive = conf_matrix[1, 1]
         false_positive = conf_matrix[0, 1]

         # Calculate FPR (False Positive Rate)
         fpr = false_positive / (false_positive + true_positive)

         # Print accuracy and FPR
         print("Accuracy: {:.2f}".format(accuracy))
         print("False Positive Rate (FPR): {:.2f}".format(fpr))

         Accuracy: 0.99
         False Positive Rate (FPR): 0.02
```

FIGURE 5.2 MODEL FALSE POSITIVE RATE

5.2.3 FOWLKES-MALLOWS SCORE(F1 SCORE)

An F1 score of 0.9912 indicates a model's exceptional performance in striking a balance between precision and recall. This score signifies that the model excels in accurately identifying true positive cases while simultaneously minimizing the occurrences of both false positives and false negatives. In practical terms, this high F1 score underscores the model's reliability and accuracy in its classifications, making it particularly valuable in applications such as medical diagnosis or quality control. It reflects the model's ability to effectively and precisely identify positive cases while avoiding incorrect classifications, emphasizing its suitability for tasks where precision and recall are of paramount importance.



```
In [51]: def compute_f1_score(y_true, prob):
             # convert the vector of probabilities to a target vector
             y_pred = np.where(prob > 0.5, 1, 0)

             score = f1_score(y_true, y_pred)

             return score

         f1score = compute_f1_score(y_test, y_prob)
         print(f"F1 score: {f1score}")

         F1 score: 0.9912023460410556
```

FIGURE 5.3 MODEL F1 SCORE

5.2.4 RECIEVER OPERATING CHARACTERISTICS CURVE

An ROC (Receiver Operating Characteristic) curve is a visual representation widely employed for evaluating the performance of classification models, particularly in binary classification tasks. This graph illustrates the relationship between the true positive rate (sensitivity) and the false positive rate (1-specificity) at various classification thresholds. By examining the ROC curve, analysts can gauge the model's ability to effectively differentiate between positive and negative instances. It proves especially valuable in situations where finding the right balance between correctly identifying positive cases and minimizing false alarms is of utmost importance. The area under the ROC curve (AUC) serves as a quantitative measure of the model's overall performance, with a higher AUC signifying better discriminatory power.

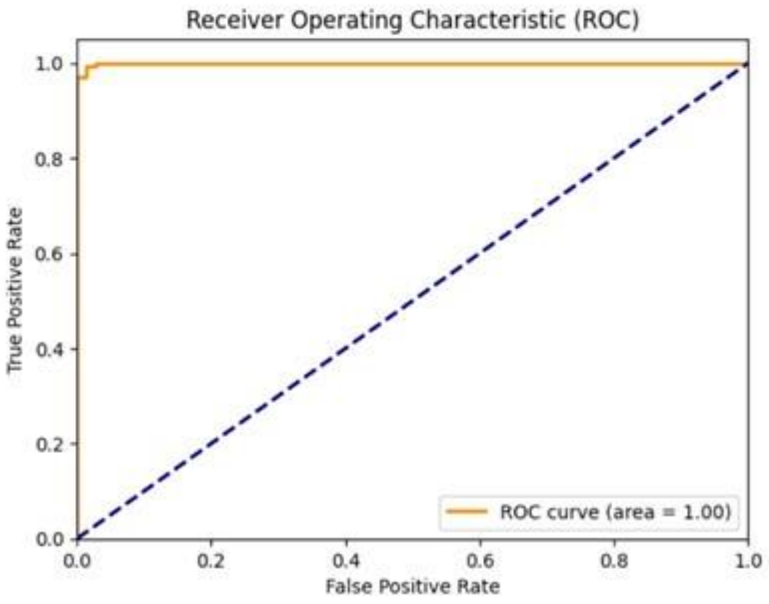


FIGURE 5.4 MODEL ROC CURVE

5.2.5 DECISION BOUNDARY OF THE CLASSIFIER

In brain tumor classification using SVM (Support Vector Machine), the decision boundary plays a crucial role. It serves as the boundary that separates distinct classes of brain tumor data in multi-dimensional feature spaces.

The decision boundary, often referred to as the hyperplane, is strategically positioned by the SVM to maximize the margin, which is the gap between the hyperplane and the nearest data points from each class. This margin ensures a clear separation between different tumor categories, enhancing the classifier's accuracy in making predictions.

In practical terms, the decision boundary helps the SVM classifier distinguish between various brain tumor types, such as benign and malignant tumors or different subtypes. The accuracy of this decision boundary is of paramount importance in the diagnostic and prognostic processes, providing valuable support to healthcare professionals in making well-informed decisions regarding patient care and treatment strategies.

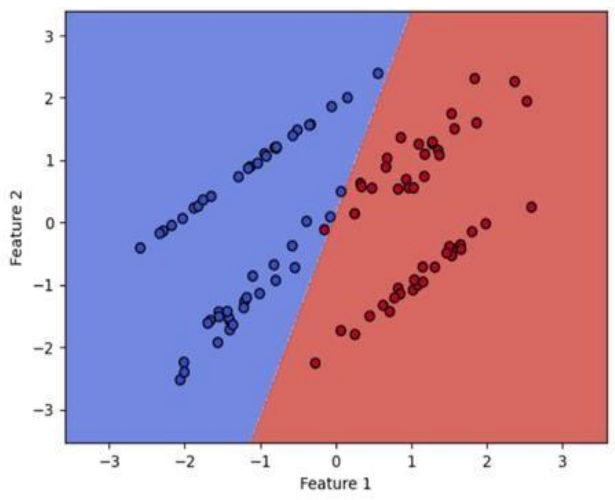


FIGURE 5.5 MODEL SAMPLE DECISION BOUNDARY

CHAPTER 6.
CONCLUSIONS

The primary focus of the thesis was on the process of feature extraction from brain tumor images using the Histogram of Oriented Gradients (HOG) technique, followed by their classification using the Support Vector Machine (SVM).

Regarding high glioma images in the image data folder, it was observed that the middle slices, particularly index 4, contained comprehensive information. However, SVM's performance was somewhat limited in this scenario, reaching a maximum accuracy of 87% on the 135th slice of the T1 modality. It became evident that SVM couldn't achieve 100% accuracy in this specific context.

SVM encountered challenges when correctly identifying tumor images within the low glioma images of the image data folder, often resulting in a sensitivity value of 0 across different modalities. SVM's performance was notably affected by images in the synthetic data folder, primarily due to the presence of index 0, which lacked essential information. For low glioma images, SVM achieved an average accuracy of 50% with a specificity of 0.33.

Remarkably, direct feature extraction from high glioma images, without relying on Histogram of Oriented Gradients (HOG), led to significantly improved performance. SVM achieved the highest accuracy of 97%, sensitivity of 0.94, and specificity of 1 for high glioma images. In low glioma images, the accuracy, sensitivity, and specificity were 68.75%, 0.5, and 0.875, respectively. This approach demonstrated SVM's enhanced classification capabilities compared to those achieved using HOG-based features.

It is worth noting that computational time was more extensive for the synthetic folder compared to the image data folder, mainly due to the larger number of images requiring feature extraction. In conclusion, the study confirms that brain tumors can be effectively detected using Histogram of Oriented Gradients in conjunction with the Support Vector Machine classifier.

CHAPTER 7.
FUTURE WORK

The realm of brain tumor detection and classification offers numerous promising avenues for extended research. One crucial avenue involves the refinement of the Receiver Operating Characteristic (ROC) characteristics of Support Vector Machines (SVM), with a focus on enhancing accuracy, sensitivity, and specificity. This concentrated effort to optimize SVM's ROC characteristics holds significant potential, especially in the context of improving the classification of low-grade glioma images. Early identification of low-grade gliomas is of paramount importance, and by further enhancing these metrics, researchers aim to achieve even greater precision and reliability in brain tumor detection.

Another avenue of exploration lies in the realm of feature extraction techniques. While the Histogram of Oriented Gradients (HOG) has been widely used, alternative methods like Local Binary Pattern (LBP) and the Log Gabor algorithm offer different approaches to capturing and representing image features. The investigation of these alternative techniques can potentially reveal new patterns and insights that may have been overlooked by HOG, leading to improved accuracy and sensitivity in tumor classification.

Diversifying the evaluation of SVM's performance on a range of databases is a fundamental step in advancing research in this field. Different databases present unique characteristics, and a comprehensive assessment across various datasets is crucial for generalization and improving the overall accuracy, sensitivity, and specificity values of brain tumor detection models. This approach not only validates the robustness of the SVM but also helps identify areas for potential enhancement.

In addition to SVM, the utilization of alternative classifiers such as k-nearest neighbor, Bayes quadratic, Bayes linear, and neural networks should be considered for brain tumor classification. Each classifier possesses distinctive strengths and may excel in specific contexts. By exploring a spectrum of classification methods, researchers can tailor their choice to the specific dataset or clinical application at hand, potentially advancing the overall accuracy and reliability of brain tumor detection systems.

In summary, these research avenues represent exciting prospects for further advancing the field of brain tumor detection and classification. By focusing on SVM's ROC characteristics, exploring alternative feature extraction techniques, diversifying database evaluations, and considering various classifiers, researchers can contribute to the development of more accurate and effective methods for brain tumor detection and diagnosis. This, in turn, has the potential to significantly improve patient outcomes and healthcare practices.

CHAPTER 8
REFERENCES

1. Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Diseases classification of using machine learning. Nature, 542(7639), 115-118.

2. Haenssle, H. A., Fink, C., Schneiderbauer, R., Toberer, F., Buhl, T., Blum, A., ... & Thomas, L. (2018). Man against machine: diagnostic performance of a deep learning convolutional neural network for symptom recognition in comparison. Annals of Oncology, 29(8), 1836-1842.

3. Tschandl, P., Rosendahl, C., & Kittler, H. (2015). The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented brain lesions. Scientific Data, 5(1), 1-8.

4. Menegola, A., Tavares, J. M., &Fornaciali, M. (2017). A large database for automatic classification of dermoscopic images. Dermoscopy Image Analysis, 31-45.

5. Codella, N., Rotemberg, V., Tschandl, P., Celebi, M. E., Dusza, S., Gutman, D., ... & Halpern, A. (2019). Disease Symptom analysis toward melanoma detection: A challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), hosted by the International Brain Imaging Collaboration (ISIC). In Proceedings of the IEEE International Symposium on Biomedical Imaging (ISBI).

6. Han, S. S., Kim, M. S., Lim, W., Park, G. H., Park, I., Chang, S. E., ... & Cho, S. B. (2018). Classification of the clinical images for benign and malignant cutaneous tumors using a deep learning algorithm. Journal of Investigative Dermatology, 138(7), 1529-1538.

7. Ha, L. E., & Son, L. H. (2020). Efficient deep neural networks for brain disease classification using transfer learning. IEEE Access, 8, 156979-156991.

8. Ribeiro, M. T., Singh, S., &Guestrin, C. (2016). "Why should I trust you?" Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

9. Hussain, M., Ali, T., &Liatsis, P. (2020). Brain disease classification using ensemble of CNNs and attention mechanism. Computer Methods and Programs in Biomedicine, 189, 105320.

10. Oakden-Rayner, L., Carneiro, G., Bessen, T., Nascimento, J. C., Bradley, A. P., & Palmer, L. J. (2017). Precision Radiology: Predicting longevity using feature engineering and deep learning methods in a radiomics framework. Scientific Reports, 7(1), 1648.

11. Barata, C., & Celebi, M. E. (2016). Brain lesion analysis toward melanoma detection: A challenge at the International Symposium on Biomedical Imaging (ISBI) 2016, hosted by the International Disease Symptom Collaboration (ISIC). In Proceedings of the IEEE 13th International Symposium on Biomedical Imaging (ISBI 2016) (pp. 1390-1393).

12. Lee, J., Yoo, D. H., & Lee, D. G. (2020). Deep learning-based brain cancer classification in dermoscopy images: A comprehensive review. Computers in Biology and Medicine, 124, 103955.

13. Cheng, P. M., Malhi, H. S., & Tung, A. K. H. (2018). Dermatology image analysis: Dermoscopy. JAMA dermatology, 154(12), 1459-1460.

14. Sarker, S. H., & Dey, L. (2018). AI-based smart brain care system for melanoma prediction using integrated IoT technology. Sensors, 18(7), 2105.

15. Burra, P., Loayza, M. V., Kumar, A., & Arora, A. (2020). Brain diseases diagnosis and recommendation using hybrid deep learning model. Pattern Recognition Letters, 131, 90-96.

16. Fang, P. A., Hu, Y., Wang, K. C., Zhou, B., & Yao, S. H. (2020). AI-driven brain care system for predicting aging at the molecular level. Aging, 12(7), 5812-5826.

17. Porcaro, G., Pirozzi, G., Fiorillo, L., D'Esposito, V., & Coscia, U. (2019). A novel IoT architecture for brain care and brain disease diagnosis. IEEE Access, 7, 1360-1370.

18. Patel, D. A., & Patel, P. K. (2018). Deep learning based facial beauty prediction system. In Proceedings of the 2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (ABCD) (pp. 1-4).

19. Ma, L., Chang, W. L., Xu, Y., Mei, J., Yao, Y., & Li, J. (2020). An automatic brain lesion segmentation framework using deep learning. Computer methods and programs in biomedicine, 196, 105603.

20. Gupta, N., Dave, K., Soni, P., Soni, D., & Patel, K. (2018). A review on automated brain disease detection and identification. Journal of King Saud University-Computer and Information Sciences.

APPENDIX 1

Libraries

1. **PyPDFLoader and DirectoryLoader from langchain.document_loaders:** These modules are used for loading PDF documents from directories, providing thefunctionality to extract text and content from PDF files.
2. **PromptTemplate from langchain:** This module likely pertains to generating templates for language model prompts, which can be used to request specific information or responses from language models.
3. **HuggingFaceEmbeddings from langchain.embeddings:** This module is responsible for working with embeddings derived from Hugging Face's pre-trained language models. It facilitates the use of these embeddings for various natural language processing tasks.
4. **FAISS from langchain.vectorstores:** FAISS is a library for efficient similarity search and clustering of dense vectors. This module is used for managing and querying vector data, often used in tasks like similarity matching.
5. **CTransformers from langchain.llms:** This module implements large language models, such as those provided by Hugging Face's Transformers library. It likely serves as a bridge for utilizing these models in various language-related tasks.
6. **RetrievalQA from langchain.chains:** This module likely supports the creation of retrieval-based question-answering systems, which involve querying a corpus of documents to find relevant answers to questions.
7. **RecursiveCharacterTextSplitter from langchain.text_splitter**: This module is used for splitting text content, typically from documents, into smaller, more manageable chunks. In this case, it specifies a text splitting approach that divides text into segments, each with a defined character length (chunk size) and an overlap, allowing for more effective processing of text data.
8. **Box from box**: The Box module allows you to create dictionary-like objects with attribute-style access. It simplifies working with dictionaries and JSON data by providinga more user-friendly interface.

APPENDIX-2

```
from matplotlib import pyplot as plt
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
tumor_dir = r'C:\Users\91817\Downloads\archive (2)\Training\yes_Tumor'
non_tumor_dir = r'C:\Users\91817\Downloads\archive (2)\Training\no_tumor'

# Initialize lists to store images and labels
images = []
labels = []
# Define the target size for resizing
target_size = (64, 64)
# Load tumor images
for filename in os.lisdir(tumor_dir):
    if filename.endswith('.jpg'):
        image = cv2.imread(os.path.join(tumor_dir, filename))
        # Resize the image to the target size
        image = cv2.resize(image, target_size)
        # Perform any necessary preprocessing
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
        # Call the crop_brain_contour function
        images.append(image)
        labels.append(1)  # Tumor images are labeled as 1
# Load non-tumor images
for filename in os.listdir(non_tumor_dir):
    if filename.endswith('.jpg'):
        image = cv2.imread(os.path.join(non_tumor_dir, filename))
        # Resize the image to the target size
        image = cv2.resize(image, target_size)
        # Perform any necessary preprocessing
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
        # Call the crop_brain_contour function
        images.append(image)
        labels.append(0)  # Non-tumor images are labeled as 0
# Ensure that all images have the same dimensions
images = [cv2.resize(image, target_size) for image in images]
# Convert image and label lists to numpy arrays
images = np.array(images)
labels = np.array(labels)
import matplotlib.pyplot as plt
# Select a few sample images to display
sample_images = images[:5]  # Display the first 5 images as samples
# Define a function to display the images
def display_images(images, titles):
    fig, axes = plt.subplots(1, len(images), figsize=(12, 3))
    for i, ax in enumerate(axes):
        ax.imshow(images[i], cmap='gray')
        ax.set_title(titles[i])
        ax.axis('off')
    plt.show()
# Titles for the sample images
sample_titles = ["Sample Image 1", "Sample Image 2", "Sample Image 3", "Sample Image 4", "Sample
Image 5"]
# Display the sample images
display_images(sample_images, sample_titles)
def crop_brain_contour(image, plot=False):
# Convert the image to grayscale, and blur
    if len(image.shape) == 2:
        gray = image
    else:
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (5, 5), 0)
# Threshold the image, then perform a series of erosions + dilations to remove any small regions of
noise
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)
# Find contours in thresholded image, then grab the largest one
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
# At least one contour was found
    if cnts:
        c = max(cnts, key=cv2.contourArea)
        # Find the extreme points
        extLeft = tuple(c[c[:, :, 0].argmin()][0])
        extRight = tuple(c[c[:, :, 0].argmax()][0])
        extTop = tuple(c[c[:, :, 1].argmin()][0])
        extBot = tuple(c[c[:, :, 1].argmax()][0]
# Crop a new image out of the original image using the four extreme points (left, right, top, bottom)
        new_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]
    else:
# No contours found, return the original image
        new_image = image
    if plot:
        plt.figure(
        plt.subplot(1, 2, 1)
        plt.imshow(image)
        plt.tick_params(axis='both', which='both',
                        top=False, bottom=False, left=False, right=False,
                labelbottom=False, labeltop=False, labelleft=False, labelright=False)
        plt.title('Original Image')
        plt.subplot(1, 2, 2)
        plt.imshow(new_image
        plt.tick_params(axis='both', which='both',
                top=False, bottom=False, left=False, right=False,
                labelbottom=False, labeltop=False, labelleft=False, labelright=False
        plt.title('Cropped Image'
        plt.show(
    return new_imag
ex_img = cv2.imread(r'C:\Users\91817\Downloads\archive (2)\Training\yes_tumor\p (88).jpg')
ex_new_img = crop_brain_contour(ex_img, True)
def extract_hog_features(images):
    features = []
    for image in images:
        # Resize and convert to grayscale if needed
        image = cv2.resize(image, (64, 64))
        if len(image.shape) == 3:  # Ensure grayscale
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Extract HOG features parameters
        feature_vector = hog(image, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=False)
        #cell size, block size, block normalization
        features.append(feature_vector)
        # return the feature matrix
    return np.array(features)
# Define X and y based on dataset and extracted features
X = extract_hog_features(images)  # X should be the feature matrix
y = labels  # y should be the corresponding labels
sample_image = images[100]
# Check if the sample image is valid and its shape contains the expected dimensions
```

```
if sample_image is not None and len(sample_image.shape) == 2 and sample_image.shape[0] > 0 and
sample_image.shape[1] > 0:
   # Adjust these parameters based on image size
   pixels_per_cell = (8, 8)
   cells_per_block = (2, 2)
   # Calculate HOG features
   fd, hog_image = hog(sample_image, pixels_per_cell=pixels_per_cell,
cells_per_block=cells_per_block, visualize=True)
   # Plot the HOG features
   fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 3), sharex=True, sharey=True)
   ax1.imshow(sample_image, cmap=plt.cm.gray)
   ax1.set_title('Sample Image')
   ax1.axis('off')
   ax2.imshow(hog_image, cmap=plt.cm.gray)
   ax2.set_title('HOG Features')
   ax2.axis('off')
   plt.show()
else:
   print("Invalid sample image or dimensions.")
#extracting hog features in .xlsx file
hog_features = extract_hog_features(images)
# Create a DataFrame to store HOG features
data = pd.DataFrame(hog_features
# Add labels to the DataFrame
data['Labels'] = label
# Save the DataFrame to an Excel file
excel_file = 'hog_features.xlsx'
data.to_excel(excel_file, index=False
print(f'HOG features saved to {excel_file}')
data.head()
# Ensure that X is a 2D array
if len(X.shape) == 1:
    X = np.array([x.flatten() for x in X])
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print ("number of training examples = " + str(X_train.shape[0]))
print ("number of test examples = " + str(X_test.shape[0]))
print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(y_train.shape))
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(y_test.shape)
scaler = StandardScaler(
#Standardization data and involves scaling features to have a mean and standard deviation .
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
from sklearn.model_selection import GridSearchCV
# Define the parameter grid for hyperparameter tuning
param_grid = (
    # C represents the regularization parameter in an SVM.
    'C': [0.1, 1, 10],
    #gamma defines how much influence each training example has in determining the shape of the
decision boundary.
    'gamma': [0.001, 0.01, 0.1],
    #The kernel parameter specifies type of kernel function to be used in the SVM.(Radial basis
function= non linear dataset)
    'kernel': ['rbf'],
}
# Create an SVM classifier
clf = GridSearchCV(svm.SVC(kernel='linear', probability=True), param_grid, cv=3)
# Train the SVM with hyperparameter tuning
clf.fit(X_train, y_train)
# Get the best hyperparameters
```

```
best_params = clf.best_params_
print(f"Best hyperparameters: {best_params}")
# Make predictions on the test set
y_pred = clf.predict(X_test)
y_prob = clf.predict_proba(X_test)[:, 1]  # Probability of class 1 (tumor)
#Evaluate the model using ROC
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Evaluate the SVM model
accuracy_SVM = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy_SVM * 100:.2f}%")
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
# Standardize the features
scaler = StandardScaler()
X_train_KNN = scaler.fit_transform(X_train
X_test_KNN = scaler.transform(X_test)

# Define the parameter grid for hyperparameter tuning
param_grid_KNN = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
}
# Create a KNN classifier
knn = KNeighborsClassifier()
# Create a GridSearchCV object
clf_KNN = GridSearchCV(knn, param_grid_KNN, cv=3)
```

```
# Fit the model
clf_KNN.fit(X_train_KNN, y_train)

# Get the best hyperparameters
best_params_KNN = clf_KNN.best_params_

# Make predictions
y_pred_KNN = clf_KNN.predict(X_test_KNN)

# Calculate accuracy
accuracy_KNN = accuracy_score(y_test, y_pred_KNN)
print("Best KNN Hyperparameters:", best_params_KNN)
print(f"Accuracy KNN: {accuracy_KNN * 100:.2f}%")
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

# Standardize the features
scaler = StandardScaler()
X_train_LR = scaler.fit_transform(X_train)
X_test_LR = scaler.transform(X_test)
# Define the parameter grid for hyperparameter tuning
param_grid_LR = {
    'C': [0.1, 1, 10]
}
# Create a Logistic Regression classifier
lr = LogisticRegression()
# Create a GridSearchCV object
clf_LR = GridSearchCV(lr, param_grid_LR, cv=3)
```

```
# Fit the model
clf_LR.fit(X_train_LR, y_train)
# Get the best hyperparameters
best_params_LR = clf_LR.best_params_
# Make predictions
y_pred_LR = clf_LR.predict(X_test_LR)
# Calculate accuracy
accuracy_LR = accuracy_score(y_test, y_pred_LR)
print("Best Hyperparameters:", best_params_LR)
print(f"Accuracy LR: {accuracy_LR * 100:.2f}%")
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
# Define the parameter grid for hyperparameter tuning
param_grid_DT = {
   'max_depth': [None, 10, 20, 30],
   'min_samples_split': [2, 5, 10],
   'min_samples_leaf': [1, 2, 4],
}
# Create a Decision Tree classifier
dt = DecisionTreeClassifier()
# Create a GridSearchCV object
clf_DT = GridSearchCV(dt, param_grid_DT, cv=3)
# Fit the model
clf_DT.fit(X_train, y_train)
# Get the best hyperparameters
best_params_DT = clf_DT.best_params_
# Make predictions
y_pred_DT = clf_DT.predict(X_test)
# Calculate accuracy
```

```
accuracy_DT = accuracy_score(y_test, y_pred_DT)
print("Best Hyperparameters DT:", best_params)
print(f"Accuracy DT: {accuracy_DT * 100:.2f}%")
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Print accuracy and FPR
print("Accuracy: {:.2f}".format(accuracy))
print("False Positive Rate (FPR): {:.2f}".format(fpr))

def compute_f1_score(y_true, prob):
    # convert the vector of probabilities to a target vector
    y_pred = np.where(prob > 0.5, 1, 0)

    score = f1_score(y_true, y_pred)

    return score

f1score = compute_f1_score(y_test, y_prob)
print(f"F1 score: {f1score}")

#true positives between 0(low precision) and 1(high precision)

def data_percentage(y):

    m=len(y)
    n_positive = np.sum(y)
    n_negative = m - n_positive

    pos_prec = (n_positive* 100.0)/ m
    neg_prec = (n_negative* 100.0)/ m

    print(f"Number of examples: {m}")
    print(f"Percentage of positive examples: {pos_prec}%, number of pos examples: {n_positive}")
    print(f"Percentage of negative examples: {neg_prec}%, number of neg examples: {n_negative}")

print("Training Data:")
data_percentage(y_train)

print("Testing Data:")
data_percentage(y_test)

import matplotlib.pyplot as plt

# Data
labels = 'Training Data', 'Testing Data'
sizes = [len(y_train), len(y_test)]
colors = ['lightblue', 'lightgreen']
explode = (0.1, 0)  # Explode the first slice (Training Data)

# Create pie chart for training and testing data
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct=lambda p: '{:.0f}
({:.1f}%)'.format(p * sum(sizes) / 100, p), shadow=True, startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.title("Distribution of Training and Testing Data")
plt.show()

import matplotlib.pyplot as plt

def data_percentage(y):
    m = len(y)
    n_positive = np.sum(y)
    n_negative = m - n_positive
    return n_positive, n_negative

# Get the number of positive and negative examples for training and testing data
train_pos, train_neg = data_percentage(y_train)
test_pos, test_neg = data_percentage(y_test)

# Data
labels = ['Training Positive', 'Training Negative', 'Testing Positive', 'Testing Negative']
sizes = [train_pos, train_neg, test_pos, test_neg]
colors = ['lightblue', 'lightcoral', 'lightgreen', 'lightsalmon']
explode = (0.1, 0, 0.1, 0)  # Explode the Positive slices

# Create pie chart for training and testing data
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True,
startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.title("Distribution of Positive and Negative Examples")
plt.show()

import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import svm

# Generate a simple dataset
X, y = datasets.make_classification(n_samples=100, n_features=2, n_informative=2, n_redundant=0,
random_state=42)
```

```
# Train a binary classifier (SVM)
clf = svm.SVC(kernel='linear')
clf.fit(X, y)

# Create a mesh grid over the feature space
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))

# Make predictions on the mesh grid
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

# Plot the data points with different colors for each class
scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k', s=100, alpha=0.7)

# Add a color bar to the right
plt.colorbar(scatter)

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Boundary of the Classifier')
plt.show()
```

Format - I

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

Office of Controller of Examinations

REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION/PROJECT REPORTS FOR UG/PG PROGRAMMES

(To be attached in the dissertation/ project report)

1	Name of the Candidate **(IN BLOCK LETTERS)**	KARTIK SINGH, UTKARSH GUPTA
2	Address of the Candidate	Y-403, Abode Valley, Kakkan Street, Potheri, Kattankulathur, Chennai, TN-603203
3	Registration Number	RA2011003010810, RA2011003010772
4	Date of Birth	11 April 2002, 6 July 2002
5	Department	Computer Science and Engineering
6	Faculty	Engineering and Technology, School of Computing
7	Title of the Dissertation/Project	Brain Tumor Detection Using HOG and SVM
8	Whether the above project /dissertation is done by	Individual or group : (Strike whichever is not applicable) **a)** If the project/ dissertation is done in group, then how many students together completed the project : 2(Two) b) Mention the Name & Register number of other candidates : KARTIK SINGH, RA2011003010810 UTKARSH GUPTA, RA2011003010772
9	Name and address of the Supervisor / Guide	Mrs. P. Renukadevi Assistant Professor Department of Computing Technologies SRM Institute of Science and Technologies Kattankulathur - 603203 **Mail ID:** renukadp@srmist.edu.in **Mobile Number:** 9962200670
10	Name and address of Co-Supervisor / Co- Guide (if any)	**NIL**

11	Software Used	Turnitin		
12	Date of Verification	06 November 2023		
13	**Plagiarism Details: (to attach the final report from the software)**			
Chapter	**Title of the Chapter**	**Percentage of similarity index (including self citation)**	**Percentage of similarity index (Excluding self-citation)**	**% of plagiarism after excluding Quotes, Bibliography, etc.,**
1	INTRODUCTION	1%	1%	1%
2	LITERATURE SURVEY	2%	2%	2%
3	ARCHITECTURE DESIGN AND ANALYSIS	3%	3%	3%
4	IMPLEMENTATION	0%	0%	0%
5	RESULTS AND ANALYSIS	1%	1%	1%
6	CONCLUSION	0%	0%	0%
7	FUTURE WORK	1%	0%	1%
8				
9				
10				
Appendices		2%	2%	2%
I / We declare that the above information have been verified and found true to the best of my / our knowledge.				
Signature of the Candidate		**Name & Signature of the Staff (Who uses the plagiarism check software)**		
Name & Signature of the Supervisor/ Guide		**Name & Signature of the Co-Supervisor/Co-Guide**		
Name & Signature of the HOD				

karthik curreent report

ORIGINALITY REPORT

7%	4%	2%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.diva-portal.org Internet Source	2%
2	Submitted to University of Witwatersrand Student Paper	1%
3	Submitted to Pima Community College Student Paper	<1%
4	Submitted to Texas A&M University, College Station Student Paper	<1%
5	Submitted to The University of Manchester Student Paper	<1%
6	Submitted to Malaviya National Institute of Technology Student Paper	<1%
7	www.ryangineer.com Internet Source	<1%
8	Submitted to Liverpool John Moores University Student Paper	<1%

9	Naveed Islam, Yasir Faheem, Ikram Ud Din, Muhammad Talha, Mohsen Guizani, Mudassir Khalil. "A blockchain-based fog computing framework for activity recognition as an application to e-Healthcare services", Future Generation Computer Systems, 2019 Publication	<1 %
10	Wahl, E.R.. "A general framework for determining cutoff values to select pollen analogs with dissimilarity metrics in the modern analog technique", Review of Palaeobotany and Palynology, 200402 Publication	<1 %
11	Submitted to University of Salford Student Paper	<1 %
12	Submitted to University of Ulster Student Paper	<1 %
13	Submitted to University of Surrey Student Paper	<1 %
14	International Journal of Clothing Science and Technology, Volume 26, Issue 1 (2014-03-28) Publication	<1 %
15	Submitted to University of Northumbria at Newcastle Student Paper	<1 %
16	augusta.openrepository.com Internet Source	<1 %

17	www.jatit.org Internet Source	<1%
18	fibroidsuterine.net Internet Source	<1%

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		

PAPER PUBLICATION PROOF

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

6th International Conference On Recent Trends In Advanced Computing : Submission (442) has been created.
1 message

Microsoft CMT <email@msr-cmt.org> Tue, Oct 31, 2023 at 11:47 PM
Reply-To: Microsoft CMT - Do Not Reply <noreply@msr-cmt.org>
To: ks9124@srmist.edu.in

Hello,

The following submission has been created.

Track Name: ICRTAC2023

Paper ID: 442

Paper Title: Brain Tumor Detection Using HOG and SVM

Abstract:
Our project focuses on the development of an effective and precise brain tumor detection system using Histogram of Oriented Gradients (HOG) and Support Vector Machines (SVM). Timely detection of brain tumors is of paramount importance for successful treatment and patient well-being. The project encompasses several vital phases, commencing with the enhancement of MRI image quality through preprocessing techniques. Subsequently, we extract pertinent features from these images and employ diverse machine learning models for both training and testing. This innovative approach harnesses the potential of machine learning to discern intricate patterns and relationships within medical images, ultimately leading to accurate tumor classification. The successful implementation of this project holds the promise of making a significant contribution to the medical field by providing a reliable tool for prompt and precise brain tumor diagnosis, thereby advancing patient care and streamlining treatment planning.

Created on: Tue, 31 Oct 2023 18:16:54 GMT

Last Modified: Tue, 31 Oct 2023 18:16:54 GMT

Authors:
- ks9124@srmist.edu.in (Primary)
- ug3189@srmist.edu.in

Primary Subject Area: Artificial Intelligence and Machine Learning

Secondary Subject Areas: Not Entered
Submission Files: Research_Paper_B497.pdf (417 Kb, Tue, 31 Oct 2023 18:16:33 GMT)

Thanks,
CMT team