BRAIN TUMOR DETECTION USING IMAGE ANALYSIS & MACHINE LEARNING ALGORITHMS

PROJECT REPORT

A Report submitted in partial fulfillment of the requirement for

Real Time Project (B.Tech)

in

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

S. SADIYA AIMAN - 23011A0415

I.GAGAN - 23011A0430

Y. POOJITHA - 23011A0446

J. MAHESH - 23011A0447

T. SWAPNARANI - 24015A0409

Under the guidance of

Mrs. B. Manasa
Assistant professor (C)

University College of Engineering, Science & Technology Hyderabad

Jawaharlal Nehru Technological University Hyderabad

Kukatpally, Hyderabad - 500 085, Telangana, India

2024– 2025

University College of Engineering, Science & Technology Hyderabad

Jawaharlal Nehru Technological University Hyderabad

Kukatpally, Hyderabad - 500 085, Telangana, India

2024 – 2025

Department of Electronics and Communications Engineering

CERTIFICATE BY THE SUPERVISOR

This is to certify that this dissertation work titled **"Brain Tumor Detection Using Image Analysis and Machine Learning Algorithms"** being submitted by **S.Sadiya Aiman (23011A0415), I.Gagan (23011A0430), Y.Poojitha (23011A0446), J.Mahesh (23011A0447), T.Swapnarani (24015A0409)** in partial fulfilment of the requirement for the award of degree of **Bachelor of Technology (Regular) in Electronics and Communication Engineering** from **JNTUHUCESTH** during the academic year 2024- 2025 is a record of Bonafide work carried out by them, under my supervision. This dissertation has not been submitted to any other university or institution for the award of any other degree. The results are verified and found to be satisfactory.

PROJECT SUPERVISOR
Mrs. B. Manasa,
Assistant Professor (C),
Department of ECE,
JNTUHUCESTH.

University College of Engineering, Science & Technology Hyderabad

Jawaharlal Nehru Technological University Hyderabad

Kukatpally, Hyderabad - 500 085, Telangana, India

2024 – 2025

Department of Electronics and Communications Engineering

CERTIFICATE BY THE HEAD OF THE DEPARTMENT

This is to certify that this dissertation work titled **"Brain Tumor Detection Using Image Analysis and Machine Learning Algorithms"** being submitted by **S.Sadiya Aiman (23011A0415), I.Gagan (23011A0430), Y.Poojitha (23011A0446), J.Mahesh (23011A0447), T.Swapnarani (24015A0409)** in partial fulfilment of the requirement for the award of degree of **Bachelor of Technology (Regular)** in **Electronics and Communication Engineering** from **JNTUHUCESTH** during the academic year 2024-2025 is a record of bonafide work carried out by them, under my supervision. This dissertation has not been submitted to any other university or institution for the award of any other degree. The results are verified and found to be satisfactory.

Dr.T.Madhavi Kumari
Professor & Head,
Department of ECE,
JNTUHUCESTH.

University College of Engineering, Science & Technology Hyderabad

Jawaharlal Nehru Technological University Hyderabad

Kukatpally, Hyderabad - 500 085, Telangana, India

2024 – 2025

Department of Electronics and Communications Engineering

DECLARATION OF THE CANDIDATE

We, **S.Sadiya Aiman (23011A0415), I.Gagan (23011A0430), Y.Poojitha (23011A0446), J.Mahesh (23011A0447), T.Swapnarani (24015A0409)** hereby certify that the project report entitled as **"Brain Tumor Detection Using Image Analysis and Machine Learning Algorithms"** is a bonafide record work done and submitted under the guidance of **Mrs.B,Manasa , Assistant Professor (C), Department of ECE, JNTUHUCESTH**, in particular fulfilment of the requirements for the award of the Degree of Bachelor of Technology (Regular) in Electronics and Communication Engineering during the academic year 2024-2025. This is a record of bonafide work carried out by us and the results embodied in this project have not been reproduced/copied from any source and have not been submitted to any other University or Institute for the award of any other degree or diploma.

S.Sadiya Aiman (23011A0415)
I. Gagan (23011A0430)
Y. Poojitha (23011A0446)
J. Mahesh (23011A0447)
T. Swapnarani (24015A0409)
Department of ECE, JNTUH UCESTH.

ACKNOWLEDGEMENTS

We wish to express our sincere thanks to our project Guide **Mrs. B. Manasa, Assistant Professor ECE, JNTUHUCESTH**, for continuous guidance and suggestions in carrying out the project work in the college.

We are Thankful to **Dr. T. Madhavi Kumari, Professor and Head, Department of ECE, JNTUHUCESTH** for supporting us to carry out the project work in the college.

We write this acknowledgement with great honor, pride, and pleasure to pay our respects to the people who enabled us either directly or indirectly in the accomplishment of this project.

We are immensely thankful to each and every faculty member of the ECE Department for their relentless contribution towards the successful completion of our course.

S.Sadiya Aiman (23011A0415)
I.Gagan (23011A0430)
Y.Poojitha (23011A0446)
J.Mahesh (23011A0447)
T.Swapnarani (24015A0409)

ABSTRACT

Accurate and early detection of brain tumors through MRI analysis is essential for effective diagnosis and treatment planning. This study presents a machine learning-based approach that exclusively utilizes Histogram of Oriented Gradients (HOG) for feature extraction, capitalizing on its ability to capture local edge and shape information critical to distinguishing tumor-affected regions in brain scans. By relying solely on HOG descriptors, the system ensures a lightweight, interpretable, and computationally efficient pipeline suitable for practical deployment.

The classification framework is built as an ensemble of three supervised learning models: K-Nearest Neighbors (KNN), Linear Support Vector Machine (SVM), and Polynomial SVM. Each classifier contributes uniquely—KNN focuses on local data distributions, Linear SVM provides efficiency for linearly separable features, and Polynomial SVM models more complex non-linear relationships. Their combined predictions, aggregated through a majority voting mechanism, enhance the system's overall robustness and generalization ability.

Extensive experimentation on a labeled brain MRI dataset demonstrates that the ensemble achieves high classification accuracy, validating the effectiveness of handcrafted HOG features in capturing tumor characteristics. The proposed method provides a compelling alternative to deep learning approaches by offering comparable performance without the need for large-scale training data or high-end computational infrastructure. This makes it especially suited for deployment in resource-constrained environments where fast and reliable diagnostic support is critical.

CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	**Full Form**
MRI	Magnetic Resonance Imaging
HOG	Histogram of Oriented Gradients
KNN	K-Nearest Neighbors
LBP	Local Binary Patterns
GLCM	Gray Level Co-occurrence Matrix
SVM	Support Vector Machine
CNN	Convolutional Neural Network
DWT	Discrete Wavelet Transform
PCA	Principal Component Analysis
ROI	Region of Interest
FLAIR	Fluid-Attenuated Inversion Recovery
BRATS	Brain Tumor Segmentation
RBF	Radial Basis Function
CART	Classification and Regression Tree
MSE	Mean Square Error
OOB	Out-Of-Bag (sample)

LIST OF SYMBOLS

Symbol	Meaning
$\|\|X1 - X2\|\|^2$	Squared Euclidean Distance between two feature vectors
γ	Kernel coefficient in RBF kernel
ξ (xi)	Slack variable in soft-margin SVM
C	Regularization parameter in SVM
w	Weight vector in SVM
x	Input feature vector
b	Bias term in SVM hyperplane equation
K(X1, X2)	Kernel function measuring similarity between X1 and X2
exp	Exponential function
a	Margin Projected to w

CHAPTER 1
INTRODUCTION

1.1 Brain tumors and their complexity

Brain tumor detection from MRI scans is a time-critical task that demands both precision and speed. Manual diagnosis, while effective in the hands of experts, is not scalable in regions with limited medical resources or high patient load. There is an increasing demand for automated systems that can assist radiologists by providing quick and accurate second opinions.

1.2 Approach Overview

This project explores a classical machine learning pipeline designed to perform binary classification of brain MRI images — identifying whether a tumor is present or not. Instead of relying on computationally intensive deep learning models, we focus on handcrafted feature extraction using Histogram of Oriented Gradients (HOG) and traditional classifiers. This makes the system lightweight, interpretable, and well-suited for environments with limited computational power.

1.3 Why Classical Machine Learning

While deep learning has become the go-to solution for image classification tasks, it often comes with high hardware requirements, large data dependencies, and reduced interpretability. In contrast, classical machine learning models — when paired with strong feature descriptors — can achieve comparable accuracy on well-structured tasks with significantly lower resource usage. This approach also provides greater transparency in decision-making, which is valuable in medical applications.

CHAPTER 2
LITERATURE REVIEW

2.1. Introduction

Brain tumors present a significant challenge in medical diagnostics due to their complexity, variability in structure, and potential to be life-threatening. Magnetic Resonance Imaging (MRI) is the most widely used non-invasive imaging technique for brain tumor analysis. To enhance diagnosis and classification accuracy, a large body of research has focused on the use of artificial intelligence (AI), particularly machine learning (ML) and deep learning (DL), in interpreting MRI data. This chapter reviews the existing literature in three major categories: traditional machine learning algorithms, deep learning-based approaches, and hybrid or ensemble methods for brain tumor classification and segmentation.

2.2. Machine Learning Algorithms for Brain Tumor Classification

Numerous studies have explored the application of traditional machine learning techniques for brain tumor detection and classification using MRI data.
Tiwari et al. (2011) proposed a support vector machine (SVM) model to classify brain tumor types based on texture features extracted from MRI scans. [1]
Zhang et al. (2016) implemented a k-nearest neighbor (KNN) approach using gray-level co-occurrence matrix features for accurate tumor segmentation. [2]
Chaplot et al. (2006) applied artificial neural networks (ANN) trained with principal component analysis (PCA) features for tumor detection. [3]
El-Dahshan et al. (2010) introduced a hybrid method combining wavelet transform, PCA, and SVM for brain tumor classification. [4]
Paul et al. (2018) utilized decision tree classifiers to separate benign and malignant tumors, showing moderate accuracy in smaller datasets. [5]

2.3. Deep Learning Approaches for Brain Tumor Detection

Deep learning models, particularly convolutional neural networks (CNNs), have shown superior performance in automated tumor identification and segmentation.
Pereira et al. (2016) proposed a deep CNN model that achieved high accuracy on the BRATS dataset for tumor segmentation. [6]
Hossain et al. (2019) developed a CNN architecture using transfer learning with pretrained models like VGG16 and ResNet. [7]
Afshar et al. (2019) introduced Capsule Networks (CapsNet) to preserve spatial features during tumor classification. [8]
Muhammad et al. (2020) proposed a hybrid deep model combining CNN and LSTM for volumetric tumor analysis in MRI. [9]
Kamnitsas et al. (2017) used 3D CNNs and fully connected CRFs for accurate multi-class tumor segmentation. [10]

2.4. Hybrid and Ensemble Techniques in Tumor Detection

Recent research has focused on hybrid approaches combining multiple machine learning and deep learning techniques to improve detection accuracy.
Raja et al. (2020) proposed a hybrid model combining SVM with CNN feature extraction for better classification results. [11]
Sajjad et al. (2019) introduced a fusion technique using CNN and gradient boosting, achieving robust performance on noisy data. [12]
Swati et al. (2019) implemented an ensemble model combining ResNet and DenseNet for

classification of glioma grades. [13]
Khawaldeh et al. (2020) developed an ensemble of transfer learning models using MobileNet and Xception. [14]
Masood et al. (2015) proposed a wavelet + ANN + AdaBoost model for enhanced brain tumor segmentation. [15]

2.5. Research Gaps

Despite extensive research, several gaps persist:

- **Limited Generalizability:** Many models are trained and tested on the same dataset (e.g., BRATS), reducing confidence in real-world clinical applicability.
- **Lack of Interpretability:** Deep learning models, especially CNNs and hybrid architectures, often function as "black boxes," which can be problematic for medical decision-making.
- **Data Imbalance:** Many studies use imbalanced datasets (e.g., fewer malignant samples), which can bias model performance.
- **Volumetric Analysis Limitations:** Few studies effectively utilize 3D volumetric data despite its importance in tumor assessment.
- **Robustness to Noise:** Existing models may not perform reliably in noisy or lower-quality scans typical of real-world environments.

2.6. Motivation for Research Work:

Early and accurate detection of brain tumors can significantly increase treatment success and patient survival rates. However, manual diagnosis through MRI is time-consuming and prone to human error. With advances in computational power and AI, there is a compelling opportunity to automate and enhance the diagnostic process. The motivation behind this research is to develop a robust and interpretable classification model that leverages the strengths of both machine learning and deep learning, addresses current limitations, and contributes to improved diagnostic accuracy in clinical settings.

2.7. Research Objectives:

The primary objectives of this research are:

1. To investigate and compare the effectiveness of traditional machine learning and deep learning techniques for brain tumor classification using MRI data.
2. To design a hybrid model that integrates handcrafted features with deep learning-based representations for improved classification performance.
3. To enhance model robustness and generalizability by training on diverse and augmented datasets.
4. To address interpretability issues by incorporating visualization techniques such as class activation maps (CAMs) to understand the decision-making process of deep models.

5. To evaluate the proposed model’s performance on both benchmark and real-world datasets, ensuring applicability in clinical environments.

2.8. Project Contributions

The key contributions of this project are as follows:

- Implementation of a HOG-based feature extraction pipeline tailored for brain MRI images.
- Performance comparison of multiple classical classifiers including KNN, SVM (Linear, RBF, Polynomial), and Random Forest.

- Development of an ensemble model that combines the strengths of top-performing classifiers to improve overall accuracy.
- Demonstration that high accuracy can be achieved using traditional machine learning techniques without the need for large datasets or GPUs.

2.9. Conclusion:

The reviewed literature indicates substantial progress in automating brain tumor classification using both traditional and modern AI approaches. Classical machine learning methods such as SVM, KNN, and decision trees have laid a foundation by demonstrating the utility of handcrafted features. However, deep learning models, particularly convolutional neural networks (CNNs), have significantly outperformed traditional approaches in segmentation and classification tasks due to their ability to learn complex patterns directly from raw data. Furthermore, hybrid and ensemble models have shown promise by integrating the strengths of multiple techniques, improving overall robustness and accuracy. Despite these advancements, key challenges such as model generalization, data scarcity, and interpretability remain.

CHAPTER 3
METHODOLOGY

3.1 Introduction:

In this chapter, we present the methodology adopted for developing our brain tumor detection system using image analysis and machine learning algorithms, it is shown in **Figure 3.1**. The goal of the project is to accurately classify brain MRI images as either tumor-affected or normal using classical machine learning techniques. To achieve this, we have structured our methodology into multiple stages including dataset preparation, image preprocessing, feature extraction, model training, and performance evaluation. Given our computational limitations, we focused on lightweight yet effective feature descriptors and algorithms that can run efficiently on standard hardware. This chapter outlines each step of our approach in detail, providing the rationale behind the choices made and their relevance to the problem at hand.

3.2 Data Collection:

The dataset used in this project consists of brain MRI images that were collected from a publicly available source. It contains a total of 3120 grayscale MRI images, out of which 1550 images represent tumor cases and 1570 images represent normal (no tumor) cases. The dataset includes axial slices of the brain with visible tumor regions in various shapes, sizes, and positions, offering sufficient diversity for meaningful analysis.

Each image was initially examined to ensure clarity and relevance. Basic preprocessing steps such as renaming files and organizing them into respective class folders (Tumor and No Tumor) were performed for ease of access during feature extraction and training.

This dataset was chosen because it provides a balanced and real-world representation of brain MRI images required for binary classification tasks. Although limited in size, it was adequate for training classical machine learning models when combined with effective feature extraction techniques.

3.3 Image Preprocessing:

The brain tumor photos were subjected to a series of preprocessing stages aimed at standardizing the dataset so that it could be used in classification problems. Here is a rundown of what was undertaken in advance: The MRI photographs were converted to grayscale, creating a monochrome version of the pictures. The data were simplified, and the computing burden was lessened as a result. Images were resized such that they all had the same 128×128 resolution. This action guaranteed that all photos were the same size, guaranteeing uniformity in the subsequent processing steps.

3.3.1 Gray Scaling:

All MRI images were converted to grayscale using OpenCV's cv2.imread function with the cv2.IMREAD_GRAYSCALE flag. This step reduces the image to a single channel, minimizing computational load while retaining important structural information relevant to brain tumor detection.

3.3.2 Resizing:

Each grayscale image was resized to a fixed dimension of **128×128 pixels** using cv2.resize. This uniform sizing ensures that all input images have the same shape, which is a requirement for most machine learning models.

3.3.3 Normalizing Pixel Values:

The pixel values of the resized images were normalized to a range of **[0, 1]** by dividing each pixel value by 255.0. Normalization helps in accelerating the training process and improves the model's convergence by ensuring that input features have a consistent scale.
At the end of preprocessing, the image data and corresponding labels were stored in NumPy arrays and saved as "data.npy" and "labels.npy" for efficient access during model training.

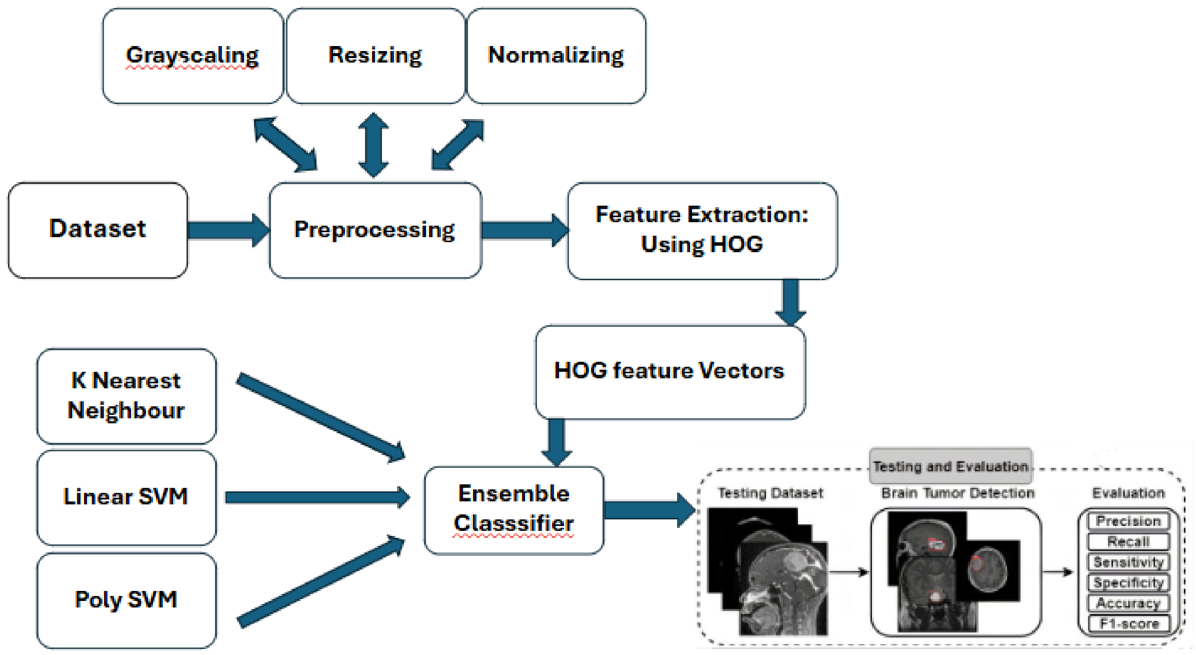


Figure 3.1 Methodology

3.4 Feature Extraction:

To effectively classify brain MRI images, it is essential to extract meaningful features that capture important structural and textural information. In this project, we used Histogram of Oriented Gradients (HOG) as the sole feature descriptor due to its efficiency and ability to highlight object shapes and edges, which are often distorted or disrupted in the presence of tumors.

HOG works by computing the gradient orientation and magnitude in localized regions of an image, forming a histogram of edge directions for each cell. These histograms are then normalized over overlapping blocks, making the feature robust to illumination and contrast variations. Since brain tumors often alter the contour and internal structures of brain tissue, HOG proved effective in capturing these anomalies.

Using the skimage library in Python, HOG features were extracted from each preprocessed grayscale MRI image, resulting in high-dimensional feature vectors that were later scaled and used for model training.

3.5 Feature Scaling:

After extracting Histogram of Oriented Gradients (HOG) features from the MRI images, the resulting feature vectors had high dimensionality with varying ranges. To ensure that all features contribute equally to the model's learning process, we applied feature scaling.

We used StandardScaler from the scikit-learn library to standardize the features so that they have a mean of 0 and a standard deviation of 1. This step is particularly important for distance-based models like K-Nearest Neighbors (KNN) and for improving the performance and convergence of Support Vector Machines (SVMs). Feature scaling thus played a crucial role in ensuring optimal model performance.

3.6 Training and Evaluation of Machine Learning Models:

The extracted HOG feature vectors were used to train multiple classical machine learning classifiers. The models trained include:

K-Nearest Neighbors (KNN)

Support Vector Machine (SVM) with Linear, RBF, and Polynomial kernels.

Random Forest Classifier

Ensemble model combining KNN, Polynomial SVM, and RBF SVM via majority voting.

Each model was trained using the scikit-learn library. After training, the models were evaluated using accuracy, precision, recall, and F1-score. These metrics were calculated on a held-out test set to ensure unbiased performance evaluation.

3.7 Final Model Selection:

After feature extraction and scaling, several machine learning models were trained and evaluated using only HOG features. The models and their respective accuracies were:

K-Nearest Neighbors (KNN): 96.47%

Linear SVM: 95.35%

RBF SVM: 96.96%

Polynomial SVM: 96.79%

Random Forest: 90.87%

To further enhance performance, we also implemented an ensemble model combining the predictions of KNN, Polynomial SVM, and RBF SVM using majority voting. This ensemble achieved the highest classification accuracy of 97.44%.

Based on these results, the ensemble model was selected as the final classifier due to its superior accuracy and robustness. It effectively leveraged the strengths of multiple high-performing models to deliver more reliable predictions.

3.8 Conclusion:

The methodology presented in this chapter forms the foundation of our brain tumor detection system. By following a systematic pipeline (from image preprocessing to feature extraction and classification) we have aimed to ensure both accuracy and efficiency. The chosen features and machine learning models were selected after careful consideration of performance, interpretability, and hardware feasibility. The subsequent chapters will discuss the results obtained using this methodology and provide insights into the effectiveness of our approach.

Chapter 4
Feature Extraction

4.1 Introduction:

In image classification tasks, extracting meaningful features from raw pixel data is crucial for improving model accuracy. One such powerful feature descriptor is the Histogram of Oriented Gradients (HOG), which captures the structure and shape information of objects in an image.

Histogram of Oriented Gradients (HOG) is a feature descriptor used to detect objects in images. It works by computing the gradients of pixel intensity and encoding the distribution of gradient orientations into histograms within small spatial regions of the image.

4.2 Step-by-Step Working of HOG:

4.2.1 Image Preprocessing:

The **Figure 4.1** shows the grayscale and resized (128x128) image, used to simplify computations.

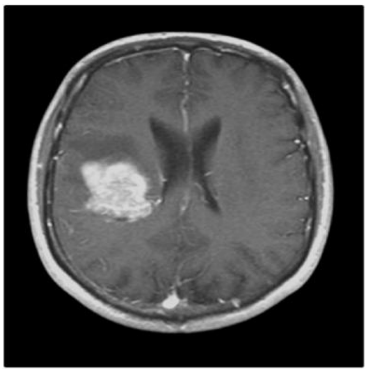


Figure 4.1 Grayscale and resized

Figure 4.2 shows the image divided into cells of (8x8) pixels.

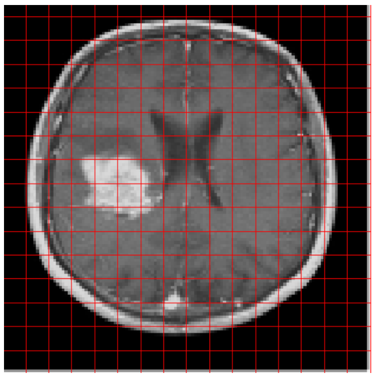
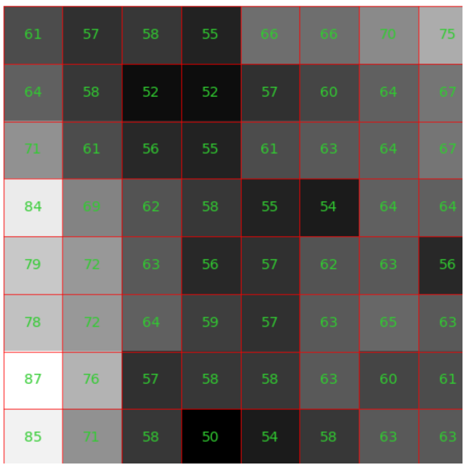


Figure 4.2 Divided into cells of 8x8 pixel

Now let's zoom into one cell and get the grayscale intensity values of each pixel as shown in **Figure 4.3.**



61	57	58	55	66	66	70	75
64	58	52	52	57	60	64	67
71	61	56	55	61	63	64	67
84	69	62	58	55	54	64	64
79	72	63	56	57	62	63	56
78	72	64	59	57	63	65	63
87	76	57	58	58	63	60	61
85	71	58	50	54	58	63	63

Figure 4.3 Grayscale intensities of the 8x8 pixel cell

4.2.2 Calculating the Gradients:

To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients.

To calculate the gradient in x-direction for a pixel, subtract the left side value from the right-side value and for gradient in y-direction subtract the bottom value from the top value.

For eg: grad(x-direction) = |52-64|=12

grad(y-direction) = |57-61|=4

Next, we can find the magnitude and direction of gradient using the following formula

$$g = \sqrt{(g_x^2 + g_y^2)}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

At every pixel, the gradient has a magnitude and direction.

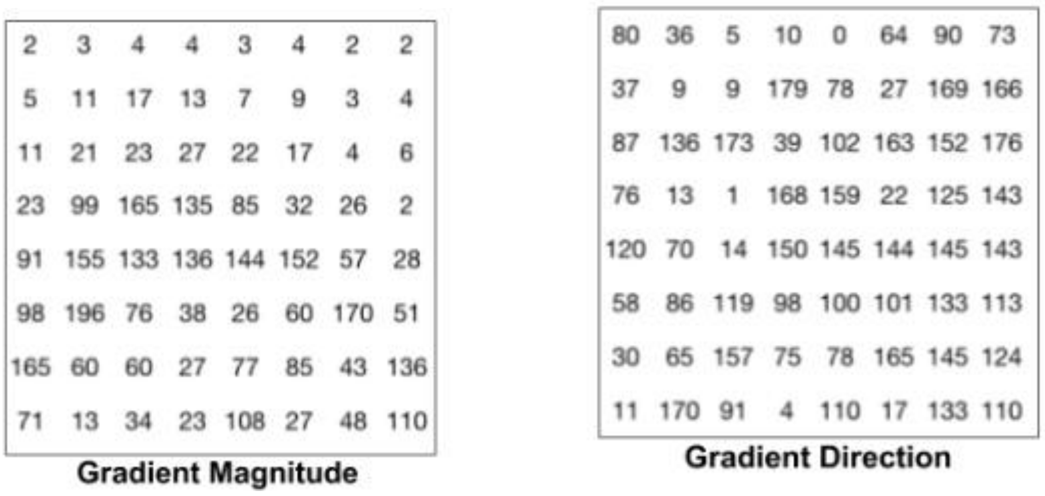
4.2.3 Calculating Histogram of Gradients for 8x8 cell:

One of the important reasons to use a feature descriptor to describe a patch of an image is that it provides a compact representation.

The gradient of this patch contains 2 values (magnitude and direction) per pixel which adds up to 8x8x2 = 128 numbers.

We will see how these 128 numbers are represented using a 9-bin histogram which can be stored as an array of 9 numbers. Not only is the representation more compact, calculating a histogram over a patch makes this representation more robust to noise. Individual gradients may have noise, but a histogram over 8×8 patch makes the representation much less sensitive to noise.

The histogram is essentially a vector (or an array) of 9 bins (numbers) corresponding to angles 0, 20, 40, 60 … 160.



Gradient Magnitude

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Direction

80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Figure 4.4 Gradients

We see the raw numbers representing the gradients in the 8×8 cells with one minor difference — the angles are between 0 and 180 degrees instead of 0 to 360 degrees. These are called "unsigned" gradients because a gradient and it's negative are represented by the same numbers. In other words, a gradient arrow and the one 180 degrees opposite to it are considered the same.

The next step is to create a histogram of gradients in these 8×8 cells. The histogram contains 9 bins corresponding to angles 0, 20, 40 … 160. The following **Figure 4.5** illustrates the process. We are looking at the magnitude and direction of the gradient of the same 8×8 patch as in the previous **Figure 4.4**
A bin is selected based on the direction, and the vote (the value that goes into the bin) is selected based on the magnitude. Let's first focus on the pixel encircled in blue. It has an angle (direction) of 80 degrees and a magnitude of 2. So, it adds 2 to the 5th bin. The gradient at the pixel encircled using red has an angle of 10 degrees and magnitude of 4. Since 10 degrees is halfway between 0 and 20, the vote by the pixel splits evenly into the two bins.

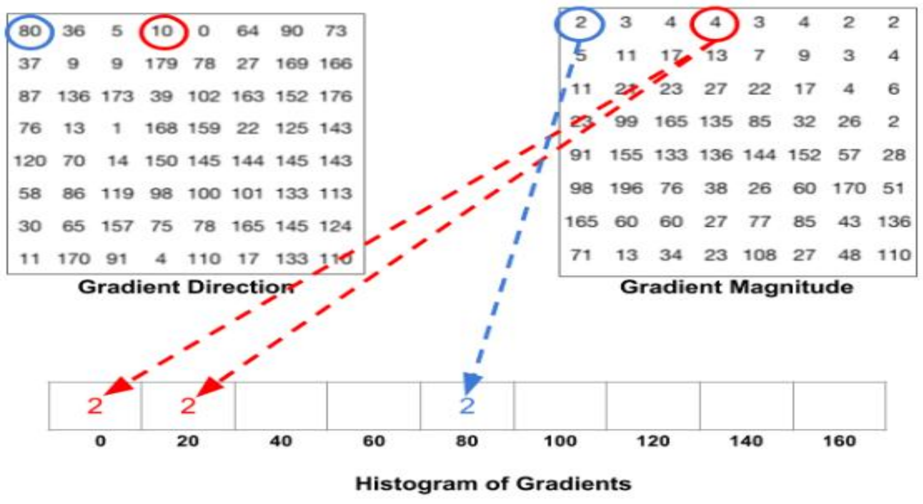
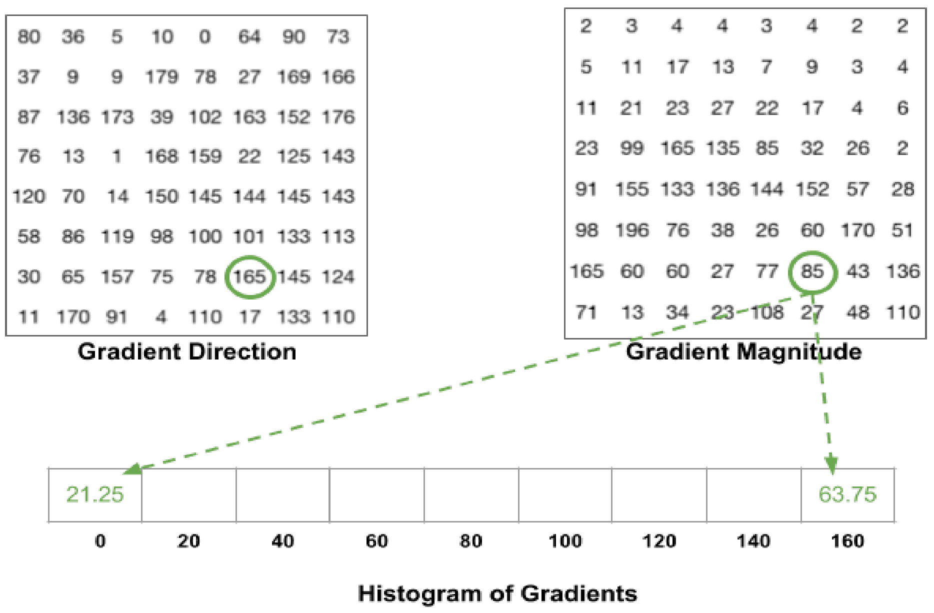


Figure 4.5 Steps to Build HOG

There is one more detail to be aware of. If the angle is greater than 160 degrees, it is between 160 and 180, and we know the angle wraps around making 0 and 180 equivalent. So, in the example below, the pixel with angle 165 degrees contributes proportionally to the 0-degree bin and the 160-degree bin.



80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

0	20	40	60	80	100	120	140	160
21.25								63.75

Histogram of Gradients

Figure 4.5 Steps to build HOG

The contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram. For the patch above, it looks like the **Figure 4.6.**

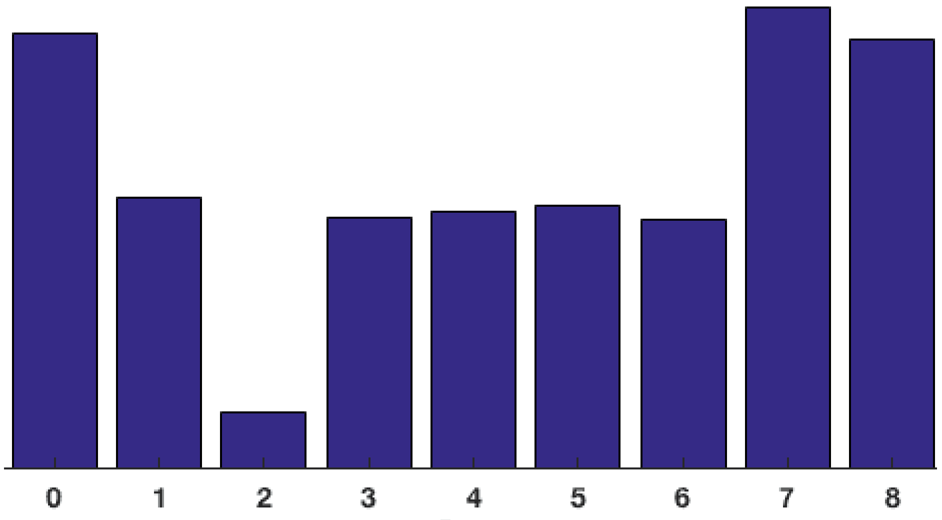


Figure 4.6 Histogram of Gradients

4.2.4 16x16 Block Normalization:

In the previous step, we created a histogram based on the gradient of the image. Gradients of an image are sensitive to overall lighting. If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half.

Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to "normalize" the histogram, so they are not affected by lighting variations.

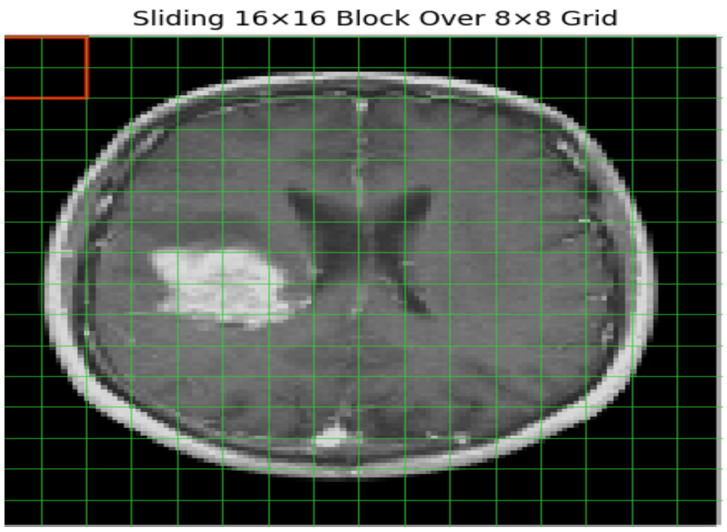


Figure 4.7 16x16 Block Normalization

A 16×16 block has 4 histograms which can be concatenated to form a 36 x 1 element vector, and it can be normalized. The window is then moved by 8 pixels and a normalized 36×1 vector is calculated over this window and the process is repeated as shown in **Figure 4.7.**

4.2.5 Calculate the Histogram of Oriented Gradients feature vector:

To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector. What is the size of this vector? Let us calculate

1. How many positions of the 16×16 blocks do we have? There are 15 horizontal and 15 vertical positions making a total of 15 x 15 = 225 positions.
2. Each 16×16 block is represented by a 36×1 vector. So, when we concatenate them all into one giant vector, we obtain a 36×225 = 8100-dimensional vector.

4.2.6 Visualization of HOG:

The HOG descriptor of an image patch is usually visualized by plotting the 9x1 normalized histograms in the 8x8 cells. In **Figure 4.8** you will notice that the dominant direction of histogram captures the shape of the tumor.

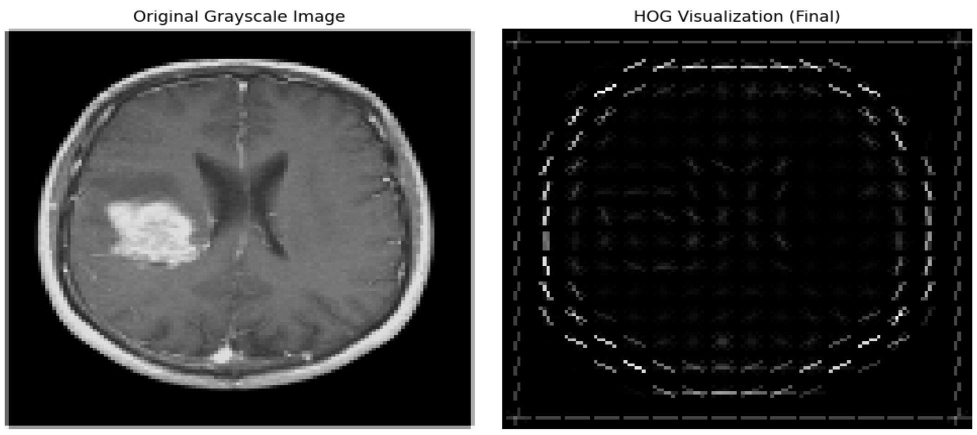


Figure 4.8 Visualization of HOG

4.3 Conclusion:

In this chapter, we explored the feature extraction process essential for transforming MRI images into meaningful numerical representations suitable for machine learning models. We adopted the Histogram of Oriented Gradients (HOG) descriptor as our sole feature extraction technique due to its proven effectiveness in capturing edge and shape information—critical in identifying structural anomalies caused by brain tumors.

The extracted HOG features provided a rich and discriminative representation of the input images while keeping computational complexity low. These features served as the foundation for the subsequent machine learning tasks, enabling our models to distinguish between tumor and non-tumor cases with high accuracy.

The success of our approach demonstrates that even with handcrafted features, robust classification performance can be achieved when the feature extraction technique is well-aligned with the nature of the data.

CHAPTER 5
TRAINING OF MACHINE LEARNING MODELS

5.1 Introduction

Once meaningful features were extracted from the brain MRI images using the Histogram of Oriented Gradients (HOG) technique, the next step involved training machine learning models to classify the images as either tumor or non-tumor. This chapter focuses on the process of training various classical machine learning algorithms using the extracted feature vectors.

A range of classifiers was explored, including K-Nearest Neighbors (KNN), Support Vector Machines (SVM) with different kernels (Linear, RBF, Polynomial), and Random Forest. Each model was trained using standardized feature inputs and evaluated using appropriate performance metrics to determine its effectiveness. Additionally, to enhance predictive accuracy, an ensemble model was developed by combining the outputs of the top-performing individual classifiers through majority voting.

This chapter details the training configurations, model-specific parameters, evaluation strategies, and the comparative analysis that guided our final model selection.

5.1.1 What is the KNN algorithm?

The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point as shown in **Figure 5.1**.
It is one of the popular and simplest classification and regression classifiers used in machine learning today.
While the KNN algorithm can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

For classification problems, a class label is assigned based on a majority vote -i.e. the label that is most frequently represented around a given data point is used. While this is technically considered “plurality voting”, the term “majority vote” is more commonly used in literature. The distinction between these terminologies is that “majority voting” technically requires a majority of greater than 50%, which primarily works when there are only two categories. When you have multiple classes—e.g. four categories, you don’t necessarily need 50% of the vote to make a conclusion about a class; you could assign a class label with a vote of greater than 25%.

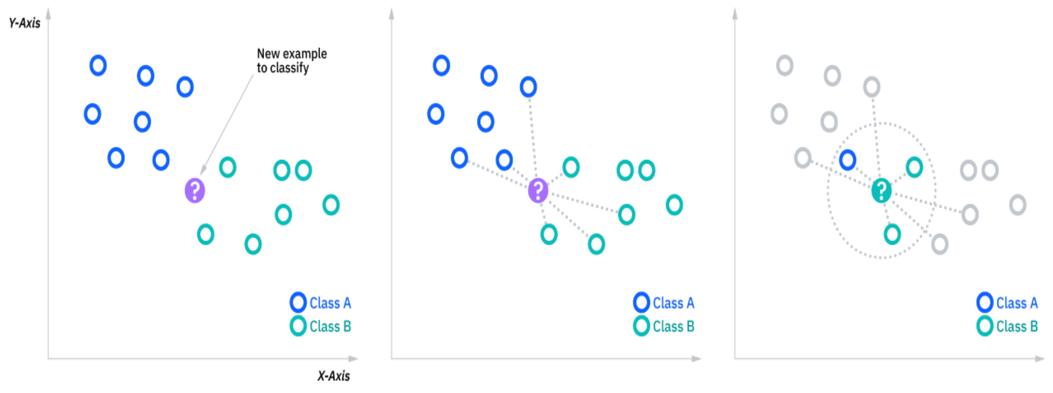


Figure 5.1 KNN Algorithm

However, before a classification can be made, the distance must be defined. Euclidean distance is most used, which we'll delve into more below.

5.1.2 Compute KNN: distance metrics

To recap, the goal of the k-nearest neighbor algorithm is to identify the nearest neighbors of a given query point, so that we can assign a class label to that point. To do this, KNN has a few requirements:

5.1.3 Determine your distance metrics:

To determine which data points are closest to a given query point, the distance between the query point and the other data points will need to be calculated. These distance metrics help to form decision boundaries, which partitions query points into different regions. You commonly see decision boundaries visualized with Voronoi diagrams.
While there are several distance measures that you can choose from, this article will only cover the following:

5.1.3.1 Euclidean distance (p=2):

This is the most used distance measure, and it is limited to real-valued vectors. Using the below formula, it measures a straight line between the query point and the other point being measured.

$$\text{Euclidean distance} \quad d(X,Y) = \sqrt{\sum_{i=1}^{n}(yi - xi)^2}$$

5.1.3.2 Minkowski distance:

This distance measure is the generalized form of Euclidean and Manhattan distance metrics. The parameter p, in the formula below, allows for the creation of other distance metrics. Euclidean distance is represented by this formula when p is equal to two, and Manhattan distance is denoted with p equal to one.

Minkowski distance $d(X,Y) = (\sum_{i=1}^{n}|xi - yi|)^{\frac{1}{p}}$

5.1.3.3 Manhattan distance (p=1):

This is also another popular distance metric, which measures the absolute value between two points. It is also referred to as taxicab distance or city block distance as it is commonly visualized with a grid, illustrating how one might navigate from one address to another via city streets.

Manhattan distance $d(X,Y) = (\sum_{i=1}^{n}|xi - yi|)$

5.1.4 Compute KNN: defining k

The k value in the k-NN algorithm defines how many neighbors will be checked to determine the classification of a specific query point. For example, if k=1, the instance will be assigned to the same class as its single nearest neighbor.
Defining k can be a balancing act as different values can lead to overfitting or underfitting as shown in **Figure 5.2**. Lower values of k can have high variance, but low bias, and larger values of k may lead to high bias and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k. Overall, it is recommended to have an odd number for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset.

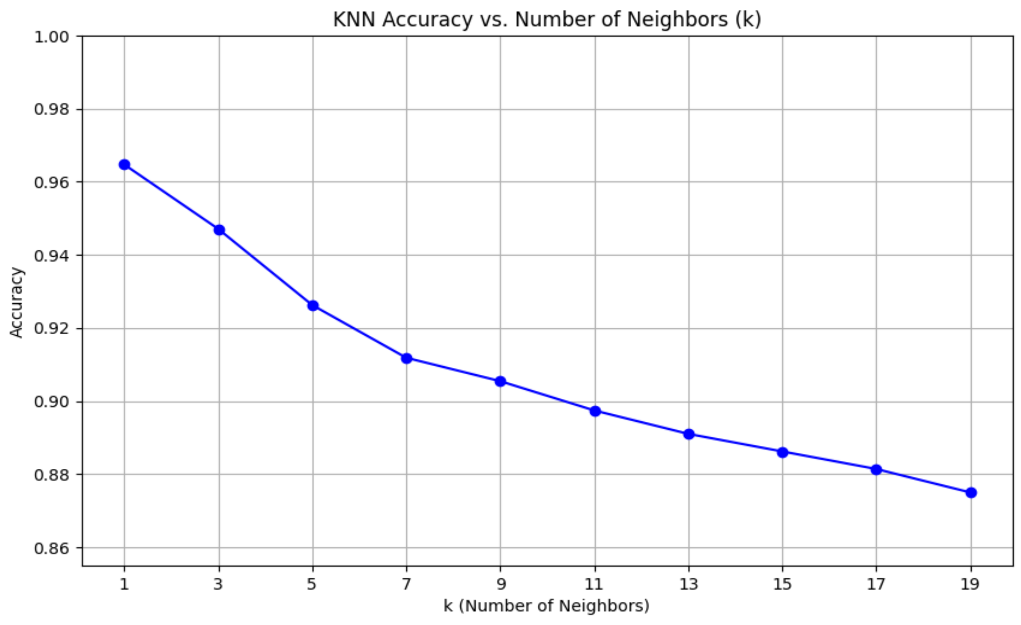


Figure 5.2 KNN Accuracy vs. No.of Neighbours(k)

5.2 Support vector machine (SVM):

A support vector machine (SVM) is a supervised machine learning algorithm that classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space.

SVMs are commonly used for classification problems. They distinguish between two classes by finding the optimal hyperplane that maximizes the margin between the closest data points of the opposite classes as shown in **Figure 5.3**. The number of features in the input data determine if the hyperplane is a line in a 2-D space or a plane in a n-dimensional space. Since multiple hyperplanes can be found to differentiate classes, maximizing the margin between points enables the algorithm to find the best decision boundary between classes. This, in turn, enables it to generalize well to new data and make accurate classification predictions. The lines that are adjacent to the optimal hyperplane are known as support vectors as these vectors run through the data points that determine the maximum margin.

The SVM algorithm is widely used in machine learning as it can handle both linear and nonlinear classification tasks. However, when the data is not linearly separable, kernel functions are used to transform the data into higher-dimensional space to enable linear separation as shown in **Figure 5.4.** This application of kernel functions can be known as the "kernel trick", and the choice of kernel function, such as linear kernels, polynomial kernels, radial basis function (RBF) kernels, or sigmoid kernels, depends on data characteristics and the specific use case.

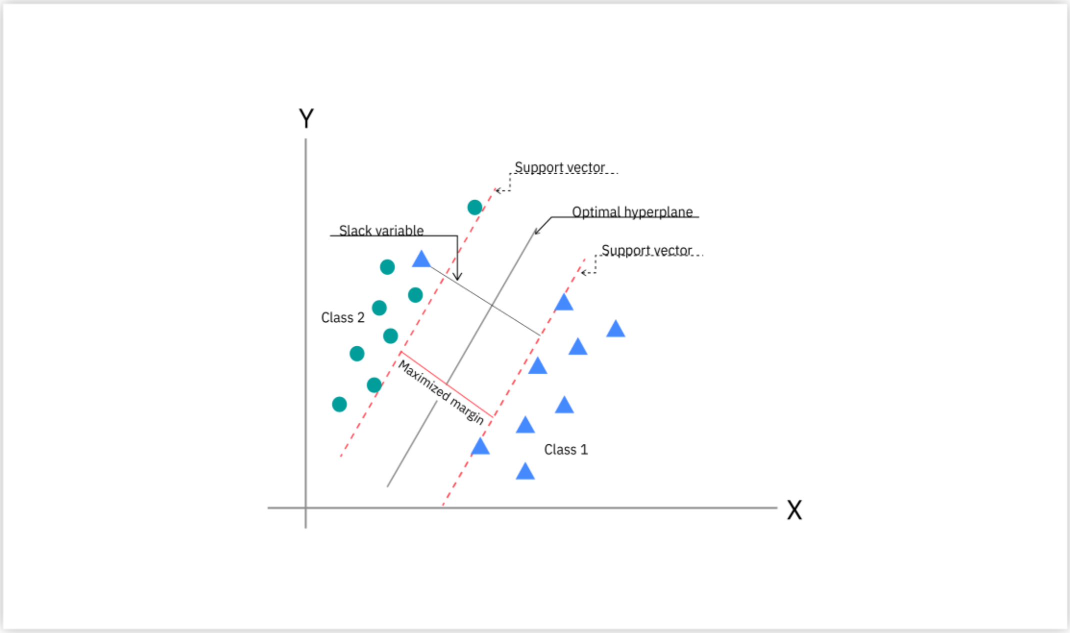


Figure 5.3 Support Vector Machine

5.3 Types of SVM classifiers:

5.3.1 Linear SVMs:

Linear SVMs are used with linearly separable data; this means that the data do not need to undergo any transformations to separate the data into different classes. The decision boundary and support vectors form the appearance of a street, and Professor Patrick Winston from MIT uses the analogy of "fitting the widest possible street"2 (link resides outside ibm.com) to describe this quadratic optimization problem. Mathematically, this separating hyperplane can be represented as:

$wx + b = 0$

where w is the weight vector, x is the input vector, and b is the bias term.

There are two approaches to calculating the margin, or the maximum distance between classes, which are hard-margin classification and soft-margin classification. If we use a hard-margin SVMs, the data points will be perfectly separated outside of the support vectors, or "off the street" to continue with Professor Hinton's analogy. This is represented with the formula,
$(wxj + b)\ yj \geq a$,
and then the margin is maximized, which is represented as: max $\gamma = a / ||w||$, where a is the margin projected onto w.

Soft-margin classification is more flexible, allowing for some misclassification through the use of slack variables (`ξ`). The hyperparameter, C, adjusts the margin; a larger C value narrows the margin for minimal misclassification while a smaller C value widens it, allowing for more misclassified data3

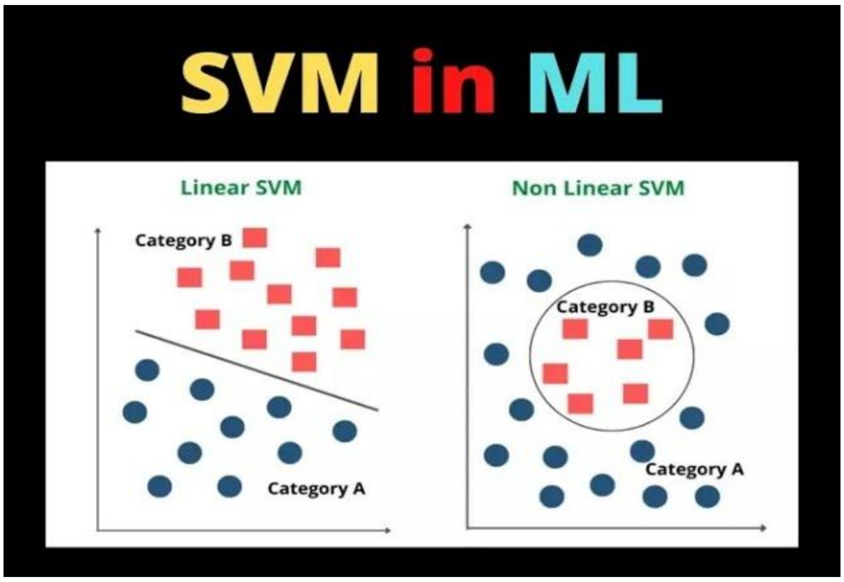


Figure 5.4 Linear vs Non-Linear SVM

5.3.2 Radial basis Function (RBF):

5.3.2.1 Introduction:

The Radial Basis Function (RBF) kernel is one of the most powerful, useful, and popular kernels in the Support Vector Machine (SVM) family of classifiers. In this article, we'll discuss what exactly makes this kernel so powerful, look at its working, and study examples of it in action. We'll also provide code samples for implementing the RBF kernel from scratch in Python that illustrates how to use the RBF kernel on your own data sets.

5.3.2.2 What are Kernels in SVM?

SVM is an algorithm that has shown great success in the field of classification. It separates the data into different categories by finding the best hyperplane and maximizing the distance between points. To this end, a kernel function will be introduced to demonstrate how it works with support vector machines. Kernel functions are a very powerful tool for exploring high-dimensional spaces. They allow us to do linear discriminants on nonlinear manifolds as shown in **Figure 5.5**, which can lead to higher accuracy and robustness than traditional linear models alone.

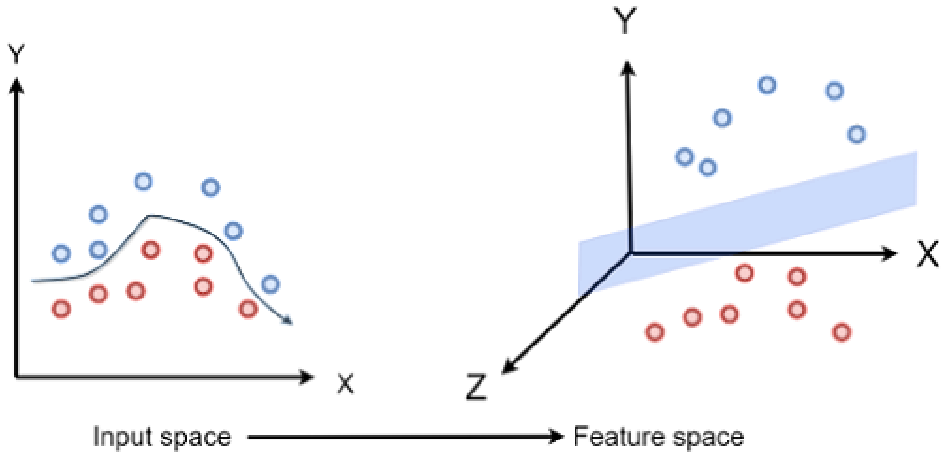


Figure 5.5 Kernels in SVM

The kernel function is just a mathematical function that converts a low-dimensional input space into a higher-dimensional space. This is done by mapping the data into a new feature space. In this space, the data will be linearly separable. This means that a support vector machine can be used to find a hyperplane that separates the data.

For example, if the input x is two-dimensional, the kernel function will map it into a three-dimensional space. In this space, the data will be linearly seperable.

In addition, they provide more features than those of other algorithms such as neural networks or tree ensembles in some kinds of problems involving handwritten recognition, face detection, etc because they extract intrinsic properties of data points through a kernel function.

5.3.2.3 The RBF Kernel:

RBF short for Radial Basis Function Kernel is a very powerful kernel used in SVM. Unlike linear or polynomial kernels, RBF is more complex and efficient at the same time that it can combine multiple polynomial kernels multiple times of different degrees to project the non-linearly separable data into higher dimensional space so that it can be separable using a hyperplane.

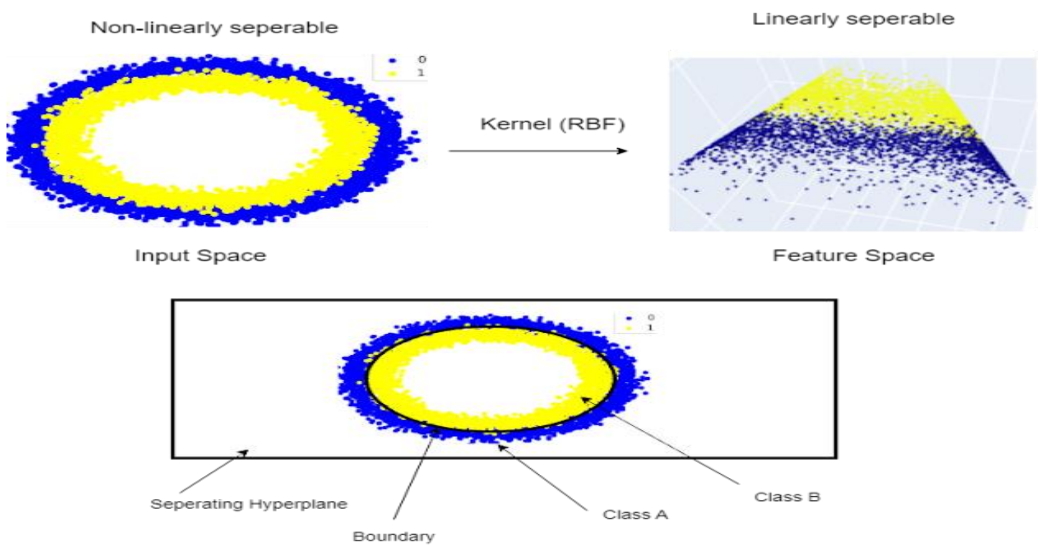


Figure 5.6 RBF Kernel

The RBF kernel works by mapping the data into a high-dimensional space by finding the dot products and squares of all the features in the dataset and then performing the classification using the basic idea of Linear SVM as shown in **Figure 5.6**. For projecting the data into a higher dimensional space, the RBF kernel uses the so-called radial basis function which can be written as:

$$K\ (X1,\ X2) = \exp\left(-\frac{(|\ \ |X1-X2|\ \ |^2)}{2\sigma^2}\right)$$

Here ||X1 – X2||^2 is known as the Squared Euclidean Distance and σ is a free parameter that can be used to tune the equation.
When introducing a new parameter γ = 1 / 2σ^2, the equation will be.

$$K\ (X1,\ X2) = \exp(-\gamma|\ \ |X1-X2|\ \ |^2)$$

The equation is simple here, the Squared Euclidean Distance is multiplied by the gamma parameter and then finding the exponent of the whole. This equation can find the transformed inner products for mapping the data into higher dimensions directly without transforming the entire dataset, which leads to inefficiency. And this is why it is known as the RBF kernel function.

As you can see, the Distribution graph of the RBF kernel looks like the Gaussian Distribution curve, which is known as a bell-shaped curve. Thus, the RBF kernel is also known as the Gaussian Radial Basis Kernel.

RBF kernel is most popularly used with K-Nearest Neighbors and Support Vector Machines.

The distribution graph of RBF Kernel will look as shown in **Figure 5.7**

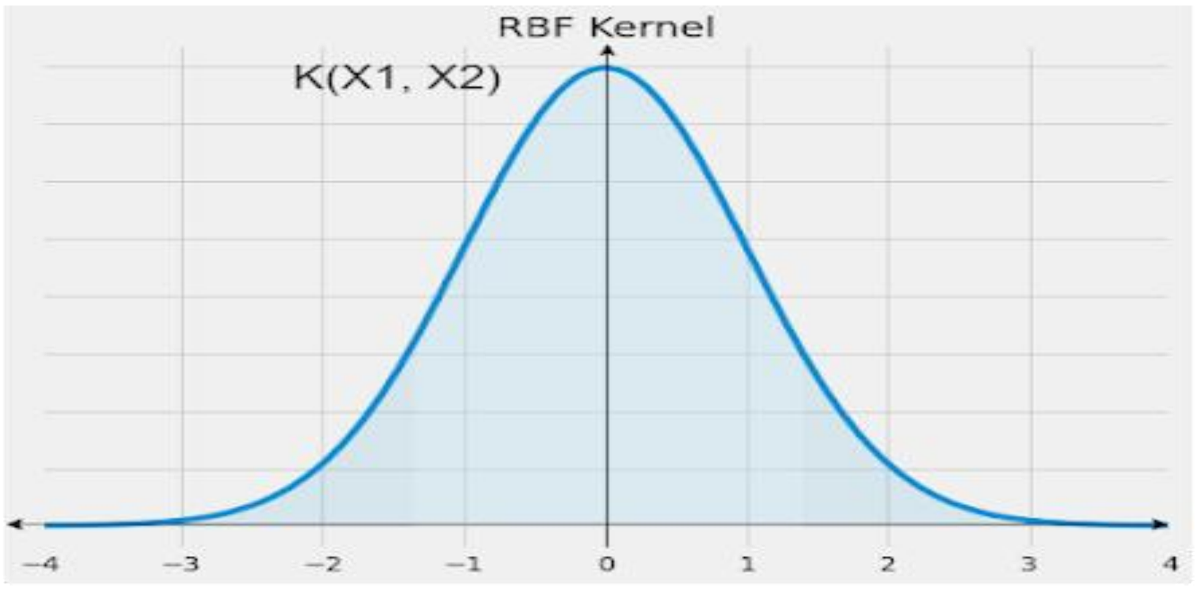


Figure 5.7 Distribution Graph of RBF Kernel

5.3.3 Polynomial SVM:

A polynomial kernel is a kind of SVM kernel that uses a polynomial function to map the data into a higher-dimensional space. It does this by taking the dot product of the data points in the original space and the polynomial function in the new space.

In a polynomial kernel for SVM, the data is mapped into a higher-dimensional space using a polynomial function. The dot product of the data points in the original space and the polynomial function in the new space is then taken. The polynomial kernel is often used in SVM classification problems where the data is not linearly separable. By mapping the data into a higher-dimensional space, the polynomial kernel can sometimes find a hyperplane that separates the classes.

The polynomial kernel has a number of parameters that can be tuned to improve its performance, including the degree of the polynomial and the coefficient of the polynomial.

For degree d polynomials, the polynomial kernel is defined as:

$$K(x1, x2) = \left(x1^T \ x2 + c\right)^d$$

where **c** is a constant and **x1** and **x2** are vectors in the original space.

The parameter **c** can be used to control the trade-off between the fit of the training data and the size of the margin. A large **c** value will give a low training error but may result in overfitting. A small **c** value will give a high training error but may result in underfitting. The degree **d** of the polynomial can be used to control the complexity of the model. A high degree **d** will result in a more complex model that may overfit the data, while a low degree **d** will result in a simpler

model that may underfit the data.
When a dataset is given containing features x1 and x2, the equation can be transformed as:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot [x_1 x_2] = \begin{bmatrix} x_1^2 & x_1 x_2 \\ x_1 x_2 & x_2^2 \end{bmatrix}$$

The important terms we need to note are **x1, x2, x1^2, x2^2**, and **x1 * x2.** When finding these new terms, the non-linear dataset is converted to another dimension that has features **x1^2, x2^2**, and **x1 * x2.**

5.4 Random Forest:

5.4.1 What is Random Forest?

Random forest is a commonly used machine learning algorithm, trademarked by Leo Breiman and Adele Cutler, that combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

5.4.2 Decision Tree:

Since the random forest model is made up of multiple decision trees, it would be helpful to start by describing the decision tree algorithm briefly. Decision trees start with a basic question, such as, “Should I surf?” From there, you can ask a series of questions to determine an answer, such as, “Is it a long period swell?” or “Is the wind blowing offshore?”. These questions make up the decision nodes in the tree, acting as a means to split the data. Each question helps an individual to arrive at a final decision, which would be denoted by the leaf node. Observations that fit the criteria will follow the “Yes” branch and those that don’t will follow the alternate path. Decision trees seek to find the best split to subset the data, and they are typically trained through the Classification and Regression Tree (CART) algorithm. Metrics, such as Gini impurity, information gain, or mean square error (MSE), can be used to evaluate the quality of the split.
This decision tree is an example of a classification problem, where the class labels are "surf" and "don't surf."

While decision trees are common supervised learning algorithms, they can be prone to problems, such as bias and overfitting. However, when multiple decision trees form an ensemble in the random forest algorithm, they predict more accurate results, particularly when the individual trees are uncorrelated with each other.

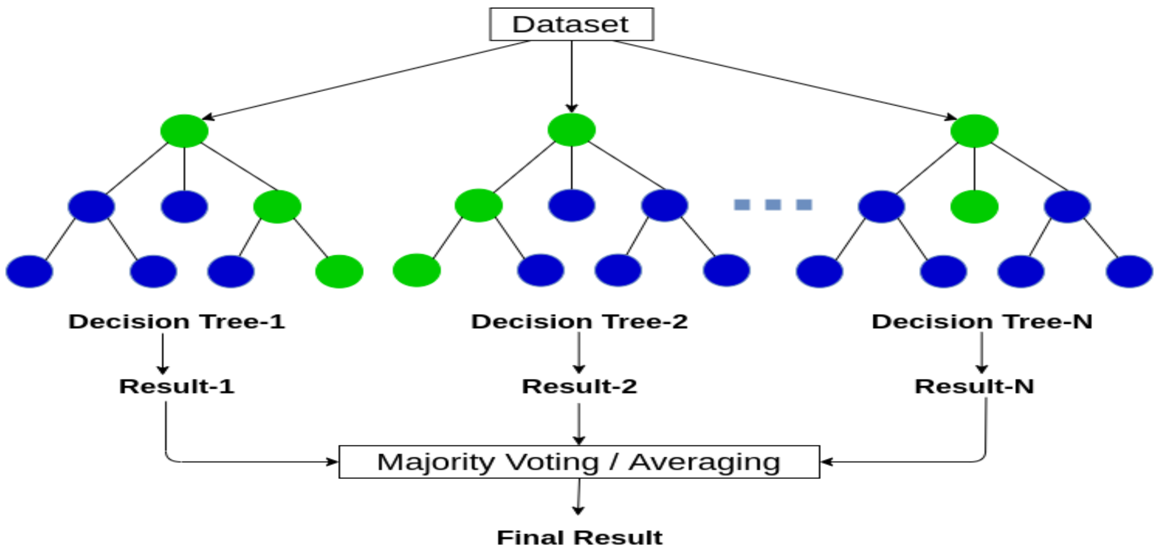


Figure 5.8 Random Forest using Decision Trees

5.4.3 Random Forest algorithm:

The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees as shown in **Figure 5.8.** Feature randomness, also known as feature bagging or "the random subspace method", generates a random subset of features, which ensures low correlation among decision trees. This is a key difference between the decision trees and random forests. While decision trees consider all the possible feature splits, random forests only select a subset of those features.

If we go back to the "should I surf?" example, the questions that I may ask to determine the prediction may not be as comprehensive as someone else's set of questions. By accounting for all the potential variability in the data, we can reduce the risk of overfitting, bias, and overall variance, resulting in more precise predictions.

5.4.4 How it works.

Random forest algorithms have three main hyperparameters, which need to be set before training. These include the node size, the number of trees, and the number of features sampled. From there, the random forest classifier can be used to solve regression or classification problems.

The random forest algorithm is made up of a collection of decision trees as shown in **Figure 5.9**, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample. Of that training sample, one-third of it is set aside as test data, known as the out-of-bag (oob) sample, which we'll come back to later. Another instance of randomness is then injected through feature bagging, adding more diversity to the dataset and reducing the correlation among decision trees. Depending on the type of problem, the determination of the prediction will vary. For a regression task, the individual decision trees will be averaged, and for a classification task, a majority vote—i.e. the most frequent categorical variable—will yield the predicted class. Finally, the oob sample is then used for cross-validation, finalizing that prediction.

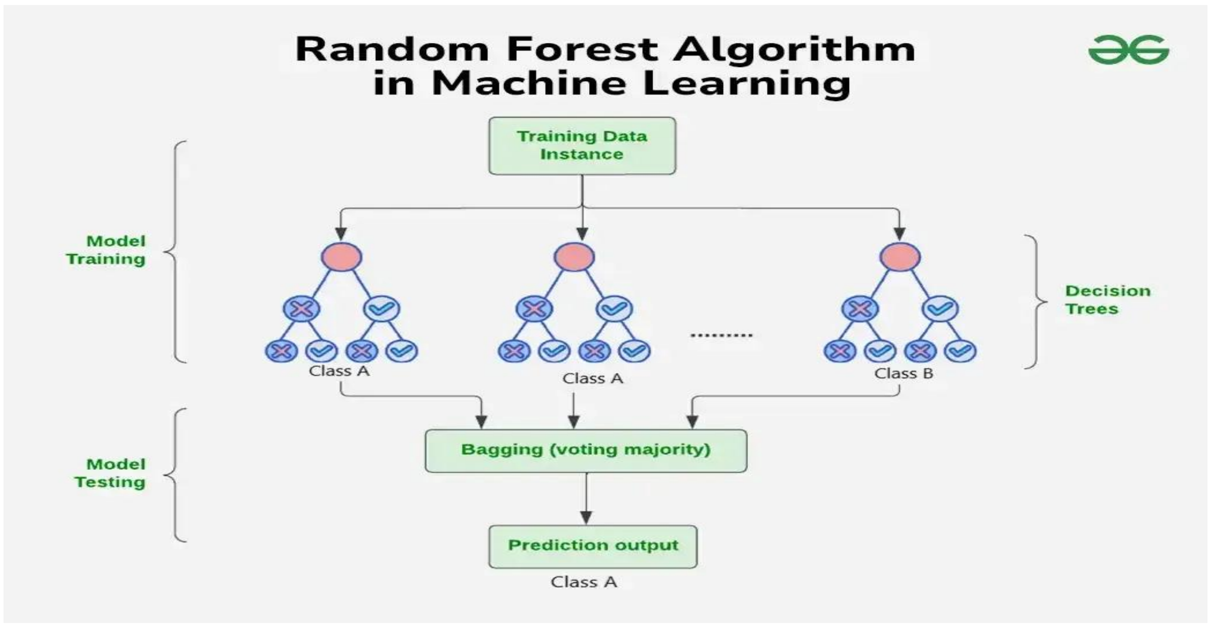


Figure 5.9 Random Forest Algorithm

5.5 Conclusion:

In this chapter, we successfully trained and evaluated multiple classical machine learning models using the HOG feature vectors derived from brain MRI images. Among the individual classifiers, RBF SVM, Polynomial SVM, and KNN demonstrated high accuracy, while Random Forest showed relatively lower performance. The ensemble model, which combined KNN, RBF SVM, and Polynomial SVM, achieved the best accuracy of 97.44%, outperforming all standalone models.

The results confirm that carefully selected classical models, when trained on well-engineered features, can deliver high performance in medical image classification tasks. This validates our design choice of using interpretable, resource-efficient models instead of more complex deep learning architectures. The trained models form the core decision-making component of our tumor detection system.

CHAPTER 6
RESULTS AND DISCUSSIONS

6.1 Introduction:

To assess the performance of our classification models, we utilize several standard evaluation metrics. These metrics help us understand different aspects of the model's performance, especially in cases where the dataset may be imbalanced.
Understanding TP, TN, FP, FN:

- True **Positive (TP):**
  The model correctly predicted the **positive class**.
  Example: It predicted "Tumor", and the image actually had a tumor.
- True **Negative (TN):**
  The model correctly predicted the **negative class**.
  Example: It predicted "No Tumor", and the image actually had no tumor.
- False **Positive (FP):**
  The model incorrectly predicted the **positive class**.
  Example: It predicted "Tumor", but the image actually had no tumor.
  (Also called a "Type I Error")
- False **Negative (FN):**
  The model incorrectly predicted the **negative class**.
  Example: It predicted "No Tumor", but the image actually had a tumor.
  (Also called a "Type II Error")

6.2 Accuracy

Accuracy is the ratio of correctly predicted observations to the total number of observations. It provides an overall effectiveness of the model in terms of correct classifications.

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

6.3 Precision

Precision measures the proportion of correctly predicted positive observations out of all observations that were predicted as positive. It is particularly useful when the cost of False Positives is high.

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

6.4 Recall (Sensitivity or True Positive Rate)

Recall measures the proportion of actual positives that were correctly identified by the model. It is crucial when the cost of False Negatives is high.

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

6.5 F1 Score

The F1 Score is the harmonic mean of precision and recall. It provides a single metric that balances both false positives and false negatives, especially useful when the class distribution is uneven.

$$\text{F1 Score} = 2 \text{ x } \frac{Precision\ x\ Recall}{Precition+Recall}$$

6.6 Confusion Matrix

The confusion matrix is a tabular representation that summarizes the performance of a classification algorithm. It provides the count of True Positives, True Negatives, False Positives, and False Negatives, giving deeper insight into the types of errors made.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

6.7 Model Evaluation Results

The performance of each classifier is evaluated using the metrics described above. Below is a detailed breakdown for each model:

6.7.1 KNN:

Accuracy: 96.47%

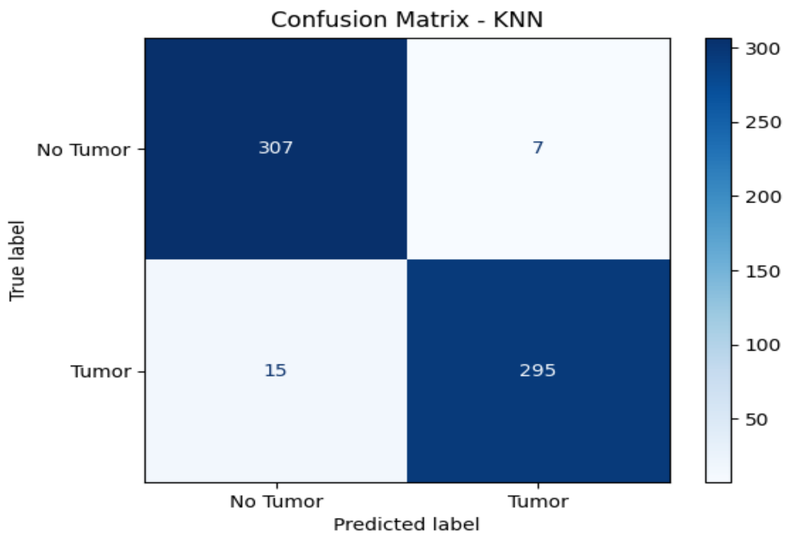
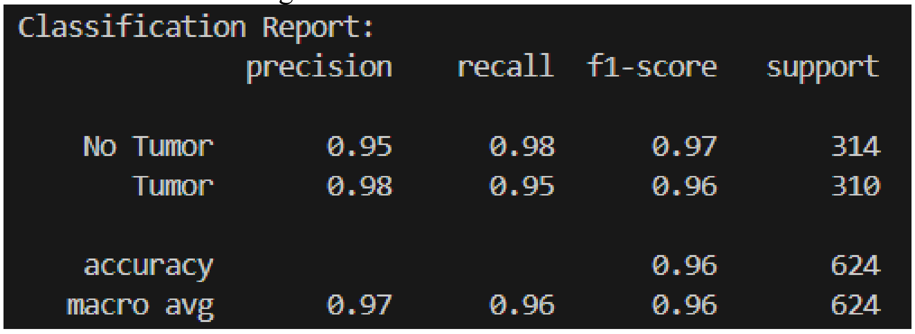


Figure 6.1 KNN Confusion Matrix



```
Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.95      0.98      0.97       314
       Tumor       0.98      0.95      0.96       310

    accuracy                           0.96       624
   macro avg       0.97      0.96      0.96       624
```

Figure 6.1 KNN Classification Report

Class-wise Metrics:

Class: "No Tumor"

- **Precision = 0.95** → Out of all images predicted as "No Tumor", 97% were correct.
- **Recall = 0.98** → Out of all actual "No Tumor" images, 98% were correctly identified.
- **F1-Score = 0.97** → A balanced average of precision and recall.
- **Support = 314** → There are 314 actual "No Tumor" images in the test set.

Class: "Tumor"

- **Precision = 0.98** → Out of all images predicted as "Tumor", 98% were correct.
- **Recall = 0.95** → Out of all actual "Tumor" images, 96% were correctly caught by the model.
- **F1-Score = 0.96** → Again, balanced score between precision and recall.
- **Support = 310** → There are 310 actual "Tumor" images in the test set.

6.7.2 Linear SVM:

Accuracy: 95.35%

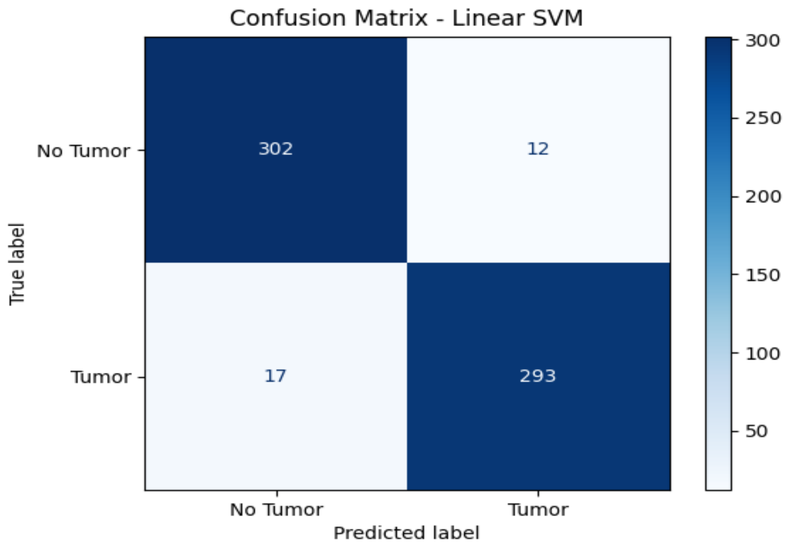
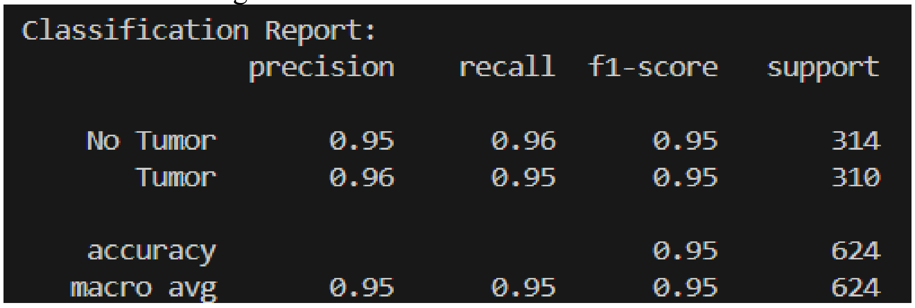


Figure 6.2 Linear SVM Confusion Matrix



```
Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.95      0.96      0.95       314
       Tumor       0.96      0.95      0.95       310

    accuracy                           0.95       624
   macro avg       0.95      0.95      0.95       624
```

Figure 6.2 Linear SVM Classification Report

Class-wise Metrics:
Class: "No Tumor"

- **Precision = 0.95** → Out of all images predicted as "No Tumor", 97% were correct.
- **Recall = 0.96** → Out of all actual "No Tumor" images, 98% were correctly identified.
- **F1-Score = 0.95** → A balanced average of precision and recall.
- **Support = 314** → There are 314 actual "No Tumor" images in the test set.

Class: "Tumor"

- **Precision = 0.96** → Out of all images predicted as "Tumor", 98% were correct.
- **Recall = 0.95** → Out of all actual "Tumor" images, 96% were correctly caught by the model.
- **F1-Score = 0.95** → Again, balanced score between precision and recall.
- **Support = 310** → There are 310 actual "Tumor" images in the test set.

6.7.3 RBF SVM:

Accuracy: 96.96%

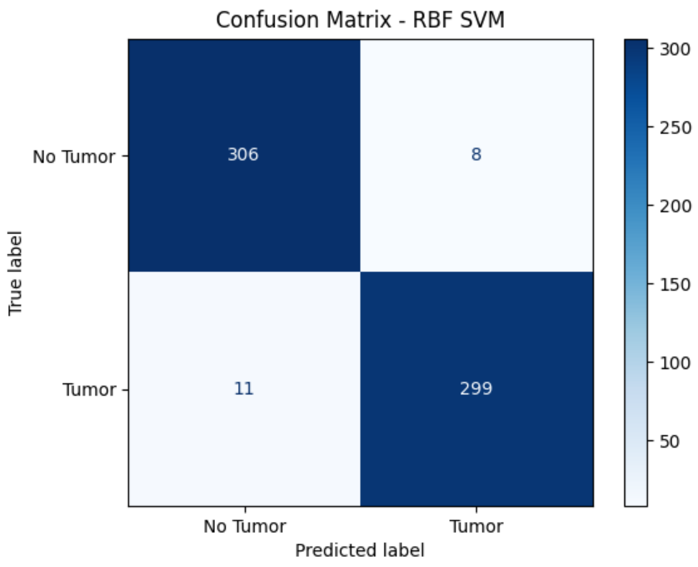
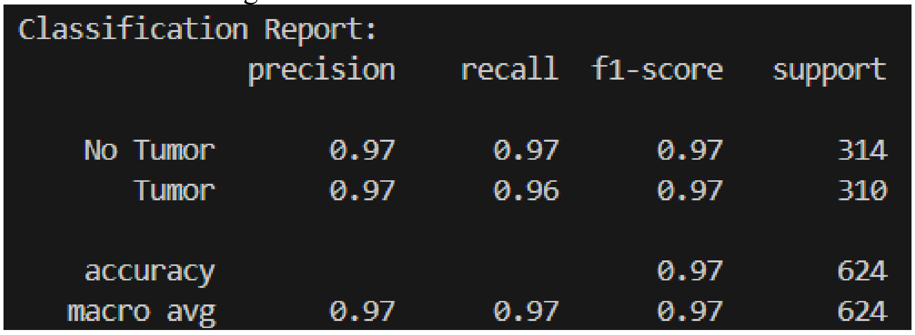


Figure 6.3 RBF SVM Confusion Matrix



```
Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.97      0.97      0.97       314
       Tumor       0.97      0.96      0.97       310

    accuracy                           0.97       624
   macro avg       0.97      0.97      0.97       624
```

Figure 6.3 RBF SVM Classification Report

Class-wise Metrics:
Class: "No Tumor"

- **Precision = 0.97** → Out of all images predicted as "No Tumor", 97% were correct.
- **Recall = 0.97** → Out of all actual "No Tumor" images, 98% were correctly identified.
- **F1-Score = 0.97** → A balanced average of precision and recall.
- **Support = 314** → There are 314 actual "No Tumor" images in the test set.

Class: "Tumor"

- **Precision = 0.97** → Out of all images predicted as "Tumor", 98% were correct.
- **Recall = 0.96** → Out of all actual "Tumor" images, 96% were correctly caught by the model.
- **F1-Score = 0.97** → Again, balanced score between precision and recall.
- **Support = 310** → There are 310 actual "Tumor" images in the test set.

6.7.4 Polynomial SVM:

Accuracy: 96.79%

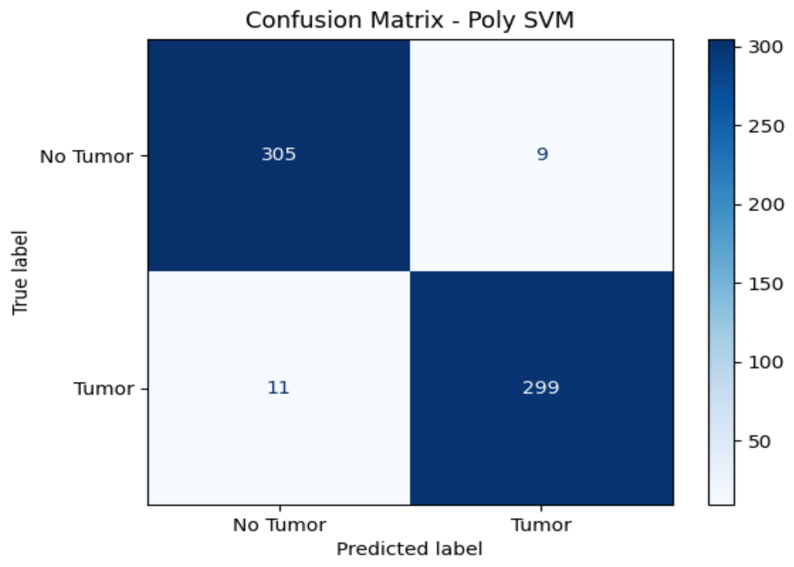
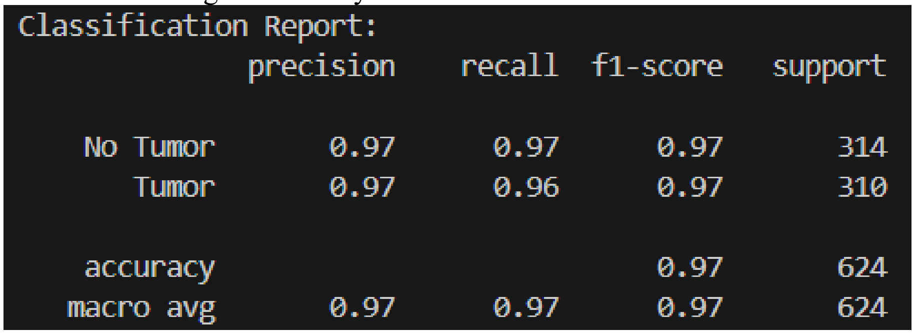


Figure 6.4 Polynomial SVM Confusion Matrix



```
Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.97      0.97      0.97       314
       Tumor       0.97      0.96      0.97       310

    accuracy                           0.97       624
   macro avg       0.97      0.97      0.97       624
```

Figure 6.4 Polynomial SVM Classification Report

Class-wise Metrics:

Class: "No Tumor"

- **Precision = 0.97** → Out of all images predicted as "No Tumor", 97% were correct.
- **Recall = 0.97** → Out of all actual "No Tumor" images, 98% were correctly identified.
- **F1-Score = 0.97** → A balanced average of precision and recall.
- **Support = 314** → There are 314 actual "No Tumor" images in the test set.

Class: "Tumor"

- **Precision = 0.97** → Out of all images predicted as "Tumor", 98% were correct.
- **Recall = 0.96** → Out of all actual "Tumor" images, 96% were correctly caught by the model.
- **F1-Score = 0.97** → Again, balanced score between precision and recall.
- **Support = 310** → There are 310 actual "Tumor" images in the test set

6.7.5 Random Forest:

Accuracy: 90.87%

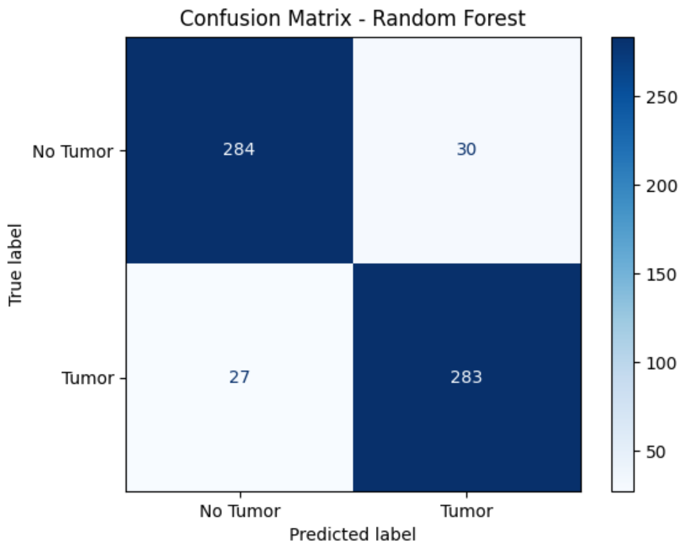
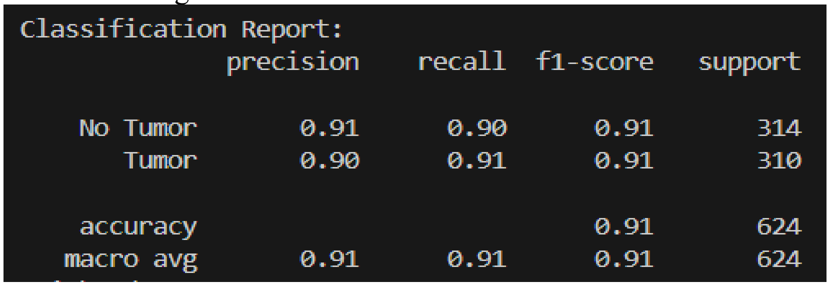


Figure 6.5 Random Forest Confusion Matrix



```
Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.91      0.90      0.91       314
       Tumor       0.90      0.91      0.91       310

    accuracy                           0.91       624
   macro avg       0.91      0.91      0.91       624
```

Figure 6.5 Random Forest Classification Report

Class-wise Metrics:

Class: "No Tumor"

- **Precision = 0.91** → Out of all images predicted as "No Tumor", 97% were correct.
- **Recall = 0.90** → Out of all actual "No Tumor" images, 98% were correctly identified.
- **F1-Score = 0.91** → A balanced average of precision and recall.
- **Support = 314** → There are 314 actual "No Tumor" images in the test set.

Class: "Tumor"

- **Precision = 0.90** → Out of all images predicted as "Tumor", 98% were correct.
- **Recall = 0.91** → Out of all actual "Tumor" images, 96% were correctly caught by the model.
- **F1-Score = 0.91** → Again, balanced score between precision and recall.
- **Support = 310** → There are 310 actual "Tumor" images in the test set.

6.7.6 Ensemble (KNN, RBF SVM and Polynomial SVM):

Accuracy: 97.44%

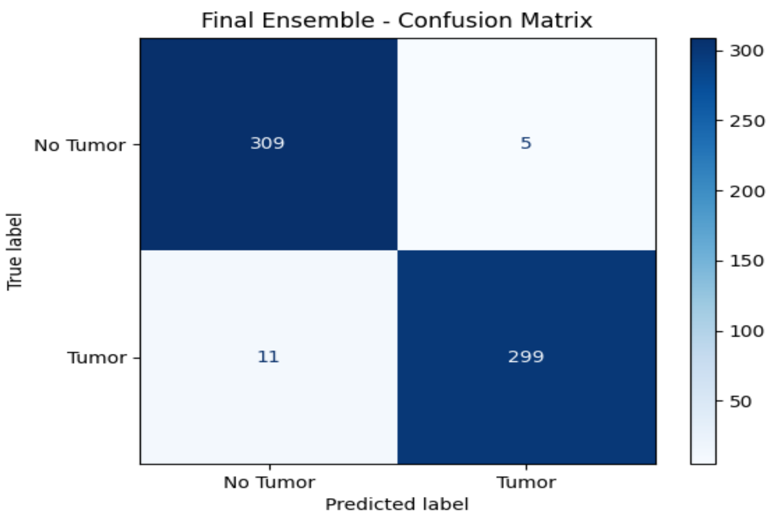
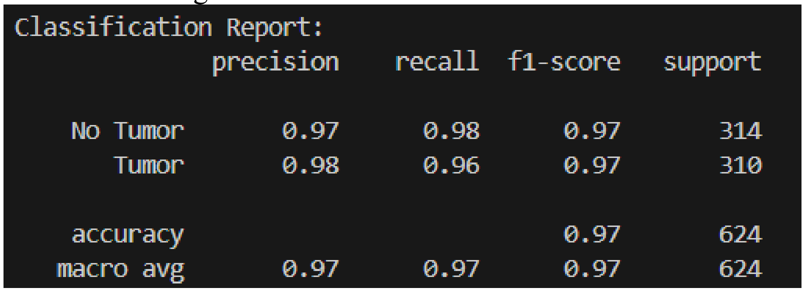


Figure 6.6 Ensemble Confusion Matrix



```
Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.97      0.98      0.97       314
       Tumor       0.98      0.96      0.97       310

    accuracy                           0.97       624
   macro avg       0.97      0.97      0.97       624
```

Figure 6.6 Ensemble Classification Report

Class-wise Metrics:
Class: "No Tumor"

- **Precision = 0.97** → Out of all images predicted as "No Tumor", 97% were correct.
- **Recall = 0.98** → Out of all actual "No Tumor" images, 98% were correctly identified.
- **F1-Score = 0.97** → A balanced average of precision and recall.
- **Support = 314** → There are 314 actual "No Tumor" images in the test set.

Class: "Tumor"

- **Precision = 0.98** → Out of all images predicted as "Tumor", 98% were correct.
- **Recall = 0.96** → Out of all actual "Tumor" images, 96% were correctly caught by the model.
- **F1-Score = 0.97** → Again, balanced score between precision and recall.
- **Support = 310** → There are 310 actual "Tumor" images in the test set.

6.8 Conclusion:

We evaluated all models using standard classification metrics including accuracy, precision, recall, F1-score. Among the classifiers tested, the **ensemble of KNN, RBF SVM, and Polynomial SVM** demonstrated the best overall performance. It consistently achieved high values across all evaluation metrics, indicating strong generalization and balanced prediction capabilities for both 'Tumor' and 'No Tumor' classes. The ensemble model achieved the highest F1-score, making it particularly effective at handling both false positives and false negatives — a crucial requirement in medical diagnosis tasks.

In contrast, the **Random Forest classifier** exhibited the lowest performance across several metrics. While it still produced reasonable results, its relatively lower recall and F1-score suggest that it was less reliable in consistently identifying tumor cases compared to the other models.

CHAPTER 7
CONCLUSION AND FUTURE SCOPE

This project successfully demonstrated the potential of traditional machine learning algorithms for brain tumor detection using image-based feature analysis. By extracting Histogram of Oriented Gradients (HOG) features and applying various classifiers, the system achieved high accuracy and robust classification performance. Among the models tested, the ensemble of KNN, RBF-SVM, and Polynomial-SVM delivered the best results, outperforming individual classifiers in both precision and recall.

While there are certain limitations, such as dataset size and the lack of tumor localization, this work provides a strong foundation for future enhancements. With further development — particularly in the areas of deep learning integration, dataset expansion, and clinical deployment — the system holds promise as a practical aid for early brain tumor diagnosis.

7.1 Advantages:

- **High Accuracy with Lightweight Models**: The system achieves high classification accuracy (~97%) using traditional machine learning models, which are computationally efficient and do not require GPU-based deep learning hardware.
- **Interpretable Feature-Based Approach**: By using handcrafted features like HOG, the decision process remains relatively interpretable compared to black-box deep learning models.
- **Balanced Performance Across Classes**: The ensemble classifier effectively handled both tumor and non-tumor images, minimizing false negatives and false positives — crucial in medical diagnosis.
- **Flexibility in Model Choice**: Multiple classifiers were evaluated, providing flexibility to choose the best model depending on constraints like computational power, speed, or precision-recall trade-offs.

Limitations:

- **Limited Dataset Size**: The model was trained and tested on a relatively small dataset (1570 no-tumor and 1550 tumor images), which may not fully represent the diversity seen in real-world cases.
- **No Tumor Localization**: The system only performs **classification**, not **segmentation** or localization, i.e., it does not indicate *where* the tumor is present in the image.
- **Feature Dependence**: The system relies heavily on the quality of extracted features (e.g., HOG). If the tumor's texture or shape varies drastically, handcrafted features might fail to generalize.
- **No Deep Learning Benchmark**: Due to hardware constraints, deep learning-based methods were not implemented or compared, which are currently state-of-the-art in medical imaging tasks.

7.2 Future Scope:

- **Incorporating Deep Learning**: Future versions can explore CNN-based models using frameworks like TensorFlow or PyTorch to achieve higher generalization and end-to-end learning.
- **Tumor Localization and Segmentation**: Adding segmentation techniques (e.g., using image masks or bounding boxes) would help in highlighting the tumor region, aiding radiologists further.
- **Larger and More Diverse Datasets**: Training on larger MRI datasets with more diversity (different scanners, patient demographics, tumor types) would improve robustness.
- **Real-time Deployment**: With optimization, the model can be deployed as part of a lightweight diagnostic tool for use in low-resource clinical settings.

7.3 Uniqueness:

In most existing brain tumor detection systems, especially academic ones, the focus is either purely on deep learning or limited to using a single machine learning model. What makes our work different is that we designed a lightweight, feature-based approach that performs comparably well without needing large datasets or high-end hardware.

Specifically, we used Histogram of Oriented Gradients (HOG) — a feature descriptor that is rarely explored for medical MRI images — and combined it with multiple traditional classifiers, then compared them using well-defined metrics.

What's even more unique is that we identified the best models individually (KNN, RBF-SVM, Polynomial-SVM), and then created a custom ensemble from them, which outperformed all individual models in terms of precision, recall, and F1-score.
Our pipeline is highly interpretable, cost-efficient, and practical for low-resource settings like small clinics or hospitals where deep learning-based solutions are difficult to deploy.

So in short — our uniqueness lies in combining classical computer vision + ensemble ML approach, tailored for efficiency, performance, and interpretability, which makes it different from deep learning-only solutions or basic single-classifier ML models.

7.4 Real World Deployment:

To enhance the usability and accessibility of our brain tumor detection system, we developed a web-based user interface that allows users to interact with the trained machine learning model in a simple and intuitive way.

7.4.1 Purpose

The primary goal of the web application is to provide a platform where:

- Medical professionals or general users can upload an MRI image,
- The system will automatically predict whether a tumor is present or not,and display the result in a clear and interpretable format.

The Webpage is shown in **Figure 7.1.**

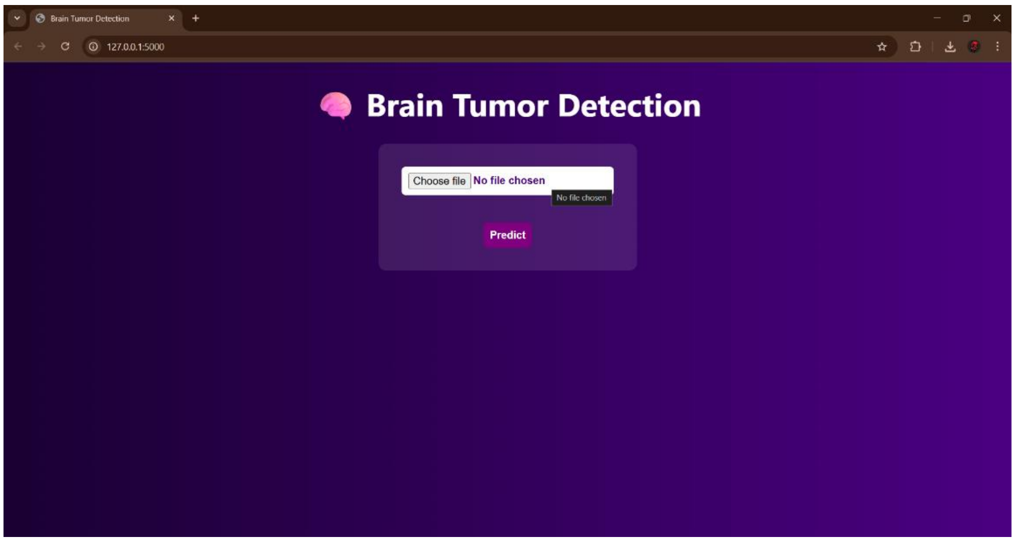


Figure 7.1

7.4.2 Web Application Interface:

When the user clicks on **"Choose File"** , File explorer opens and allows the user to select the MRI Scan Image as shown in **Figure 7.2**

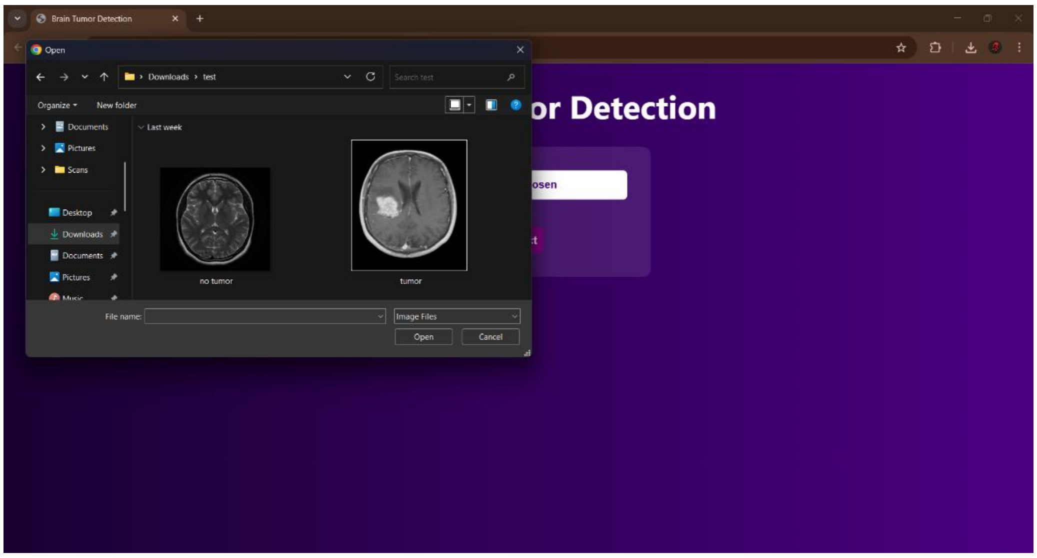
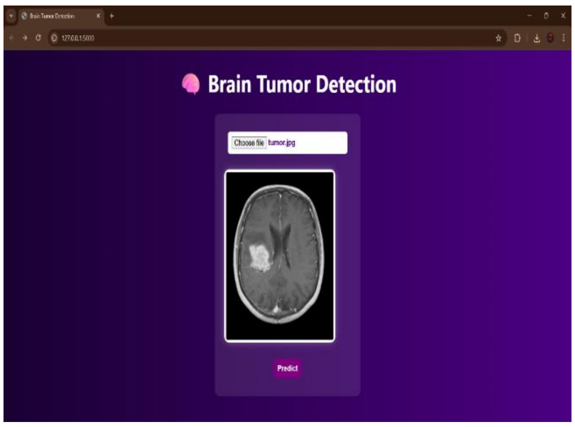


Figure 7.2



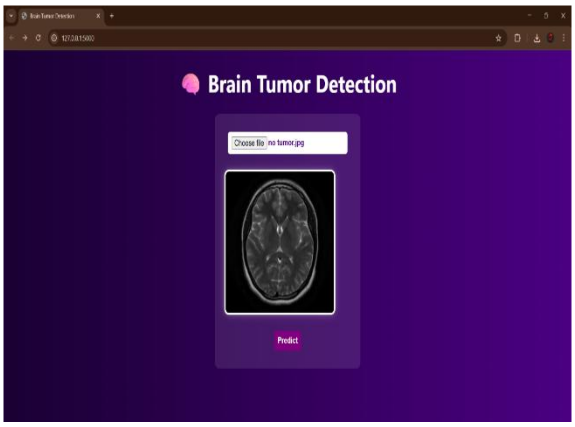
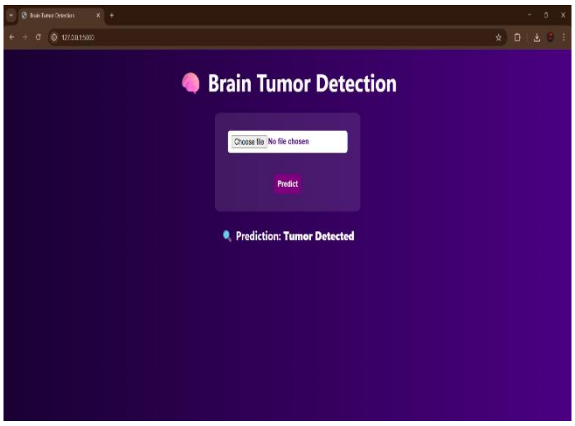


Figure 7.3 Figure 7.4



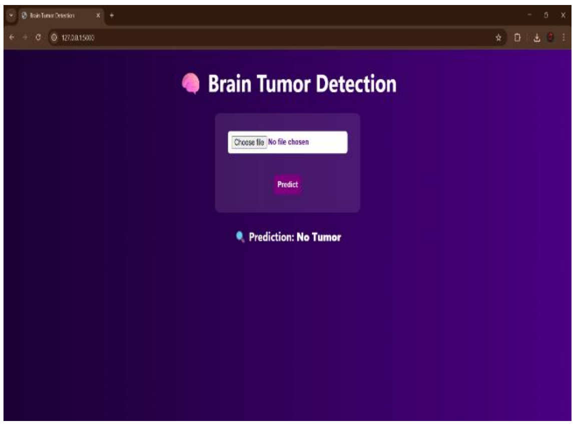


Figure 7.5 Figure 7.6

Figure 7.1-6 Web Application Interface

Figure 7.3 shows the user selecting the MRI Scan of a Brain **with** tumor and when the **"Predict"** button is clicked we get the prediction as **"Tumor Detected"** as shown in **Figure 7.5**

Figure 7.4 shows the user selecting the MRI Scan of a Brain **without** tumor and when the **"Predict"** button is clicked we get the prediction as **"No Tumor"** as shown in **Figure 7.6**

7.4.3 Key Features:

- **Image Upload Interface:** Users can upload an MRI image in .jpg or .png format.
- **Automatic Preprocessing:** Uploaded images are resized, converted to grayscale, and processed using the same feature extraction pipeline used during model training.
- **Real-time Prediction:** The processed image is passed to the trained model, and the predicted label ("Tumor" or "No Tumor") is displayed instantly.

REFERENCES:

1. Tiwari, P., Prasanna, P., Wolansky, L., et al. (2011). A texture-based machine learning algorithm for the classification of brain tumors using MRI. J. Magn. Reson. Imaging, 35(6), 1351–1359.

2. Zhang, Y., Dong, Z., Phillips, P., et al. (2016). Exploiting redundant classification for brain tumor recognition. Pattern Recognition, 63, 416–427.

3. Chaplot, S., Patnaik, L. M., & Jagannathan, N. R. (2006). Classification of magnetic resonance brain images using wavelets as input to support vector machine and neural network. Biomedical Signal Processing and Control, 1(1), 86–92.

4. El-Dahshan, E. S. A., Hosny, T., & Salem, A.-B. M. (2010). Hybrid intelligent techniques for MRI brain images classification. Digital Signal Processing, 20(2), 433–441.

5. Paul, J., Sinha, D., & Bandyopadhyay, S. (2018). Decision tree based tumor detection from brain MRI. Procedia Computer Science, 132, 1123–1130.

6. Pereira, S., Pinto, A., Alves, V., & Silva, C. A. (2016). Brain tumor segmentation using convolutional neural networks in MRI images. IEEE Trans. Med. Imaging, 35(5), 1240–1251.

7. Hossain, M. S., Muhammad, G., & Alamri, A. (2019). Brain tumor detection using transfer learning and fine-tuned CNNs. Journal of Ambient Intelligence and Humanized Computing, 10(8), 2813–2820.

8. Afshar, P., Mohammadi, A., & Plataniotis, K. N. (2019). Brain tumor type classification via capsule networks. In 2019 IEEE ICASSP, 1368–1372.

9. Muhammad, K., Khan, S., & Del Ser, J. (2020). Smart anomaly-based brain MRI segmentation using deep learning. Neural Computing and Applications, 32, 10255–10268.

10. Kamnitsas, K., Ledig, C., Newcombe, V. F., et al. (2017). Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. Med. Image Anal., 36, 61–78.

11. Raja, N., Srinivas, K., & Reddy, V. (2020). CNN-SVM hybrid model for brain tumor classification. Neural Comput & Applic, 32, 18329–18335.

12. Sajjad, M., Khan, S., & Rehman, A. (2019). Multi-grade brain tumor classification

using deep CNN with augmented data. Journal of Healthcare Engineering, 2019.

13. Swati, Z. N. K., Zhao, Q., Kabir, M., et al. (2019). Brain tumor classification for MRIs using transfer learning and fine-tuned deep CNN. Computerized Medical Imaging and Graphics, 75, 34–46.

14. Khawaldeh, S., Taher, F., Al-Mazari, S., et al. (2020). Deep learning-based classification of brain tumors using multiple transfer learning models. J. Imaging, 6(6), 48.

15. Masood, K., & Rajpoot, N. (2015). A novel framework for automated segmentation of brain tumors from MRI using ensemble techniques. Journal of Biomedical Imaging, 2015, 1–12.

APPENDIX

1) KNN CODE:

```
X = hog_features
y = labels

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

2) Linear SVM CODE:

```
X = hog_features
y = labels

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y)

svm = SVC(kernel='linear', C=0.01, random_state=42)
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)
```

3) RBF SVM CODE:

```
X = hog_features
y = labels

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y)

svm = SVC(kernel='rbf', C=10,gamma='scale', random_state=42)
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)
```

4) Poly SVM CODE:

```
X = hog_features
y = labels

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y)

svm = SVC(kernel='poly', C=1,gamma='scale',coef0=1,degree=4, random_state=42)
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)
```

5) Random Forest CODE:

```
X = hog_features
y = labels

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
   X_scaled, y, test_size=0.2, random_state=42, stratify=y)

rf = RandomForestClassifier (n_estimators=200, max_depth=20, random_state=42,
class_weight='balanced', max_features='sqrt', min_samples_leaf=1, min_samples_split=5)

rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)
```

6) Ensemble CODE:

```
X = hog_features
y = labels

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
   X_scaled, y, test_size=0.2, stratify=y, random_state=42)

svm_rbf = SVC(kernel='rbf', C=10, gamma='scale', probability=True)
svm_poly= SVC(kernel='poly', C=1, gamma='scale',coef0=1,degree=4, probability=True)
knn = KNeighborsClassifier(n_neighbors=1)

ensemble = VotingClassifier(estimators=[('svm1', svm_rbf), ('knn', knn), ('svm2',svm_poly)],
voting='soft' )

ensemble.fit(X_train, y_train)

y_pred = ensemble.predict(X_test)
```