Clinical Brain Tumor Detection: Optimized Machine Learning Frameworks for MRI Diagnosis

Q.Q. Chen (2330034001), S.Q. Huang (2330034020), M.H. Li (2330034027)
Y.R. Liang (2330034030), Y.X. Tu (2330034048)
Beijing Normal-Hong Kong Baptist University

***Abstract*—This paper presents a comprehensive machine learning framework for brain tumor detection in MRI images, addressing critical challenges in medical image analysis. Our approach leverages multiple traditional ML models (SVM, XGBoost, MLP, Logistic Regression, and KNN) optimized for clinical deployment scenarios with limited computational resources. We integrate multiple datasets and employ advanced preprocessing techniques, including bicubic interpolation-based resizing, rotation, flipping, and color jitter, to enhance data quality and diversity. The system incorporates interpretable feature extraction methods (HOG, GLCM) and implements model-specific innovations: second-order optimization in XGBoost, warm-up cosine annealing for MLP, and dynamic k-selection for KNN. We also explore how to achieve high brain tumor MRI classification accuracy using limited, heterogeneous data and simple, easily implementable machine learning models. Comparative analysis reveals the strengths and limitations of each approach in handling the challenges of medical image analysis, including high dimensionality, limited data, and image quality variations. The proposed system provides clinicians with a reliable decision support tool for tumor detection, achieving high precision while maintaining computational efficiency.**

***Index Terms*—Brain tumor detection, Magnetic Resonance Imaging (MRI), Computer-aided diagnosis (CAD), Machine Learning, Medical Image Processing, Feature Extraction**

I. INTRODUCTION

Brain tumors represent a prevalent and life-threatening condition so the early detection of brain tumors is critical for patient survival and treatment outcomes, as tumors can lead to neurological dysfunction, seizures, and even death. Magnetic Resonance Imaging (MRI), introduced in the late 1970s, has become an essential tool in medical imaging, particularly in neurology. Compared to traditional X-rays or CT scans, MRI offers superior resolution, enabling clear visualization of brain structures. Its multi-angle, multi-layered imaging capabilities provide critical information for assessing tumor characteristics, size, and location [1]. However, early-stage tumors are often small and difficult to detect in MRI scans, requiring highly sensitive diagnostic tools. Significant challenges remain in brain MRI analysis, particularly regarding automated tumor detection and early diagnosis.

Conventional MRI interpretation relies on manual assessment, which introduces subjectivity and diagnostic variability. Second, manual diagnosis is time-consuming and susceptible to human error, especially when distinguishing between tumor and healthy tissue in complex MRI images. Machine learning and deep learning offer promising approaches to address these challenges by enabling rapid, objective, and high-accuracy tumor detection. By training on large annotated MRI datasets, machine learning models can automatically detect tumor regions, extract features, and classify lesions.

Despite the potential of artificial intelligence in medical imaging, several challenges remain. Variations in imaging equipment, acquisition protocols, and platforms result in inconsistent image quality and heterogeneous data formats. These include differences in image size, shape, and the number of channels, all of which negatively impact algorithm performance [2]. In addition, clinical trial data are difficult to obtain, and the limited availability of labeled datasets makes it impractical to train deep learning models effectively. Given such low-data conditions, it is critical to improve the efficiency of learning from limited and diverse data. Therefore, image preprocessing and augmentation play an essential role in enhancing data quality, increasing variability, and enabling robust model training. Therefore, this paper presents a comprehensive machine learning framework for automated brain tumor detection, integrating advanced image preprocessing techniques with five state-of-the-art classification models: Support Vector Machines (SVM), Multilayer Perceptron (MLP), XGBoost, Logistic Regression, and KNN. We explore how to achieve high brain tumor MRI classification accuracy using limited, heterogeneous data and simple, easily implementable machine learning models.

II. RELATED WORKS

In recent years, automatic recognition and classification of brain tumor MRI images have become important research areas in medical image analysis. Many studies have employed deep learning methods, especially convolutional neural networks (CNNs), to extract high-level features from images and achieve accurate classification. For example, U-Net and its variants have demonstrated excellent performance in brain tumor segmentation. However, these methods typically rely on large-scale annotated datasets for training [3]. Due to the limited availability of brain tumor MRI datasets and variations in data acquisition conditions and equipment, the generalization ability of deep learning models on small sample sizes and heterogeneous data remains limited.

Meanwhile, some studies have explored brain tumor classification based on traditional machine learning methods combined with texture and shape features. These approaches utilize handcrafted features such as Local Binary Patterns (LBP), effectively reducing the dependence on large-scale datasets.

Moreover, the models are relatively simple to implement and easy to deploy. For example, through proper feature extraction and preprocessing, traditional classifiers like Support Vector Machines (SVM) and Random Forests have still achieved satisfactory performance with limited data. However, due to the scarcity of data and insufficient preprocessing, these methods often suffer from low accuracy and ineffective models [4].

A Study used modified fuzzy C-means (MFCM) and ANN to segment and classify brain tumor MRI images. The method achieved sensitivity, specificity, and efficiency of 98.1%, 99.8%, and 99.59%, respectively [5]. We can find that most existing studies use a single dataset, which performs well on that specific dataset but is difficult to generalize to different clinical settings due to the shape, size, channels of the images might be different from different machines. Moreover, high accuracy is often reported, but comparisons of other key metrics such as sensitivity and specificity are inconsistent, making the evaluation of model reliability more complex.

III. METHODOLOGY

A. Data Preparing

As previously mentioned, the focus of our study is to achieve high accuracy in brain tumor MRI image classification using a relatively small amount of data with varying complexities and specifications, and by employing simple, easy-to-implement machine learning models with high interpretability.

For data preparing, we have integrated information from multiple publicly available MRI database platforms. By comparing the data types, research domains, and access methods of these platforms, we select appropriate data sources for subsequent tasks such as disease-specific modeling, neuroimaging analysis, or cross-dataset integration experiments. The table below provides an overview of major MRI data platforms, highlighting their key features and access links as a reference for data source selection.

TABLE I: MRI Data Platforms Overview

Platform	Introduction	Interlinkage
TCIA	Contains MRI images of various cancers, accompanied by annotations and clinical data	https://www.cancerimagingarchive.net/
ADNI (Alzheimer's Disease Neuroimaging Initiative)	Study the brain MRI scan data of Alzheimer's disease	http://adni.loni.usc.edu/
OpenNeuro	A large amount of functional MRI (fMRI) and structural MRI data widely used in neuroscience	https://openneuro.org/
OASIS	Includes brain MRI scan data of healthy people and dementia patients	https://www.oasis-brains.org/
Kaggle	Multiple MRI-related competition data, such as brain tumor segmentation, etc.	https://www.kaggle.com/

And here are the initial source image of our research.

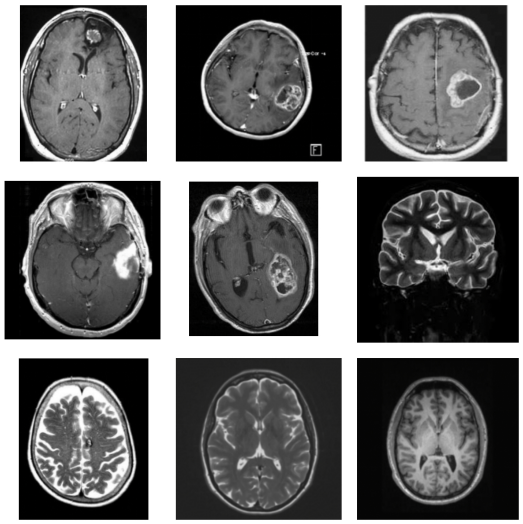


Fig. 1: Display of source images

The previously mentioned channels exhibit different issues. Pseudocolor images are artificially created by assigning distinct colors to single-channel data (typically grayscale or scalar values). Rather than displaying the true colors of objects, they utilize color variations to enhance visual discernibility. Each original data value is mapped to a specific color, thereby generating a vivid color image that aids in more intuitive understanding and analysis of the data.

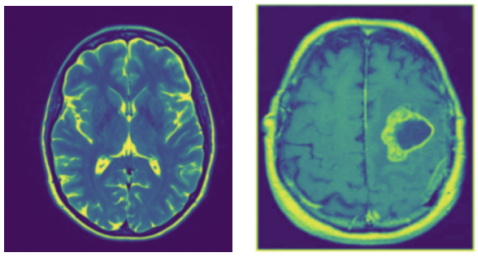


Fig. 2: Pseudocolor images of MRI images

Fig. 3: An example of pseudocolor encoding

The original MRI images before preprocessing exhibit varying sizes (height and width) and different numbers of channels. As shown in Figure 2 and Figure 4, some images have only a single channel and are therefore displayed using pseudocolor encoding for visualization purposes.

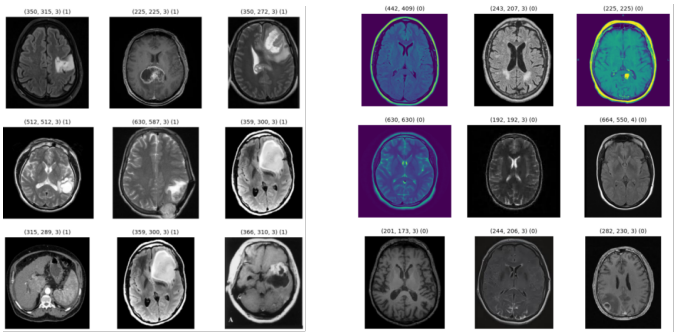


Fig. 4: Display of MRI images before preprocessing

In our dataset, the images vary in shape, featuring multiple sizes and resolutions.

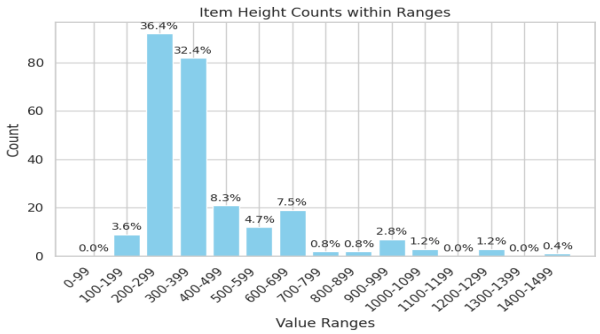


Fig. 5: Distribution of Image Heights Across Value Ranges

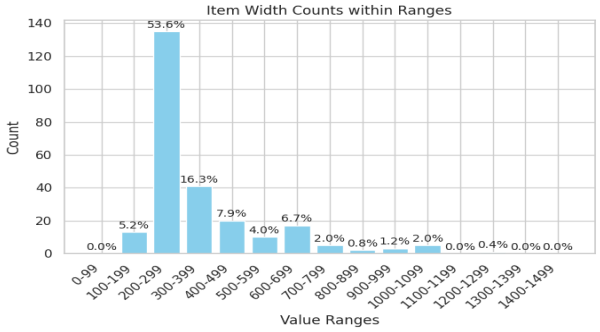


Fig. 6: Distribution of Image Widths Across Value Ranges

B. Data Preprocessing

Regarding the issues mentioned above, such as limited data volume, varying image sizes, and different numbers of channels, we propose the following data preprocessing strategies.

1. Resize the image's height and width

Image resizing algorithms based on interpolation methods alter the dimensions of an image by computing new pixel values through interpolation. These methods are widely used for enlarging or reducing image size. The core idea of interpolation is to estimate the value of a target pixel based on the values of its neighboring pixels.

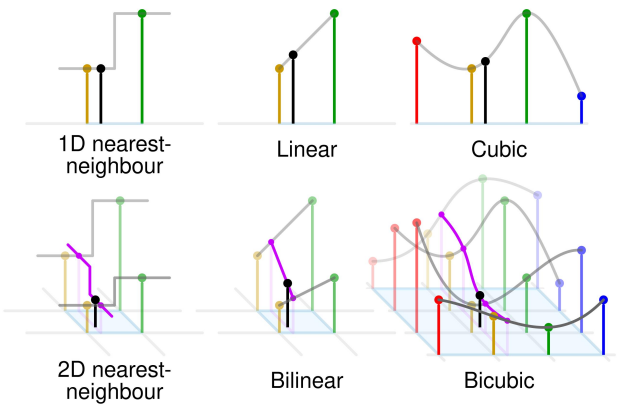


Fig. 7: Comparison of some 1- and 2-dim interpolations: *Black and red/yellow/green/blue dots correspond to the interpolated point and neighbouring samples, respectively. Their heights above the ground correspond to their values.*

There are various interpolation algorithms, as shown in the figure above. We compared several of them: for the nearest-neighbor algorithm [6], when downscaling an image, some details may be lost because multiple original pixels are mapped to a single new pixel. For standard linear interpolation methods, such as 1D interpolation and bilinear interpolation, high-frequency information is often lost because linear interpolation essentially acts as a low-pass filter. In contrast, higher-order methods like cubic and bicubic interpolation determine the value of a new pixel by fitting a 2D cubic polynomial to the values of the surrounding 4×4 original pixels, which leads to smoother results and better detail preservation. Therefore, in our paper, we chose the resize algorithm based on bicubic interpolation.

In image magnification, bicubic interpolation determines the color of a target pixel by considering the distances and color values of its 16 neighboring pixels, applying a weighted calculation to achieve smoother and more detailed enlargement. Its principle is similar to bilinear interpolation but involves a larger influence area.

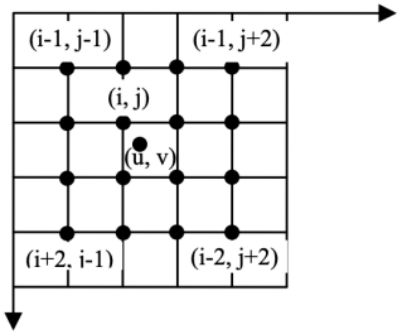


Fig. 8: Diagram of bicubic interpolation algorithm (Adapted from[7])

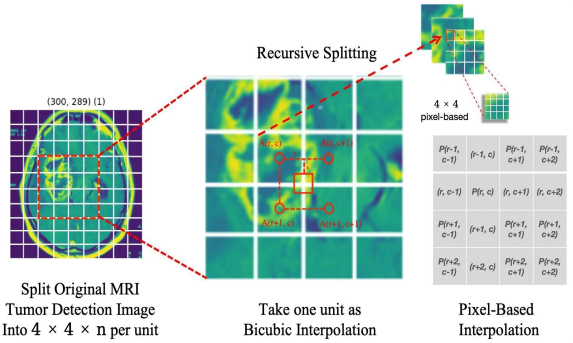


Fig. 9: Recursive Splitting Unit Based on Interpolation Computation

For the Bicubic algorithm, we aim to compute the pixel value $I'(x', y')$ at any arbitrary position (x', y') from the original image $I(x, y)$. Bicubic interpolation is essentially a two-dimensional cubic interpolation, which can be defined by a general interpolation kernel function $h(a)$.

The bicubic interpolation kernel function is defined as:

$$h(a) = \begin{cases} (1.5)|a|^3 - 2.5|a|^2 + 1, & 0 \le |a| < 1 \\ -0.5|a|^3 + 2.5|a|^2 - 4|a| + 2, & 1 \le |a| < 2 \\ 0, & |a| \ge 2 \end{cases}$$

- The term $|a|$ represents the relative distance from the center point.
- This kernel is continuous and has first-derivative continuity.

We extend the above 1D interpolation kernel function to 2D image data: For 2D images, interpolation employs the separable method:

The bicubic interpolation formula is:

$$I'(x', y') = \sum_{i=-1}^{2} \sum_{j=-1}^{2} w(i, j) \cdot I(x_0 + i, y_0 + j)$$

where:

- (x_0, y_0) is the nearest integer point to (x', y')
- $w(i, j) = h(x' - x_0 - i) \cdot h(y' - y_0 - j)$
- A total of $4 \times 4 = 16$ neighboring pixels are involved

The original MRI images had inconsistent shapes and sizes (as marked with xxx in the figure above). After performing bicubic interpolation resizing, we obtained the following standardized image shapes: $(225, 225, 1)$, $(225, 225, 3)$, and $(225, 225, 4)$.

Here is the comparison of details before and after interpolation-based resizing.

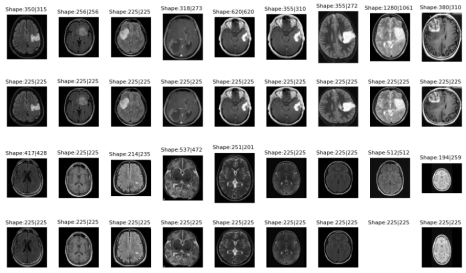


Fig. 10: Comparison of MRI images before (left) and after (right) bicubic interpolation-based resizing

Our observations indicate that bicubic interpolation successfully maintains textural details while resizing images to a uniform target resolution.

- **Upscaling Enlargement Effect Visualization**

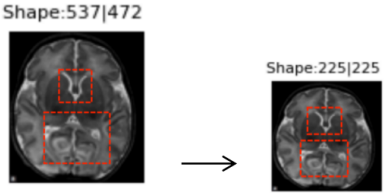


Fig. 11: Comparison image before and after resizing I: ***The U-shaped contour in the middle of the left image, as well as the complex, nearly symmetrical outline in the lower part, are still well-preserved in the right image after resizing.***

- **Downscaling Reduction Effect Visualization**

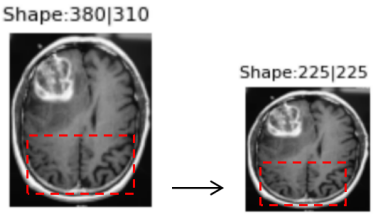


Fig. 12: Comparison image before and after resizing II: ***The left image's symmetrical sulcal patterns and fine anatomical details at the skull/gray-white matter interface remain well-preserved in the resized right image.***

- **Downscaling Reduction Effect Visualization**

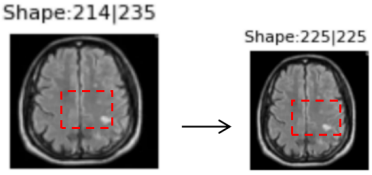


Fig. 13: Comparison image before and after resizing III: ***The left image demonstrates intact contours of the inferior horn of the lateral ventricle, with discernible parahippocampal gyrus architecture. Despite the need for contour interpolation in low-dimensional, small-scale imaging, the right image effectively preserves structural details through high-fidelity resizing techniques.***

The following is the resize algorithm based on Bicubic Interpolation.

Algorithm 1 Bicubic Interpolation Based Resize Algorithm

function U(s, a)
  if $|s| \leq 1$ **then**
    return $(a+2)|s|^3 - (a+3)|s|^2 + 1$
  else if $|s| \leq 2$ **then**
    return $a|s|^3 - 5a|s|^2 + 8a|s| - 4a$
  else
    return 0
  end if
end function

function PADDING($image, h, w, c$)
  Create $padded\_image$ of zeros $(h+4, w+4, c)$
  Copy $image$ into $padded\_image[2 : h+1, 2 : w+1, :]$
  Extend edges by replicating boundary pixels
  return $padded\_image$
end function

function BICUBIC($image, target\_h, target\_w, a$)
  Get dimensions h, w, c from $image$
  $padded \leftarrow$ PADDING($image, h, w, c$)
  Create $output$ array $(target\_h, target\_w, c)$
  $scale\_x \leftarrow w/target\_w$, $scale\_y \leftarrow h/target\_h$
  for $j \leftarrow 0$ to $target\_h - 1$ **do**
    for $i \leftarrow 0$ to $target\_w - 1$ **do**
      $x \leftarrow i \times scale\_x + 2$, $y \leftarrow j \times scale\_y + 2$
      Get 4×4 neighborhood around (x, y)
      Calculate weights using $u()$ for x and y directions
      Interpolate via weighted sum of neighborhood
      $output[j, i] \leftarrow$ interpolated value
    end for
  end for
  return $output$
end function

function BICUBIC_INTERPOLATION($image, target\_h, target\_w$)
  $result \leftarrow$ BICUBIC($image, target\_h, target\_w, -0.5$)
  Clip $result$ to $[0, 255]$
  return $result$ as uint8
end function

Main:
Read $input\_image$ from file
Set target dimensions $target\_h \leftarrow 128$, $target\_w \leftarrow 128$
$dst \leftarrow$ BICUBIC_INTERPOLATION($input\_image, 128, 128$)
Display result image and dimensions

2. Data Enhancement Methods

Image flipping is one of the most basic and effective image enhancement methods. By performing mirror-processing on the image in the horizontal or vertical direction, images with different orientations but the same semantics can be generated, thus increasing the diversity of data and reducing the model's dependence on specific directions of the image.

Let the image size be $H \times W$, with pixel coordinates (x, y):

- **Horizontal flip** (along the vertical axis):

$$I'(x, y) = I(W - 1 - x, y)$$

- **Vertical flip** (along the horizontal axis):

$$I'(x, y) = I(x, H - 1 - y)$$

Algorithm 2 Horizontal Flip

function RANDHORIZONTALFLIP($image$)
  Get dimensions H, W, C from $image$
  $flipped\_image \leftarrow$ copy of $image$
  for $i \leftarrow 0$ to $H - 1$ **do**
    $flipped\_image[i]$ and REVERSE along width axis
  end for
  return $flipped\_image$
end function

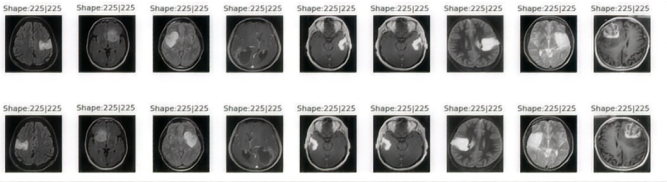


Fig. 14: A comparison picture before and after the horizontal folding operation

Image rotation is to obtain a new image perspective by rotating around the image center or an arbitrary point by a certain angle. The rotation operation changes the orientation of the image but does not change its essential semantics, which is suitable for enhancing the direction-robustness of the model.

Let (x_0, y_0) be the rotation center (typically image center), the new coordinates $[x', y']$ after rotating point (x, y) by angle θ are:

- **Random Rotation with** θ

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

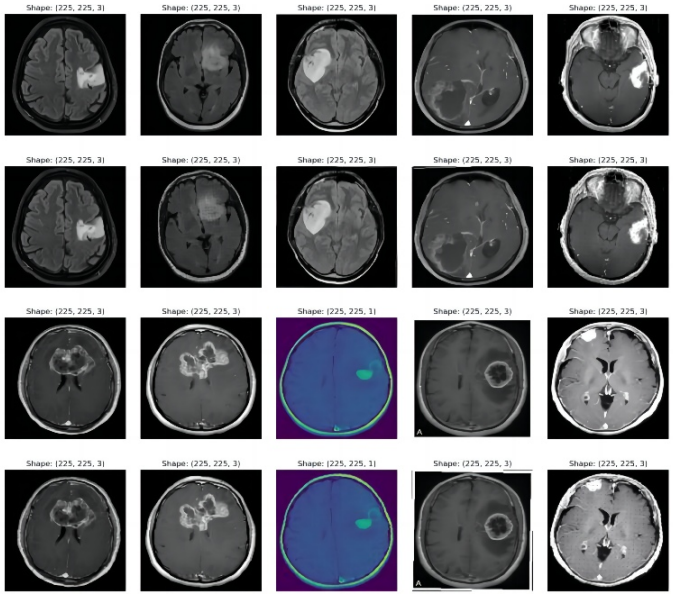


Fig. 15: The comparison diagram before and after performing the random angle selection operation rotation

Algorithm 3 Random Angle Selection-Based Rotation

function ROTATEIMAGE($image$, $angle$)
  Get H, W, C from $image$
  Convert $angle$ to radians $\rightarrow angle\_rad$
  $\cos\_angle \leftarrow \cos(angle\_rad)$, $\sin\_angle \leftarrow \sin(angle\_rad)$
  $center\_x \leftarrow W//2$, $center\_y \leftarrow H//2$
  for each pixel (i, j) in original image **do**
    $x \leftarrow j - center\_x$, $y \leftarrow i - center\_y$
    $new\_x \leftarrow \text{int}(\cos\_angle \cdot x - \sin\_angle \cdot y + center\_x)$
    $new\_y \leftarrow \text{int}(\sin\_angle \cdot x + \cos\_angle \cdot y + center\_y)$
    if new_x and new_y within image bounds **then**
      $rotated\_image[new\_y, new\_x] \leftarrow image[i, j]$
    else
      $rotated\_image[i, j] \leftarrow image[i, j]$ ▷ Fill with original
    end if
  end for
  return $rotated\_image$
end function

function RANDROTATION($image$, $angle\_range$)
  Select random $angle$ from $angle\_range$
  $rotated \leftarrow \text{RotateImage}(image, angle)$
  return $rotated$
end function

Color perturbation is an image enhancement technique that simulates external conditions such as variations in illumination and differences in photographic equipment. It randomly adjusts parameters including brightness, contrast, saturation, and hue within specified ranges to increase the color diversity of images.

Let the original pixel be $I(x, y, c)$, where $c \in \{R, G, B\}$ is the channel index.

- **Brightness Adjustment:** $I' = I \cdot b, \quad b \in [1 - \delta, 1 + \delta]$
- **Contrast Adjustment:** $I' = (I - \mu) \cdot c + \mu, \quad c \in [1 - \delta, 1 + \delta]$
  where μ is the image mean (or grayscale mean)
- **Saturation Adjustment (in HSV color space):** $S' = S \cdot s, \quad s \in [1 - \theta, 1 + \theta]$
- **Hue Adjustment (Hue shift):** $H' = (H + h) \mod 360$, $h \in [-\theta, \theta]$

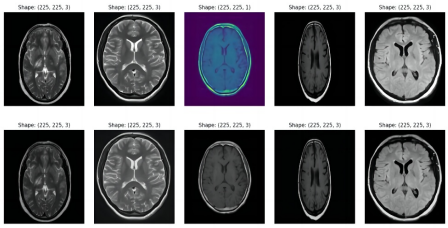


Fig. 16: Comparison before and after Color-Jitter operation

Due to the random Color-Jitter augmentation, the contour areas in the first and fifth images become significantly darker, while the central region of the second image appears brighter.

Algorithm 4 Color-Jitter Data Augmentation

function ADJUST_BRIGHTNESS($image$, $factor$)
  return CLIP$(image \times factor, 0, 255)$
end function
function ADJUST_CONTRAST($image$, $factor$)
  $mean \leftarrow \text{MEAN}(image)$ ▷ Over height and width
  return CLIP$((image - mean) \times factor + mean, 0, 255)$
end function
function ADJUST_SATURATION($image$, $factor$)
  if channels $\neq 3$ **then**
    Convert to 3-channel RGB
  end if
  $hsv \leftarrow \text{RGB\_TO\_HSV}(image/255)$
  $hsv[..., 1] \leftarrow \text{CLIP}(hsv[..., 1] \times factor, 0, 1)$
  return CLIP(HSV_TO_RGB$(hsv) \times 255, 0, 255$)
end function
function ADJUST_HUE($image$, $factor$)
  if channels $\neq 3$ **then**
    Convert to 3-channel RGB
  end if
  $hsv \leftarrow \text{RGB\_TO\_HSV}(image/255)$
  $hsv[..., 0] \leftarrow (hsv[..., 0] + factor) \bmod 1.0$
  return CLIP(HSV_TO_RGB$(hsv) \times 255, 0, 255$)
end function
function COLOR_JITTER($image$, $brightness$, $contrast$, $saturation$, hue)
  if channels $\neq 3$ **then**
    Convert to 3-channel RGB
  end if$param$ in $[brightness, contrast, saturation, hue]$
  if $param \neq 0$ **then**
    $factor \leftarrow 1 + \text{UNIFORM}(-param, param)$
    $image \leftarrow \text{adjust\_*}(image, factor)$ ▷ * is param name
  end if

  return CLIP$(image, 0, 255)$ as uint8
end function

However, we observed that although the height and width of the images are uniformly set to 225, there remains an issue: the number of channels is not consistent across all photos... After processing, we found variations such as (225, 225, 1), (225, 225, 3), and (225, 225, 4). Therefore, we need to handle the images based on their characteristics:

For grayscale images with a single channel, replicate the channel values to the R, G, and B channels to create pseudo-color images.

A grayscale image has only one channel, which is expanded into three channels (RGB) by copying the grayscale values to each channel.

Mathematical representation:

$$I_{\text{rgb}}(x, y, c) = I_{\text{Gray}}(x, y, 0), \quad \text{where } c \in \{0, 1, 2\}$$

This formula copies the grayscale intensity to all three color channels (Red, Green, Blue), producing an RGB image that appears grayscale.

Discard the 4th channel (Alpha channel) and retain the first three channels: Red, green, and blue (R, G, B).

Mathematical representation:

$$I_{\text{rgb}}(x, y, c) = I_{\text{rgba}}(x, y, c), \quad \forall c \in \{0, 1, 2\}$$

This formula represents copying the three color channels R, G and B (corresponding to $c = 0, 1, 2$ respectively) from an RGBA image (with alpha channel) directly to the corresponding channels of another RGB image (without transparency channel).

Next, we will introduce our image data preprocessing workflow.

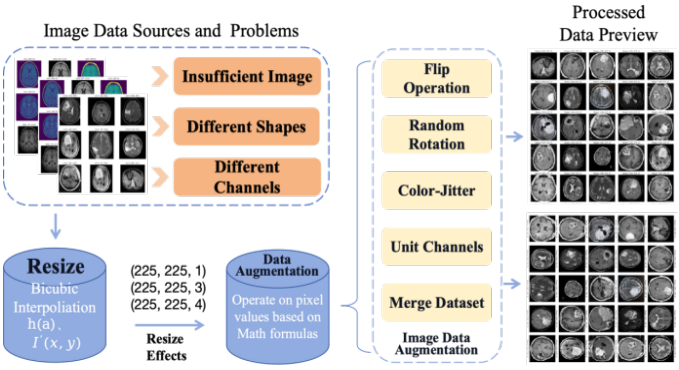


Fig. 17: Image Data Preprocessing Workflow

First, our image data sources face three main issues: insufficient image quantity, varying shapes and sizes, and inconsistent channel numbers.

To address these problems, the first step in preprocessing is to use a Bicubic Interpolation-based Resize algorithm, which standardizes all images to fixed dimensions (e.g., (225, 225, 1), (225, 225, 3), or (225, 225, 4)), ensuring consistent input data formats.

Next, the Data Augmentation phase expands the MRI image dataset through operations such as flipping (Flip), random rotation (Random Rotation), color jittering (Color-Jitter), and channel normalization (Unit Channels). Finally, the datasets are merged (Merge Dataset) to produce previews of the processed results. This workflow not only resolves the limitations of the raw data but also enhances the model's generalization capability through augmentation techniques.

The overall process is clear, systematically addressing the initial problems and ultimately delivering standardized and diversified image data.

And here are the image preview after data preprocessing.

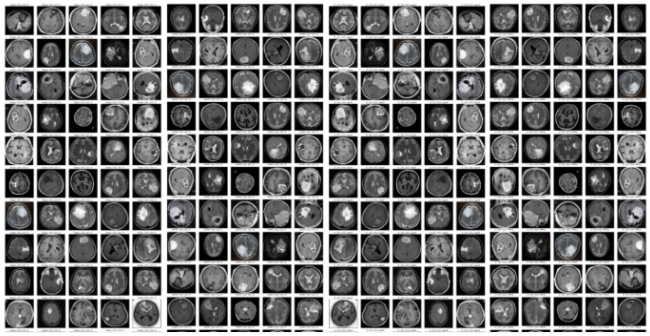


Fig. 18: A part of Image preview after data preprocessing

C. Feature Extraction

In brain tumor MRI image analysis tasks, feature extraction is a critical step for improving model performance and clinical interpretability.

Raw MRI images typically have extremely high dimensionality (as shown in Fig. [5] & Fig. [6]), which can lead to the "curse of dimensionality" if used directly. This results in high computational complexity and increased risk of overfitting. Therefore, dimensionality reduction techniques—such as texture features (e.g., Gray-Level Co-occurrence Matrix, GLCM; Local Binary Patterns, LBP), shape features (e.g., boundary irregularity, compactness), and edge information—can significantly compress the input space (scaled to (225, 225, C)), improving computational efficiency and model stability.

Moreover, these extracted high-level semantic features exhibit strong discriminative power, enabling more accurate differentiation between lesion and healthy tissues, thereby enhancing the classifier's performance in brain tumor identification. Additionally, removing redundant and noisy information while retaining key features helps improve the model's generalization ability, ensuring more robust performance on independent test sets or clinical data. This also enhances the interpretability of lesion regions, facilitating clinical acceptance of AI-assisted diagnosis.

Next, we will employ two feature extraction methods to derive HOG (Histogram of Oriented Gradients) and GLCM features from the MRI images:

1. HOG Features Extraction: Shape and Edge Orientation

In medical image analysis, particularly for brain cancer MRI processing, the Histogram of Oriented Gradients (HOG) is widely used for lesion identification and classification due to its sensitivity to edges, shapes, and local structures. As a feature-based descriptor in computer vision, HOG is specifically designed for robust object detection tasks.[6].

Gradient Computation: For an image $I(x, y)$, we compute the gradient at each pixel using the **Sobel operator** (first-order difference filter):

$$G_x(x, y) = I(x + 1, y) - I(x - 1, y)$$
$$G_y(x, y) = I(x, y + 1) - I(x, y - 1)$$

The gradient magnitude $M(x, y)$ and orientation $\theta(x, y)$ are calculated as:

$$M(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}$$
$$\theta(x, y) = \arctan\left(\frac{G_y(x, y)}{G_x(x, y)}\right)$$

For Constructing Cells, the image is divided into small **cells** (typically 8×8 pixels). For each cell:

- Gradient orientations $\theta(x, y)$ are quantized into **9 bins** (nbins=9)
- Orientation range is **[0°, 180°]**
- Each bin represents a 20 interval (e.g., $0 - 20$, $20 - 40$, etc.)
- Gradient magnitudes $M(x, y)$ are used as weights
- The cell's feature is a **9-dimensional** histogram vector

In Block Normalization, our goal is to improve robustness against lighting variations.Due to issues like local contrast variations, noise, or uneven illumination in images, HOG constructs blocks (e.g., 2×2 cells) over multiple cells and performs normalization on their internal feature vectors to enhance feature robustness.

Common normalization methods are as follows:

L2-Norm:

$$v' = \frac{v}{\sqrt{\|v\|^2 + \epsilon}}$$

L1-Norm:

$$v' = \frac{v}{\|v\|_1 + \epsilon}$$

where v is the original vector formed by concatenating all cell feature vectors in the block, and ϵ is a small constant to prevent division by zero. The block setting has cells_per_block as (2, 2), and the L1-Norm is selected for regularization, with a small constant of 1e-5 added to prevent division by zero.

In the Feature Vector Construction section, the final HOG feature vector is constructed by:

- Concatenating all normalized block vectors
- The total dimension is:

$$D = (\text{number of blocks}) \times (\text{dimension per block})$$

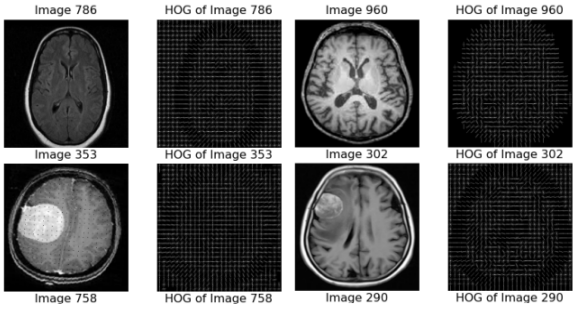


Fig. 19: HOG Feature Visualization

The overall feature extraction process achieves a processing speed of 9.24 images per second (image/s).

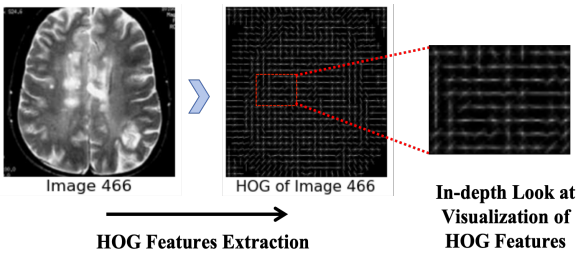


Fig. 20: In-Depth HOG Feature Visualization for Medical MRI Tumot Image Texture Analysis

Algorithm 5 Compact HOG Features Extraction

function COMPUTE_HOG(gray_image, pixels_per_cell=(8,8), cells_per_block=(2,2), nbins=9)
  Gradient Computation:
  $gx \leftarrow gray\_image[:, 2:] - gray\_image[:, :-2]$ ▷ Horizontal
  $gy \leftarrow gray\_image[2:, :] - gray\_image[:-2, :]$ ▷ Vertical
  $mag \leftarrow \sqrt{gx^2 + gy^2}$, $ori \leftarrow \text{DEG}(\arctan 2(gy, gx)) \bmod 180$
  Cell Histograms:
  $cell\_h, cell\_w \leftarrow pixels\_per\_cell$
  Initialize $histogram$ with $(H//cell\_h, W//cell\_w, nbins)$
  for each cell (i, j) **do**
    Bin cell's gradients into $nbins$ orientation bins
    Weight by magnitude and accumulate into $histogram[i, j, :]$
  end for
  Block Normalization:
  $block\_h, block\_w \leftarrow cells\_per\_block$
  for each overlapping block **do**
    $block \leftarrow \text{FLATTEN}(histogram[i : i+block\_h, j : j + block\_w, :])$
    $features \leftarrow \text{APPEND}(features, block/(\|block\|_2 + \epsilon))$
  end for
  Visualization (optional):
  Draw oriented lines in each cell proportional to histogram bins
  Normalize hog_image to [0,1] range
  return CONCATENATE($features$), hog_image
end function

2. GLCM Features Extraction

The Gray-Level Co-occurrence Matrix (GLCM) calculates the spatial relationships between pixel pairs and then extracts features from the matrix[8]. In medical image analysis, texture features are crucial for identifying the tissue structure of tumor regions. Among these, the Gray-Level Co-occurrence Matrix (GLCM) is a classic and highly effective texture analysis method used to quantify the spatial relationships between pixel intensity levels in an image.

The GLCM describes the joint probability distribution of pixel intensity levels at a specific direction and distance within the image. It captures repetitive structural patterns in the image, serving as a core feature representation of texture.

Let $I(x, y)$ be a 2D grayscale image, where the pixel intensity values range from $\{0, 1, \ldots, G-1\}$, with G being the number of gray levels (e.g., 8, 16, 256).

For a given direction θ (e.g., $0°$, $45°$, $90°$, $135°$) and distance d, the Gray-Level Co-occurrence Matrix $P_{d,\theta}(i, j)$ is defined as:

$$P_{d,\theta}(i,j) = \sum \left[\begin{cases} 1, & \text{if } I(x,y) = i, I(x+\Delta x, y+\Delta y) = j \\ 0, & \text{otherwise} \end{cases} \right]$$

where:

- (x, y) is the current pixel position
- $(x + \Delta x, y + \Delta y)$ is the neighboring pixel at direction θ and distance d
- $\Delta x = d \cdot \cos(\theta)$, $\Delta y = d \cdot \sin(\theta)$

The GLCM is typically normalized to a probability matrix:

$$P_{d,\theta}(i,j) = \frac{P_{d,\theta}(i,j)}{\sum_i \sum_j P_{d,\theta}(i,j)}$$

Algorithm 6 GLCM Feature Extraction

Require: Set of input images I, quantization levels $L = 8$
Ensure: Matrix F containing six GLCM features per image

```
for each image img ∈ I do
    Convert img to grayscale if necessary
    Normalize pixel intensities to range [0, 1]
    Quantize normalized image to L discrete levels
    Initialize empty co-occurrence matrix M of size L × L
    for each horizontal pixel pair (p1, p2) in img do
        Increment M[p1, p2]
    end for
    Make M symmetric by adding its transpose
    Normalize M to obtain probability matrix P
    Compute six statistical features:
        • Contrast: weighted sum of squared intensity differences
        • Dissimilarity: weighted sum of absolute intensity differences
        • Homogeneity: weighted sum with inverse distance weighting
        • Angular Second Moment (ASM): sum of squared probabilities
        • Energy: square root of ASM
        • Correlation: linear dependency of intensity pairs
    Store feature vector for current image
end for
return Feature matrix F for all images
```

- **Contrast**: Measures local intensity variations (higher values = rougher texture):

$$\text{Contrast} = \sum_{i,j} P(i,j) \cdot (i-j)^2$$

- **Energy (Uniformity)**: Reflects homogeneity (higher values = smoother texture):

$$\text{Energy} = \sum_{i,j} P(i,j)^2$$

- **Homogeneity**: Measures local similarity (higher values = more uniform texture):

$$\text{Homogeneity} = \sum_{i,j} \frac{P(i,j)}{1+(i-j)^2}$$

3. Traditional Hand-crafted Features Extraction

1) Intensity Features

Intensity Features describe the global brightness, contrast, and distribution of pixel intensities in an image.

- **Mean**: Measures the average brightness of the image:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

where N is the total number of pixels, and x_i is the intensity of the i-th pixel.

- **Standard Deviation (Std)**: Quantifies the dispersion of pixel intensities (higher values indicate greater contrast):

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2}$$

- **Skewness**: Measures the asymmetry of the intensity distribution:

$$\text{Skewness} = \frac{\mathbb{E}[(X-\mu)^3]}{\sigma^3}$$

  - **Skewness ¿ 0**: Right-tailed distribution (brightness concentrated in darker regions)
  - **Skewness ¡ 0**: Left-tailed distribution (brightness concentrated in brighter regions)

- **Kurtosis**: Describes the "peakedness" of the distribution (heavy-tailed or light-tailed compared to a normal distribution):

$$\text{Kurtosis} = \frac{\mathbb{E}[(X-\mu)^4]}{\sigma^4} - 3$$

Kurtosis ≥ 0 indicates a sharper peak with heavier tails, while kurtosis ≤ 0 indicates a flatter peak with lighter tails.

- **Quantiles (25%, 50%, 75%)**: Describe the spread of pixel intensities (Q2 = median)
- **Entropy**: Measures the randomness/complexity of the intensity distribution (higher entropy = more information):

$$H = -\sum_{k=1}^{B} p_k \log_2 p_k$$

where p_k is the normalized histogram bin probability.

2) GLCM Features

As mentioned above, extracting texture features using the Gray-Level Co-occurrence Matrix (GLCM) captures the spatial relationships between pixel intensities.

3) Gradient Features

Gradient features are extracted by computing changes in pixel intensity to highlight edge information, commonly used to emphasize structural boundaries in images and aid in image recognition and analysis.

- **Sobel Operator**:As mentioned earlier, the Sobel operator is an edge detection method that highlights edges in an image by computing the gradient of pixel intensity in the horizontal and vertical directions.
  Two 3×3 kernels for horizontal (G_x) and vertical (G_y) gradients:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

  - G_x detects vertical edges
  - G_y detects horizontal edges
- **Gradient Calculation**: Gradient Calculation is performed by convolving the image with Sobel kernels to compute the rate of intensity change at each pixel in the horizontal (G_x) and vertical (G_y) directions, which is used to extract edge information and detect contours and structures in the image.
  Convolves the image with Sobel kernels to compute gradients:

$$G_x[i,j] = \sum_{u=-1}^{1} \sum_{v=-1}^{1} I[i+u, j+v] \cdot G_x[u+1, v+1]$$

$$G_y[i,j] = \sum_{u=-1}^{1} \sum_{v=-1}^{1} I[i+u, j+v] \cdot G_y[u+1, v+1]$$

- **Gradient Magnitude**:Gradient Magnitude combines horizontal and vertical gradients to measure overall edge strength at each pixel, while statistical measures like mean and standard deviation of the gradient magnitude quantify the average and variability of edge intensity across the image, helping analyze texture and edge distribution.
  Combines G_x and G_y to quantify edge strength:

$$G = \sqrt{G_x^2 + G_y^2}$$

- **Statistics**:
  - **Mean gradient magnitude**: Average edge strength
  - **Std of gradient magnitude**: Variability in edge strength

The parallel coordinates plot (Fig. 21) is used to visualize medical imaging features, aiming to distinguish brain tumors (abnormalities) from normal tissues based on multidimensional feature differences. By comparing the distribution patterns of various features—such as intensity, texture, gradient, etc.—it helps researchers quickly identify which features are most valuable for classification, thereby assisting in diagnosis or feature selection.

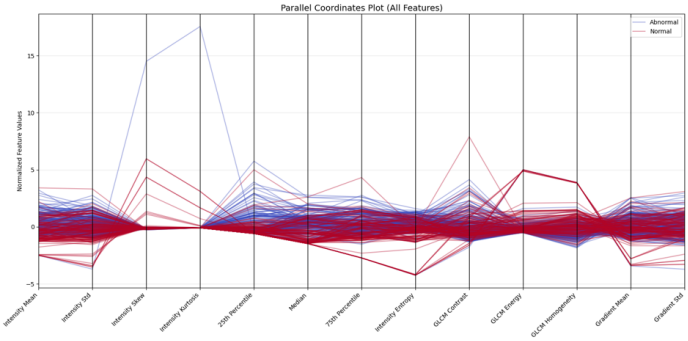


Fig. 21: Feature Distribution Visualization: Abnormal vs. Normal Brain Tissue

D. Model Implementation

Our system adopts a multi-model fusion architecture, which first performs standardized preprocessing on the input brain medical images, then conducts parallel analysis through five heterogeneous models (SVM, MLP, KNN, XGBoost, Logistic Regression), with each model assigned different weights based on historical performance. Finally, a meta-model (Meta Model) integrates the outputs of all sub-models to achieve high-precision, robust intelligent diagnostic decision-making.

- **Multi-Models Ensemble Learning Framework**

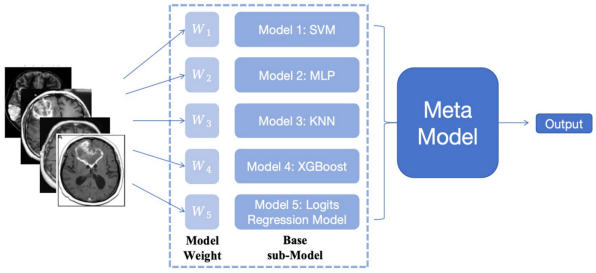


Fig. 22: Feature Distribution Visualization: Abnormal vs. Normal Brain Tissue

Subsequently, we will show the architectural framework and methodological innovations inherent to each constituent model within the ensemble.

1.Support Vector Machine (SVM)

Support Vector Machines (SVM) is a supervised learning algorithm based on statistical learning theory and convex optimization, with the core objective of finding an optimal hyperplane that maximizes the inter-class margin in a high-dimensional feature space, whether linear or nonlinear (achieved via the kernel method). The theoretical foundation of SVM originates from Vapnik-Chervonenkis (VC) theory, emphasizing structural risk minimization to ensure good generalization performance.

In the SVM model section, we employ Histogram of Oriented Gradients (HOG) for feature extraction, as previously described. Fig. 23 illustrates the demonstration of brain tumor detection performed with the Support Vector Machine (SVM) model.

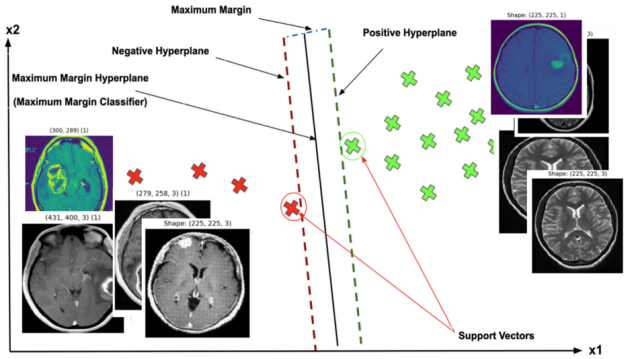


Fig. 23: Brain Tumor Detection Using SVM Margin Split

matical framework and algorithmic workflow, combined with code implementation insights.

Given a training dataset, $\{(x_i, y_i)\}_{i=1}^n$, $x_i \in \mathbb{R}^d$, $y_i \in \{-1, 1\}$,the primal optimization problem for binary classification SVM with soft margin is formulated as:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

It is subject to the constraints$y_i\big(w^\top\phi(x_i) + b\big) \geq 1 - \xi_i$, $\xi_i \geq 0$, $\forall i$, where $\phi(\cdot)$ is a feature mapping to a possibly high-dimensional space, C is the regularization parameter, and ξ_i are slack variables allowing for soft-margin classification.

Using the method of Lagrange multipliers, this primal problem is transformed into the dual formulation:

$$\max_{\alpha} \quad \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j K(x_i, x_j)$$

subject to

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n}\alpha_i y_i = 0,$$

where

$$K(x_i, x_j) = \phi(x_i)^\top\phi(x_j)$$

is the kernel function.

The dual problem is solved by the Sequential Minimal Optimization (SMO) algorithm, which iteratively updates pairs of Lagrange multipliers (α_i, α_j) to satisfy the Karush-Kuhn-Tucker (KKT) conditions:

- Select multipliers that violate the KKT conditions.
- Compute bounds $[L, H]$ for the update based on the labels of the selected pairs.
- Update multipliers using the step size

$$\eta = 2K(x_i, x_j) - K(x_i, x_i) - K(x_j, x_j),$$

ensuring a decrease in the dual objective function.

The kernel trick is key to implementing nonlinear SVMs. The following kernel functions are commonly used:

- **Linear kernel**:

$$K(x_i, x_j) = x_i^\top x_j$$

- **Polynomial kernel**:

$$K(x_i, x_j) = (x_i^\top x_j + \text{coef0})^{\text{degree}}$$

- **Gaussian kernel (RBF)**:

$$K(x_i, x_j) = \exp\big(-\gamma\|x_i - x_j\|^2\big),$$

where γ is a parameter often set inversely proportional to the input variance or manually tuned.

The bias term b is updated following the SMO heuristic, using support vectors that satisfy $0 < \alpha_i < C$ to compute an average ensuring the margin constraints hold.

The decision function for prediction is:

$$f(x) = \text{sign}\left(\sum_{i \in SV}\alpha_i y_i K(x, x_i) + b\right),$$

where support vectors (SV) are samples with α_i significantly greater than zero.

The training method iteratively optimizes the dual variables until the changes fall below a specified tolerance or the maximum number of iterations is reached. Upon completion, it stores the support vectors along with their corresponding labels and weighting parameters for use in subsequent predictions.

This algorithm relies on convex optimization, kernel function mapping, and SMO's dual gap control, guaranteeing sparsity (only support vectors influence the decision boundary) and theoretical optimality. This aligns with SVM's core principle of maximizing the margin to achieve good generalization.

2.Multilayer Perceptron (MLP)

Multilayer Perceptron (MLP) is a type of neural network that can learn complex patterns in medical imaging data to classify whether a brain tumor is present. It is suitable because it can model nonlinear relationships and extract features automatically from imaging inputs, improving detection accuracy.

In the MLP model section, we employ Histogram of Oriented Gradients (HOG) for feature extraction, as previously described.

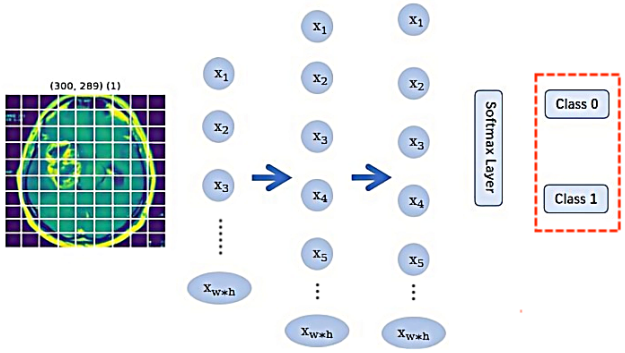


Fig. 24: Brain Tumor Detection Using MLP

During the data preprocessing stage, outlier detection is first performed on the input data $X \in \mathbb{R}^{N\times C\times H\times W}$ by establishing a mask through finiteness checking:

$$\text{mask}_{i,j,k} = \text{isfinite}(X_{i,j,k})$$

This mask can effectively identify anomalous samples containing non-numeric values (NaN/Inf). For the detected anomalies, the system provides two handling strategies:

- **Deletion mode:** directly retains valid samples $X_{\text{clean}} = X[\text{mask}]$,
- **Imputation mode:** fills missing values using the channel mean

$$\mu_c = \text{nanmean}(X_{\dots,c}),$$

which preserves data size while ensuring numerical stability.

During the data normalization stage, the tensor dimension is first adjusted to the (N, H, W, C) format, and then channel-wise statistics are computed:

$$\mu_c = \frac{1}{NHW}\sum_{i,j,k} X_{i,j,k,c}, \quad \sigma_c = \sqrt{\frac{1}{NHW}\sum_{i,j,k}(X_{i,j,k,c} - \mu_c)^2}$$

To avoid numerical instability, channels with standard deviation less than 10^{-6} are forcibly set to $\sigma_c = 1$. The final normalization uses a clipping strategy:

$$\tilde{X}_{i,j,k,c} = \text{clip}\left(\frac{X_{i,j,k,c} - \mu_c}{\sigma_c}, -3, 3\right)$$

This method effectively controls the impact of outliers while preserving the original distribution characteristics. After flattening, the feature matrix

$$X_{\text{flat}} \in \mathbb{R}^{N\times(CHW)}$$

provides standardized input for subsequent model processing.

In the traditional method part, a classical two-hidden-layer MLP architecture is used, whose forward propagation contains the following key steps:

$$a_1 = [1; X_{\text{flat}}]$$

This step adds a bias term before the input features to enhance model expressiveness. The linear transformation in the first layer is:

$$z_2 = a_1\Theta_1^\top, \quad \Theta_1 \in \mathbb{R}^{d_h\times(d_{in}+1)}$$

After passing through the nonlinear activation function $f(\cdot)$, the output is clipped within a reasonable range:

$$h = \text{clip}(f(z_2), -c)$$

This truncation prevents activation explosion. The second layer is processed similarly:

$$a_2 = [1; h], \quad p = \text{softmax}(a_2\Theta_2^\top), \quad \Theta_2 \in \mathbb{R}^{2\times(d_h+1)}$$

In the traditional implementation, activation functions mainly use ReLU-family functions, including the standard ReLU:

$$\text{ReLU}(x) = \max(0, x)$$

and the LeakyReLU with a leakage parameter:

$$\text{LeakyReLU}(x) = \begin{cases} x & x > 0 \\ \alpha x & \text{otherwise} \end{cases}$$

The training process uses the cross-entropy loss function:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{2} y_{i,c}\log p_{i,c}$$

This loss computes gradients via backpropagation and employs gradient clipping:

$$g_{\text{clipped}} = \begin{cases} g & \|g\|_2 \le \gamma \\ g\cdot\frac{\gamma}{\|g\|_2} & \text{otherwise} \end{cases}$$

This technique effectively controls the magnitude of parameter updates and prevents gradient explosion.

In the innovative method part, the focus is on improving activation functions and optimization strategies.

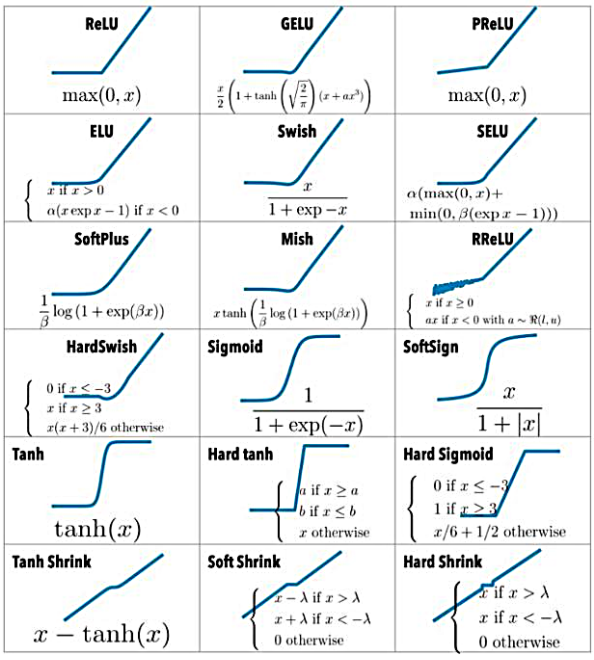


Fig. 25: Activation Functions Sets

Instead of using just one activation function, we introduced Mish activation function has unique mathematical properties and comprehensively compares the performance of three activation functions: ReLU, LeakyReLU, and Mish [**?**]:

$$\text{Mish}(x) = x\cdot\tanh(\text{softplus}(x))$$

Its gradient calculation includes a self-gating mechanism:

$$\frac{d}{dx}\text{Mish}(x) = \tanh(\text{softplus}(x)) + x\sigma(x)\left(1-\tanh^2(\text{softplus}(x))\right)$$

This structure allows gradients to maintain better flow during backpropagation.

Second, our optimizer adopts an improved AdamW algorithm with the parameter update rule. AdamW is an optimization algorithm that improves Adam by decoupling weight decay regularization from the gradient-based parameter updates to enhance generalization [10]:

$$\theta_t = \theta_{t-1} - \eta\left(\hat{v}_t + \epsilon\hat{m}_t + \lambda\theta_{t-1}\right)$$

This algorithm decouples the weight decay term from the gradient update, improving parameter update stability.

Finally, the learning rate scheduler innovatively combines warm-up with cosine annealing. Combining warm-up with cosine annealing is a learning rate schedule that gradually increases the learning rate at the start (warm-up) to stabilize training, then smoothly decreases it following a cosine curve to improve convergence [11]:

$$\eta_t = \begin{cases} \eta_{\max} \cdot \frac{t}{T_{\text{warm}}} & t \le T_{\text{warm}} \\ \eta_{\min} + 0.5(\eta_{\max} - \eta_{\min})\left(1 + \cos\left(\pi \frac{t-T_{\text{warm}}}{T-T_{\text{warm}}}\right)\right) & t > T_{\text{warm}} \end{cases}$$

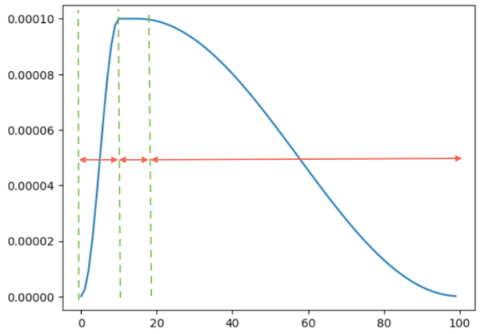


Fig. 26: Visualization Curve of Warmup and Cosine Decay

This dynamic adjustment allows the learning rate to smoothly transition, ensuring training stability in the early stage and fine-tuning precision later.

3.eXtreme Gradient Boosting (XGBoost)

XGBoost (eXtreme Gradient Boosting) is an ensemble learning algorithm based on Gradient Boosting Decision Trees (GBDT), which iteratively optimizes the objective function using an additive training strategy. Its core idea is to sequentially build multiple regression trees to progressively correct the prediction errors of previous models. Compared with traditional GBDT, XGBoost introduces a regularization term and a second-order Taylor expansion to balance model accuracy and generalization ability.

We use Traditional Hand-crafted Features Extraction in our XGBoost model. Our proposed XGBoost classifier includes three key improvements on the traditional gradient boosting trees: (1) a conditional Hessian matrix based on feature correlations and a robust weighting mechanism, (2) split gain calculation based on sample weights, (3) adaptive learning rate adjustment.

For the basic XGBoost model framework, we utilized the following formulas and concepts:

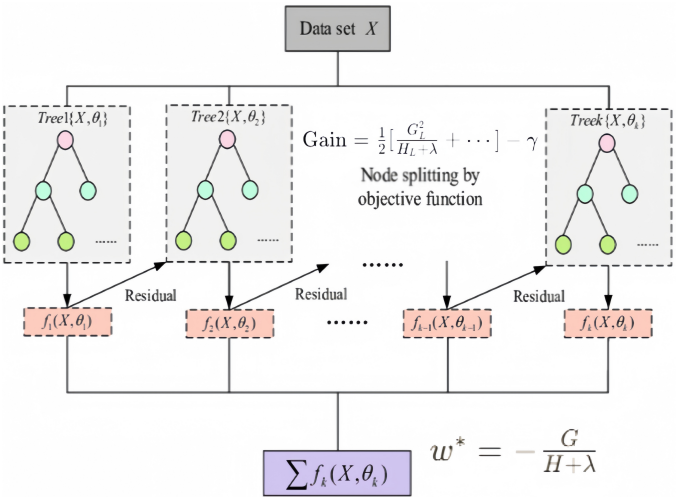


Fig. 27: Feature Distribution Visualization: Abnormal vs. Normal Brain Tissue

The objective function of XGBoost consists of the loss function and the regularization term:

$$L(\phi) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where:

- $L(y_i, \hat{y}_i)$ is the loss function (for example, the cross-entropy loss used in logistic regression). - $\Omega(f_k)$ is the complexity regularization term of the k-th tree:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

Here, T is the number of leaf nodes, and w represents the leaf weights. γ and λ are hyperparameters (gamma and lambda).

At each boosting iteration t, XGBoost minimizes the objective by adding a new tree f_t to improve the predictions. The loss is approximated using a *second-order Taylor expansion* around the current prediction $\hat{y}^{(t-1)}$:

$$L^{(t)} \approx \sum_{i=1}^{n}\left[L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)\right] + \Omega(f_t)$$

where:

- $g_i = \frac{\partial L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$ is the gradient (first-order derivative).

- $h_i = \frac{\partial^2 L(y_i, \hat{y}_i^{(t-1)})}{\partial (\hat{y}_i^{(t-1)})^2}$ is the Hessian (second-order derivative).

For *logistic regression* (binary classification), these derivatives simplify to:

$$g_i = \hat{p}_i - y_i, \quad h_i = \hat{p}_i(1 - \hat{p}_i)$$

where $\hat{p}_i = \frac{1}{1+e^{-\hat{y}_i}}$ is the predicted probability.

The Hessian h_i acts as a confidence measure—higher values indicate more reliable predictions, influencing how much the model adjusts for each sample.

To grow the tree, XGBoost evaluates potential splits using again metric, which balances improvement in prediction accuracy against added complexity:

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

where:

- $G_L = \sum_{i \in \text{left}} g_i$, $H_L = \sum_{i \in \text{left}} h_i$ are the sums of gradients and Hessians in the left subtree.
- G_R and H_R are the corresponding sums in the right subtree.

After determining the tree structure, the optimal weight w^* for each leaf is computed to minimize the loss:

$$w^* = -\frac{G}{H + \lambda}$$

where G and H are the sums of gradients and Hessians for that leaf.

Here is our algorithm design:

For $t = 1$ to T Do:

1. Calculate the first-order derivative g_{ti} and second-order derivative h_{ti} of the loss function L for the i-th sample based on $f_{t-1}(x_i)$. Compute the sums of first-order derivatives $G_t = \sum_{i=1}^{n} g_{ti}$ and second-order derivatives $H_t = \sum_{i=1}^{n} h_{ti}$ for all samples.
2. Attempt to split the regression tree based on the current node, with the default score = 0. G_t and H_t represent the sums of first- and second-order derivatives for the node to be split.

For $k = 1$ to d Do:

2.1 Initialize $G_L = 0$, $H_L = 0$.

2.2 Sort the training set samples by attribute k in ascending order to obtain a new training set: $D_s \leftarrow SortByKey(D, k)$.

For x_i in D_s Do:

5. For the current sample, calculate the corresponding left and right subtrees, and compute their sums of first- and second-order derivatives:

$\{G_L \leftarrow G_L + g_{ti}, G_R \leftarrow G_t - G_L$
$H_L \leftarrow H_L + h_{ti}, H_R \leftarrow H_t - H_L\}$

6.Update the maximum score:

$score = \max(score, \frac{1}{2}[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}] - \gamma)$

3. Split the subtree based on the feature and threshold corresponding to the maximum score.
4. If the maximum score = 0, the current regression tree is complete. At this point, compute the weights ω_{tj} for all leaf regions to obtain the weak learner $h_t(x)$, update the strong learner $f_t(x)$, and proceed to the next iteration. Otherwise, return to Step 2 to continue attempting to split the subtree.

Fig. 28: Algorithm Design For XGBoost Tree Structure

By incorporating L2 regularization and minimum split gain control, our model effectively prevents overfitting while maintaining simplicity and efficiency. Compared to traditional decision tree methods, our model enhances the gradient boosting process with second-order derivatives (Hessian matrix) optimization, improving its adaptability to imbalanced brain tumor datasets, such as when normal samples greatly outnumber tumor samples.

To address the varying clinical requirements in brain tumor detection tasks, such as emphasizing recall during screening and precision during diagnosis, our model supports dynamic adjustment of the classification threshold. This allows physicians to flexibly select the optimal decision boundary based on real-world needs without retraining the model. This feature enables our method to better adapt to diverse clinical application scenarios.

4. Logistic Regression

In our Logistic Regression model, we use HOG Features Extraction as we mentioned.

We built the model based on the traditional logistic regression formula, with additional innovations including the introduction of a momentum mechanism, an early stopping mechanism, and an automatic threshold selection mechanism based on F1 score, precision, and recall.

In binary classification problems, logistic regression is a widely used statistical model due to its probabilistic interpretation and efficient training methods. To reduce the risk of overfitting, L2 regularization is often incorporated into the loss function. The regularized logistic loss function is defined as:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\Big[-y^{(i)}\log(h_\theta(x^{(i)})) - (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\Big] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

where m is the number of training examples, λ is the regularization strength, and $\theta \in \mathbb{R}^n$ is the parameter vector. The function $h_\theta(x)$ is the hypothesis, defined as the sigmoid activation:

$$h_\theta(x) = \sigma(z) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = \theta^T x$$

To improve numerical stability during training, the value of z is clipped to the interval $[-500, 500]$ to avoid overflow in computing e^{-z}. Similarly, the output $h_\theta(x)$ is clipped to $[10^{-15}, 1-10^{-15}]$ to avoid undefined values in the logarithmic terms.

The regularization term penalizes large weights by adding the squared magnitude of the parameters (excluding the bias term θ_0) to the loss. This helps reduce model variance and prevents overfitting, especially when the number of features is large or the training data is noisy.

Parameter optimization is performed using gradient descent. The parameters are updated as:

$$\theta := \theta - \alpha \cdot \nabla_\theta J(\theta)$$

where α is the learning rate. The gradient of the regularized loss function is given by:

$$\nabla_\theta J(\theta) = \frac{1}{m}X^T(h - y) + \frac{\lambda}{m}\theta$$

To exclude the bias term θ_0 from regularization, only θ_j for $j \geq 1$ are regularized, which is typically implemented in code by applying the regularization term to `theta[:, 1:]`.

Next, we will introduce the main two innnovations of our Logistic Regression model.

Momentum is an optimization technique designed to accelerate gradient-based learning by incorporating historical gradient information into parameter updates [12]. Unlike traditional gradient descent, which relies exclusively on the current gradient at each iteration, Momentum introduces a velocity term v_t that accumulates past gradients with an exponential decay factor. This approach can be mathematically expressed as:

$$v_t = \beta v_{t-1} + (1 - \beta)\nabla_\theta J(\theta_t)$$

$$\theta_{t+1} = \theta_t - \eta v_t$$

where:

- $\nabla_\theta J(\theta_t)$ is the current gradient of the loss function J with respect to parameters θ,
- η is the learning rate,
- β (typically set between 0.8 and 0.99) controls the contribution of past gradients.

By smoothing parameter updates, Momentum helps mitigate oscillations in steep or noisy optimization landscapes, leading to faster convergence and improved stability compared to standard gradient descent. This mechanism is particularly effective in high-curvature or ravine-like regions of the loss surface, where gradients may otherwise fluctuate drastically.

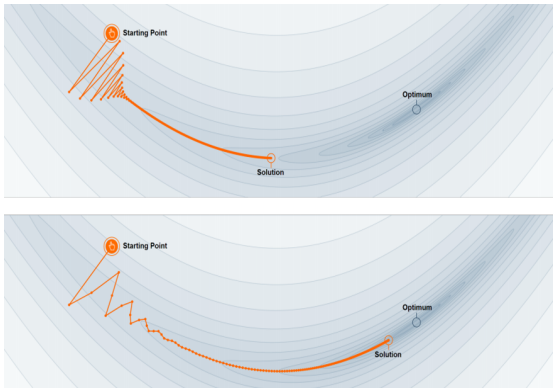


Fig. 29: The comparison diagram between using the Momentum optimization algorithm and conventional optimization methods

From Fig. 29, we can clearly observe the difference between the momentum mechanism and standard one.

Optimization algorithms with the Momentum mechanism effectively leverage the spatial continuity of medical imaging features by accumulating historical gradient information. This smooths the optimization trajectory and ensures stable convergence in complex brain tumor detection tasks, particularly in MRI multimodal feature spaces. As a result, training is accelerated,, and the model more efficiently approaches optimal detection performance. Additionally, Momentum offers greater robustness to noise and local perturbations, helping to mitigate class imbalance issues common in medical imaging.

In contrast, traditional optimization methods lack the ability to utilize historical gradients, leading to oscillating optimization paths and unstable convergence. These limitations are especially pronounced in high-dimensional, sparse medical data, where such methods are more prone to local optima and noise interference. Their sensitivity to imbalanced distributions also hampers the accurate capture of subtle pathological features, reducing detection accuracy.

5.K-Nearest Neighbors (KNN)

KNN is a non-parametric classifier that assigns a label by majority vote among the k nearest neighbors in feature space.

Texture features are extracted from each normalized MRI slice using the gray-level co-occurrence matrix (GLCM), which captures second-order statistics of gray-level spatial relationships. By quantifying properties such as contrast, homogeneity, and energy in local neighborhoods, GLCM produces robust descriptors that effectively distinguish tumor tissue from normal brain parenchyma and serve as compact inputs for KNN classification.

and define the weighted Euclidean distance

$$d(\mathbf{f}, \mathbf{f}^{(i)}) \;=\; \sqrt{\sum_{j=1}^{6} w_j \left(f_j - f_j^{(i)}\right)^2}\,.$$

For a query feature vector $\mathbf{f}$, let $N_k(\mathbf{f})$ be its k nearest neighbors under the distance above. We estimate class probabilities by

$$P(y = c \mid \mathbf{f}) = \frac{1}{k} \sum_{i \in N_k(\mathbf{f})} \mathbf{1}\{y^{(i)} = c\},$$

and predict the class with highest $P(y = c)$. The value $P(y = 1)$ is used as a confidence score for threshold-based decisions.

We perform five-fold stratified cross-validation over odd k values from 3 to 15. For each k we record the mean F1–score in figure 45

$$\overline{\mathrm{F1}}(k) = \frac{1}{5} \sum_{r=1}^{5} \mathrm{F1}_r(k)$$

and its fold-to-fold variance. We choose the smallest k that maximizes $\overline{\mathrm{F1}}$ while keeping variance low, yielding $k = 3$ as shown below.

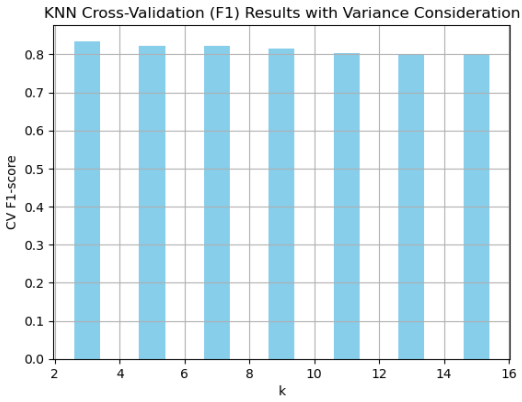


Fig. 30: Dynamic k Selection

Figure 31 presents a three-panel overview of the KNN workflow. The first panel shows the original MRI slice in gray scale. The second displays the normalized GLCM co-occurrence matrix. The third projects all training samples into two dimensions via PCA, colors them by true label (blue=no tumor, red=tumor), marks a chosen test point ($\star$), and draws a dashed circle with radius equal to its third-nearest neighbor distance.

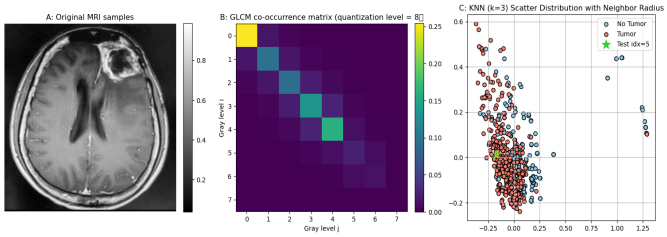


Fig. 31: The KNN decision pipeline

Weighted KNN, empowered by inverse-variance feature weighting and a data-driven dynamic choice of k, effectively leverages the local texture continuity inherent in GLCM-derived MRI descriptors. By scaling each dimension according to its stability, the classifier emphasizes clinically salient patterns—such as subtle variations in contrast and homogeneity—while suppressing noise. The cross-validated selection of k further adapts the receptive field to the heterogeneity of the dataset, yielding smooth and stable decision boundaries across folds.

IV. EXPERIMENTAL STUDY AND RESULT ANALYSIS

A. Model Evaluation

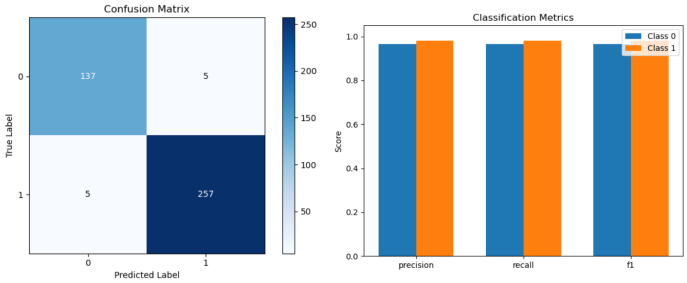
1. Model Performance Evaluation

Cross-validation (CV) is a crucial technique in machine learning for evaluating model generalization performance, selecting hyperparameters, and preventing overfitting. In our five models, we all employed 5-fold cross-validation (CV=5). We also evaluate each model using suitable metrics (e.g., accuracy, precision, recall, F1 score, RMSE).

1) SVM

Here are the model performance evaluation results of SVM:

Fig. 32 presents the confusion matrix and classification report of the Support Vector Machine (SVM) model on the test set, which are used to evaluate its classification performance.



(a) Confusion matrix (b) Classification matrix

Fig. 32: Confusion and classification matrices of SVM

Figure 33 shows the cross-validation accuracy of the Support Vector Machine (SVM) model for each fold, which helps assess the model's stability and generalization performance.

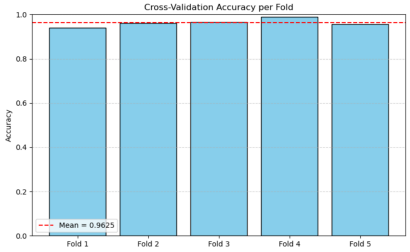


Fig. 33: Cross-Validation Accuracy per Fold of SVM

2) MLP

For the three activation functions, we conducted a comparison, and the results are shown in the figure below.

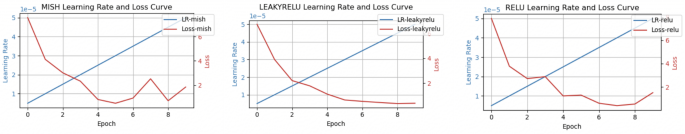


Fig. 34: Comparision of Learning Curve and Loss between three activation functions

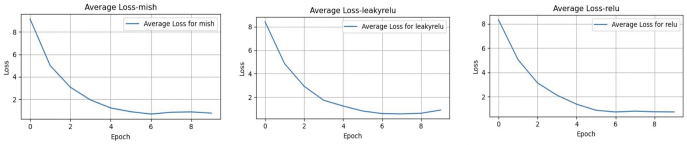


Fig. 35: Comparision of Loss between three activation functions

And here is the comparison of accuracy between three activation functions:

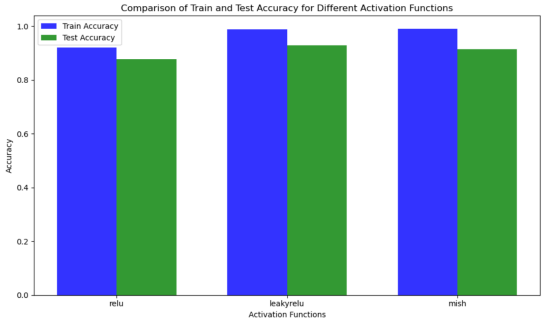


Fig. 36: Comparision of Accuracy between three activation functions

Activation Function	Training Accuracy	Test Accuracy
ReLU	0.9202	0.8768
LeakyReLU	0.9876	0.9286
Mish	0.9893	0.9147

TABLE II: Comparison of activation functions on training and test accuracy

3) XGBoost

We assess the model using complementary plots:

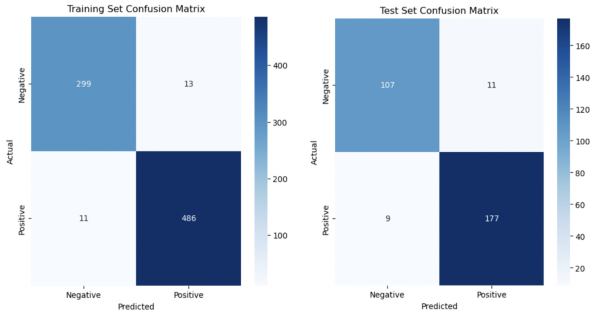


Fig. 37: Confusion matrix of XGBoost

For the threshold optimization, we plot a analysis image to decide the best threshold.

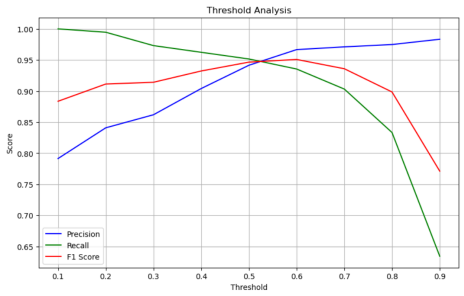


Fig. 38: Threshold analysis of XGBoost

We have output the performance of different folds in cross-validation, as shown in the figure below.

Fig. 39: Performance in cross-validation of XGBoost

4) Logistic Regression

We assess the model using complementary plots:

For the ROC Curve and Precision–Recall curve, we have:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}$$

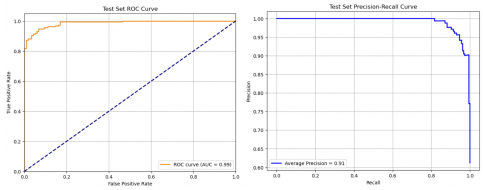


Fig. 40: ROC Curve and Precision–Recall curve of Linear Regression

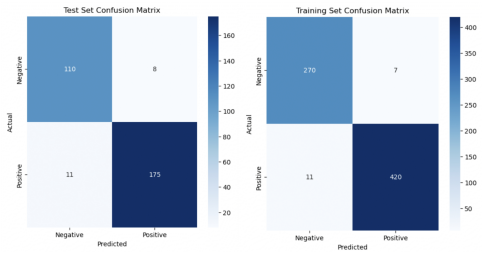


Fig. 41: Confusion matrix of Linear Regression

For the threshold optimization, we plot a analysis image to decide the best threshold.

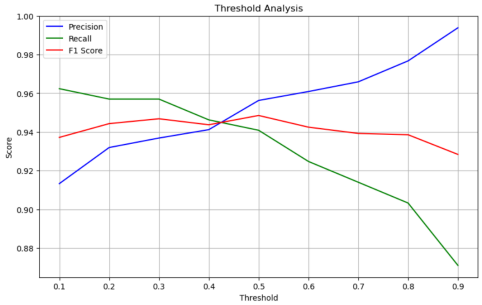


Fig. 42: Threshold analysis of Linear Regression

We have output the performance of different folds in cross-validation, as shown in the figure below.

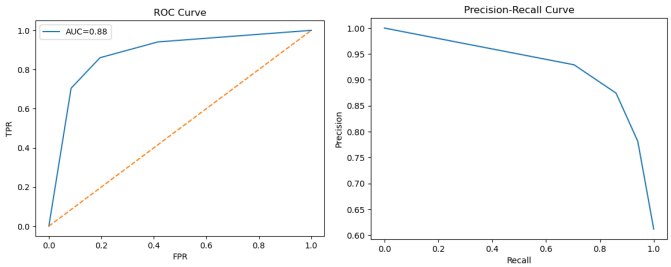
Fig. 43: Performance in cross-validation of Linear Regression

5) KNN

We assess the model using complementary plots:

We plot the Receiver Operating Characteristic curve in Fig. 44, summarizing the trade-off between true positive rate and false positive rate.

For the ROC Curve and Precision–Recall curve, we have:



(a) ROC Curve (b) Precision–Recall Curve

Fig. 44: ROC and Precision–Recall Curves of KNN

Confusion matrix: the confusion matrix shows true vs. predicted labels on the test set.

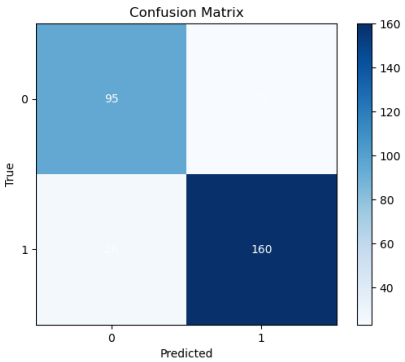


Fig. 45: Confusion matrix of KNN

Learning curve: training vs. validation accuracy as a function of training set size (Figure 46).

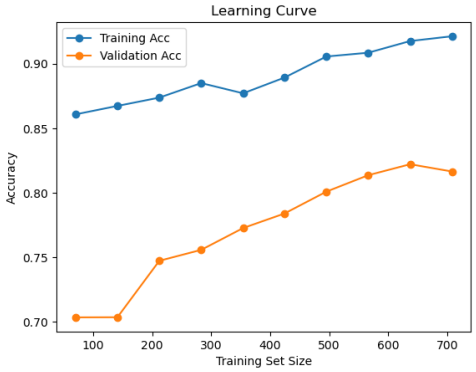


Fig. 46

2. Discussion of Strengths and Weaknesses

1) SVM

Support Vector Machines (SVM) are effective for brain tumor detection due to their solid foundation in statistical learning theory, enabling strong generalization even with limited medical imaging data. By maximizing the margin between classes, SVM reduces overfitting risks, making it suitable for small datasets. The kernel trick allows SVM to model nonlinear relationships in high-dimensional MRI features efficiently. The sparsity of support vectors also enhances inference speed. Moreover, convex optimization ensures a globally optimal solution, avoiding local minima common in neural networks.

Despite these strengths, SVM faces challenges in medical image analysis. Its performance depends on kernel choice and hyperparameter tuning such as C and γ, which can be time-consuming and require expertise. The training time increases sharply with large datasets, limiting its scalability for high-resolution 3D MRI data. Additionally, SVM lacks built-in feature extraction and relies on handcrafted inputs like Histogram of Oriented Gradients (HOG), which may overlook subtle tumor features. Class imbalance, often present in medical data, can further degrade performance without appropriate handling.

2) MLP

Multilayer Perceptrons (MLP) are well-suited for brain tumor detection because they can automatically learn complex, nonlinear patterns from high-dimensional MRI data without extensive manual feature extraction. By using multiple hidden layers, MLPs can hierarchically capture important features, improving classification accuracy. Advanced activation functions (e.g., Mish) and optimization techniques (e.g., AdamW with warm-up and cosine annealing) further enhance training efficiency and model performance. MLPs also work well with preprocessed data, such as normalized and outlier-removed inputs, increasing their robustness to noise and variability in medical images.

However, MLP also have limitations. Their fully connected structure results in high computational cost, especially with large 3D medical images. Unlike convolutional neural networks (CNNs), MLPs do not preserve spatial relationships in

the data, which can lead to loss of critical local information. In addition, MLPs require careful tuning of hyperparameters and are more prone to overfitting, particularly when training on small or imbalanced medical datasets. Techniques like dropout and regularization can help, but may not fully overcome these challenges in clinical settings.

3) XGBoost

XGBoost excels in brain tumor detection due to its high accuracy, efficiency, and robustness in handling structured medical data. By leveraging gradient boosting with second-order optimization (Hessian matrix), it effectively minimizes prediction errors while controlling model complexity through L1/L2 regularization, reducing overfitting risks—especially critical for small medical datasets. XGBoost automatically handles feature interactions and nonlinear relationships, eliminating the need for manual feature engineering required by methods like SVM. Its built-in handling of missing values and support for sample weighting make it ideal for imbalanced datasets (e.g., rare tumor cases). Additionally, dynamic threshold adjustment allows clinicians to prioritize recall (screening) or precision (diagnosis) without retraining, enhancing clinical adaptability.

However, XGBoost has limitations in medical imaging tasks. Unlike CNNs, it cannot directly process raw image data, relying on handcrafted features (e.g., HOG or radiomics) that may miss subtle spatial patterns in MRI scans. Hyperparameter tuning (e.g., learning rate, tree depth) can be complex and computationally expensive, requiring cross-validation with limited labeled data. While efficient for tabular features, its performance plateaus with high-dimensional image-derived features compared to deep learning. The model's "black-box" nature also limits interpretability for clinical decision-making, though techniques like SHAP values can partially mitigate this.

4) Logistic Regression

Logistic Regression, especially with handcrafted features like HOG, offers simplicity, interpretability, and ease of implementation. Its probabilistic output aids clinical decision-making, while momentum-based gradient descent speeds up convergence and stabilizes training in high-dimensional data. L2 regularization helps prevent overfitting—crucial for small medical datasets. The model also allows automatic threshold tuning (e.g., optimizing F1, precision, or recall), making it flexible for clinical priorities. It handles numerical stability well, using clipping to prevent computation errors.

However as a linear model, Logistic Regression struggles with complex nonlinear patterns in MRI data unless paired with strong feature engineering, which may miss subtle tumor cues. It requires careful hyperparameter tuning and is sensitive to class imbalance. Unlike CNNs or kernel-based SVMs, it can't learn spatial hierarchies from raw images and relies heavily on manual features. Its performance can suffer with redundant or correlated inputs and may not generalize well across varied imaging settings or tumor types.

5) KNN

As an instance-based, non-iterative method, KNN delivers rapid inference without extensive training, and its soft-voting probabilities allow clinicians to balance sensitivity and specificity through straightforward thresholding. Moreover, focusing on robust texture dimensions helps mitigate class imbalance and local perturbations, ensuring reliable detection of faint tumor signatures. In contrast, a traditional KNN with uniform weighting and a fixed k often produces erratic neighbor selections in high-dimensional, imbalanced imaging data, resulting in unstable classification regions, heightened sensitivity to outliers, and reduced ability to capture fine-grained pathological features.

The main drawback of KNN is its high computational cost at inference time: for each new sample, the algorithm must compute distances to all n training examples in a d-dimensional feature space, resulting in $O(n \times d)$ complexity. This makes real-time decision-making and scalability challenging as the dataset grows.

B. Model Comparision

Here are the Model Comparision:

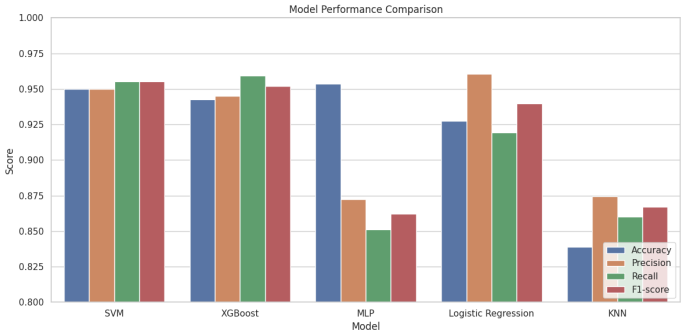


Fig. 47: Comparision Visualizationn

TABLE III: Model Performance Comparison

Metric	SVM	XGBoost	MLP	LR	KNN
Accuracy	0.9500	0.9426	0.9538	0.9276	0.8388
Precision	0.9500	0.9451	0.8722	0.9607	0.8743
Recall	0.9551	0.9594	0.8512	0.9194	0.8602
F1-score	0.9553	0.9518	0.8622	0.9396	0.8672
Runtime (s)	185.15	2185.39	61.71	474.93	8.81
Memory Before (MB)	2991.77	2559.70	969.17	1447.82	2561.73
Memory After (MB)	4164.27	2584.17	325.41	1450.39	2571.78
Memory Δ (MB)	1172.50	24.48	-643.76	2.57	10.04

From the table and Fig. 47, it is evident that both SVM and XGBoost demonstrate strong performance in terms of accuracy and recall, each reaching around 95%. This indicates that these models provide overall accurate classification while minimizing the omission of positive samples. Notably, XGBoost achieves the highest recall, making it suitable for scenarios where minimizing false negatives is critical. Meanwhile, SVM attains a slightly higher F1-score, reflecting a good balance between precision and recall. Although MLP shows a marginally higher accuracy than SVM and XGBoost, its precision and recall are relatively lower, suggesting potential

bias toward certain classes. In contrast, Logistic Regression (LR) maintains stable performance with the highest precision, making it favorable in applications where reducing false positives is important. KNN exhibits comparatively weaker overall performance, with noticeably lower accuracy and recall than the other models; despite its shortest runtime, its accuracy is insufficient for complex tasks.

Regarding resource consumption, XGBoost requires the longest runtime, likely due to the complexity of its tree-based training process. MLP, despite its high accuracy, shows relatively low runtime and memory usage, indicating good efficiency. Although SVM performs well, its longer runtime and significant memory increase may limit its usability in resource-constrained environments. KNN has minimal memory changes and the fastest speed but suffers from poor performance. Considering both performance and efficiency, MLP strikes a balance between accuracy and resource usage, making it suitable for scenarios demanding both effectiveness and efficiency. On the other hand, SVM and XGBoost are better suited for applications requiring very high predictive accuracy and where computational resources are ample.

V. FUTURE WORK

To further advance this research, several promising directions can be explored.

First, integrating hybrid architectures that combine deep learning (e.g., CNNs or Vision Transformers) with traditional machine learning models could enhance feature extraction and classification accuracy, particularly for complex or rare tumor types.

Second, expanding the framework to incorporate multi-modal MRI data (e.g., T1-weighted, T2-weighted, FLAIR, and DWI sequences) may improve diagnostic robustness by leveraging complementary imaging information.

Third, developing adaptive preprocessing pipelines that automatically adjust to variations in MRI acquisition protocols could enhance model generalizability across different clinical settings. Additionally, exploring uncertainty quantification techniques (e.g., Bayesian deep learning or ensemble methods) would improve model reliability in real-world diagnostic scenarios.

Finally, deploying the system in clinical trials for real-time validation and iterative refinement based on radiologist feedback could ensure practical utility and regulatory compliance. These advancements would bridge the gap between research and clinical adoption, ultimately improving early and accurate brain tumor detection.

VI. CONCLUSION

In conclusion, this report presents an optimized machine learning framework for brain tumor detection in MRI images, addressing key challenges in medical imaging through advanced preprocessing, feature extraction, and model-specific enhancements. By integrating multiple datasets and employing techniques such as bicubic interpolation, data augmentation, and interpretable feature descriptors (HOG, GLCM), we achieved robust classification performance across SVM, XGBoost, MLP, Logistic Regression, and KNN models. Our results demonstrate that SVM and XGBoost excel in accuracy and recall, while MLP offers a strong balance between performance and computational efficiency. The framework's adaptability to limited and heterogeneous data makes it particularly suitable for clinical deployment in resource-constrained settings. Future research could explore hybrid deep learning approaches and multi-modal fusion to further improve diagnostic precision, as well as investigate real-time deployment on edge devices for point-of-care applications. This work highlights the potential of optimized traditional machine learning models to enhance early brain tumor detection, providing a reliable, interpretable, and efficient tool for computer-aided diagnosis.

REFERENCES

[1] M. Živanović, A. Aracki Trenkić, V. Milošević, D. Stojanov, M. Mišić, M. Radovanović, and V. Radovanović, "The role of magnetic resonance imaging in the diagnosis and prognosis of dementia," Biomolecules and Biomedicine, vol. 23, no. 2, pp. 209–224, Mar. 2023, doi: 10.17305/bjbms.2022.8085.

[2] J. R. Zech, M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann, "Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study," *PLoS Medicine*, vol. 15, no. 11, p. e1002683, Nov. 2018, doi: 10.1371/journal.pmed.1002683.

[3] J. Wang, N. I. R. Ruhaiyem, and P. Fu, "A Comprehensive Review of U-Net and Its Variants: Advances and Applications in Medical Image Segmentation," arXiv preprint arXiv:2502.06895, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2502.06895.

[4] K. Kumar, K. Jyoti, and K. Kumar, "Machine learning for brain tumor classification: evaluating feature extraction and algorithm efficiency," *Discover Artificial Intelligence*, vol. 4, p. 112, 2024, doi: 10.1007/s44163-024-00214-4.

[5] M. S. Ullah, M. A. Khan, H. M. Albarakati, R. Damaševičius, and S. Alsenan, "Multimodal brain tumor segmentation and classification from MRI scans based on optimized DeepLabV3+ and interpreted networks information fusion empowered with explainable AI," *Computers in Biology and Medicine*, vol. 182, p. 109183, 2024, doi: 10.1016/j.compbiomed.2024.109183.

[6] H. S. Dadi and G. K. M. Pillutla, "Improved Face Recognition Rate Using HOG Features and SVM Classifier," IOSR Journal of Electronics and Communication Engineering (IOSR-JECE), vol. 11, no. 4, pp. 34–44, Jul.-Aug. 2016, doi: 10.9790/2834-1104013444.

[7] D. Han, "Comparison of Commonly Used Image Interpolation Methods," Proc. 2nd Int. Conf. Comput. Sci. Electron. Eng. (ICCSEE), 2013, pp. 1553–1556, doi: 10.2991/iccsee.2013.391.

[8] R. M. Haralick, K. Shanmugam, and I. Dinstein, “Textural features for image classification,” IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-3, no. 6, pp. 610–621, Nov. 1973, doi: 10.1109/TSMC.1973.4309314.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, ”Learning representations by back-propagating errors,” Nature, vol. 323, no. 6088, pp. 533–536, Oct. 1986, doi: 10.1038/323533a0.

[10] P. Janson, V. Singh, P. M. Mehrbod, A. Ibrahim, I. Rish, E. Belilovsky, and B. Thérien, ”Beyond Cosine Decay: On the Effectiveness of Infinite Learning Rate Schedule for Continual Pre-training,” arXiv:2503.02844 [cs.LG], Mar. 2025. [Online]. Available: https://arxiv.org/abs/2503.02844.

[11] Z. Liu, ”Super Convergence Cosine Annealing with Warm-Up Learning Rate,” CAIBDA 2022; 2nd International Conference on Artificial Intelligence, Big Data and Algorithms, Nanjing, China, 2022, pp. 1-7.

[12] D. Misra, ”Mish: A Self Regularized Non-Monotonic Activation Function,” arXiv:1908.08681 [cs.LG], Aug. 2019, revised Aug. 2020. [Online]. Available: https://doi.org/10.48550/arXiv.1908.08681

Name	Student ID	Contribution
陈倩琦	2330034001	20%
黄思齐	2330034020	20%
李铭豪	2330034027	20%
梁意睿	2330034030	20%
涂一心	2330034048	20%