# Brain Tumor Segmentation using Computer Vision

## 1. Introduction

Brain tumor segmentation is a critical task in medical imaging that aids in the diagnosis and treatment planning for brain-related conditions. This project uses Convolutional Neural Networks (CNNs) to classify MRI brain scans into four categories: glioma, meningioma, pituitary, and no tumor. The system includes a Gradio-based user interface for practical deployment.

**Current Trends :**

**1. Deep Learning Dominance**

* Use of **Convolutional Neural Networks (CNNs)** for tumor classification from MRI/CT scans.
* Pretrained models like **VGG, ResNet, EfficientNet** offer high accuracy and transfer learning benefits.

**2. Data Augmentation & Preprocessing**

* Techniques like rotation, flipping, and normalization are commonly used to improve model generalization, especially when data is limited.

**3. Multi-Class and Multi-Label Classification**

* Models now distinguish between multiple tumor types (glioma, meningioma, pituitary, etc.) instead of binary (tumor vs. no tumor) classification.

**4. Use of Open Datasets**

* Widespread use of public datasets such as:
  + **BRATS (Brain Tumor Segmentation)**
  + **Figshare**
  + **Kaggle brain tumor datasets**

**5. Explainable AI (XAI)**

* Techniques like **Grad-CAM** and **LIME** are used to make model predictions more interpretable for clinical use.

**6. Integration with Web Interfaces**

* Real-time demo deployment using **Gradio**, **Streamlit**, or **Flask** allows non-experts to use trained models easily.

**Future Trends :**

**1. Transformer-Based Models**

* Adoption of **Vision Transformers (ViTs)** and **Swin Transformers** is increasing due to their ability to model global context better than CNNs.

**2. Multimodal Learning**

* Combining **MRI images + patient metadata + genomic data** for more accurate diagnosis using **fusion models**.

**3. Federated Learning for Privacy**

* Training models across multiple hospitals or regions **without sharing raw data**, ensuring **data privacy and security**.

**4. Self-Supervised Learning**

* Leveraging unlabelled medical images to **pretrain models without human annotations**, reducing dependence on costly labeled datasets.

**5. Real-Time Edge Inference**

* Deployment of lightweight CNN models on **edge devices or mobile platforms** for real-time brain scan analysis in rural/remote areas.

**6. Clinical Decision Support Systems (CDSS)**

* Deep learning models will be part of **AI-assisted diagnostic tools** used in hospitals to support radiologists.

**7. 3D Imaging and Segmentation**

* Shift from 2D slice-based models to **3D CNNs and 3D U-Nets** that process entire volumes for more spatially accurate classification.

**2. Problem Statement**

The project classifies brain MRI images into one of four categories — **glioma**, **meningioma**, **pituitary**, or **no tumor** — using a custom-built CNN model. It provides a real-time image classification interface using Gradio.

**Objective**

* Classify brain MRI images into multiple tumor types.
* Use a lightweight CNN architecture suitable for fast and accurate inference.
* Build a complete pipeline from image preprocessing to model deployment.
* Provide a user-friendly web interface using Gradio.

**3. Limitations of Existing Systems**

* Dependence on large labeled medical datasets for high performance.
* Performance degradation due to image noise, artifacts, or varying resolution.
* Often limited to binary classification (tumor vs. no tumor).
* Lack of model explainability (black-box decisions).
* Poor generalization across different imaging devices or hospitals.

**4. Proposed System**

* Develop a custom CNN architecture for multi-class tumor classification.
* Use OpenCV for image reading and resizing.
* Normalize and preprocess MRI images before feeding to the CNN.
* Split dataset for training and testing (80/20 split).
* Use Gradio to build a real-time classifier demo interface.

**5. Pipeline of the System**

**System Flow Chart:**

[Image Input] --> [Preprocessing] --> [CNN Model] --> [Prediction] --> [Gradio Interface Output]

**Module Flow Chart:**

1. Data Collection
2. Preprocessing
3. EDA
4. Training
5. Testing
6. Implementation
7. Evaluation
8. Deployment (Gradio)

**Data Collection**

* Dataset used: Brain MRI images (glioma, meningioma, pituitary, notumor) stored in folders.
* Images labeled by directory name.

**Preprocessing**

* Resize images to 128x128 pixels.
* Normalize pixel values to range [0, 1].
* Apply one-hot encoding to categorical labels.
* Train-test split using 80% for training and 20% for testing.

**EDA (Exploratory Data Analysis)**

* Visual inspection of sample MRI images per class.
* Class distribution analysis to detect imbalance.
* Image dimension and color range analysis.

**Training**

* CNN architecture with:
  + 3 convolutional layers with increasing filters (32, 64, 128)
  + MaxPooling and BatchNormalization layers
  + GlobalAveragePooling and Dense layers
* Optimizer: Adam
* Loss Function: Categorical Crossentropy
* Epochs: 10
* Batch Size: 16

**Testing**

* Evaluate trained model on held-out test set.
* Track accuracy and loss on test data.

**Implementation**

* Use TensorFlow/Keras to build and train the CNN.
* Use OpenCV for image handling.
* Create real-time interface using Gradio.

**Evaluation**

* Accuracy measured on training and test datasets.
* Cross-entropy loss used during training phase.

**Coding**

* Platform: Google Colab / Jupyter Notebook
* Language: Python
* Libraries: TensorFlow/Keras, NumPy, OpenCV, scikit-learn, Gradio

**6. Dashboard / User Interface**

* Built using **Gradio**
* Allows users to upload MRI images
* Displays predicted tumor class
* Can show top 1 or top-N prediction probabilities
* Clean, user-friendly interface for demos

**7. Libraries and Architecture**

**Libraries Used**

* **TensorFlow/Keras** – CNN model building and training
* **OpenCV** – Image reading and resizing
* **NumPy** – Image normalization and array handling
* **scikit-learn** – Label binarization, train-test split
* **Gradio** – Real-time web interface for predictions

**CNN Architecture Overview**

* Input: (128, 128, 3)
* Conv2D → MaxPooling → BatchNorm (x3 blocks)
* GlobalAveragePooling2D
* Dense(128) + Dropout(0.5)
* Output Dense(4) with softmax activation

**CNN Architecture Breakdown**

The CNN is built using the Sequential API in Keras and consists of several key layers that help the model learn patterns in MRI brain images.

**1. Input Layer**

* **Shape:** (128, 128, 3) → 128x128 pixel images with 3 color channels (RGB).
* No computation happens here; it's just the input format.

**2. Convolutional Layers (Conv2D)**

* These layers apply filters (kernels) to extract features such as edges, textures, and patterns.
* Filters are small matrices (e.g., 3x3) that slide over the image.
* **Activation:** ReLU (Rectified Linear Unit) is used to introduce non-linearity.

Example:

Conv2D(32, (3, 3), activation='relu', padding='same')

**3. MaxPooling2D Layers**

* Reduces spatial dimensions (height and width) while keeping the important features.
* Helps in reducing computation and overfitting.

Example:

MaxPooling2D(pool\_size=(2, 2))

**4. BatchNormalization Layers**

* Normalizes the output of the previous layer to speed up training and improve stability.
* Helps the model converge faster.

Example:

BatchNormalization()

**5. GlobalAveragePooling2D Layer**

* Replaces fully connected flattening by taking the average of each feature map.
* Reduces the number of parameters and overfitting.

Example:

GlobalAveragePooling2D()

**6. Dense (Fully Connected) Layer**

* The first dense layer has 128 neurons with ReLU activation.
* Learns complex features from the high-level features extracted by convolutional layers.

Example:

Dense(128, activation='relu')

**7. Dropout Layer**

* Randomly disables some neurons during training to prevent overfitting.
* **Rate:** 0.5 → 50% of neurons turned off during each iteration.

Example:

Dropout(0.5)

**8. Output Layer**

* A Dense layer with softmax activation for **multi-class classification** (4 classes).
* Outputs probabilities for each class (glioma, meningioma, pituitary, notumor).

Example:

Dense(4, activation='softmax')

**Summary Flow:**

Input → Conv2D → MaxPooling → BatchNorm →

Conv2D → MaxPooling → BatchNorm →

Conv2D → MaxPooling → BatchNorm →

GlobalAvgPool → Dense(128) → Dropout →

Dense(4, softmax)

Coding:

!pip install gradio

import gradio as gr

import os

import cv2

import numpy as np

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.layers import (Conv2D, MaxPooling2D, BatchNormalization, Dropout, Dense, GlobalAveragePooling2D)

from tensorflow.keras.optimizers import Adam

# 📁 STEP 2: Mount Google Drive (if dataset is there)

from google.colab import drive

drive.mount('/content/drive')

IMG\_SIZE = 128

BATCH\_SIZE = 16

EPOCHS = 10

DATASET\_PATH = '/content/drive/MyDrive/Training'

CATEGORIES = ['glioma', 'meningioma', 'notumor', 'pituitary']

data = []

labels = []

# ✅ Load and resize images

for category in CATEGORIES:

    folder\_path = os.path.join(DATASET\_PATH, category)

    print(f"Loading {category}...")

    for img in os.listdir(folder\_path):

        img\_path = os.path.join(folder\_path, img)

        try:

            image = cv2.imread(img\_path)

            image = cv2.resize(image, (IMG\_SIZE, IMG\_SIZE))

            data.append(image)

            labels.append(category)

        except Exception as e:

            pass

# Preprocessing

data = np.array(data) / 255.0

from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()

labels = lb.fit\_transform(labels)

# Train-test split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.2, random\_state=42)

# Model

model = Sequential([

    Conv2D(32, (3, 3), activation='relu', padding='same', input\_shape=(IMG\_SIZE, IMG\_SIZE, 3)),

    MaxPooling2D((2, 2)),

    BatchNormalization(),

    Conv2D(64, (3, 3), activation='relu', padding='same'),

    MaxPooling2D((2, 2)),

    BatchNormalization(),

    Conv2D(128, (3, 3), activation='relu', padding='same'),

    MaxPooling2D((2, 2)),

    BatchNormalization(),

    GlobalAveragePooling2D(),

    Dense(128, activation='relu'),

    Dropout(0.5),

    Dense(len(CATEGORIES), activation='softmax')

])

model.compile(loss='categorical\_crossentropy', optimizer=Adam(), metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, batch\_size=BATCH\_SIZE, validation\_data=(X\_test, y\_test), epochs=EPOCHS, verbose=1)

# Function to make predictions

def predict\_image(image):

    image = cv2.cvtColor(np.array(image), cv2.COLOR\_RGB2BGR)

    image = cv2.resize(image, (IMG\_SIZE, IMG\_SIZE))

    image = np.expand\_dims(image, axis=0) / 255.0

    prediction = model.predict(image)

    label = lb.inverse\_transform(prediction)

    return label[0]

# Create Gradio interface

interface = gr.Interface(fn=predict\_image,

                         inputs=gr.Image(type='pil'),

                         outputs=gr.Label(num\_top\_classes=4),

                         title="Brain Tumor Classifier",

                         description="Upload an image to classify a brain tumor as glioma, meningioma, notumor, or pituitary.")

interface.launch()

## 8.User Interface

Gradio is used to create a simple web interface that:  
- Accepts an image upload  
- Runs prediction and shows the tumor type  
- Displays title and description

## 9.Future Enhancements

- Implement data augmentation  
- Use pre-trained models like ResNet or EfficientNet  
- Integrate Grad-CAM for model explainability  
- Add pixel-wise segmentation capabilities

## 10.Conclusion

This project shows how CNNs can be effectively applied to brain tumor classification. With further enhancements, this system can be evolved into a full-fledged diagnostic tool for real-time tumor analysis.