**Introduction :**

A brain tumor is when unusual cells grow in or around the brain. These tumors can be either non-cancerous (benign) or cancerous (malignant), and both types can be dangerous depending on their size and where they are in the brain.Doctors usually use MRI scans to find and check brain tumors. But looking at these scans by hand takes a lot of time and can sometimes lead to mistakes because people may see things differently.Using computer-based methods to detect and highlight tumors in images can help find them more quickly and accurately. This can lead to better treatment, fewer mistakes, and help doctors make better decisions.

**What is CNN /LLM ?**

**CNN :**CNN (Convolutional Neural Network) is a deep learning algorithm specifically designed for visual data. It is highly effective in tasks like image classification, object detection, and segmentation.

**LLM :**LLM (Large Language Model) refers to AI models like ChatGPT that work with text. While LLMs are revolutionizing natural language processing, they are not typically used for image segmentation. Hence, CNNs are the primary choice for this medical imaging task.

**How CNN is Used in Brain TumorSegmentation :**

In this project, a U-Net-based CNN architecture is used for precise segmentation of tumor regions from MRI scans. The steps include:

1 .**Preprocessing** MRI images and corresponding binary masks.

1. **Training** a U-Net model using the Kaggle-sourced Figshare Brain Tumor Dataset.
2. **Segmenting** tumor regions by predicting pixel-level masks.
3. **Evaluating** the model using metrics like Dice Coefficient and IoU.
4. **Deploying** the model through a Streamlit or Gradio app to enable real-time predictions on user-uploaded images.

This deep learning approach offers high accuracy in identifying tumor boundaries and supports clinical workflows through automation.

**Current Scenario (Simplified)**

Using CNNs (Convolutional Neural Networks) for medical image analysis is growing fast. Hospitals and research centers are using deep learning to help doctors and make diagnoses more reliable. For brain tumors, models like U-Net are commonly used in research and hospitals.Public datasets, like the Figshare Brain Tumor Dataset, help speed up the development of open-source models. Tools such as Gradio and Streamlit allow doctors to test models easily with real cases.This project is part of these efforts by using CNNs and creating a simple app for users to test tumor detection models**.**

**PROBLEM STATEMENT:**

**MainObjective**

The primary objective of this project is to develop an automated brain tumor segmentation system using Convolutional Neural Networks (CNNs), specifically a U-Net architecture, to accurately identify and segment tumor regions from MRI scans. This model is trained on the publicly available Figshare Brain Tumor Dataset and deployed using a user-friendly interface (Gradio/Streamlit) that allows medical professionals or users to upload MRI images and receive real-time tumor segmentation results.

By automating this process, the goal is to:

1. Reduce human error in tumor identification,
2. Support radiologists with fast and accurate results,
3. Enable consistent and reproducible diagnostic workflows,
4. And ultimately improve early detection and treatment planning

**Existing System**

In the existing medical diagnostic workflow:

* MRI images are manually reviewed by radiologists.
* Tumor regions are segmented by hand or with basic software.
* Diagnosis often depends on the expertise and attention of individual radiologists.

**Limitations of the Existing System:**

* **Time-consuming:** Manual segmentation takes significant time per patient.
* **Inconsistency:** Different radiologists may interpret the same scan differently.
* **Limited accuracy:** Early or small tumors may go undetected due to visual fatigue or subtle patterns.
* **Lack of automation:** Most hospital systems still lack AI-integrated workflows for real-time segmentation

**Proposed System:**

The proposed system leverages a deep learning-based U-Net model to perform precise brain tumor segmentation. It introduces:

* Fully automated segmentation of tumor regions from MRI images.
* Use of a labeled dataset (Figshare Brain Tumor Dataset) for robust training.
* Preprocessing pipeline to normalize, resize, and enhance MRI images.
* Evaluation metrics such as Dice Score and IoU to ensure model quality.
* User interface deployment via Gradio/Streamlit to make it accessible for medical or research use.

**Advantages of the Proposed System:**

* High accuracy through deep learning.
* Fast inference allows near real-time results.
* Consistency in tumor detection regardless of user.
* Scalable and adaptable for real-world medical systems.
* Interactive deployment that makes it usable even for non-technical users.

**DATA/PROJECT PIPELINE:**

The brain tumor segmentation system follows a structured pipeline that ensures efficient data handling, model training, testing, and deployment. Below are the key stages:

1. **Data Collection :**

Source: Kaggle –Figshare Brain Tumor Dataset

Content: MRI brain images with corresponding masks (tumor = white pixels, non-tumor = black pixels)

Classes: Glioma, Meningioma, Pituitary Tumor, No Tumor

1. **Data Preprocessing :**

Normalization of pixel values to [0,1]

Image resizing (e.g., 256x256 pixels)

Splitting into training, validation, and test sets

Augmentation (e.g., flipping, rotation) to increase model robustness

1. **Model Training :**

Model: U-Net CNN architecture

Input: Preprocessed MRI images

Output: Pixel-wise tumor segmentation mask

Loss Function: Dice Loss / Binary Cross-Entropy

Optimizer: Adam

1. **Model Testing &Evaluation :**

Metrics: Dice Coefficient, Intersection over Union (IoU), Precision, Recall

Use of unseen test data to measure generalization

Visual comparison between predicted and actual segmentation

1. **Error Detection :**

Identify misclassified or poorly segmented samples

Analyze failure cases using confusion matrices and visual inspection

Retraining or fine-tuning model if necessary

1. **Deployment :**

Build an interactive user interface using Gradio or Streamlit

Allow real-time MRI image upload and instant tumor segmentation

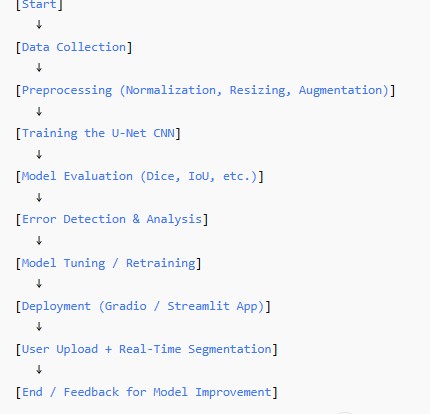
Display results and download options for further clinical review

1. **Feedback Loop :**

Use real-world predictions to improve and retrain the model over time

Enable model updates with new data for continuous learning

**Flow Chart :**



**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

IMG\_SIZE = 128

BATCH\_SIZE = 16

EPOCHS = 10

DATASET\_PATH = '/content/drive/MyDrive/Brain Tumor Segmentation/Training'

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

data = []

labels = []

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

data = np.array(data) / 255.0

from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()

labels = lb.fit\_transform(labels)

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 = 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'])

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

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]

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()

**LibrariesUsed:**

**1.TensorFlowKeras** – For building and training the CNN model.

**Modules**: Sequential, Conv2D, MaxPooling2D, Dense, etc.

**2.Scikit-learn** – For encoding labels and splitting data.

**Modules**: LabelBinarizer, train\_test\_split.

**3.OpenCV** (**cv2**) – For reading and resizing images.

**4.NumPy** – For array handling and normalization.

**5.Gradio** – For creating a web interface for model predictions.

**6.GoogleColabDrive** – To access dataset stored in Google Drive.

**7.OS** – To handle file paths and directories.

**Architecture Used:**

The model is a **Convolutional Neural Network (CNN)** built using the Keras Sequential API. Below are the layers used:

1. **Input Layer:**

Accepts images of size **128×128×3** (RGB images).

**2. Convolutional + Pooling Blocks:**

* **1st Block**:
  + Conv2D: 32 filters, 3×3 kernel, ReLU activation.
  + MaxPooling2D: 2×2 pool size.
  + BatchNormalization: Stabilizes learning.
* **2nd Block**:
  + Conv2D: 64 filters, 3×3 kernel, ReLU activation.
  + MaxPooling2D: 2×2 pool size.
  + BatchNormalization
* **3rd Block**:
  + Conv2D: 128 filters, 3×3 kernel, ReLU activation.
  + MaxPooling2D: 2×2 pool size.
  + BatchNormalization

**3.Global Average Pooling:**

GlobalAveragePooling2D: Reduces each feature map to a single number.

**4. Fully Connected Layers:**

* Dense: 128 neurons, ReLU activation.
* Dropout: 0.5 dropout rate to prevent overfitting.
* Dense: Output layer with 4 neurons (one for each tumor class), **softmax** activation.

**LIMITATIONS:**

1. **Small Dataset**
   * The model may not work well on new images if it’s trained on too few examples.
2. **Uneven Image Counts**
   * If some tumor types have more images than others, the model might be biased toward those types.
3. **No Image Changes Used**
   * The model sees only the original images. Adding changes like flips or rotations could help it learn better.
4. **Simple Model Used**
   * The CNN model is basic. More advanced models like ResNet could give better results.
5. **No Pre-Trained Help**
   * The model is trained from scratch. Using pre-trained models could improve accuracy, especially with small datasets.
6. **Basic Testing Only**
   * It uses a simple train-test split. More advanced testing (like cross-validation) could give more reliable results.
7. **No Error Checks**
   * The model may not handle bad or incorrect images properly.
8. **Grayscale Images Not Supported**
   * It only works with color images (RGB). Black-and-white images might cause errors.

## FUTURE ENHANCEMENT:

1. **Use Pre-Trained Models**
   * Use models like ResNet or VGG to get better results with less data.
2. **Add Image Variations**
   * Use flips, rotations, and zooms to help the model learn better.
3. **Make a Web or Mobile App**
   * Turn the project into an app for real-world use.
4. **Show Tumor Location**
   * Add features to find and highlight the tumor area.
5. **Support Hospital Image Formats**
   * Allow DICOM images used by doctors.
6. **Add Confidence and Visual Explanations**
   * Show how sure the model is and what part of the image it focused on.
7. **Improve Interface**
   * Make the app easier for doctors to use.
8. **Add Better Evaluation**
   * Use more metrics like precision and recall to check performance.
9. **Use Cross-Validation**
   * Split the data in smarter ways to test the model better.
10. **Automate the Whole Process**

* Build a full system that handles everything from training to prediction.

**CONCLUSION:**

This project shows how deep learning can help doctors detect brain tumors by analyzing MRI images. The model can classify four types of brain conditions and is easy to use with a simple interface. While it works well, it can be improved in many ways to make it more useful in real-life situations, like hospitals or diagnostic apps.