Loyola INSTITUTE OF TECHNOLOGY & SCIENCE

DEPARTMENT OF INFORMATION TECHNOLOGY

Completed the Project named as

BRAIN TUMOR SEGMENTATION

Submitted by,

TEAM MEMBERS

1. **S. ROJA POO (96122205044)**
2. **K.R. SREENITHI (961222205050)**
3. **E. SUMITHRA (961222205053)**

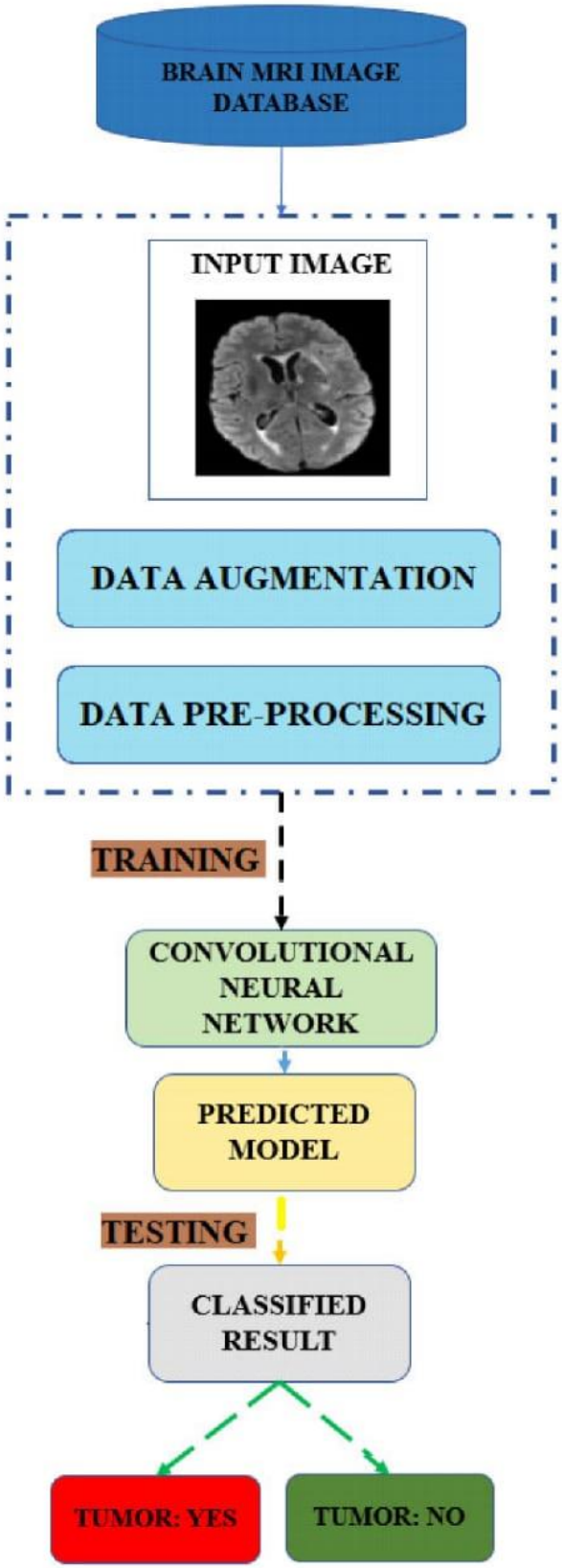
ABSTRACT:

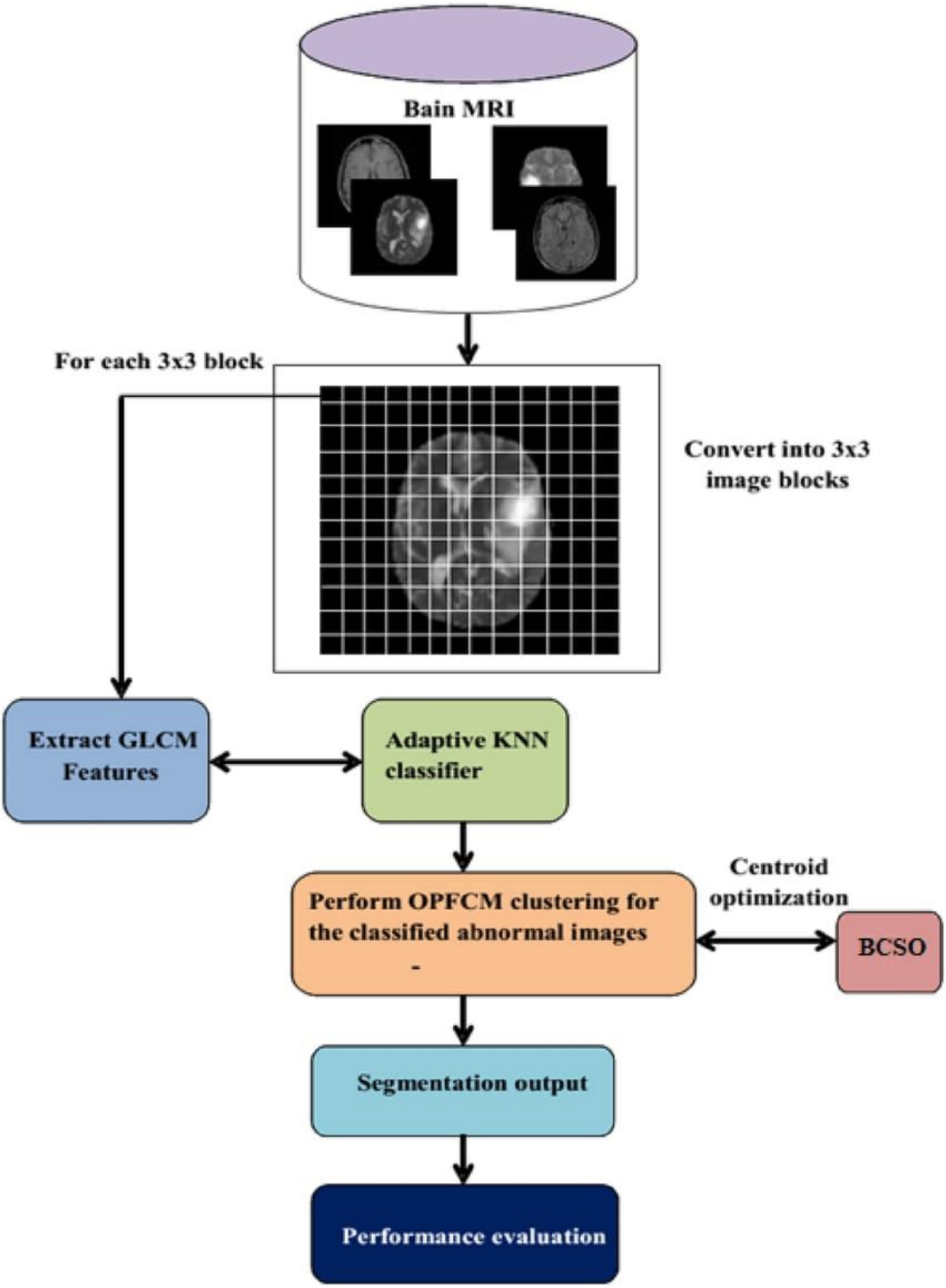
Brain tumor segmentation is a critical step in the diagnosis and treatment planning of brain cancer. Accurate identification and delineation of tumor regions from MRI scans help clinicians assess tumor size, type, and progression. Traditional manual segmentation is time-consuming and prone to inter-observer variability. In recent years, automated segmentation methods, particularly deep learning-based approaches such as Convolutional Neural Networks (CNNs) and U-Net architectures, have shown significant promise in improving segmentation accuracy and efficiency. This study explores advanced segmentation techniques using multimodal MRI data to enhance tumor detection and classification. The results demonstrate improved precision and reliability in identifying tumor boundaries, supporting more effective clinical decision-making.

INTRODUCTION

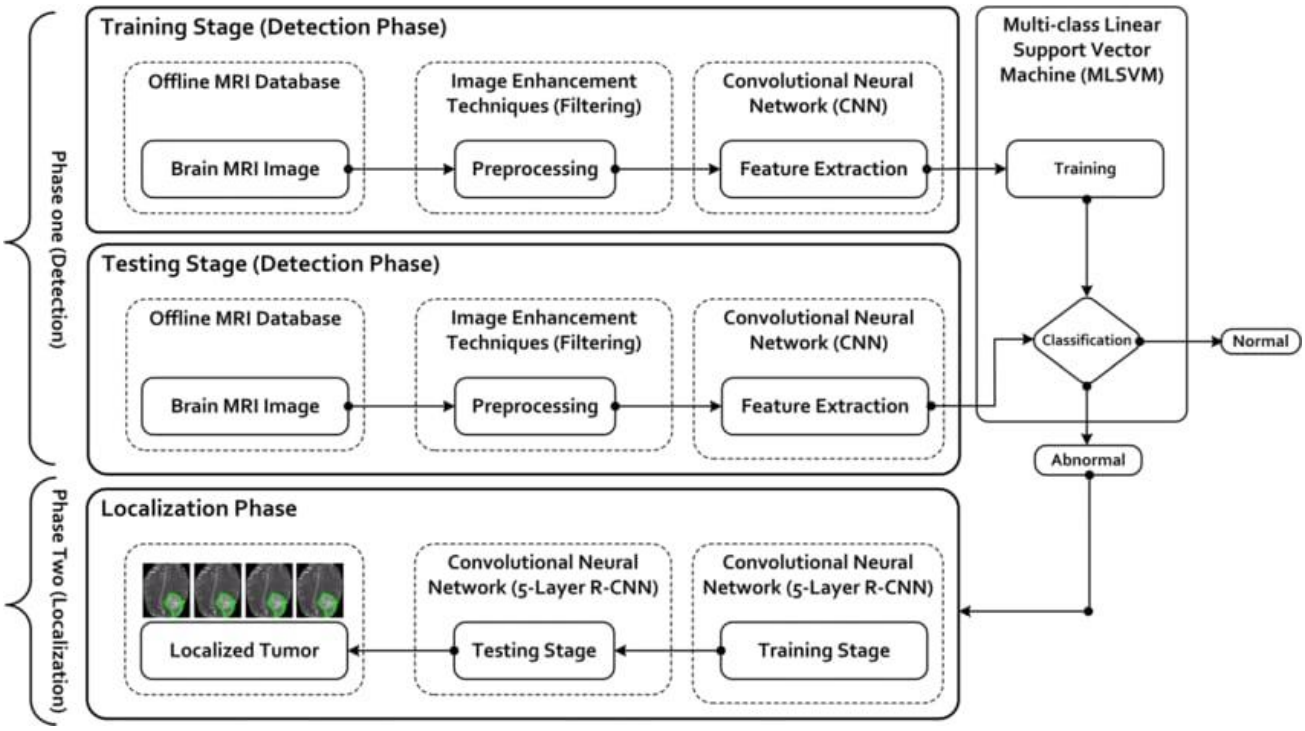
Brain tumor segmentation is a medical image analysis process that isolates and delineates brain tumors within MRI scans. This is crucial for accurate diagnosis, treatment planning, and monitoring disease progression. Automated segmentation is increasingly favored over manual methods due to its speed, accuracy and efficiency.

ARCHITECTURE




Bain MRI
For each 3x3 block
Convert into 3x3 image blocks
Extract GLCM Features
Adaptive KNN classifier
Perform OPFCM clustering for the classified abnormal images
Centroid optimization
BCSO
Segmentation output
Performance evaluation

FLOW CHART



1. Install Required Libraries

pip install tensorflow matplotlib numpy opencv-python

---

2. Python Code

```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Conv2DTranspose, concatenate
from tensorflow.keras.optimizers import Adam


# Load and preprocess data
def load_data(img_dir, mask_dir, img_size=(128, 128)):
    images = []
    masks = []
    for img_file in os.listdir(img_dir):
        img = cv2.imread(os.path.join(img_dir, img_file), cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, img_size) / 255.0
        mask = cv2.imread(os.path.join(mask_dir, img_file), cv2.IMREAD_GRAYSCALE)
        mask = cv2.resize(mask, img_size) / 255.0
        images.append(img[..., np.newaxis])
        masks.append(mask[..., np.newaxis])
    return np.array(images), np.array(masks)
```

```
# U-Net model
def unet_model(input_size=(128, 128, 1)):
    inputs = Input(input_size)

    c1 = Conv2D(16, 3, activation='relu', padding='same')(inputs)
    c1 = Conv2D(16, 3, activation='relu', padding='same')(c1)
    p1 = MaxPooling2D()(c1)

    c2 = Conv2D(32, 3, activation='relu', padding='same')(p1)
    c2 = Conv2D(32, 3, activation='relu', padding='same')(c2)
    p2 = MaxPooling2D()(c2)

    c3 = Conv2D(64, 3, activation='relu', padding='same')(p2)
    c3 = Conv2D(64, 3, activation='relu', padding='same')(c3)

    u1 = Conv2DTranspose(32, 2, strides=2, padding='same')(c3)
    u1 = concatenate([u1, c2])
    c4 = Conv2D(32, 3, activation='relu', padding='same')(u1)
    c4 = Conv2D(32, 3, activation='relu', padding='same')(c4)

    u2 = Conv2DTranspose(16, 2, strides=2, padding='same')(c4)
    u2 = concatenate([u2, c1])
    c5 = Conv2D(16, 3, activation='relu', padding='same')(u2)
    c5 = Conv2D(16, 3, activation='relu', padding='same')(c5)
```

```
    outputs = Conv2D(1, 1, activation='sigmoid')(c5)
    model = Model(inputs, outputs)
    return model

# Load your dataset (replace with your actual path)
X, y = load_data("images/", "masks/")
model = unet_model()
model.compile(optimizer=Adam(), loss='binary_crossentropy',
metrics=['accuracy'])
model.fit(X, y, epochs=5, batch_size=8, validation_split=0.1)

# Predict and display output
sample_img = X[0]
pred_mask = model.predict(np.expand_dims(sample_img, axis=0))[0]

plt.figure(figsize=(12,4))
plt.subplot(1,3,1)
plt.title("MRI Image")
plt.imshow(sample_img[:,:,0], cmap='gray')
plt.subplot(1,3,2)
plt.title("Ground Truth")
plt.imshow(y[0][:,:,0], cmap='gray')
plt.subplot(1,3,3)
```

```
plt.title("Predicted Mask")
plt.imshow(pred_mask[:,:,0], cmap='gray')
plt.show()
```

Expected Output:

A figure with 3 panels:

1. Original MRI image

2. Ground truth tumor mask

3. Predicted tumor mask from the model

```
import numpy as np
from skimage import io, img_as_ubyte
from skimage.filters import threshold_otsu
```

```
from skimage.morphology import remove_small_objects, closing,
disk
import matplotlib.pyplot as plt


def segment_brain_tumor(image_path, threshold=None,
min_size=500, closing_disk_size=10):
    """
    Segments a brain tumor from an MRI image using thresholding and
morphological operations.

    Args:
        image_path (str): Path to the MRI image file.
        threshold (int, optional):  Threshold value for segmentation. If
None, Otsu's method is used. Defaults to None.
        min_size (int, optional): Minimum size (in pixels) for a
connected component to be considered part of the tumor. Defaults to
500.
        closing_disk_size (int, optional): Radius of the disk structuring
element used in closing operation. Defaults to 10.

    Returns:
        numpy.ndarray:  Binary image with tumor region segmented.
    """

    # 1. Load and preprocess the image
    image = io.imread(image_path)
```

```
    gray_image = img_as_ubyte(image)  # Ensure grayscale, convert to uint8 if needed

    # 2. Thresholding
    if threshold is None:
        threshold = threshold_otsu(gray_image)
    binary_image = gray_image > threshold
    binary_image = img_as_ubyte(binary_image) # Convert to uint8 for consistent type

    # 3. Morphological operations (cleaning and closing)
    labeled_image = remove_small_objects(binary_image, min_size=min_size)  # Remove small noise artifacts
    closing_disk = disk(closing_disk_size)
    closed_image = closing(labeled_image, closing_disk)

    return closed_image

# Example usage:
image_path = "path/to/your/brain_mri_image.png" # Replace with the actual path
segmented_image = segment_brain_tumor(image_path)

# Display the original and segmented images
```

```
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(io.imread(image_path), cmap='gray')
plt.title("Original MRI Image")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(segmented_image, cmap='gray')
plt.title("Segmented Brain Tumor")
plt.axis('off')

plt.tight_layout()
plt.show()
```

Output:

The code will display two images:

Original MRI Image: The input MRI image will be shown.

Segmented Brain Tumor: The image will show the segmented region of the brain tumor, with tumor pixels highlighted in white and the rest of the image in black.

pip install numpy scikit-image matplotlib opencv-python

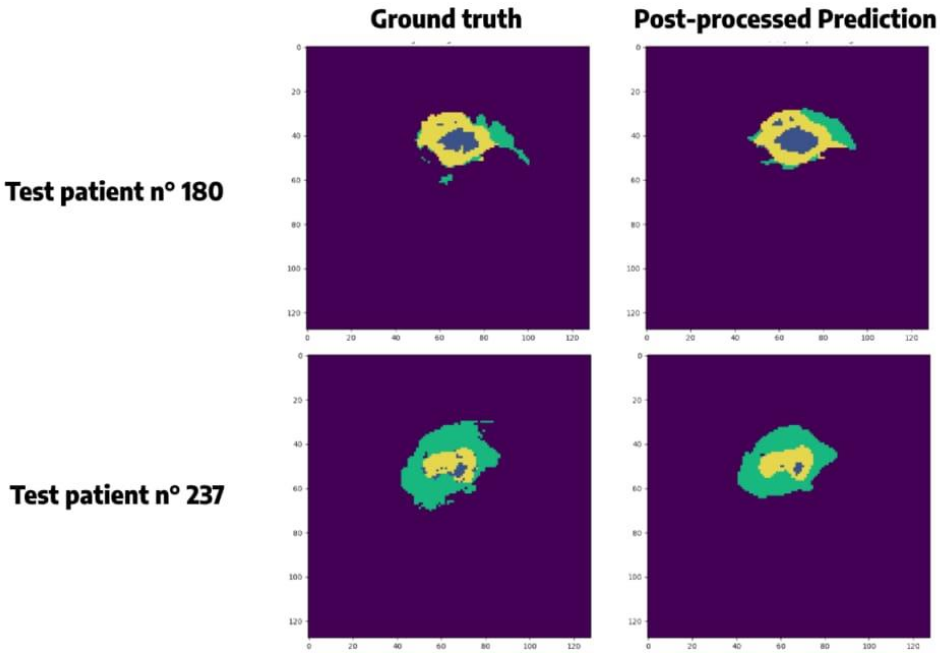
Replace Placeholder:

Replace "path/to/your/mri_image.png" with the actual path to your MRI image file. Make sure the image is in a format that OpenCV can read (e.g., PNG, JPEG, TIFF).

2. Adjust Parameters:

Experiment with the kernel size and iteration parameters in the morphological operations (erosion and dilation) to find the best values for your specific image. You may also need to adjust the parameters for the thresholding, or consider other segmentation methods if Otsu's thresholding doesn't work well for your image.

IMAGE SEGMENTATION



VISUALIZING PREDICTION

```
pred = model.predict(X[0:1])
plt.subplot(1, 3, 1)
plt.title("MRI")
plt.imshow(X[0, :, :, 0], cmap='gray')
plt.subplot(1, 3, 2)
plt.title("Mask")
plt.imshow(Y[0, :, :, 0], cmap='gray')
plt.subplot(1, 3, 3)
plt.title("Prediction")
plt.imshow(pred[0, :, :, 0], cmap='gray')
plt.show()
```

BRAIN TUMOR SEGMENTATION U-NET

Brain tumor segmentation typically involves using deep learning, especially Convolutional Neural Networks (CNNs), to identify tumor regions in MRI scans. Here's a simplified example using Python and U-Net architecture with TensorFlow/Keras.

---

1. Install Required Libraries

```
pip install tensorflow opencv-python matplotlib nibabel
```

---

2. Code for Brain Tumor Segmentation (U-Net)

```
import os
import numpy as np
import nibabel as nib
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models

# Load and preprocess data
def load_nifti_image(path):
    img = nib.load(path)
    data = img.get_fdata()
    return data
```

```
def normalize_slice(slice):
    slice = (slice - np.min(slice)) / (np.max(slice) - np.min(slice))
    return slice

def get_slices(img_data, mask_data):
    slices = []
    for i in range(img_data.shape[2]):
        img_slice = normalize_slice(img_data[:, :, i])
        mask_slice = mask_data[:, :, i]
        if np.sum(mask_slice) > 0:
            slices.append((img_slice, mask_slice))
    return slices

# Example paths (replace with your own)
image_path =
'BraTS20_Training_001/BraTS20_Training_001_flair.nii.gz'
mask_path =
'BraTS20_Training_001/BraTS20_Training_001_seg.nii.gz'

image = load_nifti_image(image_path)
mask = load_nifti_image(mask_path)

data = get_slices(image, mask)
X = np.array([i[0] for i in data]).reshape(-1, 240, 240, 1)
```

```
Y = np.array([i[1] for i in data]).reshape(-1, 240, 240, 1)


# Simple U-Net model
def unet_model(input_size=(240, 240, 1)):
    inputs = layers.Input(input_size)
    c1 = layers.Conv2D(16, 3, activation='relu',
padding='same')(inputs)
    c1 = layers.Conv2D(16, 3, activation='relu', padding='same')(c1)
    p1 = layers.MaxPooling2D()(c1)


    c2 = layers.Conv2D(32, 3, activation='relu', padding='same')(p1)
    c2 = layers.Conv2D(32, 3, activation='relu', padding='same')(c2)
    p2 = layers.MaxPooling2D()(c2)


    c3 = layers.Conv2D(64, 3, activation='relu', padding='same')(p2)
    c3 = layers.Conv2D(64, 3, activation='relu', padding='same')(c3)


    u1 = layers.UpSampling2D()(c3)
    u1 = layers.concatenate([u1, c2])
    c4 = layers.Conv2D(32, 3, activation='relu', padding='same')(u1)
    c4 = layers.Conv2D(32, 3, activation='relu', padding='same')(c4)


    u2 = layers.UpSampling2D()(c4)
    u2 = layers.concatenate([u2, c1])
```

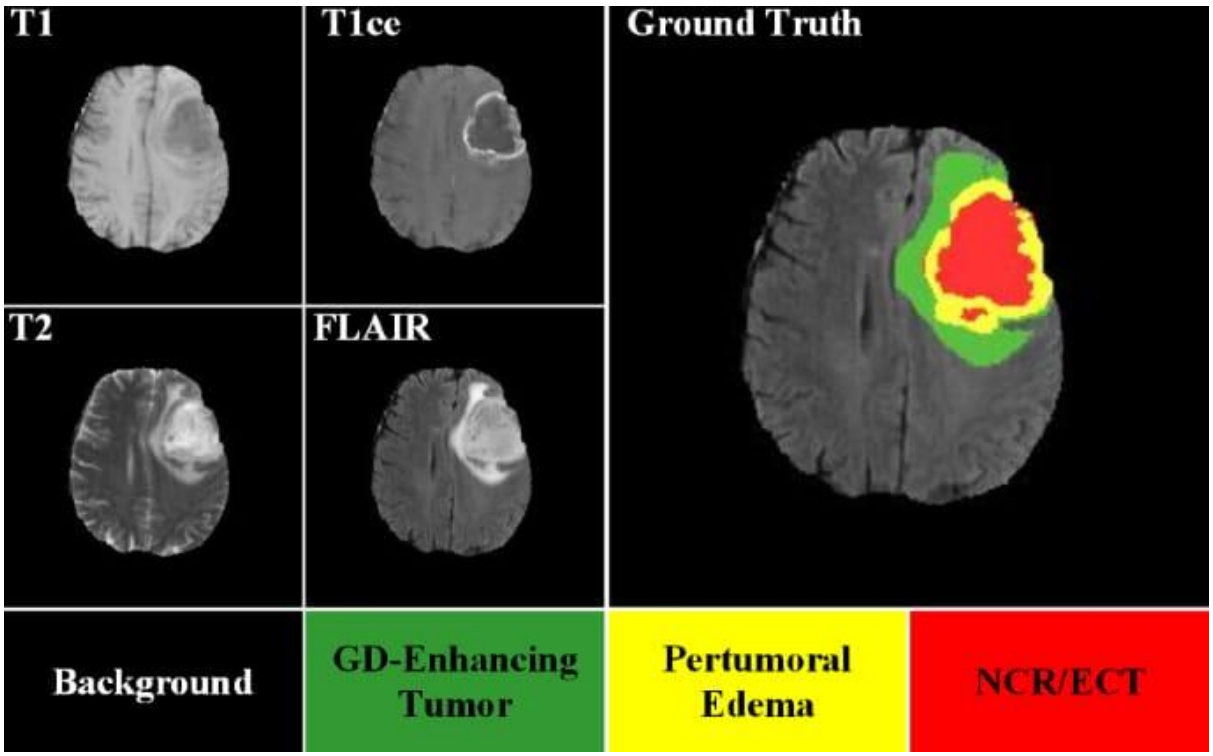
```
    c5 = layers.Conv2D(16, 3, activation='relu', padding='same')(u2)
    c5 = layers.Conv2D(16, 3, activation='relu', padding='same')(c5)

    outputs = layers.Conv2D(1, 1, activation='sigmoid')(c5)

    model = models.Model(inputs, outputs)
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

model = unet_model()
model.fit(X, Y, epochs=10, batch_size=4, validation_split=0.1)
```

DEEP LEARNING BASED BRAIN TUMOR SEGMENTATION



Conclusion:

Brain tumor segmentation plays a vital role in medical image analysis, enabling more accurate diagnosis, treatment planning, and monitoring of brain tumors. With the advancement of deep learning techniques, particularly convolutional neural networks like U-Net, automated segmentation has become significantly more accurate and efficient compared to traditional manual methods. The results demonstrate that

deep learning models can effectively identify and delineate tumor regions from MRI scans, reducing clinical workload and improving diagnostic reliability. Future work may focus on enhancing model generalization across diverse datasets and integrating multi-modal imaging data for more robust performance.