Experiment 7

Title: To develop a brain tumor classification system from MRI images using deep learning techniques.

Objective: The objective of this experiment is to design and implement a deep learning-based image classification system capable of identifying and classifying brain tumors from MRI images. The system will classify the MRI images into two categories: Tumor and No Tumor, thus facilitating early detection of brain tumors for medical diagnosis and treatment planning.

Theory:
Medical image classification plays a vital role in healthcare, particularly in the detection and diagnosis of diseases. One of the most important applications of image classification in healthcare is in the detection of brain tumors from MRI scans. MRI scans are non-invasive imaging techniques that provide detailed images of the brain's internal structures, and detecting tumors in these images is crucial for early diagnosis and treatment.

In recent years, deep learning techniques, especially Convolutional Neural Networks (CNNs), have become the state-of-the-art approach for image classification tasks. These methods automatically learn features from the data without requiring manual feature extraction, making them ideal for complex medical image classification tasks such as tumor detection.

What is Brain Tumor Classification?

Brain tumor classification is the task of identifying and categorizing MRI images into two distinct classes:

1. Tumor: Images that contain abnormal growths or tumors in the brain.
2. No Tumor: Images that do not show any abnormal growth.

This task is critical in the medical field as it assists radiologists in identifying brain tumors early, which can lead to faster treatment and improved patient outcomes. Early detection can also significantly increase the success rate of treatment, making the role of accurate classification highly important.

Deep Learning Approaches for Tumor Detection:

To address the challenges of brain tumor classification, deep learning techniques, especially Convolutional Neural Networks (CNNs), are widely used. CNNs are well-suited for image-based tasks due to their ability to learn hierarchical features from raw images.

1. **Convolutional Neural Networks (CNNs):**

- CNNs consist of multiple layers such as convolutional layers, pooling layers, and fully connected layers, which enable them to learn increasingly abstract features from images.
- Convolutional layers detect local patterns such as edges and textures, while deeper layers capture more complex structures like objects or tumors.

2. **Pre-trained Models**:
   - Using pre-trained models like VGG16, ResNet, and Inception can significantly improve classification performance, as they have already been trained on large image datasets and learned useful features.
3. **Transfer Learning**:
   - **Transfer learning** allows the use of pre-trained models and fine-tuning them on specific datasets (e.g., brain tumor images). This approach reduces the need for a large amount of training data and speeds up model development.
4. **Data Augmentation**:
   - Data augmentation techniques, such as rotating, flipping, and zooming, are used to artificially expand the dataset and increase model generalization, especially in the case of limited data.

Workflow of Brain Tumor Classification System:

1. **Data Collection and Preparation:**
   - A dataset of brain MRI images is collected, typically from public datasets such as the Brain Tumor Segmentation (BraTS) dataset.
   - Images are preprocessed by resizing, normalization, and augmentation to improve the model's ability to generalize.
2. **Model Selection:**
   - A suitable deep learning model is selected, often starting with a simple CNN architecture or utilizing pre-trained models such as VGG16, ResNet, or EfficientNet.
3. **Model Training:**
   - The model is trained on labeled MRI images (Tumor vs. No Tumor). Techniques like cross-entropy loss and Adam optimizer are commonly used for training the model.
4. **Model Evaluation:**
   - After training, the model is evaluated using validation data. Performance metrics such as accuracy, precision, recall, and F1-score are computed to assess the model's ability to classify the images correctly.
5. **Testing on New Data:**
   - The trained model is tested on new MRI images that the model has not seen before. The model predicts whether the image contains a tumor or not.
6. **Model Improvement (if necessary):**

- If performance is not satisfactory, additional techniques like fine-tuning the model, adding more layers, or adjusting the hyperparameters (e.g., learning rate) are employed.

Software Requirements and Tools:

Software / Library	**Version / Notes**	**Purpose**
Python	3.8 or above	Programming language
TensorFlow	Latest	Deep learning frameworks for building models
Numpy	Latest	Numerical operations
Matplotlib	Optional	Visualization of results and training curves
OpenCV	Latest	Image processing and visualization
scikit-learn	Latest	Metrics evaluation and performance analysis
Keras	Latest	High-level neural networks API

Experiment Procedure:

Step 1: Import Libraries:

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
Dropout, Flatten, Dense
```

Step 2: Load and preprocess dataset:

```
yes_path = "train/yes"
no_path = "train/no"

yes_files = os.listdir(yes_path)
no_files = os.listdir(no_path)

X_data, y_data = [], []
```

```
# YES (tumor) images
for file in yes_files:
    img = cv2.imread(os.path.join(yes_path, file))
    if img is None:
        continue
    img = cv2.resize(img, (32, 32))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    X_data.append(img)
    y_data.append(1)

# NO (no tumor) images
for file in no_files:
    img = cv2.imread(os.path.join(no_path, file))
    if img is None:
        continue
    img = cv2.resize(img, (32, 32))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    X_data.append(img)
    y_data.append(0)

# Convert to numpy arrays
X = np.array(X_data, dtype="float32") / 255.0
y = np.array(y_data)
```

Step 3: Build the CNN model and Train the Model:

```
model = Sequential([
     Conv2D(16, kernel_size=9, padding='same', activation='relu',
input_shape=(32,32,3)),
    MaxPooling2D(pool_size=2),
    Dropout(0.45),

                Conv2D(16,    kernel_size=9,    padding='same',
activation='relu'),
    MaxPooling2D(pool_size=2),
    Dropout(0.25),

                Conv2D(36,    kernel_size=9,    padding='same',
activation='relu'),
    MaxPooling2D(pool_size=2),
    Dropout(0.25),
```

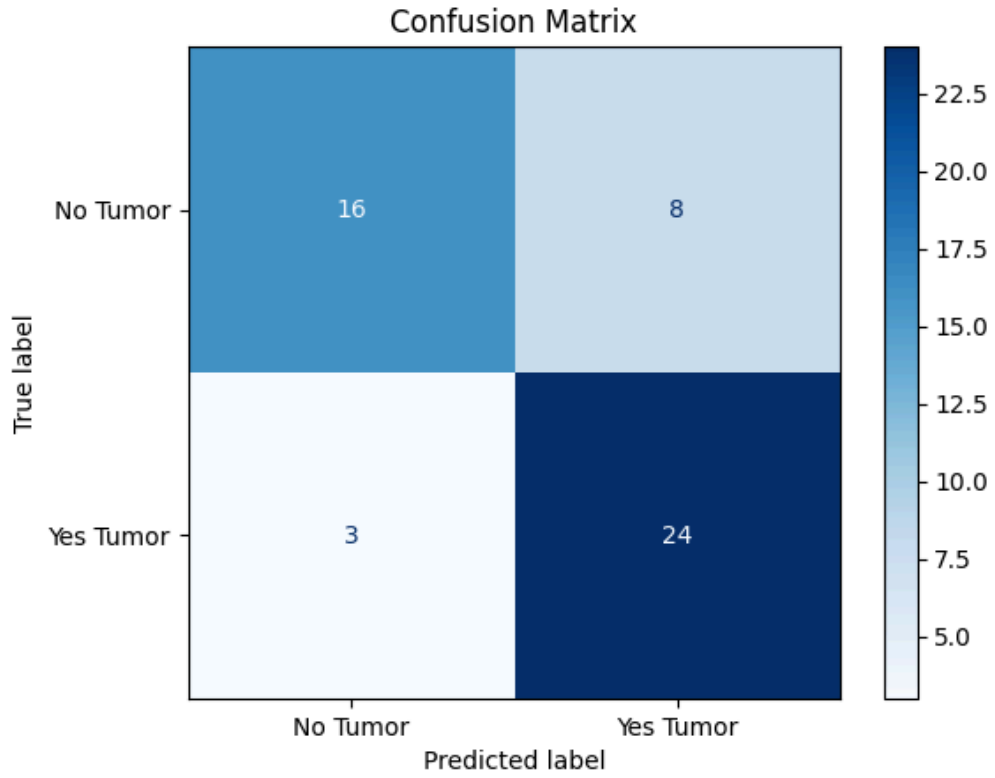
```
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.15),
    Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
history = model.fit(
    x_train, y_train,
    batch_size=32,
    epochs=50,
    validation_data=(x_valid, y_valid),
    verbose=1
)
```

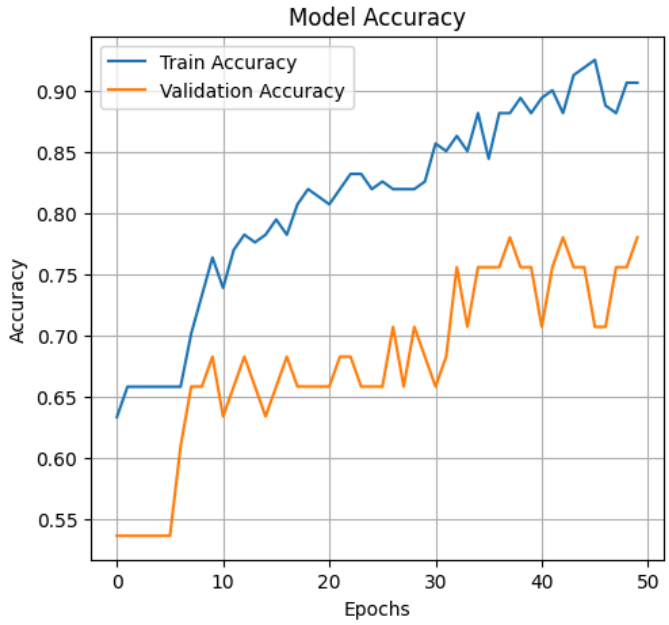
Step 4: Evaluate the model:

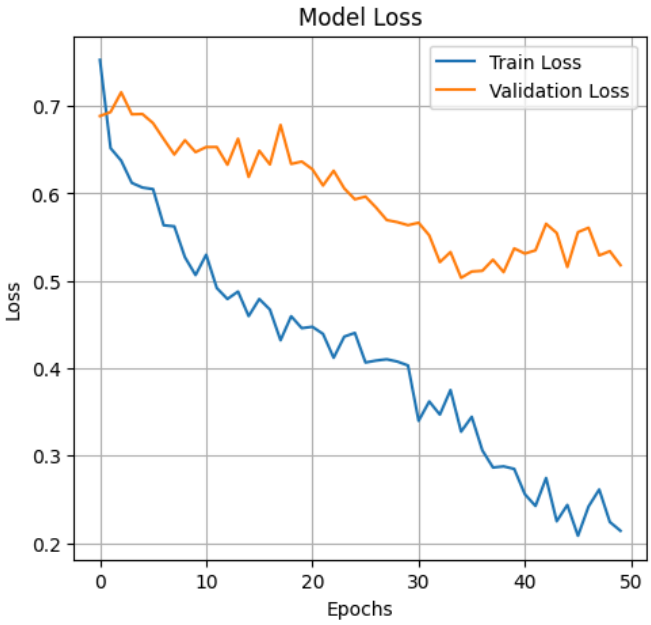
```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print("\nTest accuracy:", test_acc)

from sklearn.metrics import classification_report,
confusion_matrix, ConfusionMatrixDisplay

# Convert predictions to binary (0 or 1)
y_pred = (y_hat > 0.5).astype("int32")

print("\n Classification Report:")
print(classification_report(y_test, y_pred, target_names=["No
Tumor", "Yes Tumor"]))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=["No Tumor", "Yes Tumor"])

plt.figure(figsize=(6,6))
disp.plot(cmap=plt.cm.Blues, values_format='d')
plt.title("Confusion Matrix")
plt.show()
```

```
plt.figure(figsize=(12,5))

# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'],       label='Validation
Accuracy')
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)

# Loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Model Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

plt.show()
```

```
Test accuracy: 0.7647058963775635

 Classification Report:
              precision    recall  f1-score   support

    No Tumor       0.77      0.71      0.74        24
   Yes Tumor       0.76      0.81      0.79        27

    accuracy                           0.76        51
   macro avg       0.77      0.76      0.76        51
weighted avg       0.77      0.76      0.76        51
```







Step 4: Predict on test images:

```
y_hat = model.predict(x_test)
labels = ["No Tumor", "Yes Tumor"]
```

```
figure = plt.figure(figsize=(15, 8))
for i, index in enumerate(np.random.choice(x_test.shape[0],
size=6, replace=False)):
    ax = figure.add_subplot(3, 3, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[index]))
    predict_index = 1 if y_hat[index] > 0.5 else 0
    true_index = y_test[index]
                    ax.set_title("Pred:     {}     |     Actual:
{}".format(labels[predict_index],

labels[true_index]),
                    color=("green" if predict_index == true_index
else "red"))
plt.show()
```