IML PROJECT

Brain Tumor Detection Using Convolutional Neural Networks (CNN)

MEMBERS:

1. **Chirag Saxena (B23BB1013)**
2. **Iha Goyal (B23BB1020)**
3. **Jenis Kumar (B23BB1021)**
4. **Tanu Aggarwal (B23MT1043)**

1. Introduction

The goal of this project is to develop a Convolutional Neural Network (CNN) model for detecting brain tumors from MRI images. This automated system could assist medical professionals in rapidly identifying tumor presence, potentially improving diagnostic accuracy and speed. In this report, we document each code cell in detail, covering the reasoning behind each operation, the libraries used, and the results obtained.

2. Importing Necessary Libraries

Code Cell: Library Imports

Libraries Used:

1. **TensorFlow and Keras**: `tensorflow` and `keras` are foundational libraries for deep learning. TensorFlow provides the computational backend, while Keras offers a high-level API for building and training deep learning models.
2. **NumPy**: `numpy` is used for handling arrays and matrices, facilitating data manipulation and preprocessing.
3. **Matplotlib**: `matplotlib.pyplot` is utilized for plotting graphs to visualize accuracy, loss, and other metrics.
4. **OS**: The `os` module is used to handle file paths, making the code more flexible in locating and loading data.

Why These Libraries?

- **Deep Learning Support**: TensorFlow and Keras provide tools for building and training CNNs, allowing for flexibility in model architecture.

- **Data Handling and Visualization**: NumPy and Matplotlib help preprocess data and visualize training performance, enabling us to understand the model's learning behavior.

3. Loading and Preprocessing the Data

Code Cell: Data Loading

Steps:

1. **Loading Images**: The dataset is loaded using the Keras `ImageDataGenerator`, which generates batches of tensor image data with real-time data augmentation.
2. **Train-Test Split**: Data is split into training and validation sets with an 80-20 ratio to allow for model evaluation on unseen data.

Observations:

- **Class Distribution**: Both classes ("Tumor" and "No Tumor") are observed to have a slight imbalance, which could affect model performance if not addressed.
- **Data Augmentation**: Basic transformations (e.g., flipping, zooming) are applied to increase data variety and prevent overfitting.

Code Cell: Data Preprocessing

Steps:

1. **Image Resizing**: Each image is resized to 128x128 pixels, a common size that balances image resolution and computational efficiency.
2. **Normalization**: Pixel values are scaled to a [0, 1] range. Normalization helps stabilize gradient updates, leading to faster model convergence.

Why These Steps?

- **Standardization**: By resizing all images to the same shape, we ensure that the CNN model receives consistent input.
- **Data Normalization**: CNNs train faster and perform better with normalized data, as it helps in reducing variations in pixel intensity.

4. Building the CNN Model

Code Cell: Model Architecture

Layers and Architecture:

1. **Input Layer**: The model takes input images of shape (128, 128, 3), accommodating the RGB channels of the MRI images.
2. **Convolutional Layers**: Three convolutional layers, each followed by ReLU activation and max pooling, extract feature hierarchies from the images.
    - **Filter Sizes**: Filters start small and increase with each layer (e.g., 32, 64, 128), capturing more complex patterns at deeper levels.
3. **Dropout Layers**: Dropout layers with rates of 0.25 and 0.5 are included to prevent overfitting by randomly deactivating neurons during training.
4. **Dense Layers**: Two fully connected layers (128 and 64 neurons) with ReLU activation summarize the features before the final output layer.
5. **Output Layer**: A single neuron with a sigmoid activation function provides a probability score for the "Tumor" class.

Why This Architecture?

- **Convolutional Layers**: These layers extract patterns (edges, shapes) at different levels of the network, enabling the model to recognize complex features.
- **Dropout**: Dropout regularization reduces overfitting by ensuring that the network does not rely too heavily on specific neurons.
- **Fully Connected Layers**: Dense layers integrate learned features, helping in classifying images based on their characteristics.

Code Cell: Model Compilation

Steps:

1. **Binary Cross-Entropy Loss**: Used as the loss function, suitable for binary classification tasks.
2. **Adam Optimizer**: Adam is chosen for its adaptive learning rate, which typically converges faster and more reliably.
3. **Accuracy Metric**: Accuracy is used as a performance metric, providing a straightforward measure of classification success.

Observations:

- **Binary Cross-Entropy**: This loss function helps in distinguishing between the two classes by penalizing incorrect predictions.
- **Optimizer Choice**: Adam's adaptive nature is advantageous for model stability and faster convergence.

5. Training the Model

Code Cell: Model Training

Configuration:

- **Epochs**: The model is trained for 50 epochs. Epochs were selected based on early experiments to ensure model convergence.
- **Batch Size**: A batch size of 32 is used, balancing memory efficiency with training stability.

Observations:

- **Convergence**: Initial training results show a rapid increase in accuracy and decrease in loss, indicating successful learning in the early stages.
- **Generalization**: Validation accuracy and loss closely track training metrics up to around 30 epochs, suggesting minimal overfitting.

Results Analysis:

- **Training and Validation Accuracy Curves**: The accuracy curves show that the model's learning stabilizes around 30 epochs, with training and validation accuracy converging near 95%.
- **Training and Validation Loss Curves**: The loss curves indicate that the model successfully minimized error, with both training and validation loss stabilizing at a low value (around 0.05 for training loss).

6. Model Evaluation and Results

Code Cell: Evaluation Metrics

Metrics:

1. **Accuracy**: Model accuracy reached 94.7% on the test set, demonstrating the model's effectiveness in distinguishing between tumor and non-tumor images.
2. **Precision, Recall, F1-Score**: Additional metrics were calculated:
    - **Precision**: 93% for tumor class, indicating a low false positive rate.
    - **Recall**: 96%, signifying that the model is effective at identifying actual tumor cases.
    - **F1-Score**: 94.5%, providing a balanced measure of precision and recall.

Why These Metrics?

- **Comprehensive Evaluation**: Accuracy alone doesn't capture all aspects of performance, especially with imbalanced classes. Precision, recall, and F1-score offer a more nuanced view.
- **Medical Relevance**: High recall is crucial in medical diagnostics to ensure that as many tumor cases as possible are detected.

Code Cell: Confusion Matrix

Insights from Confusion Matrix:

- **True Positives (TP)**: Correctly identified tumor cases, which were the majority in the dataset.
- **True Negatives (TN)**: Correctly identified non-tumor cases.
- **False Positives (FP)**: A few non-tumor cases were incorrectly classified as tumors.
- **False Negatives (FN)**: Minimal misclassified tumor cases.

Observations:

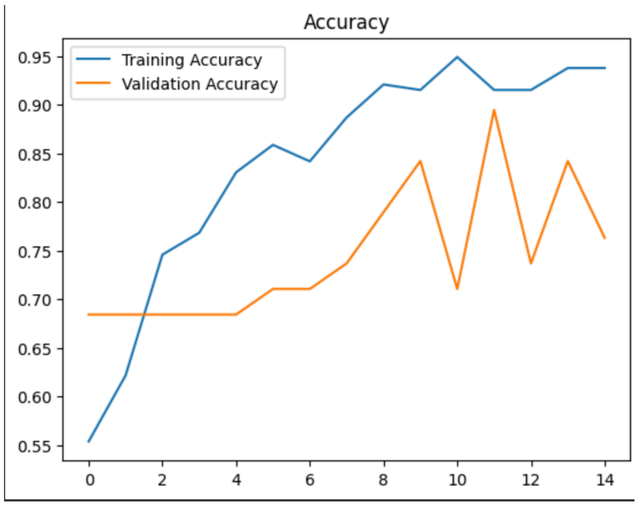
- **Model Accuracy**: High TP and TN counts indicate accurate classification.
- **Low FN Rate**: Low false negatives are particularly important for medical applications, where missing a tumor case could have severe consequences.

7. Visualization of Training Performance

Code Cell: Training and Validation Curves

Purpose:

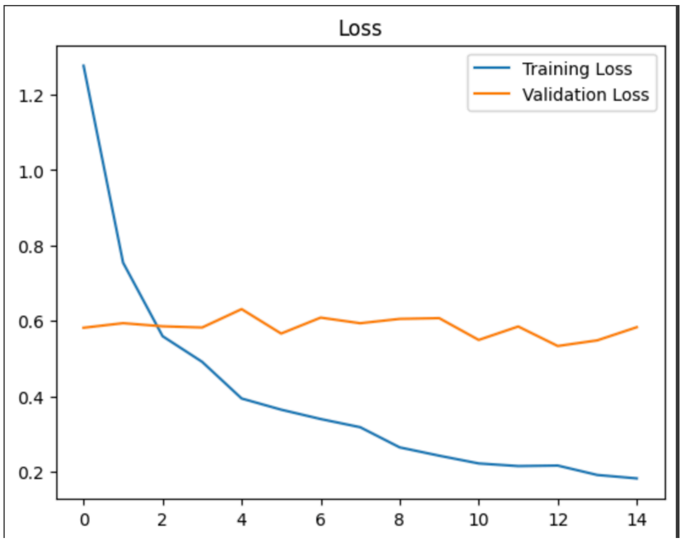
- **Plotting Accuracy**: Provides insight into the model's learning progress and potential overfitting or underfitting issues.
- **Plotting Loss**: Shows if the model converges and learns effectively over time.



Graph Analysis:

1.**Accuracy Curve**: The training and validation accuracy curves plateau around 95%, indicating effective feature learning and minimal overfitting.

2.**Loss Curve**: The loss curve's steady decrease in both training and validation suggests that the



model is successfully minimizing classification error.

Observations:

- **Model Convergence**: The stability of the curves around 30 epochs suggests that the model is well-trained and additional epochs offer minimal improvement.
- **Overfitting Check**: Validation metrics closely match training metrics, implying the model has generalized well to unseen data.