**Project Report – Brain Tumor Classification Using Deep Learning Techniques**

**Team Member:**

Saketh Tatipally

**1. Introduction**

Brain tumors are among the most critical medical conditions, requiring precise and early diagnosis. Detecting brain tumors from MRI scans can save lives by enabling timely and effective treatment. This project aims to build a deep learning-based classification system to automate the diagnosis process and develop a user-friendly graphical user interface (GUI) to enhance accessibility for medical practitioners.

**2. Problem Statement**

The manual inspection of MRI scans by radiologists is labor-intensive, time-consuming, and prone to human error. This project proposes a solution by leveraging deep learning to create an automated classification system for brain tumors. The system ensures faster and more accurate diagnosis, aiding medical professionals in making informed decisions.

**3. Literature and Technology Review**

**3.1 Literature Review**

In recent years, medical imaging has benefited immensely from advancements in deep learning. Convolutional Neural Networks (CNNs), such as VGG16 and DenseNet, have demonstrated remarkable success in image classification tasks. These models leverage their pre-trained architectures to extract complex features from medical images, making them ideal for tumor classification.

**3.2 Technology Review**

* **Python**: Serves as the primary programming language for implementing models and building the web application.
* **TensorFlow/Keras**: Libraries used to construct, train, and evaluate deep learning models.
* **Flask**: A lightweight web framework to deploy the GUI.
* **Google Colab**: Provides an accessible environment for model development and training.
* **HTML/CSS**: Utilized for designing the user interface.
* **Pre-trained Models**: Leveraging VGG16 and DenseNet121 trained on ImageNet for transfer learning.

**4. Dataset Description**

The project uses the Brain Tumor MRI Dataset from Kaggle, which contains MRI images of brain tumors categorized into four classes: glioma, meningioma, no tumor, and pituitary tumors.

**4.1 Data Overview**

* **Dataset Size**: Approximately 7,022 images.
* **Classes**: Glioma, Meningioma, No Tumor, and Pituitary Tumors.
* **Image Format**: Grayscale, converted to RGB for uniformity.
* **Image Size**: Resized to 150x150 pixels for model input.

**4.2 Data Augmentation**

To improve model generalization, the following augmentation techniques were applied:

* Rescaling pixel values to [0, 1].
* Random rotation, brightness adjustment, and slight shifts.

Link to the dataset:

<https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset>

**5. Methodology**

**5.1 Data Collection**

The dataset includes MRI images along with class labels, carefully curated for training and testing purposes.

**5.2 Data Preprocessing**

To enhance the dataset’s quality, preprocessing steps include:

* Noise reduction using bilateral filters.
* Converting grayscale images to RGB.
* Resizing images to ensure uniformity.

**5.3 Model Design**

The project employs CNN-based models—VGG16 and DenseNet121—as base architectures, modified to include custom classification layers for this task.

**Vgg16 model description:**

**Key Features of VGG16:**

* **Architecture:** The model consists of 16 layers—13 convolutional layers and 3 fully connected layers.
* **Convolutional Layers:** The convolutional layers use small 3x3 filters with a stride of 1 and padding of 1. This allows the network to capture spatial features in images with a relatively small number of parameters compared to larger filters.
* **Max Pooling:** After every few convolutional layers, a 2x2 max-pooling layer is applied with a stride of 2. This reduces the spatial dimensions of the image progressively.
* **Fully Connected Layers:** After the convolutional layers, the model has three fully connected (dense) layers, with the last one having 1000 units corresponding to the number of ImageNet categories.
* **Activation Function:** ReLU (Rectified Linear Unit) is used as the activation function for the convolutional and fully connected layers.

**VGG16 Model Structure:**

1. **Input Layer:** 224x224x3 (RGB image of size 224x224 pixels)
2. **Convolutional Block 1:** 2 Conv layers (64 filters each) → Max Pooling
3. **Convolutional Block 2:** 2 Conv layers (128 filters each) → Max Pooling
4. **Convolutional Block 3:** 3 Conv layers (256 filters each) → Max Pooling
5. **Convolutional Block 4:** 3 Conv layers (512 filters each) → Max Pooling
6. **Convolutional Block 5:** 3 Conv layers (512 filters each) → Max Pooling
7. **Fully Connected Layer 1:** 4096 units
8. **Fully Connected Layer 2:** 4096 units
9. **Fully Connected Layer 3:** 1000 units (Softmax output for classification)

**DenseNet121 model description:**

**Key Features of DenseNet121:**

* **Dense Connections:** In DenseNet, each layer is connected to every other layer in a feed-forward fashion. This is different from traditional CNNs, where each layer connects only to the previous layer. This connectivity pattern leads to improved gradient flow during backpropagation and reduces the number of parameters needed.
* **Growth Rate:** DenseNet models have a growth rate (denoted k), which controls the number of filters each layer contributes to the subsequent layers. In DenseNet121, the growth rate is 32, meaning each convolutional layer outputs 32 feature maps (filters), and each subsequent layer receives input from all previous layers.
* **Block Structure:** DenseNet121 is organized into 4 dense blocks, each consisting of several convolutional layers. The number of layers in each block decreases as the model progresses, while the number of feature maps increases.
* **Transition Layers:** Between dense blocks, there are transition layers that perform a 1x1 convolution followed by a 2x2 average pooling operation. These layers reduce the number of feature maps and downsample the feature map dimensions.
* **Bottleneck Layers:** DenseNet121 incorporates bottleneck layers (1x1 convolutions) to reduce the number of channels before performing the 3x3 convolutions, which helps in reducing computational cost while maintaining performance.
* **Compression:** DenseNet models are more parameter-efficient due to the dense connectivity. The model reuses features from all previous layers, which significantly improves the efficiency of the network.

**DenseNet121 Architecture Overview:**

1. **Input Layer:** 224x224x3 (RGB image of size 224x224 pixels)
2. **Initial Convolution:** 7x7 Convolution with 64 filters → 2x2 Max Pooling
3. **Dense Block 1:** 6 layers with growth rate 32
4. **Transition Layer 1:** 1x1 Convolution → 2x2 Average Pooling
5. **Dense Block 2:** 12 layers with growth rate 32
6. **Transition Layer 2:** 1x1 Convolution → 2x2 Average Pooling
7. **Dense Block 3:** 24 layers with growth rate 32
8. **Transition Layer 3:** 1x1 Convolution → 2x2 Average Pooling
9. **Dense Block 4:** 16 layers with growth rate 32
10. **Global Average Pooling:** 1x1 Global Pooling
11. **Fully Connected Layer:** Softmax layer with the number of output units equal to the number of classes (for classification tasks)

**5.4 Model Training**

* **Optimizer**: Adam with a learning rate of 0.001.
* **Loss Function**: Categorical cross-entropy.
* **Epochs**: 20.
* **Batch Size**: 32.

**5.5 Model Evaluation**

Standard metrics, including accuracy and loss, were used to evaluate performance on the test set. Validation accuracy and loss were monitored during training.

**5.6 Fine-Tuning**

Hyperparameter tuning and retraining were performed to enhance model performance when initial results were suboptimal.

**6. Results and GUI Application**

**6.1 Model Results**

* **VGG16 Model**: Achieved an accuracy of 97 % on the test set.
* **DenseNet Model**: Achieved an accuracy of 96 % on the test set.

**6.2 GUI Application**

The GUI enables medical practitioners to interact with the system seamlessly:

* **File Upload**: Users can upload MRI images in PNG, JPG, or JPEG format.
* **Preprocessing**: Uploaded images are resized and normalized for predictions.
* **Model Prediction**: The GUI displays the predicted tumor category.
* **Implementation**: Flask was used to integrate the models with a web-based interface.
* Before Running the GUI Application upload the updated model path.

**7. Instructions for Code Execution**

**7.1 Model Development**

1. Open the Jupyter notebook (brain\_tumofr.ipynb) on Google Colab.
2. Upload the dataset to Google Drive and link it to the notebook.
3. Execute all cells sequentially to train the models and save the .h5 files.

**7.2 Web Application**

1. The GUI\_Application.zip files contains all the of GUI.
2. Install dependencies using pip install -r requirements.txt.
3. Place vgg\_brain.h5 and densenet\_brain.h5 in the model directory.
4. Launch the Flask application with:

python app.py

1. Access the application at http://127.0.0.1:5000.

**8. Conclusion**

This project demonstrated the potential of deep learning in automating brain tumor diagnosis. The integration of VGG16 and DenseNet121 models with a user-friendly web interface simplifies the diagnostic process for medical practitioners. Future work could focus on:

* Expanding the dataset to include more diverse cases.
* Improving model performance through advanced tuning techniques.
* Adding explainability features for enhanced trust and usability.