Brain Tumor Classification using Transfer Learning with ResNet and VGG

*A Project Based Learning Report Submitted in partial fulfilment of the requirements for the award of the degree*

*of*

Bachelor of Technology

in The Department of ECE

DEEP NETWORK ARCITECTURE

23EC2224F

Submitted by

Roll.no: 2310040114

Roll.no: 2310040115

Roll.no: 23100400116

Under the guidance of

DR. Ravi Boda



Department of Electronics and Communication Engineering

Koneru Lakshmaiah Education Foundation, Aziz Nagar

Aziz Nagar – 500075

APRIL - 2025.

Introduction

Brain tumor classification is crucial for early diagnosis and treatment. Deep learning, especially transfer learning, enhances accuracy by leveraging pre-trained models. ResNet (Residual Network) improves feature extraction using skip connections, while VGG (Visual Geometry Group) offers a simple yet effective CNN architecture. These models are fine-tuned on MRI datasets to classify brain tumors efficiently. Transfer learning reduces training time and computational costs while improving precision. This approach aids radiologists by providing automated, accurate tumor detection, enhancing medical diagnosis. Using ResNet and VGG, we aim to achieve high classification accuracy, making brain tumor identification faster and more reliable.

Role of Transfer Learning in Brain Tumor Classification

Transfer learning allows pre-trained deep learning models to be adapted for specific tasks without training them from scratch. This technique is beneficial when dealing with medical imaging, as large labeled datasets are often scarce. By using models pre-trained on large datasets like ImageNet, we can extract relevant features from MRI scans and fine-tune them for brain tumor classification.

Popular Architectures

CNN-Based Approaches

Two of the most commonly used Convolutional Neural Networks (CNNs) in medical imaging are ResNet and VGG:

1. **ResNet (Residual Network)** – It employs residual learning with skip connections to allow deeper network training without vanishing gradient issues. This improves feature extraction and classification accuracy.
2. **VGG (Visual Geometry Group)** – A simpler architecture with uniform convolutional layers, making it easy to fine-tune for various image classification tasks. It extracts spatial features effectively, aiding in precise tumor identification.

Methodology

* **Dataset**: MRI images of brain tumors are used, categorized into different tumor types (e.g., glioma, meningioma, and pituitary tumors).
* **Preprocessing**: Image resizing, normalization, and augmentation are applied to improve model generalization.
* **Feature Extraction**: Pre-trained ResNet and VGG models extract key features from MRI images.
* **Fine-tuning**: The last layers are modified, and the models are trained on tumor-specific data.
* **Classification**: The models predict tumor categories with high accuracy.

.

Dataset Preparation and Training

1. **Dataset Selection**

For this study, we will use the **Kaggle Brain MRI dataset**, which contains MRI images labeled into different brain tumor types (such as glioma, meningioma, pituitary tumor) and normal cases. This dataset provides a well-structured collection of images necessary for training deep learning models.

1. **Dataset Preparation**
2. **Data Preprocessing**
   * Convert images to grayscale (if required) or keep RGB format.
   * Resize images to match the input size of ResNet and VGG (typically 224×224 pixels).
   * Normalize pixel values (scale between 0 and 1).
   * Augment data (rotation, flipping, zooming) to improve model generalization.
3. **Data Splitting**
   * **Training Set (70%)** – Used to train the model.
   * **Validation Set (15%)** – Used for hyperparameter tuning.
   * **Test Set (15%)** – Used for final evaluation.
4. **Model Training**
5. **Load Pre-Trained Models**
   * Use **ResNet50** and **VGG16/VGG19**, pre-trained on ImageNet.
   * Remove fully connected layers and replace them with custom dense layers for classification.
6. **Fine-Tuning**
   * Freeze early layers to retain pre-trained features.
   * Train only the last few layers to adapt to MRI-specific features.
   * Use **Adam** or **SGD** optimizer with **categorical cross-entropy** loss.
7. **Training Process**
   * Train models for **10-50 epochs** (depending on convergence).
   * Use **batch size 16/32** for optimal performance.
   * Apply **early stopping** to prevent overfitting.
8. **Evaluation Metrics**

* **Accuracy**
* **Precision, Recall, F1-Score**
* **Confusion Matrix**

Once trained, the models will be tested on unseen MRI images to evaluate performance.

Implementation in Google Collab

 **Import Required Libraries**

* TensorFlow, Kera’s, OpenCV, NumPy, Matplotlib, Pandas.

 **Load and Preprocess Data**

* Load the **Kaggle Brain MRI dataset**.
* Resize images to **224×224** (for ResNet and VGG).
* Normalize pixel values between **0 and 1**.
* Apply data augmentation (rotation, flipping, zooming).

 **Define Model Architecture**

* Use **ResNet50** and **VGG16/VGG19** pre-trained models.
* Replace the final dense layers with custom layers for tumor classification.
* Use **SoftMax** for multi-class classification.

 **Train the Model**

* Freeze initial layers and fine-tune later layers.
* Use **Adam** optimizer and **categorical cross-entropy** loss.
* Train for **10-50 epochs** with **early stopping**.

 **Evaluate and Test**

* Use **accuracy, precision, recall, F1-score** for evaluation.
* Compare predictions with ground truth using a **confusion matrix**.
* Test on unseen MRI images.

**Code**

import os

import zip file

import NumPy as np

import pandas as pd

import matplotlib. pyplot as plt

import tensor flow as tf

from tensor flow. keras. preprocessing. image import ImageDataGenerator

from tensorflow.keras.applications import ResNet50, VGG16

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam

from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

# 1. Define path to your zipped dataset

dataset\_path = '/content/DL-DATASET.zip’ # Correct path to your zip file

# 2. Extract the zip file to a temporary directory

with zipfile.ZipFile(dataset\_path, 'r') as zip\_ref:

    zip\_ref. extract all('/content/dataset’) # Extract to '/content/dataset'

# 3. Update paths for train and test directories

train\_dir = os. path. Join ('/content/dataset', 'Training’) # Corrected path

test\_dir = os.path.join('/content/dataset', 'Testing')    # Corrected path

img\_width, img\_height = 224, 224

batch\_size = 32

# Data augmentation for training

train\_datagen = ImageDataGenerator(

    rescale=1./255,

    rotation\_range=20,

    width\_shift\_range=0.2,

    height\_shift\_range=0.2,

    shear\_range=0.2,

    zoom\_range=0.2,

    horizontal\_flip=True,

    fill\_mode='nearest'

)

# Only rescaling for testing

test\_datagen = ImageDataGenerator(rescale=1./255)

# Load training and testing data

train\_generator = train\_datagen.flow\_from\_directory(

    train\_dir,

    target\_size= (img\_width, img\_height),

    batch\_size=batch\_size,

    class\_mode='categorical'

)

test\_generator = test\_datagen.flow\_from\_directory (

    test\_dir,

    target\_size= (img\_width, img\_height),

    batch\_size=batch\_size,

    class\_mode='categorical',

    shuffle=False # Important for confusion matrix and classification report

)

# Model Selection (ResNet50 or VGG16)

def build\_model(base\_model, input\_shape=(img\_width, img\_height, 3),

num\_classes=4):

    """Builds a model using transfer learning."""

    for layer in base\_model.layers:

        layer.trainable = False  # Freeze base layers

    x = base\_model.output

    x = GlobalAveragePooling2D () (x)

    x = Dense (1024, activation='relu’) (x)

    x = Dropout (0.5) (x)

    predictions = Dense (num\_classes, activation='softmax’) (x)

model = Model (inputs=base\_model. input, outputs=predictions)

    return model

# Choose Resnet50 or VGG16

base\_model = ResNet50(weights='imagenet', include\_top=False, input\_shape= (img\_width, img\_height, 3)) #or VGG16

#base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape= (img\_width, img\_height, 3))

model = build\_model (base\_model, num\_classes=len(train\_generator.class\_indices))

# Compile the model

model.compile(optimizer=Adam(learning\_rate=0.0001), loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

epochs = 3

history = model.fit(

    train\_generator,

    epochs=epochs,

    validation\_data=test\_generator

)

# Evaluate the model

loss, accuracy = model.evaluate(test\_generator)

print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")

# Classification Report and Confusion Matrix

y\_true = test\_generator.classes

y\_pred = model.predict(test\_generator)

y\_pred\_classes = np. argmax (y\_pred, axis=1)

print ("Classification Report:")

print (classification\_report (y\_true, y\_pred\_classes, target\_names=test\_generator.class\_indices.keys()))

print ("Confusion Matrix:")

cm = confusion\_matrix (y\_true, y\_pred\_classes)

plt. figure (figsize= (8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=test\_generator.class\_indices.keys(), yticklabels=test\_generator.class\_indices.keys())

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.title('Confusion Matrix')

plt.show()

# Plot training history

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.ylim([0, 1])

plt.legend(loc='lower right')

plt.show()

plt.plot(history.history['loss'], label='loss')

plt.plot(history.history['val\_loss'], label = 'val\_loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend(loc='upper right')

plt.show()

Future Scope

1. **Improved Model Performance** 
   * Implement advanced deep learning models like **Efficient Net** or **Vision Transformers (ViTs)** for better accuracy.
   * Use **self-supervised learning** to reduce dependency on large labeled datasets.
2. **3D MRI Image Analysis**
   * Extend the model to analyze **3D volumetric MRI scans** instead of 2D slices for more precise tumor localization.
   * Integrate **3D CNNs or Transformer-based architectures**.
3. **Explainable AI (XAI) in Medical Diagnosis**
   * Use techniques like **Grad-CAM** and **SHAP values** to provide visual explanations for model predictions, helping radiologists trust AI decisions.
4. **Integration with Clinical Systems**
   * Deploy AI models into **hospital imaging systems** for real-time tumor detection.
   * Develop **mobile or web-based applications** for remote diagnosis.
5. **Multi-Modal Learning**
   * Combine MRI data with **histopathology images, genomic data, and clinical reports** for more comprehensive diagnosis.
6. **Federated Learning for Privacy Preservation**
   * Train models across multiple hospitals without sharing sensitive patient data, ensuring privacy and compliance with healthcare regulations.
7. **Automated Treatment Planning**
   * Extend classification models to **predict tumor grade and suggest treatment plans**, assisting oncologists in decision-making.