Brain Tumor Classification Using Convolutional Neural Networks: A Comprehensive Study

K. Eswar
211FA18004
ACSE

K. Visleksha
211FA18045
ACSE

***Abstract*—Brain tumors are among the most lethal forms of cancer, and their early and accurate diagnosis is crucial for effective treatment. This paper presents an extensive study on the use of Convolutional Neural Networks (CNNs) for classifying brain tumors from MRI images. We explore various components of the project, including data preprocessing, model architecture, training techniques, evaluation metrics, and the deployment of a user-friendly application using Streamlit. The aim is to provide an in-depth understanding of each aspect to aid researchers and practitioners in the field.**

***Index Terms*—Brain Tumor, Convolutional Neural Networks, MRI, Data Augmentation, Deep Learning, Streamlit**

I. Introduction

A. Background

Brain tumors can be broadly classified into primary and secondary tumors. Primary brain tumors originate in the brain, while secondary (metastatic) tumors spread to the brain from other parts of the body. The most common types of primary brain tumors include gliomas, meningiomas, and pituitary tumors. Accurate classification of these tumors is essential for determining appropriate treatment strategies.

B. Motivation

Manual analysis of MRI scans by radiologists is time-consuming and prone to human error. Automated systems using machine learning can significantly improve the speed and accuracy of diagnosis. CNNs have shown remarkable success in image classification tasks, making them suitable for brain tumor classification.

II. Methodology

A. Dataset

The dataset used in this study comprises MRI images categorized into four classes: glioma, meningioma, no tumor, and pituitary. Each class represents a distinct type of brain tissue or tumor. The dataset is split into training, validation, and test sets to facilitate model development and evaluation.

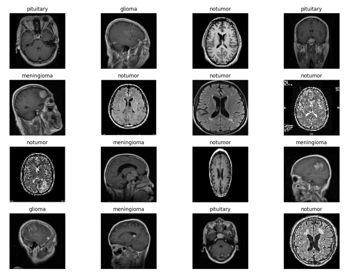


Fig. 1: some images from dataset

B. Data Preprocessing

Data preprocessing is a critical step to ensure that the input images are in a format suitable for the CNN model. Key preprocessing techniques include:

- *Rescaling*: Normalizing pixel values to a range of 0 to 1 to ensure uniformity.
- *Resizing*: Standardizing image dimensions to a fixed size to match the input requirements of the CNN.
- *Data Augmentation*: Applying transformations such as horizontal flipping and rotation to increase the diversity of the training data, thereby improving the model's generalization capability.

1) Code Snippet: Data Augmentation:

```
from keras.preprocessing.image
import ImageDataGenerator

# Data Augmentation for Training Dataset
traingen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    validation_split=0.15
)

# Data Augmentation for Testing Dataset
testgen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True
)
```

C. Convolutional Neural Networks (CNNs)

CNNs are specialized neural networks designed for image processing tasks. They consist of several types of layers, each performing a specific function:

1) Convolutional Layers: These layers apply convolutional filters to the input image, detecting features such as edges, textures, and patterns. Convolutional layers help in capturing spatial hierarchies in images.

2) Batch Normalization: Batch normalization layers normalize the input to each layer, speeding up training and stabilizing the learning process. This technique reduces internal covariate shift by maintaining the mean and variance of the input.

3) Max Pooling: Max pooling layers reduce the spatial dimensions of the feature maps, retaining the most significant features and reducing computational complexity. This layer helps in achieving spatial invariance.

4) Dropout: Dropout layers randomly set a fraction of the input units to zero during training, preventing overfitting and improving the model’s generalization.

5) Fully Connected Layers: Fully connected layers, or dense layers, perform the final classification based on the features extracted by the convolutional layers. These layers integrate all the features and output the class probabilities.

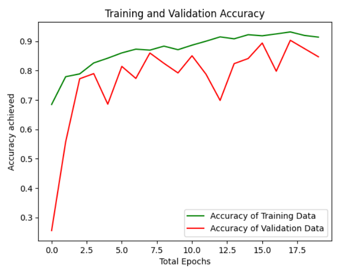
D. Model Architecture

The CNN model used in this study consists of several convolutional layers followed by fully connected layers. The architecture is designed to balance complexity and performance, ensuring that the model can learn intricate patterns without overfitting.

1) Code Snippet: Model Architecture:

```
from keras.models import Sequential
from keras.layers import Conv2D,
MaxPooling2D, Flatten, Dense,
Dropout, BatchNormalization

cnn = Sequential([
    Conv2D(filters=32,
    kernel_size=(3, 3),
    padding=’same’, activation=’relu’,
    input_shape=(130, 130, 3)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(filters=64,
    kernel_size=(3, 3),
    padding=’same’, activation=’relu’),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(filters=128,
    kernel_size=(3, 3),
    padding=’same’, activation=’relu’),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(256, activation=’relu’),
    BatchNormalization(),
    Dropout(0.5),
    Dense(128, activation=’relu’),
    BatchNormalization(),
```



```
    Dropout(0.5),
    Dense(64, activation=’relu’),
    BatchNormalization(),
    Dense(4, activation=’softmax’)
    # Assuming 4 classes for classification
])
```

E. Training the Model

The model is trained using the Adam optimizer, which is efficient for training deep networks. The categorical cross-entropy loss function is used to measure the performance of the model during training. The training process involves iterating over the training data for a specified number of epochs, updating the model parameters to minimize the loss.

1) Code Snippet: Training the Model:

```
cnn.compile(optimizer=’adam’,
loss=’categorical_crossentropy’,
metrics=[’accuracy’])
history = cnn.fit(trainds,
validation_data=valds,
epochs=20, batch_size=16, verbose=1)
```

F. Evaluation

Model performance is evaluated using metrics such as accuracy, precision, recall, and F1-score. These metrics provide a comprehensive understanding of the model’s classification ability. Additionally, confusion matrices are used to visualize the performance across different classes.

1) Code Snippet: Evaluating the Model:

```
cnn.evaluate(testds)
```

G. Results

The trained CNN model achieved high accuracy on the test dataset, demonstrating its effectiveness in classifying brain tumors from MRI images. The training and validation accuracy curves indicate the model’s learning progress and help in identifying any potential overfitting.

1) Plot: Training and Validation Accuracy:

```
import matplotlib.pyplot as plt

epochs = range(len(history.history[’accuracy’]))
plt.plot(epochs,
history.history[’accuracy’], ’green’,
label=’Accuracy of Training Data’)
plt.plot(epochs,
history.history['val_accuracy'], 'red',
label='Accuracy of Validation Data')
plt.xlabel('Total Epochs')
plt.ylabel('Accuracy achieved')
plt.title('Training and Validation Accuracy')
plt.legend(loc=0)
plt.show()
```

III. Discussion

A. Significance of Results

The high accuracy achieved by the model underscores the potential of CNNs in medical image classification. Automated systems like this can significantly aid radiologists by providing quick and reliable initial assessments, leading to better patient outcomes.

B. Challenges

- *Data Quality*: The quality of MRI images can vary, and artifacts in the images can affect model performance.
- *Data Quantity*: Larger datasets with diverse examples can further improve model accuracy.
- *Computational Resources*: Training deep learning models requires substantial computational power, which may not be accessible to all researchers.

C. Future Work

Future work can focus on improving the model by exploring advanced architectures such as ResNet or Inception. Additionally, integrating other modalities of medical imaging and using multi-task learning can enhance the system's diagnostic capabilities.

IV. Streamlit Application

A. Overview

Streamlit is an open-source app framework that allows for the rapid development of web applications for machine learning and data science projects. The Streamlit application developed in this study provides an intuitive interface for users to upload MRI images and receive tumor classification results.

B. User Interface

The user interface is designed to be simple and user-friendly. Users can upload an MRI image, and the application displays the uploaded image along with the predicted tumor class.

1) Code Snippet: Streamlit App:

```
import streamlit as st
from keras.models import load_model
from keras.preprocessing.image
import img_to_array
from PIL import Image, ImageOps
import numpy as np

model = load_model
        ('brain_tumor_model.h5')
class_names = ['glioma', 'meningioma',
               'notumor', 'pituitary']

def preprocess_image(image):
    size = (130, 130)
    image = ImageOps.fit
    (image, size, Image.ANTIALIAS)
    img_array = img_to_array(image)
    img_array = img_array / 255.0
    img_array = np.expand_dims
    (img_array, axis=0)
    return img_array

st.title("Brain Tumor Classification")
uploaded_file = st.file_uploader
("Choose an image...",
type=["jpg", "jpeg", "png"])

if uploaded_file is not None:
    image = Image.open(uploaded_file)
    st.image(image,
    caption='Uploaded Image',
    use_column_width=True)
    st.write("")
    st.write("Classifying...")

    img_array = preprocess_image(image)
    prediction = model.predict(img_array)
    predicted_class =
    class_names[np.argmax(prediction)]
    st.write(f"Prediction: {predicted_class}")
```

C. Deployment

The Streamlit app can be deployed on various platforms, including local servers, cloud services, and containerized environments. This ensures that the tool is accessible to healthcare providers and researchers.

D. Benefits

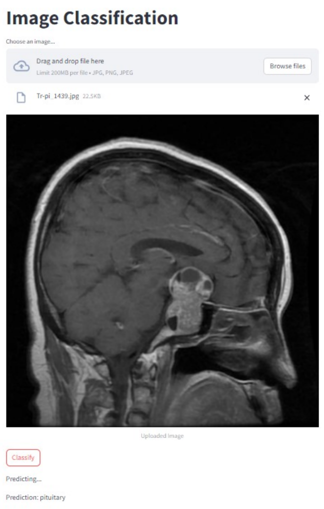
- *Accessibility*: Provides a user-friendly interface for non-technical users.
- *Efficiency*: Automates the classification process, saving time for radiologists.
- *Scalability*: Can be deployed on various platforms to reach a wider audience.

V. Conclusion

This study demonstrates the application of Convolutional Neural Networks for classifying brain tumors from MRI images. The model achieved high accuracy, indicating its potential for real-world medical applications. Deploying the model using a Streamlit application makes it accessible to non-technical users, providing a practical tool for aiding in the diagnosis of brain tumors.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems, vol. 25, 2012.



Predicting...

Prediction: pituitary

Fig. 2: application

[2] K. Simonyan and A. Zisserman, ”Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv preprint arXiv:1409.1556, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, ”Deep Residual Learning for Image Recognition,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778, 2016.