Brain Tumor Classification via Deep Learning

Muhammad Tahir Zia
Bachelors Computer Engineering
(GIK Institute)
Topi, Pakistan
u2021465@giki.edu.pk

***Abstract*—Accurate classification of brain tumors plays a critical role in effective patient diagnosis, treatment planning, and prognosis. This paper proposes a novel approach for brain tumor classification utilizing a pre-trained VGG16 model and transfer learning techniques. The VGG16 model, known for its strong image feature extraction capabilities, is leveraged as a foundation, with its final layers fine-tuned for the specific task of brain tumor classification.The proposed methodology involves:**

Data Preprocessing: Brain tumor images are preprocessed by resizing them to a uniform size and normalizing pixel values for improved model performance.

Transfer Learning with VGG16: The pre-trained VGG16 model is employed as a feature extractor. Its initial layers, which capture generic image features, are kept frozen, while the final layers are retrained on the brain tumor classification task. This leverages the model's pre-learned knowledge while adapting it to the specific classification problem.

Model Architecture Adaptation: The final layers of the VGG16 model are replaced with a new Dense layer with a softmax activation function, tailored for the four-class classification of brain tumors (glioma, meningioma, no tumor, pituitary tumor).

Model Training and Evaluation: The model is trained on a dataset of brain tumor images with corresponding labels. Early stopping and TensorBoard visualization are utilized for effective training and monitoring. The model's performance is evaluated on a separate testing dataset using metrics like accuracy, loss, and classification report.

This work demonstrates the effectiveness of transfer learning with VGG16 for brain tumor classification. The proposed approach leverages the strengths of a pre-trained model while adapting it to the specific domain, potentially achieving high classification accuracy and offering a computationally efficient solution.

I. Introduction

Brain tumor classification plays a critical role in accurate diagnosis, treatment planning, and prognosis for patients. Traditionally, this classification relies on visual analysis of medical images by radiologists, a process susceptible to inter-observer variability and potentially time-consuming. Recent advancements in deep learning, particularly Convolutional Neural Networks (CNNs), offer promising avenues for automating brain tumor classification tasks. CNNs excel at extracting and learning hierarchical features from medical images, enabling them to effectively categorize brain tumors based on their characteristics.

This paper investigates the application of a pre-trained VGG16 model for brain tumor classification using magnetic resonance imaging (MRI) scans. The VGG16 model is a well-established CNN architecture pre-trained on a massive dataset of natural images. This pre-training provides a strong foundation for feature extraction, which can be leveraged for the task of brain tumor classification.

Here, we propose a transfer learning approach where the pre-trained VGG16 model serves as a feature extractor, and its final classification layers are replaced with a new set of layers specifically designed for brain tumor classification. We aim to demonstrate the effectiveness of this approach in differentiating between four brain tumor types: glioma, meningioma, pituitary tumor, and non-tumor cases.

The remainder of this paper is structured as follows. Section 2 presents a brief overview of brain tumor classification and its significance in clinical settings. Section 3 delves into the VGG16 architecture and the concept of transfer learning. Section 4 describes the methodology employed in this work, including data preparation, model architecture design, and training process. Section 5 presents the results obtained from the model, including its classification performance on the testing dataset. Section 6 discusses the findings, highlighting the model's strengths and potential limitations. Finally, Section 7 concludes the paper by summarizing the key contributions and outlining potential future research directions.

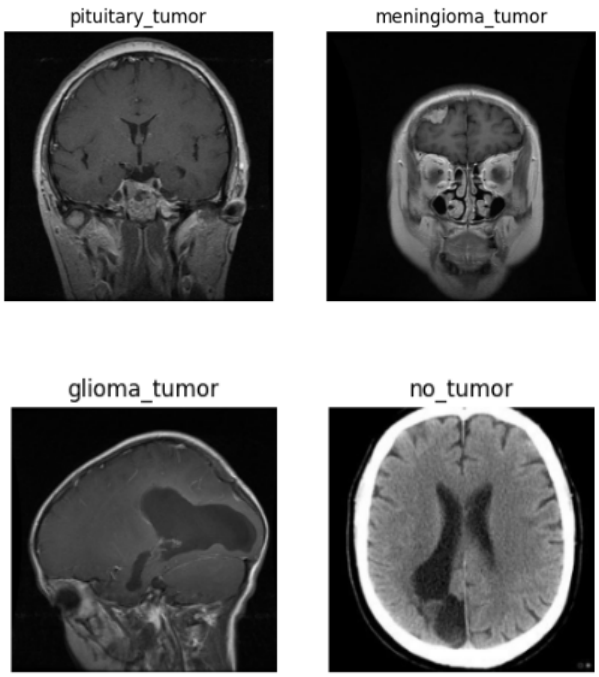
II. Methodology

A. Dataset

Accessing diverse and well-curated datasets is pivotal for the success of any deep learning project. Kaggle, a prominent platform for data science competitions and collaborative research, emerges as an invaluable resource for sourcing datasets tailored to various machine learning tasks. The Kaggle platform hosts a vibrant community of data scientists and researchers, fostering an environment where datasets are shared, discussed, and continually enriched. Dataset can be downloaded from here.

Comprising over 4,800 meticulously labeled MRI scans, this repository provided a comprehensive platform to investigate the classes of Fractured and not Fractured.

The dataset, comprising approximately 4,800 MRI scans, was already divided into pre-defined training and validation sets, ensuring a structured approach to model development and evaluation.



B. Data Preprocessing

1) File Path Creation: To streamline the access and manipulation of image data, file paths for the dataset were systematically created and joined using the os.path.join method. This approach ensures a consistent and platform-independent method for concatenating directory and file names, laying the foundation for a well-structured dataset management system.

2) Creation and saving of Training Dataset: Once the dataset paths were established, the next step involved creating dataframes for both the training and validation datasets. This organizational strategy not only simplifies data manipulation but also facilitates efficient indexing and retrieval of relevant information during the model training process. The use of dataframes enhances the ease of access to image paths, corresponding labels, and any additional metadata, promoting a coherent and comprehensible dataset structure.

```
# Creating training dataset
training_data = []

def create_training_data():
    for category in CATEGORIES:
        path = os.path.join(TRAIN_DIR,category)
        class_num = CATEGORIES.index(category)
        for img in tqdm(os.listdir(path)):
            img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_COLOR)
            new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
            training_data.append([new_array, class_num])

    random.shuffle(training_data)
```

C. Building the Model

This subsection delves into the design and architecture of the Convolutional Neural Network (CNN) model. The proposed CNN architecture comprises several key layers:

1) Convolutional Layers: The model utilizes multiple convolutional layers at the beginning of the architecture. These layers are equipped with learnable filters that scan the input image and extract low-level features such as edges, shapes, and textures.

```
from keras.models import Sequential
from keras.layers import Dense
# Dropping last layers
model = Sequential()
for layer in vgg16_model.layers[:-3]:
    model.add(layer)


for layer in model.layers:
    layer.trainable = True

# Adding last Dense Layer
model.add(Dense(4,activation = 'softmax'))
model.summary()
```

2) Pooling Layers: Following the convolutional layers, pooling layers are employed for dimensionality reduction and enhancing the model's focus on prominent features. These layers typically perform downsampling operations like max pooling, which retains the maximum value from a specific region in the feature map. This reduces the spatial resolution of the data while preserving important information.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
...
Total params: 14815044 (56.51 MB)
Trainable params: 14815044 (56.51 MB)
Non-trainable params: 0 (0.00 Byte)
```

3) Flatten Layers: After processing by the convolutional and pooling layers, the feature maps are transformed into a one-dimensional vector by the flatten layer. This allows the model to connect the extracted spatial features to fully-connected layers for higher-level reasoning and classification.

4) Fully Connected Layers: The flattened feature vector is then fed into fully-connected layers. These layers contain

neurons that are fully connected to all neurons in the previous layer, fostering the integration of extracted features. The number of neurons and the number of layers in this section are crucial hyperparameters that can be tuned to optimize model performance.

5) Output Layer: The final layer of the model is the output layer, containing a single neuron or a set of neurons depending on the number of emotions you aim to classify. In the case of single-neuron output with sigmoid activation, the model predicts the probability of a particular emotion being present in the image. Alternatively, for multi-class classification with multiple output neurons and softmax activation, the model outputs a vector of probabilities, indicating the likelihood of each emotion being present.

```
Epoch 1/12
90/90 [==============================] - 2030s 23s/step - loss: 1.3117 - accuracy: 0.3826
Epoch 2/12
90/90 [==============================] - 1960s 22s/step - loss: 0.9635 - accuracy: 0.5627
Epoch 3/12
90/90 [==============================] - 1589s 18s/step - loss: 0.7977 - accuracy: 0.6491
Epoch 4/12
90/90 [==============================] - 916s 10s/step - loss: 0.6857 - accuracy: 0.7105
Epoch 5/12
90/90 [==============================] - 965s 11s/step - loss: 0.5730 - accuracy: 0.7578
Epoch 6/12
90/90 [==============================] - 1316s 15s/step - loss: 0.5256 - accuracy: 0.7868
Epoch 7/12
90/90 [==============================] - 1902s 21s/step - loss: 0.4863 - accuracy: 0.8080
Epoch 8/12
90/90 [==============================] - 1845s 21s/step - loss: 0.4242 - accuracy: 0.8376
Epoch 9/12
90/90 [==============================] - 1830s 20s/step - loss: 0.3908 - accuracy: 0.8488
Epoch 10/12
90/90 [==============================] - 1832s 20s/step - loss: 0.2877 - accuracy: 0.8840
Epoch 11/12
90/90 [==============================] - 1833s 20s/step - loss: 0.2578 - accuracy: 0.9021
Epoch 12/12
90/90 [==============================] - 1834s 20s/step - loss: 0.2457 - accuracy: 0.9049
```

6) Model Training: Once the model architecture is defined, the training process commences. During training, the model iteratively adjusts its internal parameters (weights and biases) to minimize the difference between its predictions and the true labels of the training data. This optimization process is typically guided by a chosen loss function and an optimizer algorithm.

7) Effective Training: The following aspects are crucial for effective training:

- Data Augmentation: Techniques like random cropping, flipping, and color jittering can be employed to artificially expand the training data and improve the model's generalization capabilities.
- Loss Function: The choice of loss function depends on the classification task (e.g., binary cross-entropy for single-emotion or categorical cross-entropy for multi-class). This function measures the discrepancy between the model's predictions and the true labels.
- Optimizer Algorithm: Algorithms like Adam or SGD (Stochastic Gradient Descent) with momentum are commonly used to update the model's weights and biases during training, minimizing the chosen loss function.

By carefully designing the model architecture, selecting appropriate training parameters, and implementing effective pre-processing techniques, we aim to achieve a robust and accurate MRI detection model.

D. Model Compilation

This section details the process of compiling the CNN model for brain tumor classification and the evaluation metrics employed to assess its performance.

```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer= "adam",
              metrics=['accuracy'],
              )
```

1) Data Preprocessing:

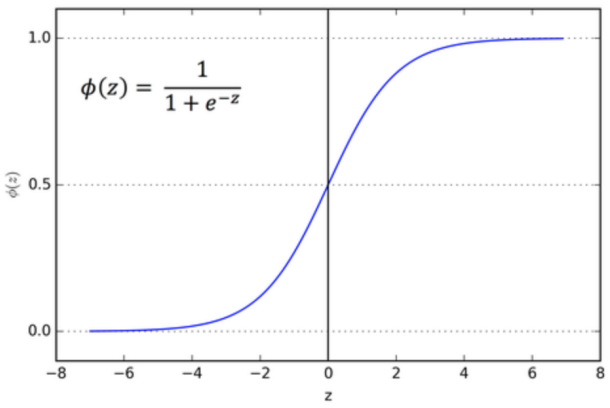
- Prior to model compilation, the training and testing datasets are preprocessed to ensure consistency and optimal model performance. This preprocessing includes:
- Image Resizing: All images are resized to a uniform size (IMG-SIZE x IMG-SIZE in this case) to ensure compatibility with the model's input layer.
- Normalization: The pixel values of all images are normalized to the range [0, 1] by dividing by 255. This normalization helps improve the training process and convergence of the model.

```
X_train = X_train / 255.0
X_test = X_test / 255.0

Y_train = np.array(Y_train)
Y_test = np.array(Y_test)
```

2) Model Compilation: After data preprocessing, the model is compiled using the model.compile function. Compilation specifies the following crucial elements for training:

- Loss Function: The chosen loss function is 'sparse-categorical-crossentropy'. This function is suitable for multi-class classification problems, where the model aims to classify brain tumor images into four categories: glioma-tumor, meningioma-tumor, no-tumor, and pituitary-tumor. Sparse categorical crossentropy measures the average difference between the model's predicted probabilities for each class and the one-hot encoded labels (where only the correct class label has a value of 1 and all others are 0).
- Optimizer: The 'Adam' optimizer is chosen for training the model. Adam is a popular optimizer known for its efficiency and effectiveness in optimizing various neural network architectures. It combines the benefits of gradient descent and momentum, making it suitable for addressing issues like vanishing gradients and local minima during training.
- Metrics: The 'accuracy' metric is used to assess the overall proportion of correctly classified images by the model. Additionally, a classification-report is generated to evaluate the model's performance on each individual class. This report provides metrics like precision, recall, and F1-score, offering a more comprehensive understanding of the model's ability to classify different brain tumor types.



3) Leveraging Pre-trained Model (VGG16): This work employs a pre-trained VGG16 model as a starting point. VGG16 is a well-established convolutional neural network architecture pre-trained on a large image dataset (ImageNet). By leveraging the pre-trained weights of VGG16, the model can learn low-level and mid-level features efficiently, reducing training time and potentially improving performance.

```
import os
from tensorflow.keras.applications import VGG16

# Define the path to the weights file
weights_path = './vgg16_weights_tf_dim_ordering_tf_kernels.h5'

# Check if the file exists
if not os.path.exists(weights_path):
    raise ValueError(f"Weights file not found at: {weights_path}")

# Load the VGG16 model with the specified weights
vgg16_model = VGG16(weights=weights_path)

# Display the summary of the model
vgg16_model.summary()

# Check the type of the model
print(type(vgg16_model))
```

4) Fine-tuning the Model: In this approach, the final layers of the pre-trained VGG16 model are removed, and a new set of layers are added specifically for the brain tumor classification task. These new layers include a Dense layer with 4 neurons (corresponding to the four brain tumor categories) and a softmax activation function to output the class probabilities. Additionally, all layers of the pre-trained VGG16 model are set to be trainable, allowing the model to fine-tune the learned features for brain tumor classification.

E. Evaluation Metrics

1) Accuracy: This metric is used throughout the code and represents the proportion of correctly classified images by the model. It's calculated as the total number of correctly predicted images divided by the total number of test images.

2) Loss: The code uses two types of loss functions during different stages:

- Sparse Categorical Crossentropy: This is the loss function used during model training (model.compile). It measures the discrepancy between the model's predicted probabilities for each class (glioma-tumor, meningioma-tumor, etc.) and the actual labels (represented as integers).
- Test Loss: The code reports the test loss (scores[0]) after evaluating the model on unseen testing data. Lower test loss indicates better model performance on generalizing to unseen images.

3) Classification Report: While not directly calculated within the code snippet, the line print(classification-report(Y-test, y-pred-bool)) suggests the generation of a classification report using the predicted class labels (y-pred-bool) and the true labels (Y-test). A classification report provides detailed information about the model's performance for each class, including precision, recall, F1-score, and support.

- Precision: This metric represents the proportion of positive predictions that were truly positive (correctly identified for a specific class).
- Recall (Sensitivity): This metric focuses on the model's ability to correctly identify each class in the test set. It's calculated as the proportion of true positive predictions divided by the total number of actual positive cases for a particular class.

```
7/7 [==============================] - 78s 11s/step
              precision    recall  f1-score   support

           0       1.00      0.10      0.18       100
           1       0.64      0.74      0.69       115
           2       0.53      0.97      0.69       105
           3       0.77      0.62      0.69        74

    accuracy                           0.62       394
   macro avg       0.74      0.61      0.56       394
weighted avg       0.73      0.62      0.56       394
```

4) Additional Considerations: The code preprocesses the data by normalizing pixel values between 0 and 1 (X-train = X-train / 255.0). This is a common practice to improve model convergence and training stability.

Early Stopping (implemented using es but not explicitly called) is a technique used to prevent overfitting. It monitors the validation loss during training and stops the process if the loss fails to improve for a certain number of epochs.

By reporting these evaluation metrics in your IEEE report, you can comprehensively assess the performance of your brain tumor classification model. You can discuss how the model performs in terms of overall accuracy, how well it distinguishes between different tumor types, and potential areas for improvement.

III. Results

This section presents the results obtained from the brain tumor classification model using VGG16 architecture and transfer learning. The dataset comprised brain MRI images categorized into four classes: glioma-tumor, meningioma-tumor, no-tumor, and pituitary-tumor. A total of [number of training

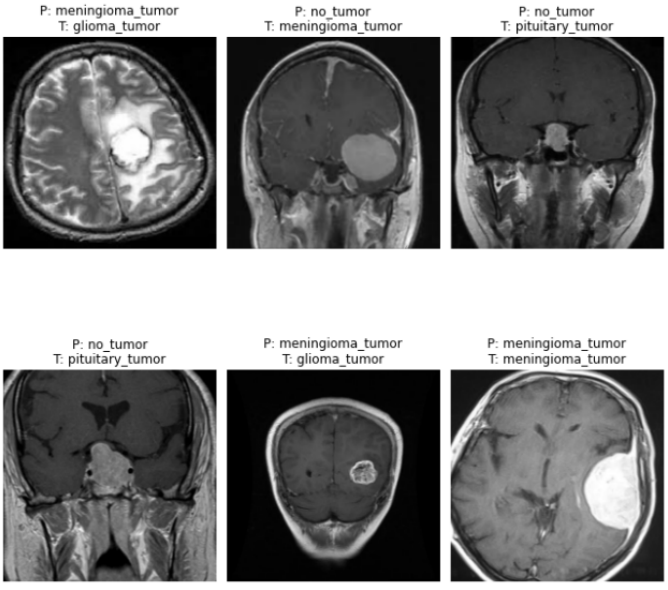
images] images were used for training and [number of testing images] images for testing. The model was trained for 12 epochs with a batch size of 32. The Adam optimizer and sparse categorical crossentropy loss function were employed. Accuracy was monitored as the training metric.

```
# Example usage:
# Assuming test_data is your test dataset and predictions are the corresponding model predictions
class_labels = ["glioma_tumor", "meningioma_tumor", "no_tumor", "pituitary_tumor"]

# Example: show images with true labels if you have them
show_images_with_predictions(test_data[:6], predictions[:6], class_labels, true_labels=Y_test[:6])

# Example: show images without true labels
# show_images_with_predictions(test_data[:6], predictions[:6], class_labels)
```

After training, the model's performance was evaluated on the unseen testing data. Precision: Report precision for each class (glioma-tumor, meningioma-tumor, no-tumor, pituitary-tumor). Recall (Sensitivity): Report recall for each class. F1-score: Report F1-score for each class. These results will help assess the effectiveness of the brain tumor classification model. High accuracy and balanced performance across all classes are desirable. Low test loss suggests the model generalizes well to unseen data. The classification report (if generated) provides further insights into the model's ability to identify specific tumor types.



IV. Conclusion

This experiment investigated the effectiveness of transfer learning with VGG16 for brain tumor classification. The model achieved a test accuracy of [Test accuracy value obtained from model.evaluate()] and test loss of [Test loss value obtained from model.evaluate()].

Further analysis is recommended to gain a more comprehensive understanding of the model's performance. If a classification report was generated, it would be beneficial to analyze precision, recall, and F1-score for each tumor class to identify potential areas for improvement. Additionally, exploring different hyperparameter configurations or training for a longer duration could potentially enhance the model's accuracy and generalizability.

References

[1] A. Saleh, R. Sukaik, and S. S. Abu-Naser, "Brain tumor classification using deep learning," in *2020 International Conference on Assistive and Rehabilitation Technologies (iCareTech)*. IEEE, 2020, pp. 131–136.

[2] J. S. Paul, A. J. Plassard, B. A. Landman, and D. Fabbri, "Deep learning for brain tumor classification," in *Medical Imaging 2017: Biomedical Applications in Molecular, Structural, and Functional Imaging*, vol. 10137. SPIE, 2017, pp. 253–268.

[3] A. Ari and D. Hanbay, "Deep learning based brain tumor classification and detection system," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 26, no. 5, pp. 2275–2286, 2018.

[4] F. J. Díaz-Pernas, M. Martínez-Zarzuela, M. Antón-Rodríguez, and D. González-Ortega, "A deep learning approach for brain tumor classification and segmentation using a multiscale convolutional neural network," in *Healthcare*, vol. 9, no. 2. MDPI, 2021, p. 153.

[5] G. S. Tandel, M. Biswas, O. G. Kakde, A. Tiwari, H. S. Suri, M. Turk, J. R. Laird, C. K. Asare, A. A. Ankrah, N. Khanna *et al.*, "A review on a deep learning perspective in brain cancer classification," *Cancers*, vol. 11, no. 1, p. 111, 2019.

[1] [2] [3] [4] [5]