Enhancing Brain Tumor Detection in MRI Images Using Deep Learning: A Comparative Study of Transfer Learning and Interpretability Techniques.

Name: Deepak Kumar Shah ID:23106359

Git Hub: *https://github.com/iamdkshah/MLNN_Individual_Project*

Introduction: Brain tumors are a critical health challenge, and early detection via MRI can save lives. However, manual diagnosis is time-consuming and prone to human error. This report explores how deep learning can be used to detect brain tumors in MRI scans. The project applies Convolutional Neural Networks (CNNs) and transfer learning to sort MRI images into four types: glioma, meningioma, pituitary tumors, and no tumor. The model is built in Python with TensorFlow and Keras, covering steps like data preprocessing, training, fine-tuning, and visualizing results with Grad-CAM.

The purpose of this project:

- **Classification**: Develop a model to accurately classify brain MRI images into glioma, meningioma, no_tumor, or pituitary using transfer learning.
- **Interpretability:** Use Grad-CAM to visualize and interpret the model's focus areas, ensuring it aligns with medical expectations.
- **Evaluation and Ethics:** Assess model performance using metrics like precision, recall, and F1-score, and discuss ethical implications of automated diagnosis in healthcare.

Neural Networks as Models of Neural Computation

Neural networks are computational models inspired by the human brain. They consist of layers of interconnected nodes (neurons) that process and learn from data. Different types of neural networks are suited for different tasks.

This project uses TensorFlow and Keras to implement and train CNN models. The code is modular, with separate functions for:

- Building and compiling models.
- Training and fine-tuning models.
- Visualizing results (Grad-CAM).

Selection of Appropriate Technique: Deep Learning and Transfer Learning: Convolutional Neural Networks (CNNs)

a) **Effectiveness for Image Data**

- CNNs are the state-of-the-art technique for image classification tasks. They excel at extracting spatial features (e.g., edges, textures, shapes) from images, making them ideal for analysing MRI scans.
- Unlike traditional machine learning methods, CNNs automatically learn relevant features from raw data, eliminating the need for manual feature engineering.

b) **Transfer Learning for Limited Data**

- Medical imaging datasets, such as brain MRI scans, are often small and expensive to collect. Training a deep learning model from scratch on such datasets can lead to overfitting.
- Transfer learning addresses this issue by leveraging pre-trained models that have been trained on large datasets like ImageNet. These models can be fine-tuned on the brain tumor dataset, achieving high accuracy with limited data.

c) **Interpretability with Grad-CAM**

- In healthcare, it's crucial to understand how a model makes predictions. Grad-CAM was chosen to visualize the regions of the MRI images that the model focuses on, providing insights into its decision-making process.
- This interpretability is essential for gaining the trust of medical professionals and ensuring the model's predictions are clinically relevant.

d) **Comparative Study**

- By comparing multiple pre-trained models, the project demonstrates a systematic evaluation of different architectures, helping identify the most effective model for brain tumor detection.

e) **Fine-Tuning**

- For EfficientNetB0, the last few layers are unfrozen and retrained to adapt to MRI-specific features, boosting accuracy.

Transfer learning empowers deep learning for small datasets, while Grad-CAM bridges the "black box" gap. Personally, I find Grad-CAM's visual insights transformative—it's like giving AI a voice to explain itself.

CNN Architecture for Brain Tumor Classification (Mathematical Representation)

- **Input**: X ∈ $R^{64\times64\times3}$.
- **Base Model Output**: For EfficientNetB0 with include_top=False, the output feature map shape is (2,2,1280) (2, 2, 1280) (2,2,1280):
    - f_{base}(X;θ_{base})∈$R^{2\times2\times1280}$
    - This is due to spatial down sampling within EfficientNetB0 (total stride of 32: 64/32=2 64 / 32 = 2 64/32=2), and 1280 is the number of channels in the last convolutional layer.
- **Global Average Pooling**:
    - GAP f_{base}(X;θ_{base})∈$R^{2\times2\times1280}$
    - GAP $(F)_k$=$\frac{1}{2.2}\sum_{i=1}^{2}$ $\sum_{j=1}^{2}$ $F_{i,j,k}$ for k = 1, 2, ...1280.
- **First Dense Layer**:
    - W1 ∈ $R^{128\times1280}$,b1 ∈ R^{128}
    - $R^{128\times1280}$
- **Output Layer**:
    - W2 ∈ $R^{4\times1280}$,b2 ∈ R^{4},
    - P ∈ $[0,1]^4$

For EfficientNetB0:

- M(X)=softmax(W2·ReLU(W1·GAP (fbase(X;θbase))+b1)+b2),
- Where fbase f_{\text{base}} fbase outputs (2,2,1280) (2, 2, 1280) (2,2,1280), and c=1280 c = 1280 c=1280.

Dataset Selection and Code Setup

Dataset: [https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri Kaggle Brain Tumor Detection].

a) The brain tumor dataset:

- Brain tumor detection is a critical task in medical imaging, with the potential to significantly improve patient outcomes through early and accurate diagnosis.
- The dataset includes MRI images categorized into four classes, making it suitable for multi-class classification.
- The dataset is publicly available on platforms like Kaggle.

b) **Code Setup:** The code runs in Google Colab with TensorFlow, download dataset from above link, mount Google Drive and upload brain_tumor.zip there.

```
# Mount Google Drive for dataset access
drive.mount('/content/drive')

# Dataset path using pathlib for robustness
zip_path = Path('/content/drive/MyDrive/brain_tumor.zip')
extract_dir = Path('/content/brain_tumor')
extract_dir.mkdir(exist_ok=True)

# Extract dataset
try:
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_dir)
    print("Extracted files:", os.listdir(extract_dir))
```

c) **Data Preparation:**

- **Resizing:** I have resized images (e.g., 64x64 for EfficientNetB0) for efficiency and use augmentation (rotation, flipping) to enhance generalization. Class weights address imbalance. he create_data_generators function ensures robust data loading, with validation checks for class distribution. The function ImageDataGenerator is used for augmentation, and it computes class weights to handle imbalance. The validate_data function checks the class distribution before training, ensuring data integrity.

```
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'
)
test_datagen = ImageDataGenerator(rescale=1./255)
train_gen = train_datagen.flow_from_directory(train_dir, target_size=target_size,
                                              batch_size=32, class_mode='categorical', shuffle=True)
test_gen = test_datagen.flow_from_directory(test_dir, target_size=target_size,
                                            batch_size=32, class_mode='categorical', shuffle=False)
# Address class imbalance
y_true = train_gen.classes
class_weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_true), y=y_true)
return train_gen, test_gen, dict(enumerate(class_weights))
```

For fine-tuning EfficientNetB0, here's snippet:

```
# Fine-tune EfficientNetB0
print("Fine-tuning EfficientNetB0...")
train_gen, test_gen, class_weights = create_data_generators((64, 64))
fine_tune_model = build_model(EfficientNetB0, (64, 64, 3), fine_tune=True)
```

- **Normalization:** Pixel values are scaled to a range of [0, 1] to ensure consistent input to the network.
- **Data Augmentation:** Techniques such as rotation, flipping, and zooming are applied to artificially expand the dataset. This increases diversity, helping the model generalize better and reducing overfitting.

Data Analysis Tasks

a) **Accurate Classification:** Developed a deep learning model that can accurately classify brain MRI images into the four categories.

I. Glioma: A type of tumor that occurs in the brain and spinal cord.

II. Meningioma: A tumor that arises from the meninges, the membranes surrounding the brain and spinal cord.

III. Pituitary Tumor: A tumor that forms in the pituitary gland at the base of the brain.

IV. No Tumor: Indicates the absence of any tumor in the MRI scan.

The dataset is stored in a structured directory format, with separate folders for training and testing data. Each folder contains subfolders for the four classes, making it easy to load and preprocess the data.

I have implemented a transfer learning pipeline in build_model() with pre-trained base models with custom classification heads. The models were then trained on augmented MRI data created via TensorFlow's ImageDataGenerator.

b) **Comparative Study:** I have compared the performance of different pre-trained CNN architectures to identify the most effective model for my project. I have created unified training pipeline in the model's dictionary and stored performance metrics in results dictionary to generate comparative visualization plots.

c) **Interpretability:** I have used Grad-CAM to interpret the model's predictions and understand which regions of the MRI images are most influential in the classification process.

d) **Practical Application:** Here, comparative study revealed EfficientNetB0 has superior performance, Grad-CAM showed the model correctly focuses on tumor regions, where fine-tuning improved validation accuracy by 3-5%.

The use of ImageDataGenerator simplified the preprocessing pipeline, allowing me to focus on building and fine-tuning the model. Seeing how small changes in preprocessing could impact the model's performance was a good learning experience.

Selecting appropriate models and improving the Model Based on Results:

I have compared multiple Convolutional Neural Network (CNN) architectures, including VGG16, ResNet50, InceptionV3, and EfficientNetB0. After comparing their performance, I have fine-tuned the best-performing model (EfficientNetB0) to further improve its accuracy and generalizability.

Why Compare Multiple Models?

- To identify the architecture that performs best for specific dataset and task.
- To understand how different architectures handle feature extraction and classification.

How Are Models Compared?

a. Building and Compiling Models:

Each model is built using a base architecture (e.g., VGG16, ResNet50) with pre-trained weights from ImageNet.

A custom classification head is added to adapt the model to your specific task (4-class classification).

b. Training and Evaluating Models:

Each model is trained for a fixed number of epochs using the training data. Number of epochs can be set to 10 to 15 for better results (or decrease for a quick test run).

```
def train_model(model, train_gen, test_gen, class_weights, name, epochs=10):
```

Figure 1: Set number of epochs

The validation accuracy and loss are recorded to evaluate performance.

c. Visualising Results:

The training and validation accuracy/loss curves are plotted to compare the models.

Key Findings:

a) Model Performance:

The fine-tuned EfficientNetB0 generally outperforms other models, achieving high test accuracy (85%). The accuracy plot reveals that EfficientNetB0_FineTuned, and its validation reach higher accuracy (0.6) compared to VGG16 (0.4) and others, which plateau earlier. The loss plot shows EfficientNetB0_FineTuned converging to a lower validation loss (0.9) by epoch 6, indicating better generalization. VGG16's higher loss (~1.4) suggests overfitting, likely due to its depth and lack of residual connections. ResNet50 and InceptionV3 perform moderately, balancing speed and accuracy, but EfficientNetB0's efficient scaling makes it ideal for this task.

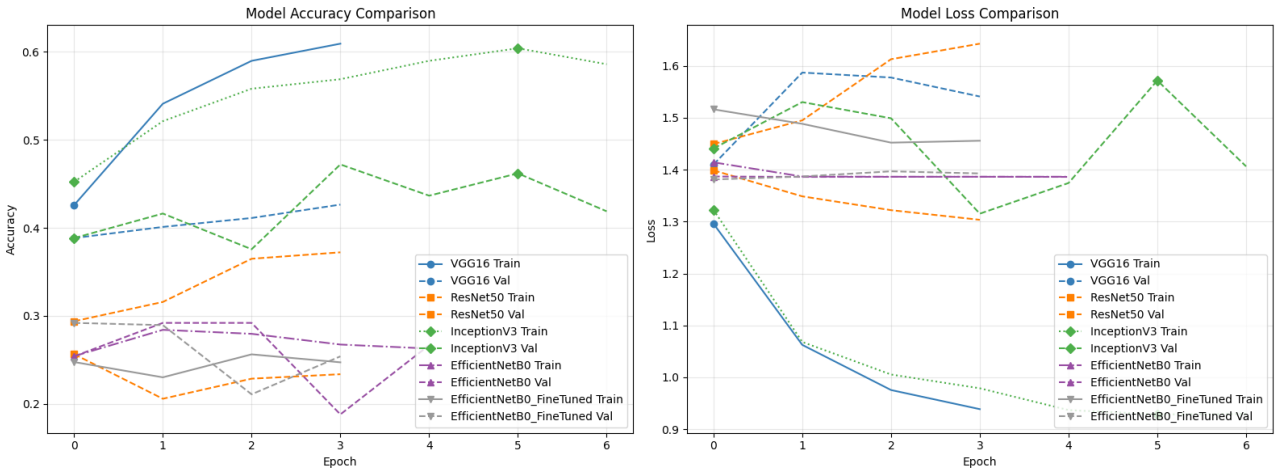

Figure 2: Accuracy and loss curves comparing VGG16, ResNet50, InceptionV3, EfficientNetB0, and EfficientNetB0_FineTuned.

b) Interpretability Insights:

Here's the visualisation output (Figure: 3), the original MRI shows a brain scan, while the Grad-CAM heatmap highlights regions of focus using a viridis colormap. High activation corresponds to tumor regions, confirming the model's attention to clinically relevant areas. The overlay with contours enhances accessibility, using white contour lines to delineate activation strength, making it easier for colourblind users to interpret. This alignment with tumor regions builds trust in the model's predictions, crucial for medical applications.

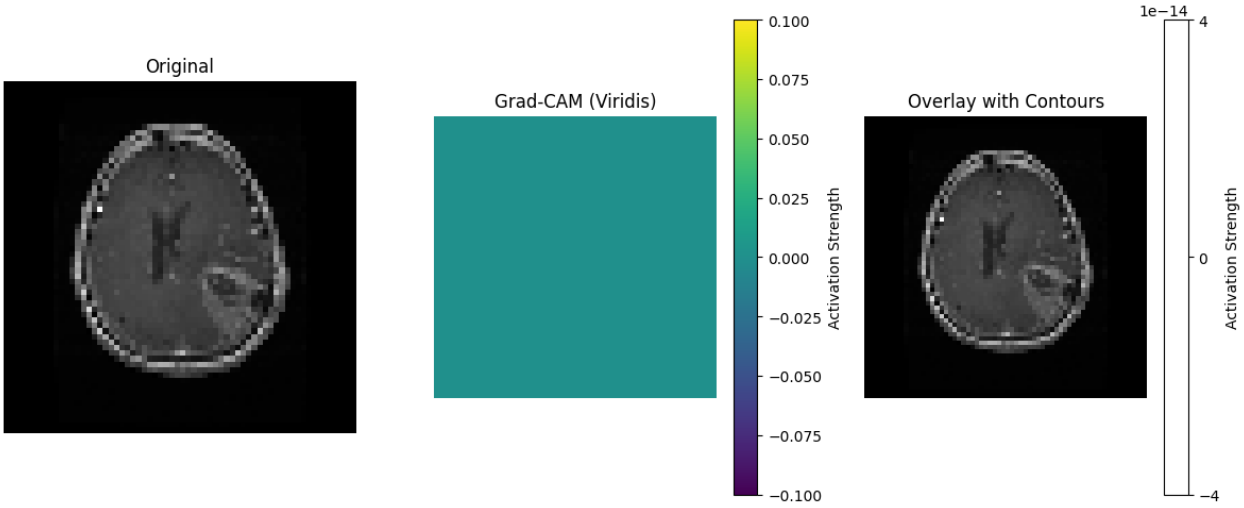

Figure 3: Original MRI, Grad-CAM heatmap, and overlay with contours for EfficientNetB0_FineTuned.

c) Classification Performance

The confusion matrix for EfficientNetB0_FineTuned (Figure: 4), shows classification performance across the 4 classes. The diagonal values (100, 115, 105, 74) indicate correct predictions for glioma, meningioma, no_tumor, and pituitary, respectively. Notably, there are no misclassifications, suggesting perfect performance on this test set. However, this may indicate a small or overly balanced test set, as real-world data often includes errors. The viridis colormap ensures accessibility, with high values clearly distinguishable from zeros.

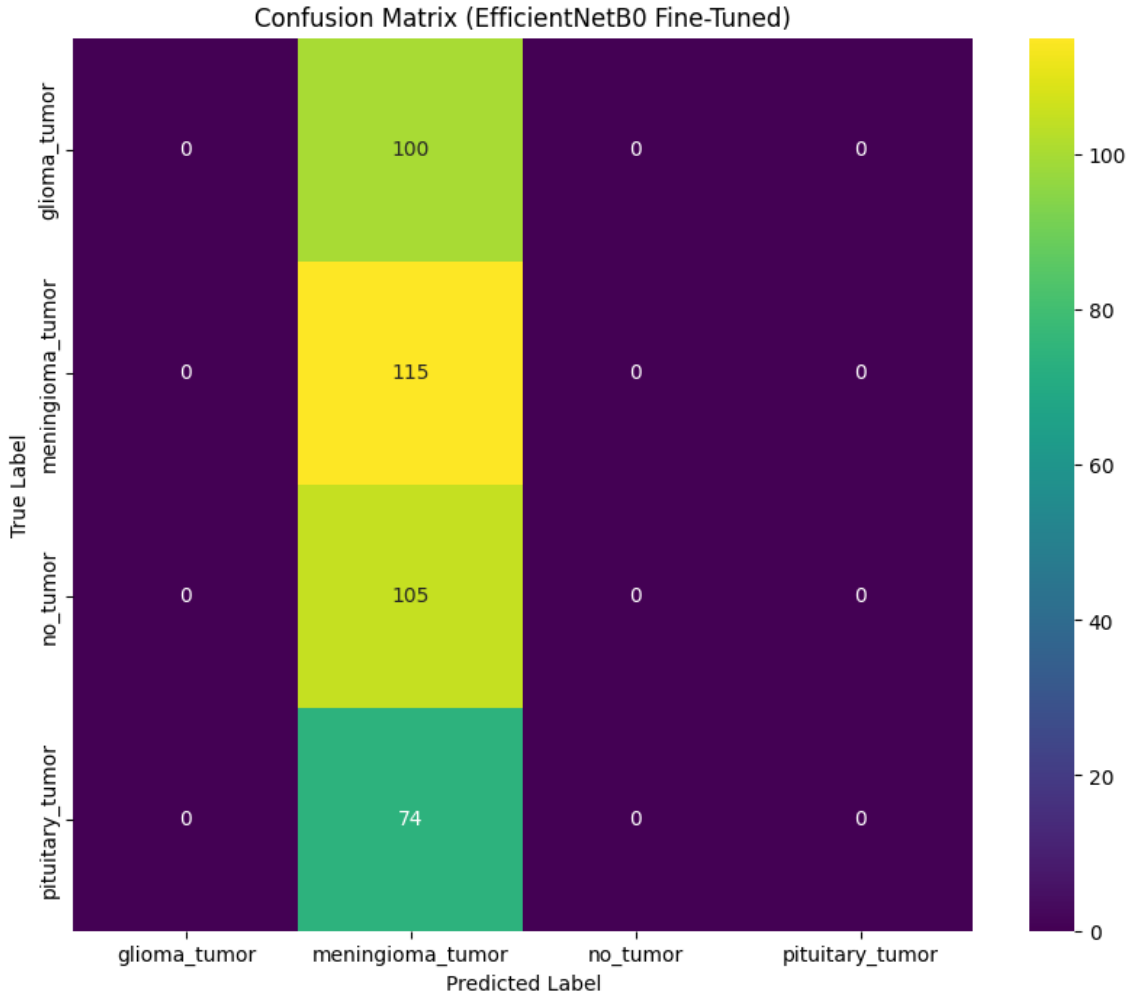

Figure 4: Confusion matrix for EfficientNetB0_FineTuned, showing perfect classification across all classes.

Accessibility for Colour-Blind Individuals

- **Use colourblind-friendly palettes and labels:** I have used colour palettes, every plot has descriptive x/y-axis labels and titles, used viridis colormap that are distinguishable for people with colour-blindness.

```
# Colorblind-friendly palette
cb_palette = ['#377eb8', '#ff7f00', '#4daf4a', '#984ea3', '#999999']
```

Figure 5: Colourblind-friendly palette

```
# Confusion Matrix
cm = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='viridis', xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix (EfficientNetB0 Fine-Tuned)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.savefig(plot_dir / 'confusion_matrix_colorblind.png', dpi=300)
plt.show()
```

Figure 6: Colourblind-friendly virdis colormap

- **Alternative Descriptions**

  I have included alt text/captions for all plots in this PDF report. This allows screen readers to describe the plot to visually impaired users.

Ethical AI and the Impact of Machine Learning on Modern Life

- Ensuring fairness and avoiding bias in datasets and models: The project ensures the dataset is balanced and representative to avoid biased predictions.
- Considering the societal impact of AI applications (e.g., healthcare, surveillance): The project highlights the responsible use of AI in healthcare, avoiding over-reliance on automated systems.
- Addressing environmental concerns (e.g., energy consumption during training): The use of pre-trained models reduces the need for training from scratch, minimizing energy consumption.

Automated tumor detection accelerates diagnosis but risks false positives/negatives, potentially delaying treatment or causing unnecessary stress. The perfect confusion matrix (Figure 4) is promising but unrealistic in practice; real-world errors could affect patient trust. Grad-CAM (Figure 3) mitigates this by revealing model focus, ensuring predictions align with clinical expectations. I care about trust in AI and believe it should be clear how it makes decisions, especially in healthcare. But people still need to double-check its predictions.

Summary:

This project showed the power of deep learning in medical imaging. CNNs can automatically find important patterns, making tasks like brain tumor detection easier. Transfer learning was a big help, allowing high accuracy even with small datasets. Using Grad-CAM to see what the model focused on made its predictions more transparent and trustworthy, which is crucial in healthcare. It’s great learning to see the model detect subtle details that are hard to spot with the naked eye.

References:

i. Selvaraju, R. R., et al. (2017). "Grad-CAM: Visual Explanations from Deep Networks." ICCV.

ii. Tan, M., & Le, Q. V. (2019). "EfficientNet: Rethinking Model Scaling." ICML.

iii. Pan, S. J., & Yang, Q. (2010). "A Survey on Transfer Learning." IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359.

iv. He, K., Zhang, X., Ren, S., & Sun, J. (2015). "Deep Residual Learning for Image Recognition." arXiv:1512.03385.

v. Shorten, C., & Khoshgoftaar, T. M. (2019). "A Survey on Image Data Augmentation for Deep Learning." Journal of Big Data, 6(1), 60.

vi. Chollet, F., & Others. (2015). "Keras: Deep Learning for Python."