UNIVERSITY OF WOLVERHAMPTON

Artificial Intelligence and Machine Learning

(6CS012)

Brain Tumor Classification Using CNNs and Transfer Learning with VGG16

Assessment Report I

Full name : Ananda Neupane

Student ID : 2329810

Group Name : L6CG2

Submission date : March 10, 2025

Tutor : Shiv Kumar yadav

Abstract

In this project, I work over the problem of brain tumor classification using DL techniques. I aim to build a model that can accurately distinguish brain MRI images into four categories: glioma, meningioma, pituitary tumor and normal. Since early identification of brain tumors is essential for patient outcomes, this task has significant real-world value.

To tackle this, I start by developing two custom convolutional neural networks (CNNs): a simple baseline model and a deeper version with regularization techniques. I also apply data preprocessing, augmentation, and use both the Adam and SGD optimizers for comparison. After experimenting with these architectures, I move on to transfer learning by integrating the VGG16 model pre-trained on ImageNet. I first use it as a feature extractor and then fine-tune its top layers to better adapt it to my dataset.

The fine-tuned VGG16 model performs the best, achieving 86% accuracy with balanced precision and recall across all tumor classes. This is a significant improvement compared to models trained from scratch. My results show that transfer learning greatly improves classification performance and helps overcome limitations related to small and imbalanced medical datasets.

In general, this work represents the practical value of combining deep CNNs with transfer learning to build robust medical imaging solutions.

Table of Contents

Table of Figures

1. Introduction

This project includes distinguishing tumors from MRI images using DL. Different tumors like glioma tumor, meningioma tumor, and pituitary can be hard to identify, and error during diagnosis effect of being costly. Main challenges that we encounter is that manual interpretation of scans isn't always consistent, especially in places with fewer trained radiologists. That's where I see the potential of AI in assisting with identification.

I plan to develop Convolutional Neural Networks (CNNs) because they're good at finding patterns in images without needing much hand-crafted preprocessing. They've already been applied in fields like face recognition and self-driving cars, but also in healthcare for example, in spotting pneumonia or classifying skin conditions. It makes sense to apply the same principles here, with the goal of helping to detect different types of brain tumors.

While some past work uses pre-trained networks like ResNet or VGG, I want to test both approaches: building my own model from scratch, and then also fine-tuning a well-known pre-trained model. I'm curious to see how much deeper model or a different optimizer changes performance, and whether transfer learning really gives an edge when working with a smaller dataset.

I use an MRI dataset that includes four categories: glioma, meningioma, pituitary tumor, and normal brain scans. My plan is to compare multiple models, track training times and accuracy, and explore which combinations of architecture and training method give the best results. In the end, I want to learn not just what works best, but why—and how these models might one day support real clinical decisions.

2. Dataset

1. Source

The dataset is publicly available brain MRI dataset sourced from Kaggle. The dataset is specifically design for brain tumor classification tasks and includes images label into four categories: glioma tumor, meningioma tumor, pituitary tumor, and normal. These classes

represent some of the most identify conditions found in clinical brain MRI scans. I received this customize data from my teacher to work on an assignment project.

2. Shape and Size

The training data set contains 2,475 images, and test data set includes 621 images. During preprocessing, I find that 28 images in the training set are either corrupted or unreadable. I remove these images, so I continue with 2,447 clean training images, while the test set stays the same. The image contains are of pixels value 256*256.

3. Class Distribution

The dataset is mostly balanced across the tumor categories, but the "normal" class has fewer images. The tables below show the class distribution before and after cleaning:

Training Set (Before Cleaning)

Classes	Count
Glioma Tumor	720
Meningioma Tumor	730
Pituitary Tumor	675
Normal	350
Total	2,475

Training Set (After Cleaning - 28 corrupt images removed)

Classes	Count
Glioma Tumor	713
Meningioma Tumor	723
Pituitary Tumor	688
Normal	343
Total	2,447

Test Sets

Classes	Count
Glioma Tumor	181
Meningioma Tumor	183
Pituitary Tumor	169
Normal	88
Total	621

4. Preprocessing

To start, I make sure the dataset is clean by checking for corrupted or unreadable images. I write a small script using Python's PIL library to open and verify each image in both the training and test folders. If an image fails to load, I remove it from the dataset. This step helps prevent unexpected errors during model training. In total, I found 28 corrupted images and removed them from the training set.

Once I finish removing the corrupt image, I proceed with resizing all images to 150×150 pixels dimensions as I need to make them consistent for model input. Moving on, I normalize the pixel into range of [0, 1] by dividing by 255. This helps the models train faster and more reliably.

For training, I use Keras' ImageDataGenerator to apply real-time data augmentation. This includes random rotation, zoom, horizontal flips, and small shifts in width and height. These techniques improve the model's generalization by exposing it to more variations of the training images. I also split 20% of the training for validation using the same generator, but without augmentation.

For the test set, I only apply normalization. I use a separate generator with rescale=1./255 and avoid shuffling to keep the test order consistent during evaluation.

In cases where I need to use pre-trained models i.e. VGG, which expect 224×224 input dimensions, I prepare an additional test generator that resizes images accordingly but keeps all other preprocessing steps the same.

3. Methodology

In this project, I use two different CNN architectures: a baseline model that I build from scratch and a deeper version that introduces regularization techniques. Both models are trained and evaluated using the same dataset and workflow to ensure fair comparison.

3.1. Baseline Model Architecture

Conv2D is a convolutional layer that scans an image with filters to extract visual features. MaxPooling2D decrease the spatial size of feature maps, helping the model generalize

better and speed up training. Firstly, Flatten layer transform 2D matrices into a 1D vector and send into dense layers. Dense layers like fully connected layers are responsible for the last final classification decisions, while the Softmax output layer assigns a probability to each class as we have 4 different classes.

The baseline CNN model includes of three convolutional layers, then max pooling, and lastly three FCN leading to a softmax output. Here's how the structure looks:

1. Conv2D (32 filters) → ReLU → MaxPooling2D
2. Conv2D (64 filters) → ReLU → MaxPooling2D
3. Conv2D (128 filters) → ReLU → MaxPooling2D
4. Flatten
5. Dense (128 units) → ReLU
6. Dense (64 units) → ReLU
7. Dense (32 units) → ReLU
8. Output layer: Dense (4 units, SoftMax)

This model is compiled using the Adam optimizer, with categorical cross-entropy as the loss function. I train the model for up to 50 epochs using early stopping (patience = 4) and model checkpointing based on validation loss.

- Input shape: (150, 150, 3)
- Total parameters: ~4.8 million
- Training time: ~2.81 minutes

From the training plots, I observe that the model achieves about 71% test accuracy, with an F1-score around 0.70 (weighted). The training accuracy increases steadily, while validation accuracy fluctuates. This may suggest slight overfitting as training progresses. The figure below displays the accuracy and loss trends over the training epochs for the baseline model.

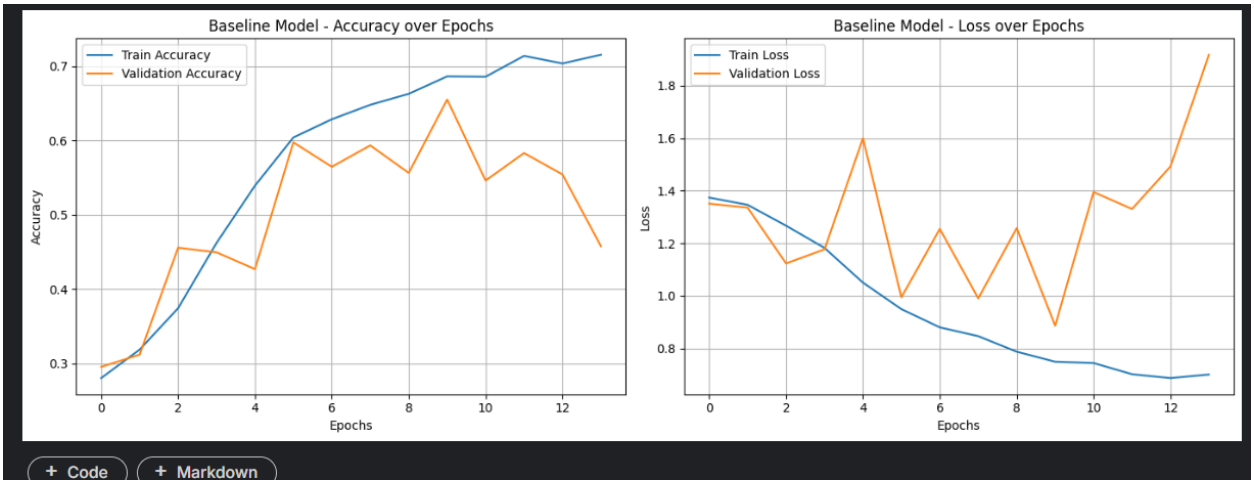

Figure 1: Baseline CNN model - Training vs Validation Accuracy

Classification Report (Test Set):

1. "Glioma Tumor": Precision → 0.91, Recall → 0.53
2. "Meningioma Tumor": Precision → 0.60, Recall → 0.61
3. "Pituitary Tumor": Precision → 0.70, Recall → 0.95
4. Normal: Precision → 0.72, Recall → 0.83

3.2. Deeper CNN with Regularization

To enhance the model, I add Batch Normalization layers after each convolution. These normalize the inputs to each layer to improve stability and convergence. I also apply Dropout after fully connected layers, which randomly disables those neurons during training phase to prevent overfitting and improve data generalization.

I build a deeper CNN by adding an additional convolutional layer and inserting Batch Normalization after each convolution. I also introduce Dropout layers after the fully connected layers to reduce overfitting.

1. 4 Conv2D layers with increasing filters: 32 → 64 → 128 → 256
2. BatchNormalization after each Conv2D
3. MaxPooling2D after each block

4. Flatten
5. Dense (256) → Dropout(0.5)
6. Dense (128) → Dropout(0.3)
7. Output layer: Dense(4, Softmax)

This model is compiled using the Adam optimizer, with categorical cross-entropy as the loss function. I train the model for up to 50 epochs applying with early stopping (patience = 4) and model checkpointing based on validation loss.

- Input shape: (150, 150, 3)
- Total parameters: ~3.6 million
- Training time: ~2.85 minutes

The deeper model performs worse than expected. It reaches 51% test accuracy with unstable validation loss curves. Although regularization is included, the model appears to struggle with generalization.

The diagram below illustrates the training and validation curves for the deeper model, highlighting its unstable learning behavior.

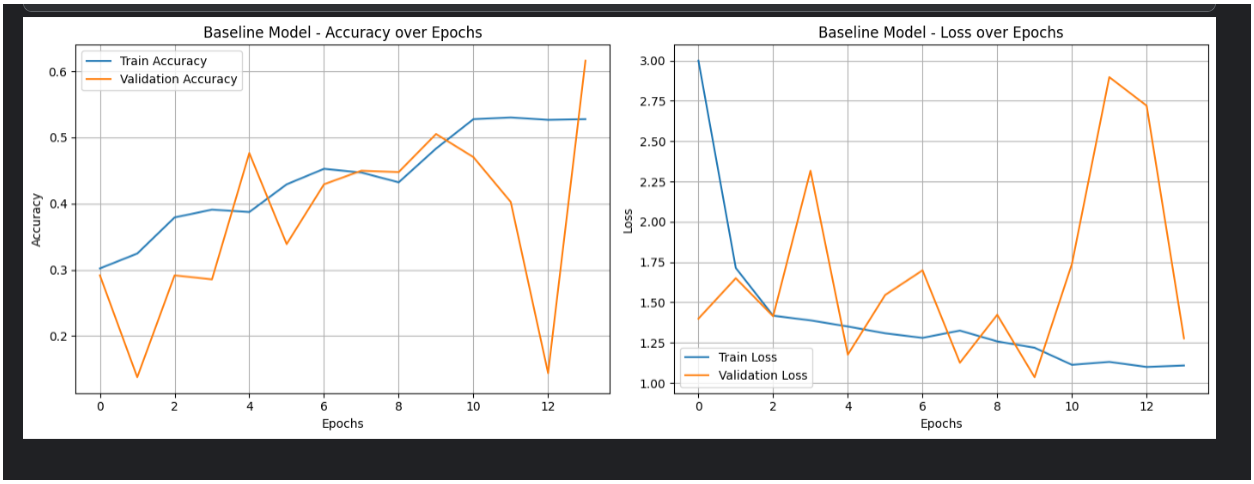

Figure 2: Deeper CNN (Adam/SGD) Training vs Validation Accuracy and Loss

Classification Report (Test Set):

1. “Glioma Tumor”: Precision → 1.00, Recall → 0.07
2. “Meningioma Tumor”: Precision → 0.39, Recall → 0.70
3. “Pituitary Tumor”: Precision → 0.69, Recall → 0.69
4. Normal: Precision → 0.53, Recall → 0.65

These results suggest that deeper architecture alone does not guarantee better performance and might require more fine tuning or balanced data to improve.

Hyperparameters Used

I compile both models using categorical cross-entropy, which is ideal for multiple class classification problems. The optimizer I choose is Adam, an adaptive method that adapts learning rates during training. To avoid overfitting, I use EarlyStopping, which halts training if the validation loss don’t show improve for a set number of epochs, and ModelCheckpoint to save only the best-performing model.

1. Optimizer: Adam
2. Loss Function: Categorical Cross-Entropy
3. Batch Size: 32
4. Epochs: 50 (early stopping applied)
5. Input Size: 150x150 (RGB)
6. Optimizer: Adam
7. Loss Function: Categorical Cross-Entropy
8. Batch Size: 32
9. Epochs: 50 (early stopping applied)
10. Input Size: 150x150 (RGB)

4. Experiments and Results

4.1. Baseline vs. Deeper Architecture

To compare performance, I analyze all of training time, validation accuracy and loss curves for both the baseline and deeper models. The baseline model achieves better overall validation accuracy and exhibits more stable loss behavior.

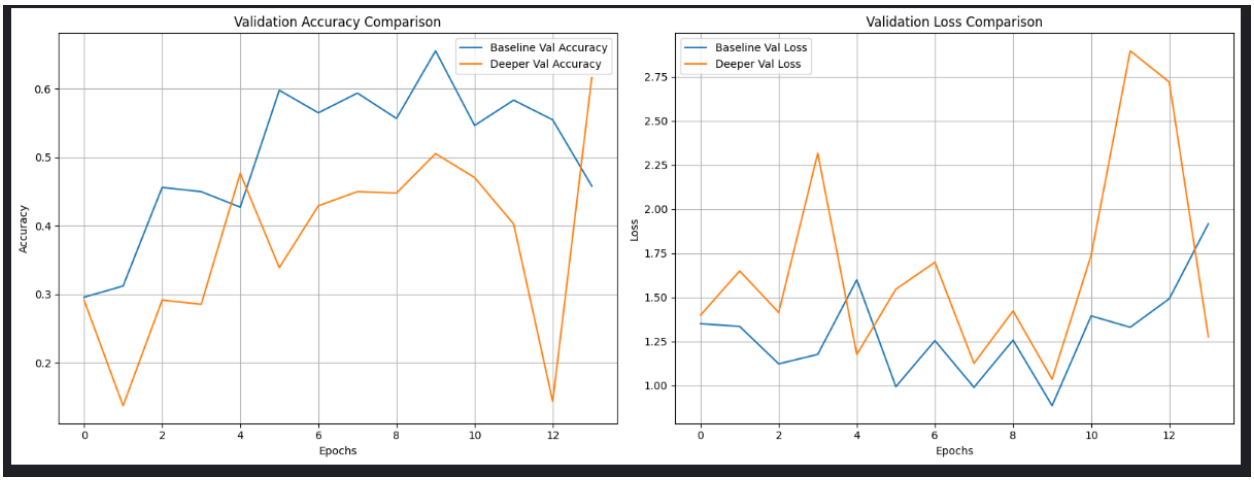

Figure 3: Accuracy and Validation Loss

The baseline reaches a validation accuracy of around 65%, while the deeper model shows unstable accuracy and higher loss values, despite having more layers and regularization.

Model	Optimizer	Accuracy	Training Time	Parameters	F1-Score (Weighted)
Baseline CNN	Adam	71%	2.81 minutes	~4.8M	0.70
Deeper CNN	Adam	51%	2.85 minutes	~3.6M	0.45

4.2. Computational Efficiency

I train both models using Kaggle TGPU acceleration. The training times are fairly close:

1. Baseline Model (Adam): ~2.81 minutes
2. Deeper Model (Adam): ~2.85 minutes

While the deeper model has more layers, regularization and batch normalization reduce training instability. However, it does not offer an efficiency advantage over the baseline.

4.3. Training with Different Optimizers

To examine the effect of optimization algorithms, I retrain the deeper model using SGD with momentum instead of Adam.

- Deeper Model (SGD): ~2.42 minutes

SGD is a traditional optimization method that adapts model weights based on each batch and uses momentum to carry forward updates, which can lead to better generalization. In contrast, Adam updates the learning rate to single parameter individually and often converges faster but can sometimes overfit on small datasets.

With SGD, the validation accuracy of the deeper model improves slightly compared to Adam. The F1-score increases from 0.45 to 0.53, and the performance becomes more balanced across classes. This shows that even though Adam performs better on training accuracy, SGD can sometimes lead to better generalization.

One thing I notice is that the model with SGD reaches its optimal performance earlier, suggesting quicker convergence in this case. However, early performance gains also come with increased fluctuations in training stability, which highlights the importance of learning rate tuning when using SGD.

4.4. Classification Report (Deeper CNN with SGD):

“Glioma Tumor”: Precision → 0.82, Recall → 0.40

“Meningioma Tumor”: Precision → 0.45, Recall → 0.69

“Pituitary Tumor”: Precision → 0.57, Recall → 0.82

Normal: Precision → 0.82, Recall → 0.10

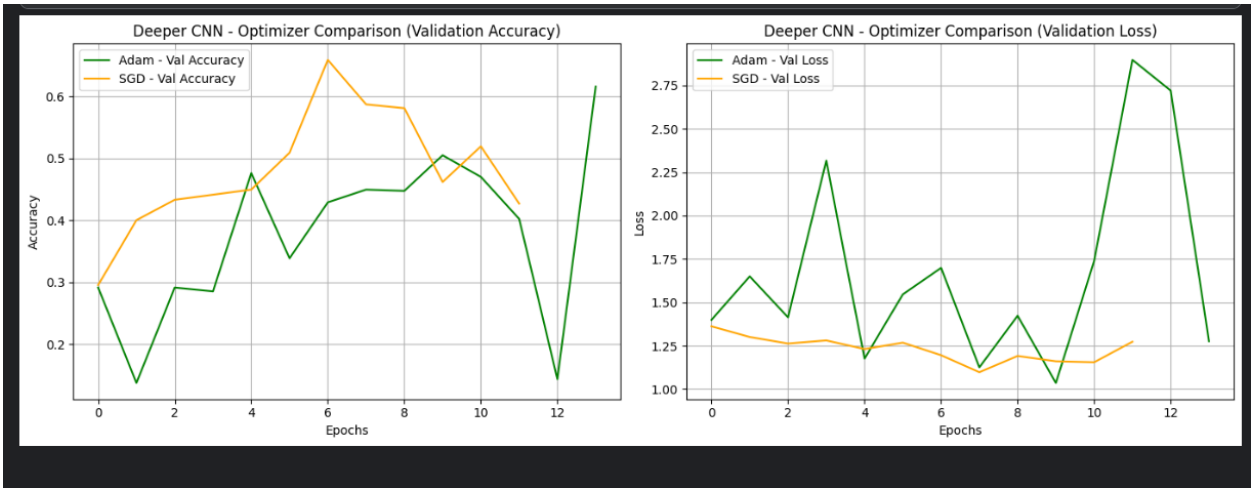

Figure 4: Accuracy and Validation of Deeper CNN

The model trained with SGD displays slightly faster convergence and better generalization in some classes, but Adam remains more stable overall, particularly for complex datasets with uneven class distributions.

4.5. Challenges in Training

Throughout training, I encounter several common deep learning challenges:

1. Overfitting in deeper models, especially without tuning learning rates or dropout ratios. This is seen when training accuracy improves while validation performance remains poor or degrades.

2. Unstable validation loss curves even with regularization such as dropout and batch normalization. This indicates that deeper models may require more fine-tuned hyperparameters or more training epochs to converge properly.

3. Data imbalance, particularly affecting the "normal" class with fewer samples, causes lower recall and precision compared to tumor classes.

4. Hyperparameter Sensitivity: Both models are sensitive to choices like learning rate, dropout rates, and optimizer settings. Slight changes can lead to drastically different training behaviors.

5. Evaluation Bias: Relying on accuracy alone would be misleading due to the class imbalance. That's why I rely more heavily on F1-score and class-wise precision/recall.

Using callbacks like EarlyStopping and ModelCheckpoint helps mitigate some of these issues. Also, training with SGD provides alternative convergence behavior, though not always superior to Adam. Regular experimentation and tracking of multiple metrics are crucial to guiding model development and debugging issues effectively.

5. Fine-Tuning or Transfer Learning

Finally, for improving performance and reducing training time, I use transfer learning technique with the VGG16 model, a re-known pre-trained CNN architecture originally trained over ImageNet. I load the model without the top layers and freeze all its convolutional layers to utilize it as a fixed feature extractor. Then, I assign custom classification head consisting of a Flatten layer, a 'Dense layer' with 256 units and ReLU activation, next a Dropout layer (rate = 0.5), and a final Dense output layer with actication softmax for four-class classification.

All input images are resized to 224×224 pixels to align VGG16's expected input size. I use Keras' ImageDataGenerator for data preprocessing and augmentation.

5.1. Feature Extraction Phase (Frozen Layers)

During initial training, I keep all VGG16 layers frozen and train only the custom head. The model performs significantly better than my custom CNNs:

Classification Report (Feature Extraction Only):

- "Glioma Tumor": Precision → 0.76, Recall → 0.86
- "Meningioma Tumor": Precision → 0.80, Recall → 0.51
- "Pituitary Tumor": Precision → 0.78, Recall → 0.96
- Normal: Precision → 0.84, Recall → 0.85

5.2. Fine-Tuning Phase

After the initial training, I unfreeze the last few layers of VGG16 (specifically the last 8 layers, excluding BatchNormalization layers) to enable the model to fine-tune high-level feature representations. This selective unfreezing allows adaptation to the brain tumor dataset while preserving the valuable weights learned from ImageNet.

Unfrozen Layers in Fine-Tuning:

1. block5_conv1
2. block5_conv2
3. block5_conv3
4. flatten
5. dense_12
6. dropout_5
7. dense_13

I compile the model using the Adam optimizer with its learning rate value to 1e-5 and train it again using both early stopping and model checkpoint.

Model Summary After Fine-Tuning:

- Trainable Parameters: ~13.5M
- Non-trainable Parameters: ~7.6M
- Total Parameters: ~21.1M

Classification Report (Fine-Tuned VGG16):

- “Glioma Tumor”: Precision → 0.93, Recall =→0.78, F1-Score →0.85
- “Meningioma Tumor”: Precision → 0.76, Recall → 0.81, F1-Score → 0.79
- “Pituitary Tumor”: Precision → 0.89, Recall → 0.96, F1-Score → 0.92

- Normal: Precision → 0.88, Recall → 0.91, F1-Score → 0.89

Fine-tuning improves performance and further stabilizes training. Compared to models trained from scratch, the VGG16-based model achieves:

Model	Accuracy	Precision (Avg)	Recall (Avg)	F1-Score (Avg)
Baseline CNN	71%	0.73	0.71	0.70
Deeper CNN (Adam)	51%	0.65	0.53	0.45
VGG16 (Frozen)	78%	0.79	0.80	0.78
VGG16 (Fine-Tuned)	86%	0.86	0.87	0.86

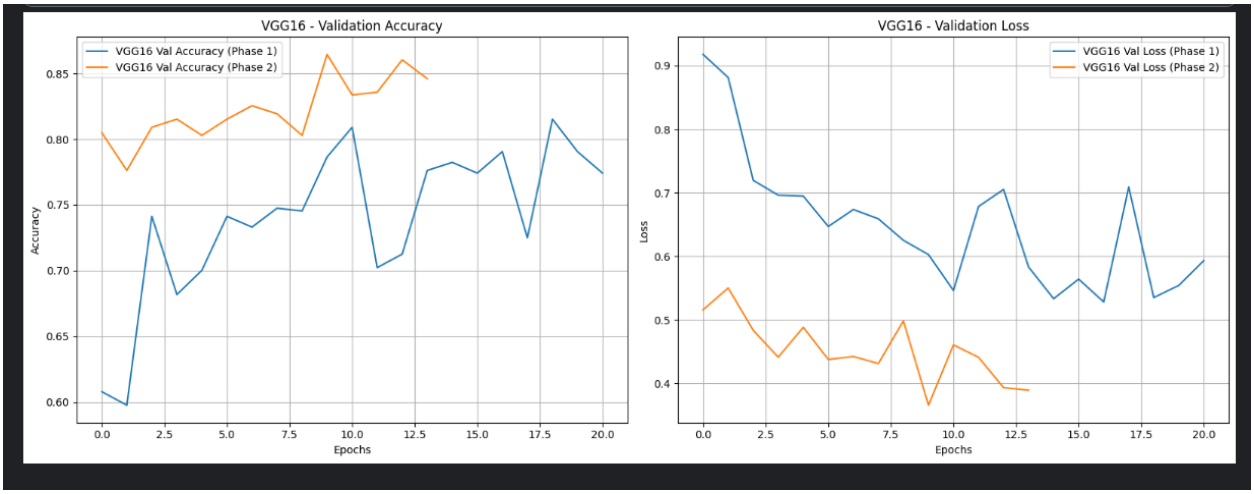

Figure 5: VGG16 Validation Accuracy and Loss Curves

Transfer learning not only outperforms the models built from scratch but also requires less training effort. The validation accuracy improves significantly after fine-tuning, as shown in Figure 6. Initially, accuracy plateaus around 75–78%, but once I unfreeze part of the base model and lower the learning rate, the model reaches up to 86% validation accuracy with reduced loss.

This clearly shows how fine-tuning leverages prior knowledge while adapting to the specific dataset, making it a highly effective approach when dealing with limited data or

imbalanced classes. but also requires less training effort. The use of pre-trained features greatly benefits generalization, especially when working with smaller datasets.

6. Conclusion and Future Work

In concluding, I explore multiple convolutional neural network architectures—from custom-built CNNs to pre-trained VGG16—for classifying brain MRI images into four categories. My experiments show that deeper architecture alone do not guarantee better performance unless properly regularized and tuned. While the baseline CNN performs reasonably well with 71% accuracy, the deeper CNN struggles to generalize, even when using dropout and batch normalization.

The introduction of transfer learning through VGG16 significantly boosts performance. By first using feature extraction and then fine-tuning the deeper layers, I get the best accuracy of 86%, with best precision and recall score across all classes. This reinforces the value of leveraging pre-trained models when working with medical datasets, which often have limited samples and high-class imbalance.

6.1. Key Takeaways

1. Simple, well-regularized models often outperform deeper ones when data is limited.
2. Transfer learning offers a powerful solution for medical image classification.
3. Optimizer choice and training strategy (SGD vs. Adam) can significantly influence performance.
4. Monitoring multiple metrics like precision, recall, and F1-score is essential due to data imbalance.

6.2. Future Work

1. I plan to explore other advanced pre-trained models such as ResNet or EfficientNet.

2. Adding more data or applying advanced augmentation techniques could further improve model robustness.
3. Implementing explainability tools like Grad-CAM can provide interpretability for clinical use.
4. Hyperparameter optimization using tools like Keras Tuner or Optuna might yield better results.

Overall, this project demonstrates how deep learning, especially when combined with transfer learning, can offer accurate and scalable solutions for medical image classification.