UNVEILING BRAIN TUMORS: AN APPROACH USING GRAPH THEORY IN MEDICAL IMAGING

Project report submitted to the Amrita Vishwa Vidyapeetham in partial fulfilment of the requirement for the Degree of

BACHELOR of TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

NIHAAS REDDY RAVULA- AM.EN.U4CSE20157
DHEERAJ REDDY B – AM.EN.U4CSE20115
AISWARYA SUDHIR – AM.EN.U4CSE20106
KARTHEEK REDDY T– AM.EN.U4CSE20167

AMRITA SCHOOL OF COMPUTING

AMRITA VISHWA VIDYAPEETHAM
(Estd. U/S 3 of the UGC Act 1956)
AMRITAPURI CAMPUS

KOLLAM -690525

MAY 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM
(Estd. U/S 3 of the UGC Act 1956)

Amritapuri Campus

Kollam -690525

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Unveiling Brain Tumors: An Approach Using Graph Theory in Medical Imaging "** submitted by **NIHAAS REDDY RAVULA (AM.EN.U4CSE20157),DHEERAJ REDDY (AM.EN.U4CSE20115),AISWARYA SUDHIR (AM.EN.U4CSE20106),KARTHEEK REDDY T (AM.EN.U4CSE20167)**, in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Computer Science and Engineering from Amrita Vishwa Vidyapeetham, is a bonafide record of the work carried out by them under my guidance and supervision at Amrita School of Computing, Amritapuri during Semester 8 of the academic year 2023-2024.

Ms Asha Ashok
Project Guide

Mr Sarath S
Project Coordinator

Dr. Swaminathan J
Chairperson
Dept. of Computer Science & Engineering

Ms Shilpa C Vijayan
Reviewer

Place : Amritapuri
Date : 10th May 2024

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

(Estd. U/S 3 of the UGC Act 1956)

Amritapuri Campus

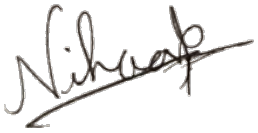
Kollam -690525

DECLARATION

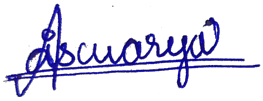
We,**NIHAAS REDDY RAVULA (AM.EN.U4CSE20157), DHEERAJ REDDY B (AM.EN.U4CSE20115), AISWARYA SUDHIR (AM.EN.U4CSE20106), KARTHEEK REDDY T (AM.EN.U4CSE20167)** hereby declare that this project entitled **"Unveiling Brain Tumors: An Approach Using Graph Theory in Medical Imaging"** is a record of the original work done by us under the guidance of **Ms Asha Ashok**, Dept. of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, that this work has not formed the basis for any degree/diploma/associationship/fellowship or similar awards to any candidate in any university to the best of our knowledge.

Place : Amritapuri

Date : 10th May 2024



T. Karthick vishal Reddy



Signature of the student

Signature of the Project Guide

Acknowledgements

First and foremost, we express our heartfelt gratitude to Mata Amritanan- damayi Devi (Amma) for her divine blessings and constant inspiration. Amma's compassionate guidance and spiritual presence have been a source of strength and motivation throughout our academic and research endeavors. We extend our sincere thanks to our esteemed guide, Ms Asha Ashok, from the Department of Computer Science and Engineering at Amrita Vishwa Vidyapeetham, Amritapuri campus.Her insightful feedback and constructive guidance have significantly enriched our work. Additionally, we would like to acknowledge and thank the entire Department of Computer Science and Engineering at Amrita Vishwa Vidyapeetham, Amritapuri campus, for providing us with an excellent academic environment and research facilities. The department's commitment to excellence in education and research has been instrumental in our growth and development. We are also grateful to our families, friends, and research participants for their encouragement, patience, and support throughout this journey. This project would not have been possible without the blessings, guidance, and contributions from all these individuals and entities. We are deeply appreciative of their invaluable assistance and encouragement.

Abstract

This research provides a novel method for classifying brain tumours in MRI images that combines graph neural networks (GNN) with convolutional neural networks (CNN). Pre-processed MRI images are translated into graphical representations to capture spatial dependencies, and feature extraction is facilitated by transfer learning with pre-trained CNN architectures. The core model employs a light-weighted network (GAT) to reveal information in network connections, resulting in a valuable learning experience for tumour classification. GAT employs a tracking strategy to choose input from neighbouring individuals, hence enhancing the model's capacity to detect correlations in visual patterns. To better comprehend data features and representations, use qualitative data analysis and visualisation tools. Three versions of this model were created and tested, each based on a different previous CNN design. Overall, the hybrid method shows good potential for brain tumor classification in MRI scans.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

The importance of early discovery in brain tumour instances cannot be emphasised, since it is critical to successful treatment and a better patient prognosis. This necessity has accelerated the development of computer-assisted detection systems, which provide healthcare workers with sophisticated diagnostic capabilities. One such innovation is the use of graph theory, a powerful approach for analysing the intricate interconnections and patterns found in medical pictures, particularly MRI and CT scans of the brain.

Brain tumours appear as anomalies in the brain's complex network of connections. Despite their importance, conventional image processing approaches may overlook subtle alterations that indicate the existence of small tumours. Graph-based techniques solve this problem by capturing both local changes and the larger context surrounding possible tumours, resulting in more reliable identification.

1.2 Objective

This research suggests the successful use of graph-based analysis in detecting brain tumours has far-reaching consequences. It has the potential to accelerate advances in disease modelling and computer-aided diagnostics for a wide range of medical disorders. This research not only intends to expand the understanding of medical image processing using graph theory, but also to encourage expertise in deep learning and graph algorithms, which are becoming increasingly essential in the sectors of healthcare and medical technology.

1.3 Motivation

The motivation for this research extends beyond the immediate clinical benefits. Brain tumours, whether malignant or benign, place tremendous pressure on the skull, resulting in severe symptoms such as headaches and dizziness. Therefore, early identification is critical to prevent these problems and improve the patient's quality of life.

Chapter 2

Problem Definition

2.1 Existing System

In the realm of brain tumor classification, a diverse array of methodologies has been explored to enhance diagnostic accuracy and streamline the characterization process. Traditional machine learning models, including Support Vector Machines (SVM), Random Forests, and Decision Trees, have long been stalwarts in the field, leveraging handcrafted features derived from medical imaging data to discern tumor types. Concurrently, deep learning paradigms, notably Convolutional Neural Networks (CNNs), have surged to the forefront, demonstrating prowess in automatic hierarchical feature extraction from raw input data. Architectures like VGG, ResNet, DenseNet, and EfficientNet have been harnessed to varying degrees of success, showcasing the adaptability and efficacy of deep learning in this domain.

Ensemble methods have emerged as formidable contenders, amalgamating multiple models to bolster predictive performance and fortify model robustness. Transfer learning, a cornerstone in contemporary machine learning,

has found fertile ground in brain tumor classification, leveraging pre-trained models on extensive datasets to expedite model training and enhance classification accuracy. Hybrid models, adept at fusing disparate neural network architectures or combining deep learning with conventional machine learning techniques, exemplify the synergistic potential of integrating diverse methodologies.

Moreover, computer-aided diagnosis (CAD) systems have materialized as indispensable allies to radiologists and clinicians, furnishing automated analyses and interpretations of medical imaging data. These systems, underpinned by machine learning or deep learning algorithms, facilitate tumor detection, segmentation, and classification, augmenting diagnostic capabilities and expediting clinical workflows. Complemented by a suite of software tools and platforms tailored for data preprocessing, model development, and result visualization, the landscape of brain tumor classification continues to evolve, propelled by the relentless march of innovation and technological advancement.

2.2 Problem Definition

The problem revolves around developing an accurate and efficient brain tumor classification system using MRI images. The challenge lies in leveraging graph neural networks (GNN) and convolutional neural networks (CNN) to effectively model spatial information, capture complex tumor features, and differentiate between different tumor classes.

Chapter 3

Related Work

3.1 Related Work

There has been increasing interest in the development of hybrid approaches combining deep networks with other computing methods. Alzubaidi et al. [7] introduced MedNet, a pre-trained convolutional neural network model specifically designed for medical imaging tasks, including brain tumor classification. These types of hybrid approaches improve accuracy by integrating features and strategies. However, they may also introduce new complexities and problems in model training and deployment. Many challenges arise in understanding the mixed contributions of different components

Researchers have also explored integrating deep learning techniques with emerging technologies like the Internet of Things (IoT) for brain tumor detection and classification. Krishnan et al. [3] expressed a cnn-based multi-class classification approach which offers automated diagnosis but also being prone to overfitting. A 3D scaffold-based model for glioma stem cell was proposed by Jeena et al. [4] providing platform for drug testing.

Furthermore, researchers have focused on enhancing the robustness and

generalization ability of deep learning models for brain tumor classification through techniques such as data augmentation and adversarial training. Dehkordi et al. [9] proposed an evolutionary convolutional neural network approach that incorporates genetic algorithms to optimize model parameters and improve classification accuracy. Additionally, Sravani et al.[2] Ghassemi et al. [10] explored the use of generative adversarial networks (GANs) for pre-training deep neural networks on MRI data, aiming to enhance the model's capacity to capture complex patterns and variations in brain tumor images. While these techniques show promise in improving classification performance, they also necessitate careful testing to ensure their reliability and effectiveness across different patient populations and imaging protocols.

Ongoing research efforts have been directed towards addressing the challenges associated with the understanding of deep learning models for brain tumor classification. Likewise, Ge et al. [11] investigated the use of pairwise generative adversarial networks (GANs) to expand the training dataset and improve the robustness of molecular-based brain tumor classification models. All contributing to the ongoing pursuit of more effective diagnosis and treatment strategies.

Chapter 4

Requirements

4.1 Software

Software requirements define the necessary software resources and prerequisites that must be installed on a computer to ensure optimal functioning of an application. These prerequisites are typically not included in the software installation package and need to be installed separately before installing the software.

1. Python: Ensure Python is installed on your system. You can download it from the official Python website or use a distribution like Anaconda, which comes with many commonly used packages for data science and machine learning.

2. PyTorch: Install PyTorch library, which provides tensors and dynamic neural networks in Python with strong GPU acceleration.

3. Torchvision: Install Torchvision, which includes popular datasets, model architectures, and common image transformations for computer vision

tasks.

4. EfficientNet-PyTorch (if applicable): If you decide to use EfficientNet as one of the backbone architectures, you'll need to install the EfficientNet-PyTorch library.

5. Other Python libraries: Depending on your specific requirements, you may need additional Python libraries such as NumPy, Matplotlib, scikit-learn, etc.

6. Integrated Development Environment (IDE) or Text Editor: Choose an IDE or text editor for writing and running your Python code. Popular options include PyCharm, Visual Studio Code (VSCode), Sublime Text, and Atom.

4.2 Hardware Requirements:

1. CPU: A multi-core CPU with a clock speed of at least 2.0 GHz is suitable for running the code. However, for faster training of deep learning models, consider a CPU with more cores and higher clock speed.

2. GPU (optional but recommended): Training deep learning models, especially with large datasets and complex architectures, can be significantly accelerated using Graphics Processing Units (GPUs). NVIDIA GPUs are commonly used for deep learning tasks, and having a GPU with CUDA support can speed up training times.

3. RAM: Ensure you have sufficient RAM to load datasets, store model parameters, and perform computations during training and inference.

At least 8 GB of RAM is recommended, but more RAM is beneficial for larger datasets and complex models.

4. Storage: Adequate storage space is necessary for storing datasets, model checkpoints, and other project-related files. Solid State Drives (SSDs) are preferred over Hard Disk Drives (HDDs) for faster read/write speeds, which can be beneficial during data preprocessing and model training.

Chapter 5

Proposed System

The study combines GNN and CNN to effectively model the spatial information in MRI images. By treating MRI as images, this overcomes the limitations of regular pixel grid representation, allowing us to capture spatial dependencies and contextual relationships between pixels or areas.

In this study, firstly the MRI images are pre-processed to ensure consistency and quality before training the model. This includes image resolution normalization. Image augmentation techniques such as rotation, translation and scaling are applied to enhance data.

The pre-processed MRI images are then transformed into graph representations to capture spatial dependencies and contextual relationships. This involves treating image features as nodes and dynamically constructing edges based on a k-nearest neighbor algorithm, thereby establishing connections between regions with similar characteristics.

To accelerate training and enhance feature learning, the study employs transfer learning with pre-trained CNN architectures – VGG-16, DenseNet-121, and ResNet-18. These architectures have been trained on large image datasets, enabling us to extract powerful features.

The core of the model consists of the Graph Attention Network (GAT), which uses attention mechanisms to focus on relevant nodes during information dissemination. This allows the model to be more efficient at finding and learning distinctive features that are important for identifying tumors and their boundaries in MRI scans.

Data analysis is performed to understand the distribution of tumors throughout the process and identify potential biases. Use visualization techniques such as t-SNE and PCA to explore high-dimensional space and gain insight into representation and class separation.

To identify the optimal backbone architecture for the task, three variations of the core model are developed:

GAT-VGG16: Employs a pre-trained VGG-16 network for feature extraction.

GAT-Densenet121: Leverages a pre-trained DenseNet-121 network, known for its efficient feature reuse and strong performance in image classification.

GAT-Resnet18: Integrates a pre-trained ResNet-18 network, aiming to benefit from residual connections and depth in feature representation.

5.1 System Design

5.1.1 High-Level Design

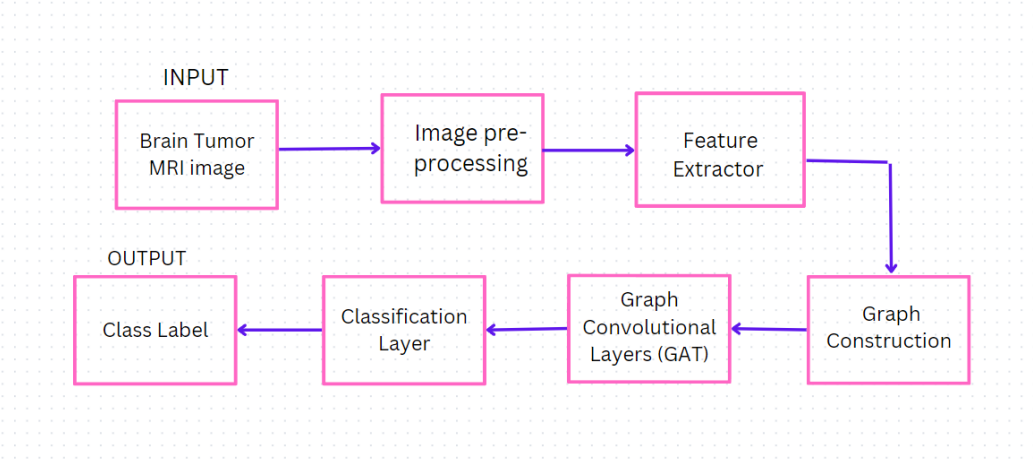


Figure 5.1: high level design

5.2 Model Architecture and Algorithms

The models share a common structure:

- Feature Extraction: One of VGG-16, DenseNet-121, or ResNet-18 extracts initial image features.
  - VGG-16:The network architecture operates on 224 x 224 RGB images, initially subjecting them to a pre-processing step involving mean subtraction for RGB value normalization. It comprises multiple stacks of convolutional layers, each stack featuring several convolutional layers with 3 x 3 receptive fields. These stacks vary in the number of filters per layer, with the first two stacks housing 64 filters, the third consisting of three layers with 256 filters, and the final two stacks containing three layers each with 512 filters. Convolutional layers maintain spatial resolution with a fixed stride

of 1 pixel and padding of 1 pixel, followed by Rectified Linear Unit (ReLU) activation functions to introduce non-linearity. The GAT layers capture the contextual relationships between nodes in the graph, allowing for adaptive feature learning based on attention mechanisms. This enables the model to focus on relevant regions while processing the image features extracted by the convolutional layers.

To downsample feature maps and reduce computational complexity, max-pooling is employed after every two convolutional layers, utilizing 2 x 2-pixel regions with a stride of 2 pixels. This pooling operation effectively halves the size of the feature maps at each stage. Consequently, after traversing all stacks, the final output size is 7 x 7 x 512, representing both the spatial dimensions and the number of filters in the resulting feature maps.

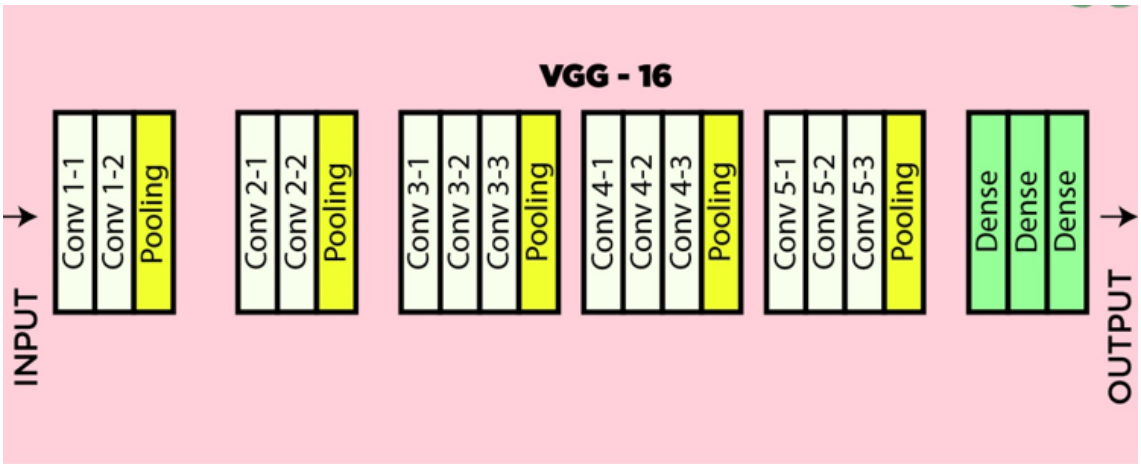


Figure 5.2: VGG-16

– DenseNet-121 Graph Attention Network (GAT) and Convolutional Neural Network (CNN) architecture, namely Dense Convolutional Network, are combined in the GDenseNet121-GAT model. The goal of this integration is to maximise feature interaction by di-

rectly linking all layers and optimising information flow across them. DenseNet121's dense connections are used to condense features through concatenation before passing them to next layers. In contrast to conventional CNNs, these links allow for strong feature propagation and improve feature reuse across the network, which results in effective parameter utilisation.

Through the integration of DenseNet121 into the GAT framework, the model effectively disseminates information across graph nodes through attention mechanisms, facilitating the extraction of complex spatial characteristics from images of brain tumours.By enabling the model to concentrate on pertinent graph regions, the attention mechanism improves the model's capacity to identify minute patterns and variances in tumour features. In the end, this synergistic strategy improves the model's performance in brain tumour classification tasks by facilitating thorough feature extraction and fostering a deeper grasp of the underlying tumour features.

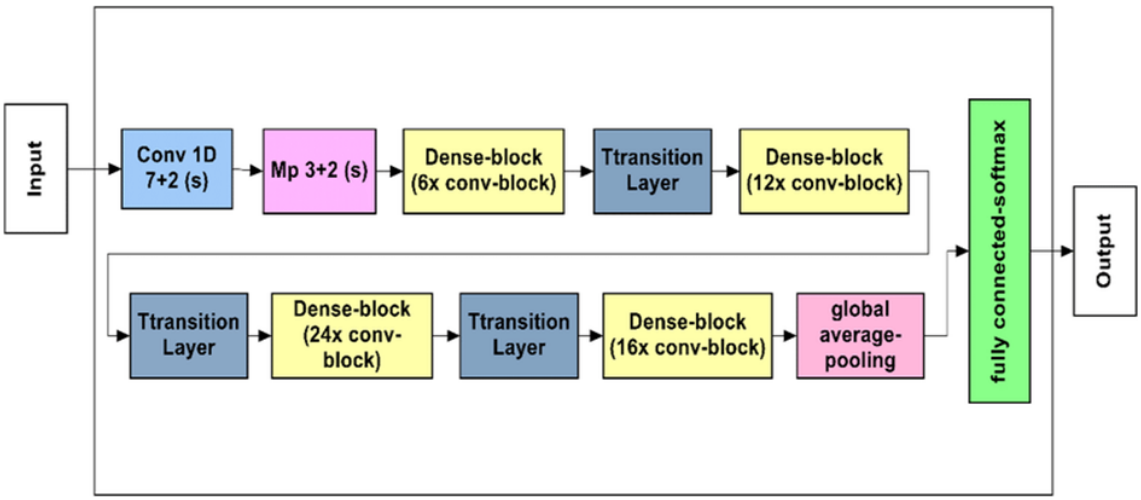


Figure 5.3: DenseNet-121

– ResNet-18 An inventive combination of the ResNet-18 architec-

ture and the Graph Attention Network (GAT) framework is exemplified by the GResNet18-GAT model. The model attempts to take advantage of ResNet18's residual connections (Figure [8]), which are well-known for their ability to mitigate the vanishing gradient problem, by incorporating it into the GAT structure. In order to train deeper networks and enable more efficient learning of complicated characteristics, residual connections provide alternate pathways for gradient movement. The unique architecture of ResNet-18 is made up of several residual building blocks with identity mappings and convolutional layers in each. These building pieces facilitate effective feature extraction and hierarchical representation learning by promoting smooth information flow throughout the network. Furthermore, ResNet-18 has become well-known in both academia and business for its outstanding results in a variety of computer vision applications, such as semantic segmentation, object detection, and image classification.

The GResNet18-GAT model gains these benefits by incorporating ResNet-18 into the GAT framework, improving its performance for tasks related to brain tumour classification and other areas.

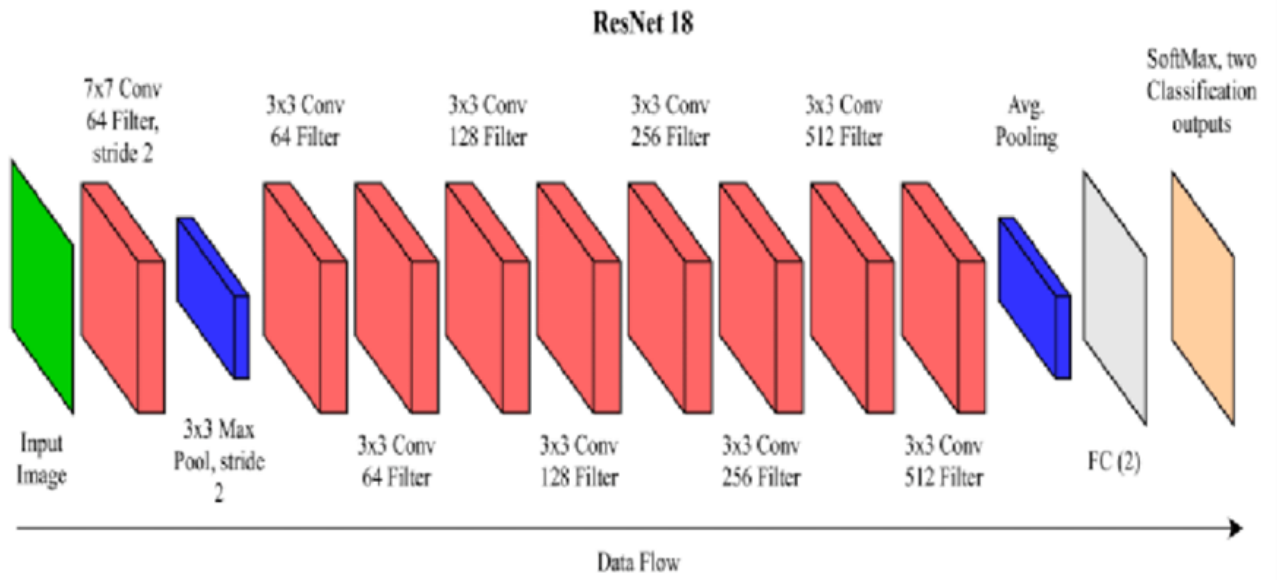


Figure 5.4: ResNet-18

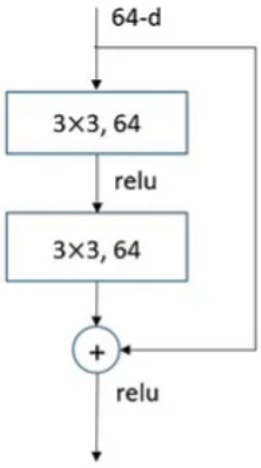


Figure 5.5: Residual blocks

- Graph Construction (`convert_graph`):
  - Calculate pairwise distances between extracted feature maps.
  - Construct a k-nearest neighbor graph to define relationships between image regions.
- GAT Layers (`forward_gcn`): Two GAT layers process the graph, leveraging attention mechanisms to learn spatial features.
- Dropout: Dropout is applied for regularization.
- Output: Models produce classification probabilities for brain tumor classes.

5.2.1 Key Parameters

- k (graph construction): Number of neighbors per node in the graph used for constructing the k-nearest neighbor graph to define relationships between image regions.
- GAT Layer Dimensionality: Dimension of feature representations in the GAT layers. This parameter determines the size of the output feature vectors produced by the GAT layers.
- Dropout Rate: Probability within dropout regularization.
- Learning Rate: The rate at which the optimizer adjusts the model parameters during training. It controls the step size in the parameter space search, influencing the convergence and optimization performance of the model.

5.2.2 Key Innovations

- Enhanced Image Representation: This approach overcomes the limitations of traditional pixel grid representations in MRI analysis for tumor classification. By transforming MRI images into graphs, spatial dependencies are captured and contextual relationships, allowing for more effective feature learning.

- Integration of GNN and CNN Strengths: The model strategically combine the strengths of Graph Neural Networks (GNNs) and Convolutional Neural Networks (CNNs) in the model architecture. GNNs enable spatial learning by propagating information across graph nodes, while pre-trained CNNs facilitate robust feature extraction from MRI images, leveraging their learned representations from large-scale image datasets.

- Systematic Backbone Exploration: The study systematically explores the effectiveness of different backbone architectures, including VGG-16, DenseNet121, and ResNet-18, within the GNN-based brain tumor classification framework. By evaluating the performance of each backbone architecture, the aim is to identify the most effective feature extractor for the specific task, optimizing the overall classification accuracy and robustness of the model.

Chapter 6

Result and Analysis

Lets compare the results between the three models using different evaluation metrics.

6.1 Evaluation Metrics

6.1.1 Confusion Matrix

It offers a thorough analysis of how the model's predictions and the actual labels for each class compare. There are four sections in the matrix: True Positive(TP), False Positive(FP), True Negative(TN), False Negative(FN). The model's performance in terms of both accurate and inaccurate predictions can be clearly seen thanks to the confusion matrix, which also offers insights into the model's shortcomings. The ResNet, DenseNet, and VGG-16 matrices are shown below. Figures[6.1],[6.2], and [6.3] display the heatmaps in the same sequence.

$$\begin{bmatrix} 28 & 52 & 20 & 0 \\ 0 & 115 & 0 & 0 \\ 0 & 1 & 104 & 0 \\ 0 & 15 & 6 & 53 \end{bmatrix}$$

$$\begin{bmatrix} 28 & 53 & 19 & 0 \\ 0 & 115 & 0 & 0 \\ 0 & 1 & 104 & 0 \\ 0 & 14 & 6 & 54 \end{bmatrix}$$

$$\begin{bmatrix} 28 & 50 & 22 & 0 \\ 0 & 115 & 0 & 0 \\ 0 & 2 & 103 & 0 \\ 0 & 14 & 6 & 54 \end{bmatrix}$$

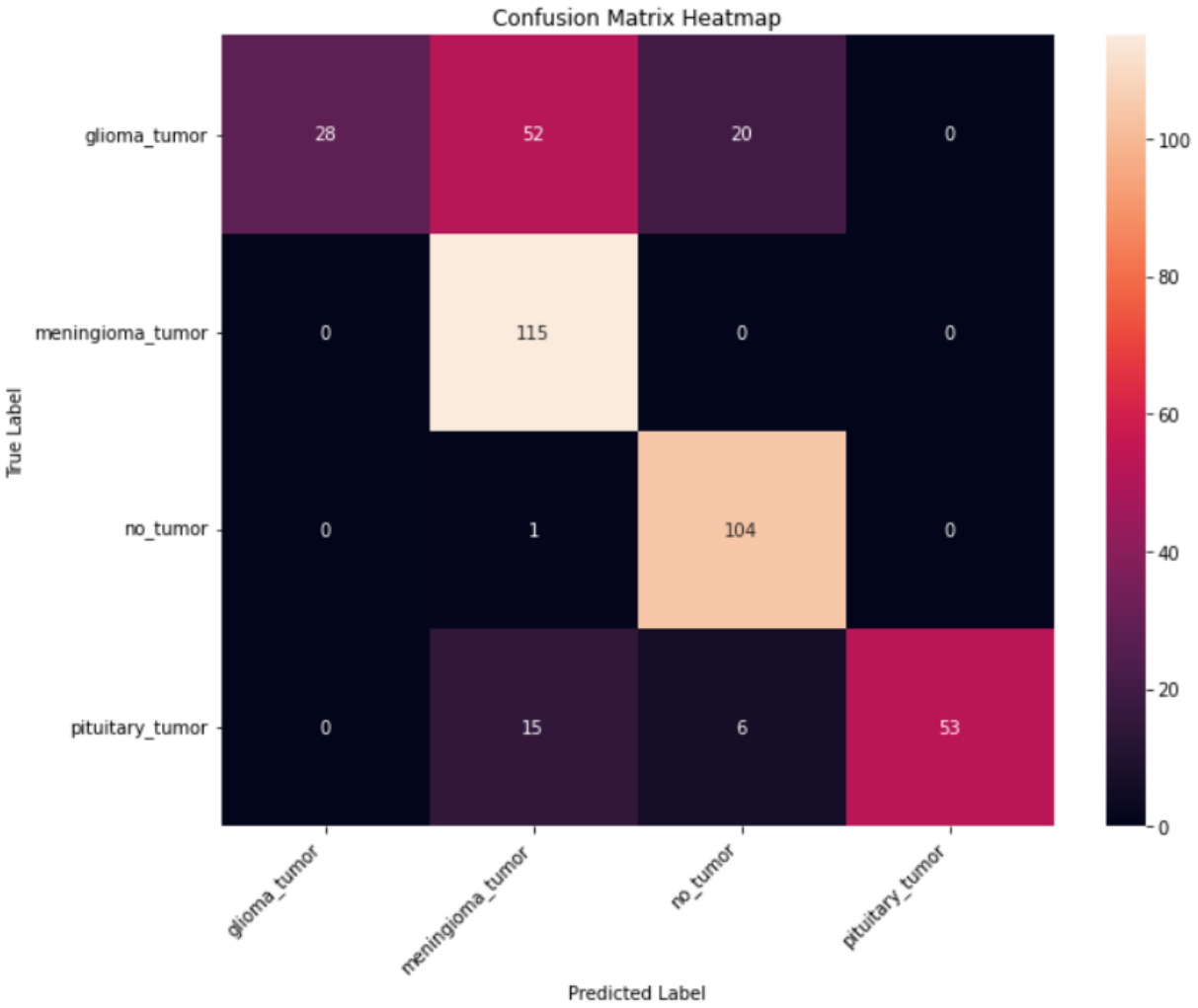


Figure 6.1: Confusion Matrix HeatMap for ResNet_GAT

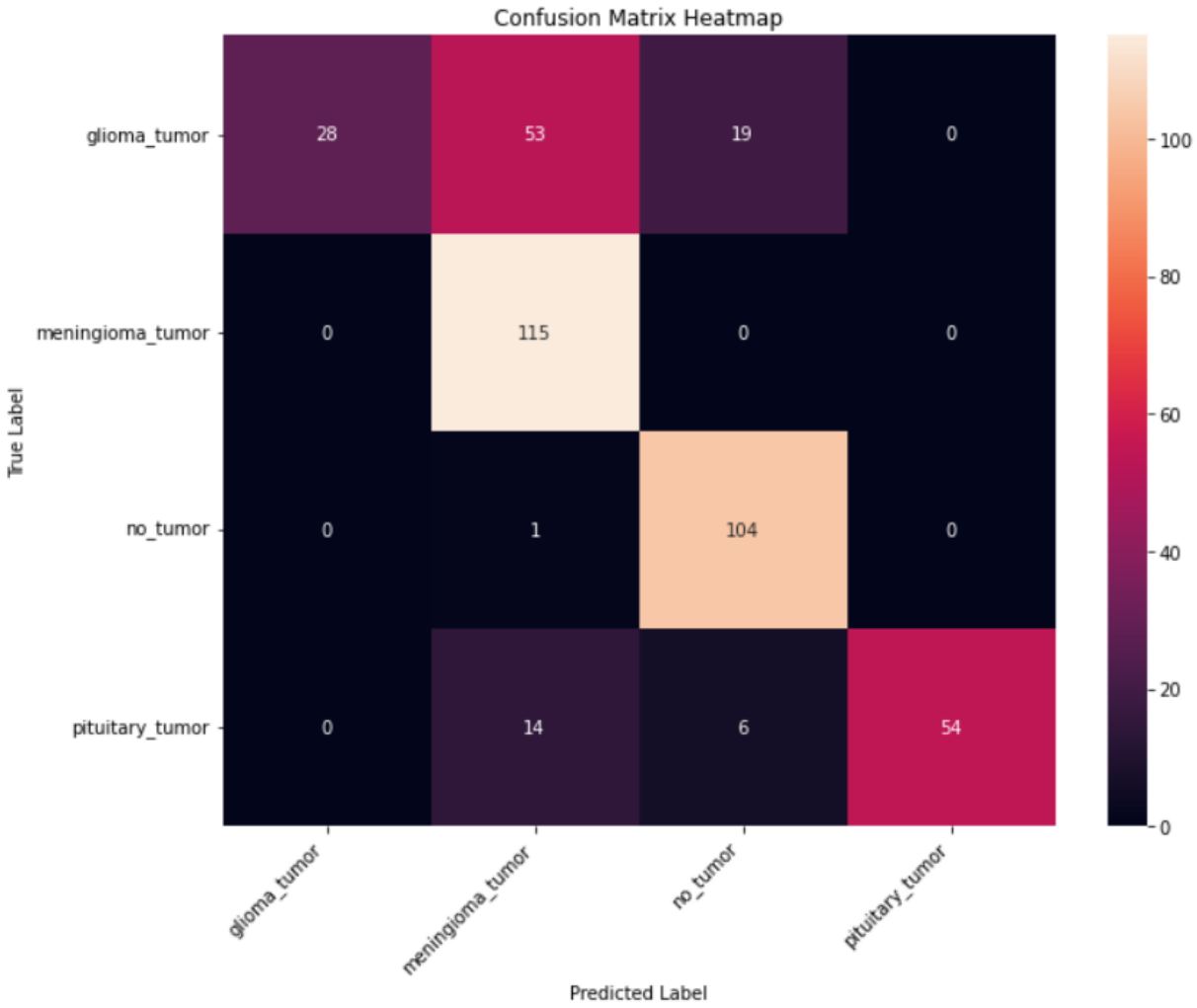


Figure 6.2: Confusion Matrix HeatMap for DenseNet_GAT

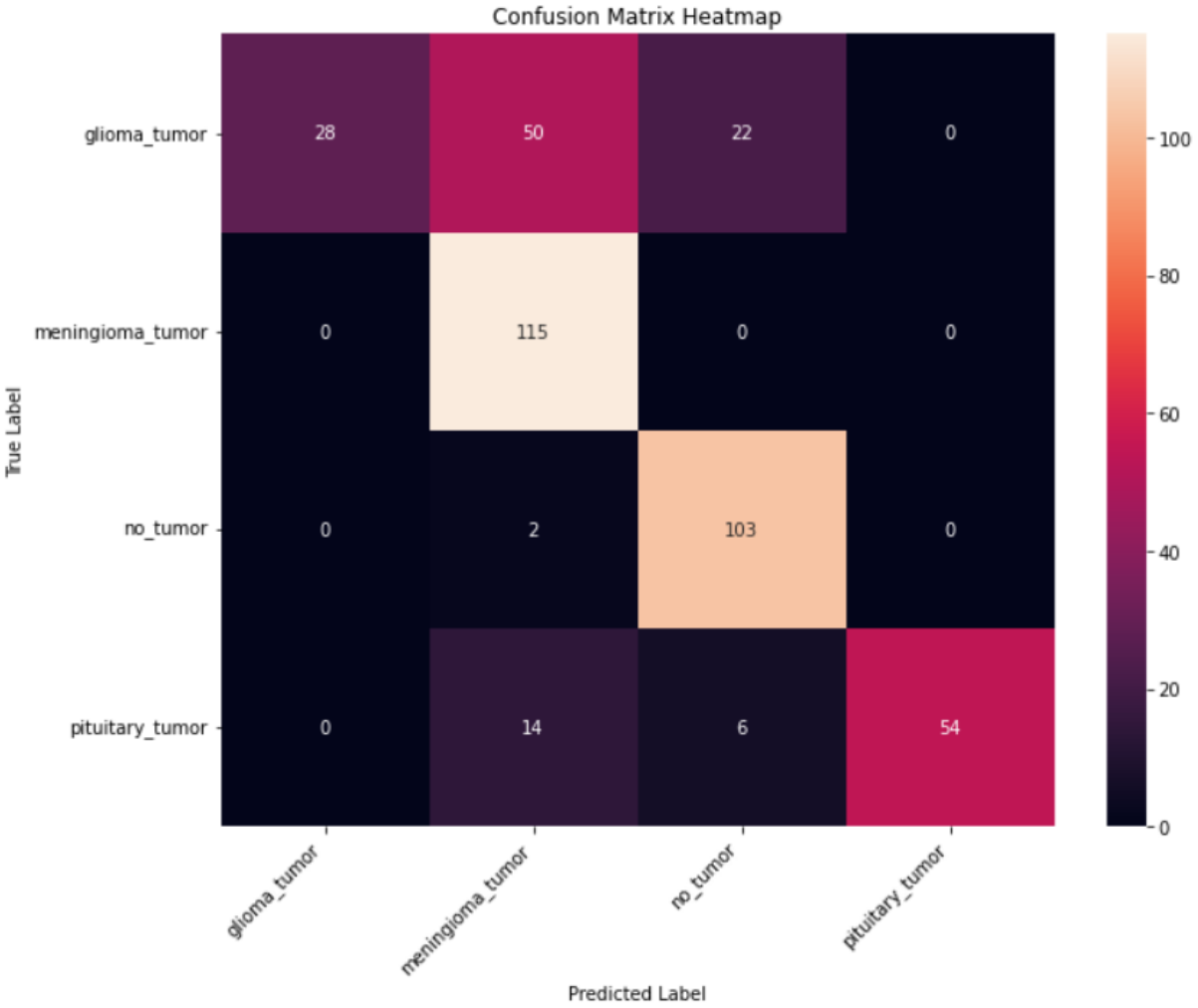


Figure 6.3: Confusion Matrix HeatMap for VGG-16_GAT

6.1.2 ROC curves

The ROC (Receiver Operating Characteristic) curve is a graphical representation of a binary classification model's diagnostic capacity under various threshold levels. It compares the true positive rate (TPR) to the false positive rate (FPR) at different thresholds. A ROC curve plots True Positive Rate (TPR), often called sensitivity or recall, on the y-axis. It indicates the fraction of actual positive cases that the model accurately identified as positive. The x-axis depicts the False Positive Rate (FPR). It indicates the

percentage of true negative cases that the model mistakenly identified as positive. They are depicted in Figures [6.4], [6.5], and [6.6], respectively.

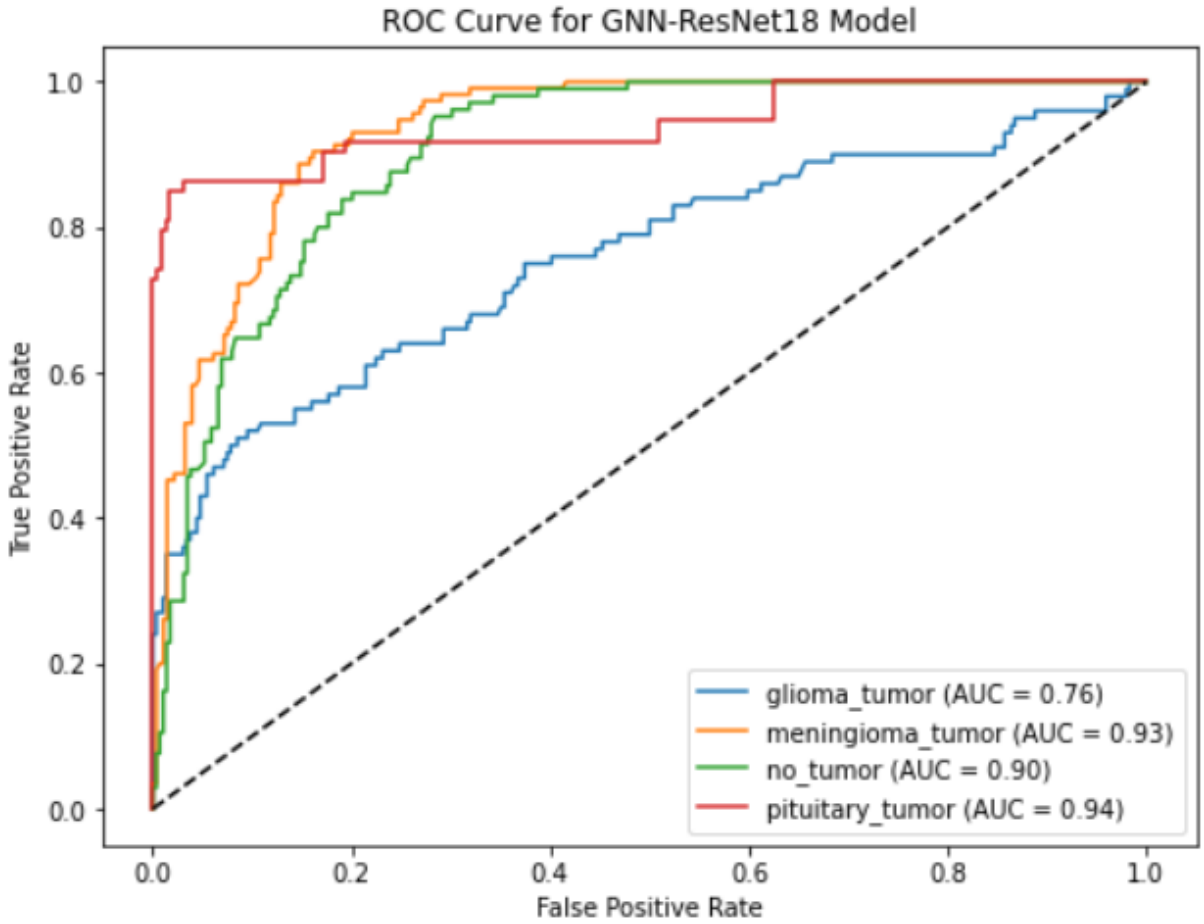


Figure 6.4: ROC curves for ResNet_GAT

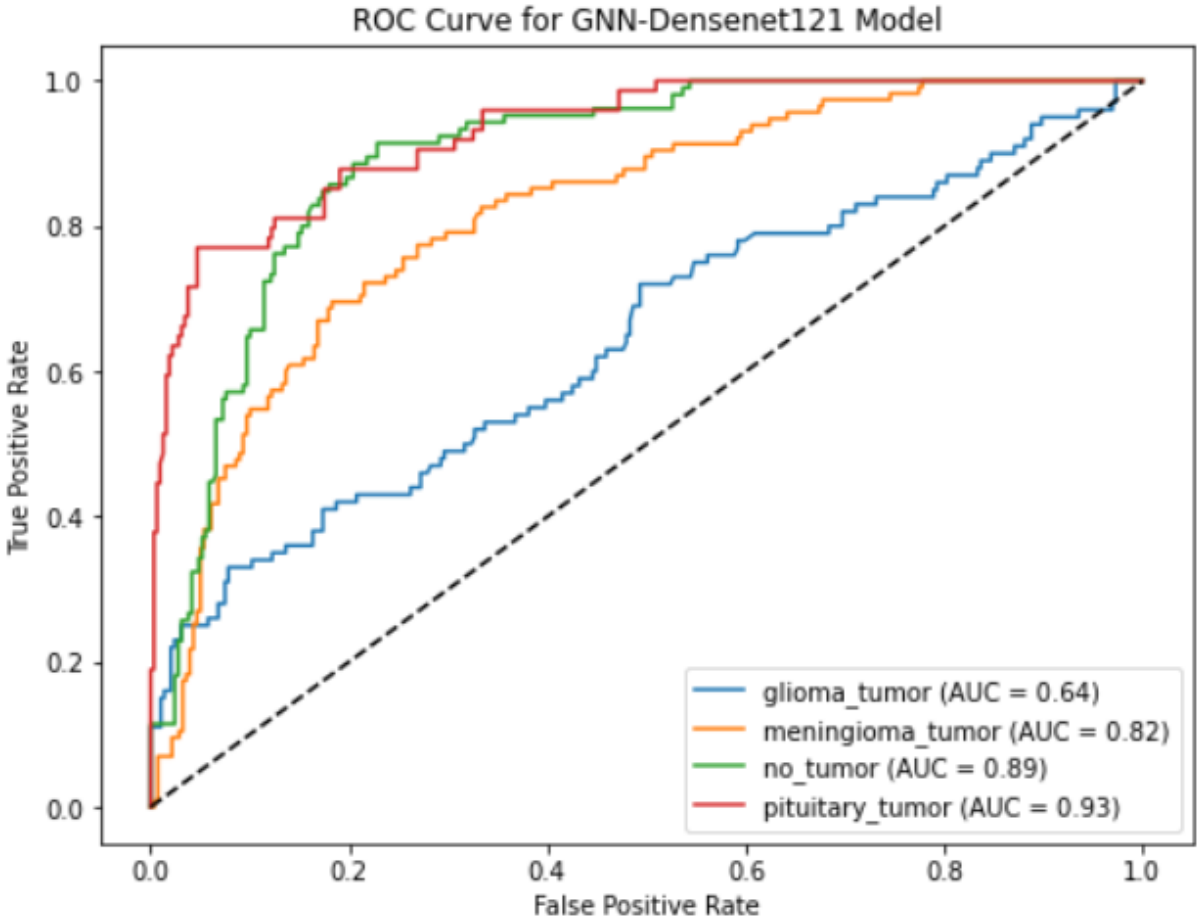


Figure 6.5: ROC curves for DenseNet_GAT

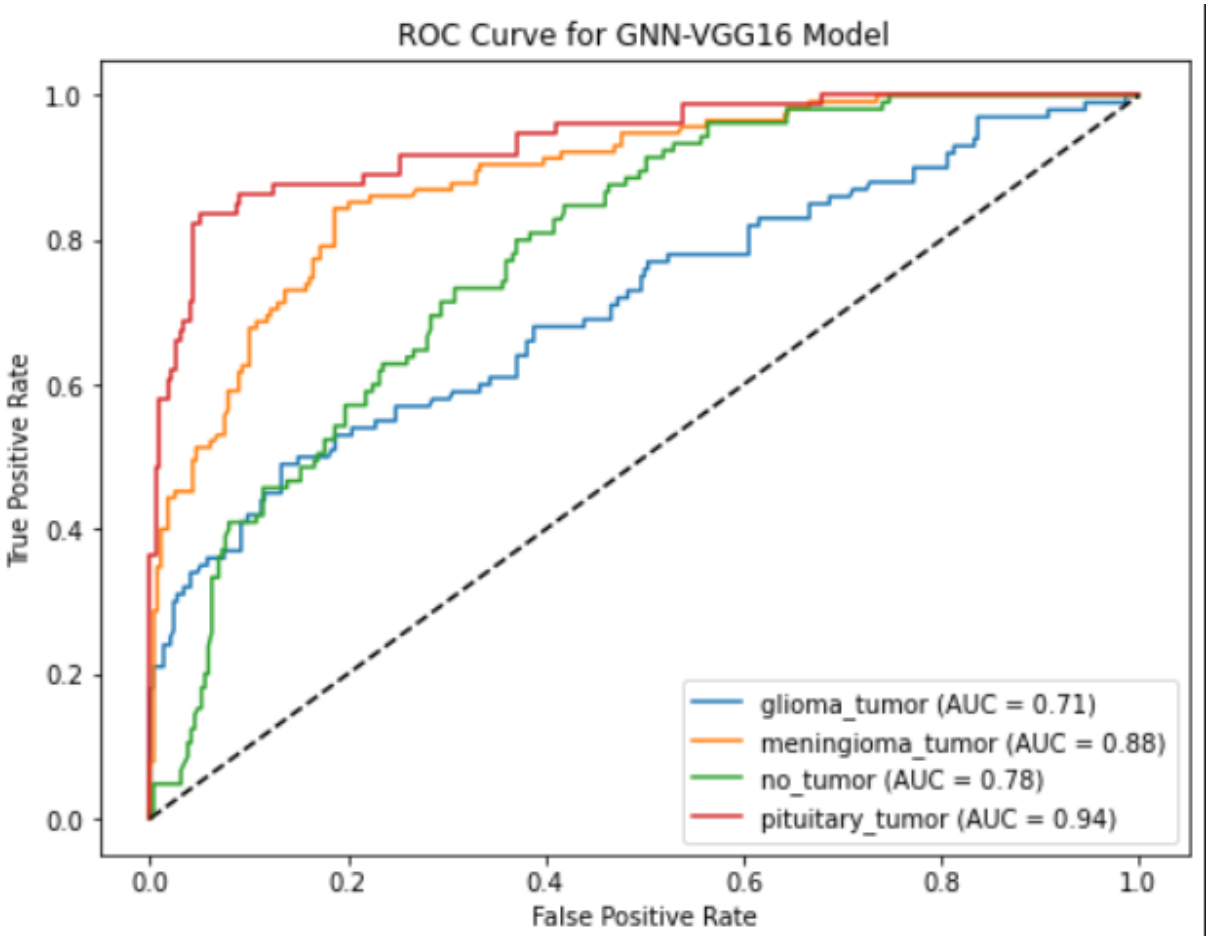


Figure 6.6: ROC curves for VGG_GAT

6.1.3 Loss and Accuracy

Accuracy is one of the most commonly used metrics for assessing categorization methods. It assesses the model's ability to properly forecast an observation's class among all observations. In classification tasks, loss is commonly defined as the difference between predicted class probabilities and true class labels. Lower loss numbers indicate higher performance, implying that the model's predictions are more accurate. They are depicted in Figures [6.7], [6.8], and [6.9].

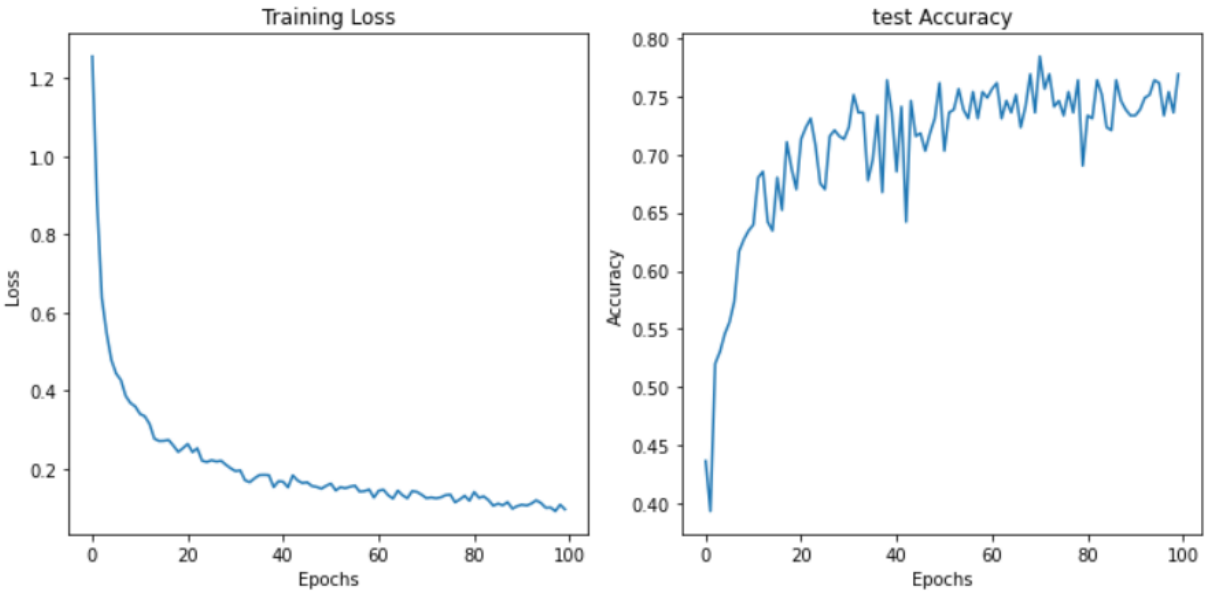


Figure 6.7: Loss and Accuracy curves for ResNet_GAT

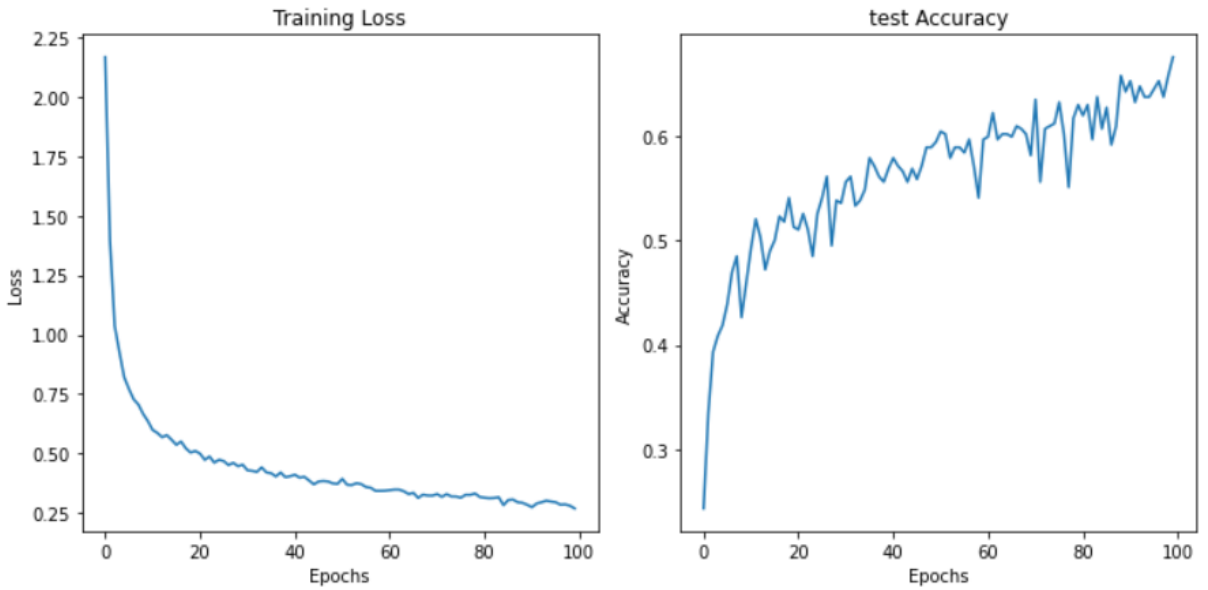


Figure 6.8: Loss and Accuracy curves for Densenet_GAT

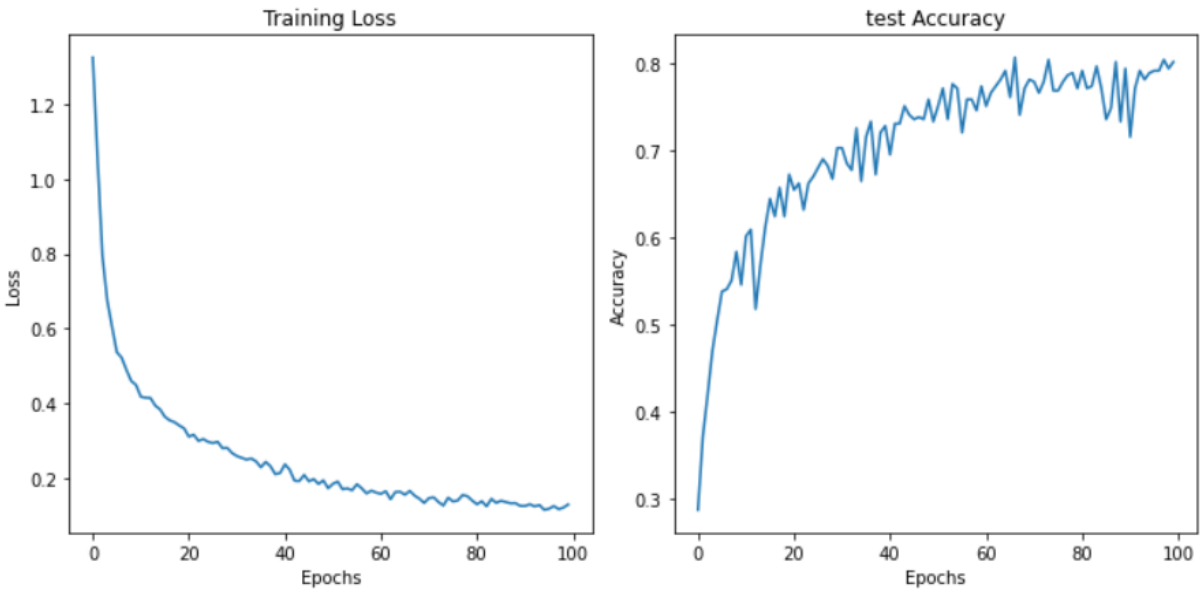


Figure 6.9: Loss and Accuracy curves for VGG-16_GAT

6.1.4 Performance Metrics

Precision

Precision evaluates the fraction of correctly classified instances or samples among the ones classified as positives.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall

Recall is a metric in machine learning that measures the ability of a model to identify all relevant instances of a particular class. It is the ratio of correctly predicted positive observations to the total actual positives, providing insights into a model's completeness in capturing instances of a given class.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 Score

The F1 Score is the harmonic mean of precision and recall, offering a balanced measure that considers both false positives and false negatives, making it suitable for imbalanced datasets.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy

Accuracy is the proportion of correct predictions in a classification task, measuring the overall correctness of a model's predictions(Table 6.1).

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Predictions}}$$

All the performance metrics are listed in Classification reports.

Table 6.1: Test Accuracy for Different Models

Model	**Test Accuracy**
GAT_VGG16	0.796954
GAT_Densenet121	0.667513
GAT_Resnet18	0.756345

Classification Report

The following are the Classification Reports of ResNet, DenseNet, VGG-16 respectively.

	Precision	Recall	F1-score	Support
Class 0	1.00	0.28	0.44	100
Class 1	0.63	1.00	0.77	115
Class 2	0.80	0.99	0.89	105
Class 3	1.00	0.72	0.83	74
Accuracy			0.76	394
Macro Avg	0.86	0.75	0.73	394
Weighted Avg	0.84	0.76	0.73	394

	Precision	Recall	F1-score	Support
Class 0	1.00	0.28	0.44	100
Class 1	0.63	1.00	0.77	115
Class 2	0.81	0.99	0.89	105
Class 3	1.00	0.73	0.84	74
Accuracy			0.76	394
Macro Avg	0.86	0.75	0.74	394
Weighted Avg	0.84	0.76	0.73	394

	Precision	Recall	F1-score	Support
Class 0	1.00	0.28	0.44	100
Class 1	0.64	1.00	0.78	115
Class 2	0.79	0.98	0.87	105
Class 3	1.00	0.73	0.84	74
Accuracy			0.76	394
Macro Avg	0.86	0.75	0.73	394
Weighted Avg	0.84	0.76	0.73	394

6.2 Comparison and Findings

In this section, lets compare the performance of the Graph Attention Network (GAT) models employing different backbone architectures, notably VGG-16, DenseNet121, and ResNet-18, for brain tumour classification in MRI scans. Lets understand more by comparing the findings to those from earlier studies using Graph Convolutional Networks (GCNs) employing the same architectures, to better understand the influence of switching from GCNs to GATs on classification accuracy.

6.2.1 Comparison of GAT Models

GraphCNN_VGG16: With the introduction of GATs, the GraphCNN_VGG16 model exhibits a substantial improvement in test accuracy, achieving a remarkable accuracy of 79.69% from 71.31% . This signifies the efficacy of GATs in capturing complex spatial dependencies within MRI images, leading to enhanced feature representations.

GraphCNN_Densenet121: The GraphCNN_Densenet121 model improves the accuracy from 64.21% to 66.75%. This suggests that the dense

connectivity pattern inherent in DenseNet121 architecture synergizes well with the spatial learning capabilities of GATs.

GraphCNN_Resnet18: Similarly, the GraphCNN_Resnet18 model exhibits a notable increase in test accuracy with the adoption of GATs, achieving an accuracy of 75.63% from 71.57%. The Table 6.2 and Table 6.3 show the accuracy for GAT and GCN models respectively.

Table 6.2: Test Accuracy for GAT Model

Model	**Test Accuracy**
GAT_VGG16	0.796954
GAT_Densenet121	0.667513
GAT_Resnet18	0.756345

Table 6.3: Test Accuracy of GCN Model

Model	**Test Accuracy**
GraphCNN_VGG16	0.713198
GraphCNN_Densenet121	0.642132
GraphCNN_Resnet18	0.715736

6.2.2 Comparative Analysis

When GAT models are compared to their GCN counterparts, a consistent trend of improved accuracy emerges across all backbone designs. The inclusion of attention mechanisms in GATs improves information propagation and feature learning, resulting in better classification performance.

Chapter 7

Conclusion

7.1 Conclusion

The findings demonstrate the effectiveness of GATs in harnessing spatial dependencies in MRI images to accurately classify brain tumours. The synergy between GNNs and CNNs, especially when combined with attention processes, improves the model's ability to detect subtle patterns and fluctuations in tumour features.

The choice of backbone architecture is critical in deciding the model's performance, with VGG-16 emerging as the best performer in the tests. However, DenseNet121 and ResNet-18 both perform well, demonstrating GATs' adaptability in supporting various network architectures.

7.2 Future Scope

Future work involves refining and optimizing the algorithms and exploring the optimization techniques, data augmentation strategies. Integrating additional imaging modalities such as functional MRI (fMRI), diffusion tensor

imaging (DTI), and spectroscopy to provide complementary information for more comprehensive tumor characterization. Integrating the developed models into Clinical Decision Support System (CDSS) to assist radiologists and clinicians in making accurate and timely diagnoses. This could involve building user-friendly interfaces and integrating with existing hospital systems. Collaborating with medical researchers, radiologists, and data scientists to collect and annotate larger and more diverse datasets, fostering the development of more robust and generalizable models.

References

[1] Reddy, Digvijay, V. Bhavana, and H. K. Krishnappa. "Brain tumor detection using image segmentation techniques." In 2018 International conference on communication and signal processing (ICCSP), pp. 0018-0022. IEEE, 2018.

[2] Sravani, Maddineni, S. Aparna, J. Sabarinath, and Yamani Kakarla. "Enhancing Brain Tumor Diagnosis with Generative Adversarial Networks." In 2024 14th International Conference on Cloud Computing, Data Science Engineering (Confluence), pp. 846-851. IEEE, 2024.

[3] Krishnan, Rohith, P. G. Gokul, Gopikrishnan Sujith, T. Anjali, and S. Abhishek. "Enhancing Brain Tumor Diagnosis: A CNN-Based Multi-Class Classification Approach." In 2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI), vol. 2, pp. 1-6. IEEE, 2024.

[4] Jeena, Kottarapat, Cheripelil Abraham Manju, Koythatta Meethalveedu Sajesh, G. Siddaramana Gowd, Thangalazhi Balakrishnan Sivanarayanan, Deepthi Mol C, Maneesh Manohar, Ajit Nambiar, Shantikumar V. Nair, and Manzoor Koyakutty. "Brain-tumor-regenerating 3D scaffold-based primary xenograft models for

glioma stem cell targeted drug screening." ACS Biomaterials Science Engineering 5, no. 1 (2019): 139-148.

[5] Kavitha, K. R., Amritha S. Nair, and Vishnu Narayanan Harish. "Detection Of Brain Tumour Using Deep Convolutional Network." In 2023 4th IEEE Global Conference for Advancement in Technology (GCAT), pp. 1-6. IEEE, 2023.

[6] Varthakavi, Sai Sasank, Devisetty Rohith Prasanna Babu, Amsitha MB, and Sasi Jyothirmai Bonu. Analysis of Classification Algorithms for Brain Tumor Detection. No. 1810. EasyChair, 2019.

[7] Alzubaidi, L. et al. "MedNet: Pre-trained convolutional neural network model for medical imaging tasks." arXiv, 2021. eprint: 2110.06512. `https://arxiv.org/abs/2110.06512`

[8] Raza, A. et al. "A hybrid deep learning-based approach for brain tumor classification." Electronics, vol. 11, no. 7, p. 1146, 2022.

[9] Dehkordi, A. A., Hashemi, M., Neshat, M., Mirjalili, S., and Sadiq, A. S. "Brain tumor detection and classification using a new evolutionary convolutional neural network." arXiv, 2022. eprint: 2204.12297. `https://arxiv.org/abs/2204.12297`

[10] Ghassemi, N., Shoeibi, A., and Rouhani, M. "Deep neural network with generative adversarial networks pre-training for brain tumor classification based on MR images." Biomed. Signal Process. Control, vol. 57, p. 101678, 2020.

[11] Ge, C., Gu, I. Y. H., Jakola, A. S., Yang, J. "Enlarged training dataset by pairwise GANs for molecular-based brain tumor classification." IEEE Access, 8, 22560–22570 (2020).

[12] Deepak, S., and Ameer, P. M. "Brain tumor classification using deep CNN features via transfer learning." Comput. Biol. Med., vol. 111, p. 103345, 2019.

[13] Sajjad, M. et al. "Multi-grade brain tumor classification using deep CNN with extensive data augmentation." J. Comput. Sci., vol. 30, pp. 174182, 2019.

[14] Mehrotra, R., Ansari, M. A., Agrawal, R., and Anand, R. S. "A transfer learning approach for AI-based classification of brain tumors." Mach. Learn. Appl., vol. 2, p. 100003, 2020.

[15] Ullah, Z., Farooq, M. U., Lee, S. H., and An, D. "A hybrid image enhancement based brain MRI images classification technique." Med. Hypotheses, vol. 143, p. 109922, 2020.

[16] Çinar, A., and Yildirim, M. "Detection of tumors on brain MRI images using the hybrid convolutional neural network architecture." Med. Hypotheses, vol. 139, p. 109684, 2020.

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

listings

Appendix A

Source code

A.1 Dataset Details

MRI data were obtained from Kaggle, which contains approximately 3264 MR images. The MRI , T1, T2 and FLAIR type 35 presented in this document are a combination of different patients and are generally divided into 4 groups: glioma tumor, meningioma tumor, no tumor, pituitary tumor shown in Figure[A.1]. MRI stands for magnetic resonance imaging. MRI uses magnetic fields instead of X-rays to create detailed images of the body being imaged. Sometimes an MRI can be used to evaluate the size of the tumor. MRI produces better, more detailed images than CT scans and is therefore preferred over CT scans when examining the brain. For the diagnosis of a brain tumor (which may be of primary or secondary origin), MRI of the brain and/or spinal cord may be performed, depending on the type of tumor. This database contains 3094 brain MRIs that distinguish between tumor and non-tumor images.

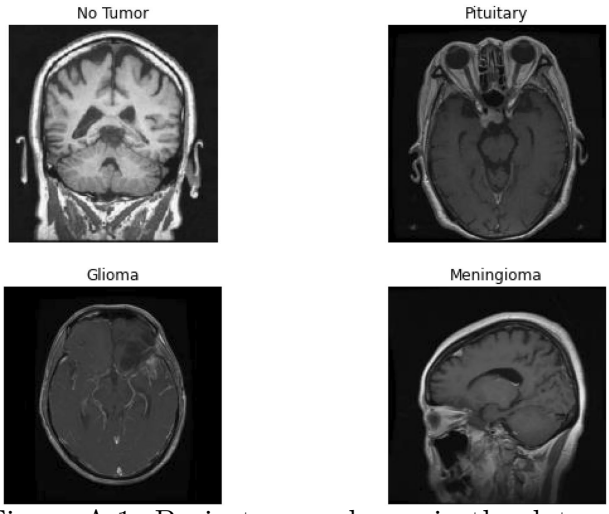


Figure A.1: Brain tumor classes in the dataset

You can find the code repository for brain tumor detection using graphing at: https://github.com/NIHAAS2002/brain-tumor-detection-using-graphing.

A.2 Installing the packages

```
import subprocess
whls = [
    "/kaggle/input/pyg-cp37-pt111/torch_cluster-1.6.0-
       cp37-cp37m-linux_x86_64.whl",
    "/kaggle/input/pyg-cp37-pt111/torch_scatter-2.1.0-
       cp37-cp37m-linux_x86_64.whl",
    "/kaggle/input/pyg-cp37-pt111/torch_sparse-0.6.16-
       cp37-cp37m-linux_x86_64.whl",
    "/kaggle/input/pyg-cp37-pt111/torch_spline_conv
       -1.2.1-cp37-cp37m-linux_x86_64.whl",
    "/kaggle/input/pyg-cp37-pt111/torch_geometric-2.2.0-
       py3-none-any.whl",
    "/kaggle/input/pyg-cp37-pt111/ruamel.yaml-0.17.21-
```

```
        py3-none-any.whl",
]
for w in whls:
    print("Installing", w)
    subprocess.call(["pip", "install", w, "--no-deps",
        "--upgrade"])
```

A.3 Importing the Modules

```
import torch
import torch.utils.data as data_utils
import torch_geometric
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader
import torch
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.optim
import torch_geometric
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv
from torch_geometric.nn import GATConv

import torch.nn.functional as F
import torchvision
from torchvision import datasets, models, transforms
import tqdm
```

```
from IPython import display
from types import MethodType
import numpy as np
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import networkx as nx
import torchvision.models as models
from sklearn.metrics import confusion_matrix,
   classification_report
import seaborn as sns
from torch.cuda.amp import autocast
from tabulate import tabulate
from skimage.segmentation import slic
from skimage.segmentation import slic, mark_boundaries
```

A.4 Image Transformations

```
transforms = torchvision.transforms
data_transform = {
    "train": transforms.Compose([
                                    transforms.
                                       RandomResizedCrop
                                       (224),
                                       transforms.Resize
                                          ((224, 224)),
                                    transforms.
                                       RandomHorizontalFlip
```

```
                                    (),
                                     transforms.ToTensor
                                        (),
                                  transforms.Normalize
                                     ((0.5, 0.5, 0.5),
                                     (0.5, 0.5, 0.5))
                                 ]),
    "tes": transforms.Compose([
                                transforms.Resize((224,
                                   224)),
                                transforms.
                                   RandomHorizontalFlip()
                                   ,
                                transforms.ToTensor(),
                                transforms.Normalize
                                   ((0.5, 0.5, 0.5),
                                   (0.5, 0.5, 0.5))
    ])}
```

A.5 Loading the Dataset

```
train_dataset = torchvision.datasets.ImageFolder("/
   kaggle/input/brain-tumor-classification-mri/Training
   ", transform=data_transform["train"])
train_loader = DataLoader(train_dataset, batch_size=32,
   shuffle=True)
test_dataset = torchvision.datasets.ImageFolder("/kaggle
   /input/brain-tumor-classification-mri/Testing",
```

```
    transform=data_transform["tes"])
test_loader = DataLoader(test_dataset, batch_size=32,
    shuffle=False)
```

A.6 Dataset Visualization

```
def display_dataset_samples(dataset, num_samples=4):
    # Display some random samples from the dataset
    indices = torch.randint(0, len(dataset), (
        num_samples,))
    sample_loader = DataLoader(dataset, batch_size=
        num_samples, sampler=torch.utils.data.sampler.
        SubsetRandomSampler(indices))

    for images, labels in sample_loader:
        plt.figure(figsize=(12, 5))
        for i in range(num_samples):
            plt.subplot(1, num_samples, i + 1)
            plt.imshow(transforms.ToPILImage()(images[i
                ]))
            plt.title(f' {dataset.classes[labels[i].item
                ()]}')  # Use dataset.classes to get the
                label name
            plt.axis('off')
        plt.show()
        break

display_dataset_samples(train_dataset, num_samples=5)
```

```
def plot_category_distribution(train_dataset,
   test_dataset):
    # Get the class names/categories from the dataset
    classes = train_dataset.classes
    print("Classes: ", classes)

    # Count occurrences of each category in the training
        dataset
    train_counts = [0] * len(classes)
    for _, label in train_dataset:
        train_counts[label] += 1

    # Count occurrences of each category in the test
       dataset
    test_counts = [0] * len(classes)
    for _, label in test_dataset:
        test_counts[label] += 1

    # Plot the bar graph
    width = 0.35
    x = np.arange(len(classes))

    fig, ax = plt.subplots(figsize=(10, 6))
    rects1 = ax.bar(x - width/2, train_counts, width,
       label='Train Dataset')
    rects2 = ax.bar(x + width/2, test_counts, width,
       label='test Dataset')
```

```
    # x-axis labels.
    ax.set_xlabel('Categories')
    ax.set_ylabel('Number of Samples')
    ax.set_title('Category Distribution in Train and
       test Datasets')
    ax.set_xticks(x)
    ax.set_xticklabels(classes, rotation=45, ha="right")
    ax.legend()

    plt.show()



plot_category_distribution(train_dataset, test_dataset)
print("Number of Training Data points: ",len(
   train_dataset))
print("Number of Testing Data points: ",len(test_dataset
   ))
```

A.7 Testing and Training Functions

```
dtype = torch.float16
device = torch.device("cuda:0")
def train(model: nn.Module, num_epochs, train_loader,
   test_loader, lr):
    model.type(dtype)
    model.to(device)
```

```
model.train()
optimizer = torch.optim.SGD(model.parameters(), lr)
loss = nn.CrossEntropyLoss()
lrc = torch.optim.lr_scheduler.CosineAnnealingLR(
   optimizer, T_max=num_epochs, eta_min=lr/10)
loss_array = []
acc_array = []

for epoch in tqdm.tqdm(range(num_epochs)):
    running_loss, c = 0, 0

    for inputs, target in train_loader:
        inputs = inputs.type(dtype)
        inputs = inputs.to(device)
        target = target.to(device)

        y_pred = model(inputs)
        l = loss(y_pred, target)

        optimizer.zero_grad()
        l.backward()
        optimizer.step()

        running_loss += l
        c += 1

    loss_array.append(running_loss.item() / c)
    acc_array.append(test(model, test_loader))
```

```
    return loss_array, acc_array

@torch.no_grad()
def test(model,test_loader):
    correct=0
    total=0
    model.eval()
    model.to(device)
    model.type(dtype)

    with torch.no_grad():
        for data in test_loader:
            image,lables=data
            outputs=model(image.type(dtype).to(device))
            _,predicted=torch.max(outputs.data,dim=1)
            total+=lables.size(0)
            correct+=(predicted==lables.to(device)).sum
               ().item()
    acc = correct/total
    return acc
```

A.8 Defining the Models

```
def _forward_impl(self, x):
    x = x.type(dtype)
    x = self.conv1(x)
    x = self.bn1(x)
```

```
    x = self.relu(x)
    x = self.maxpool(x)
    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)
    return x
```

A.9 VGG16 with GAT

```
* VGG-16 100, lr == 0.001

class GraphCNN_VGG16(nn.Module):
    def __init__(self,k, num_classes):
        super().__init__()
        self.k = k
        self.num_classes = num_classes
        self.vgg16 = models.vgg16(pretrained=True)  #
           Load pre-trained VGG-16
        self.conv1 = torch_geometric.nn.GATConv(512,
           256)  # Adjust input dimension here
        self.dropout = nn.Dropout(0.5)
        self.conv2 = torch_geometric.nn.GATConv(256,
           num_classes)

    def forward_features(self,X):
        features = self.vgg16.features(X)  # Extract
           features using VGG-16
```

```
#         print("VGG16 Features Shape:", features.shape)
#         print("Features: ", features)
        return features

    def convert_graph(self, x):
        b,c,h,w = x.shape
        device = x.device
        k = self.k
        x = torch.permute(x,(0,2,3,1)).reshape((b,h*w,c)
           )
        y = torch.cdist(x,x,2)  # [b, hw, hw]
        _,idx = torch.topk(y,k,-1)
        source = torch.arange(h*w,device=device).
           repeat_interleave(k).repeat(b)
        target = torch.flatten(idx)
        step = torch.arange(b,device=device).
           repeat_interleave(h*w*k)*h*w
        adj = torch.row_stack([source,target]) + step
        return x.reshape((b*h*w,c)), adj.long()

    def froward_gcn(self,X,adj):
        X = self.conv1(X,adj)
        X = self.dropout(X)
        X = self.conv2(X,adj)  # (B*7*7, num_classes)
        return X

    def forward(self,X):
        batch = X.shape[0]
        X = self.forward_features(X)
```

```
        X,adj = self.convert_graph(X)
        X = self.froward_gcn(X,adj)
        X = torch.reshape(X,(batch, -1, self.num_classes
           ))
        X = torch.mean(X,1) # (B, num_classes)
        return X
```

A.10 DenseNet 121 with GAT

```
* Densenet -121 100 Lr = 0.0001

class GraphCNN_Densenet121(nn.Module):
    def __init__(self,k, num_classes):
        super().__init__()
        self.k = k
        self.num_classes = num_classes
        self.densenet = models.densenet121(pretrained=
           True)  # Load pre-trained DenseNet-121
        self.conv1 = torch_geometric.nn.GATConv(1024,
           256)
        self.dropout = nn.Dropout(0.5)
        self.conv2 = torch_geometric.nn.GATConv(256,
           num_classes)

    def forward_features(self,X):
        features = self.densenet.features(X)  # Extract
           features using DenseNet-121
```

```
#         print("DenseNet121 Features Shape:", features.
   shape)
#         print("Features: ", features)
        return features

    def convert_graph(self, x):
        b,c,h,w = x.shape
        device = x.device
        k = self.k
        x = torch.permute(x,(0,2,3,1)).reshape((b,h*w,c)
           )
        y = torch.cdist(x,x,2)  # [b, hw, hw]
        _,idx = torch.topk(y,k,-1)
        source = torch.arange(h*w,device=device).
           repeat_interleave(k).repeat(b)
        target = torch.flatten(idx)
        step = torch.arange(b,device=device).
           repeat_interleave(h*w*k)*h*w
        adj = torch.row_stack([source,target]) + step
        return x.reshape((b*h*w,c)), adj.long()

    def froward_gcn(self,X,adj):
        X = self.conv1(X,adj)
        X = self.dropout(X)
        X = self.conv2(X,adj)  # (B*7*7, num_classes)
        return X

    def forward(self,X):
        batch = X.shape[0]
```

```
        X = self.forward_features(X)
        X,adj = self.convert_graph(X)
        X = self.froward_gcn(X,adj)
        X = torch.reshape(X,(batch, -1, self.num_classes
           ))
        X = torch.mean(X,1) # (B, num_classes)
        return X
```

A.11 ResNet18 with GAT

```
* Resnet-18 30 lr = 0.001

class GraphCNN_Resnet18(nn.Module):
    def __init__(self,k, num_classes):
        super().__init__()
        self.k = k
        self.num_classes = num_classes
        self.resnet = torchvision.models.resnet18(True)
        self.resnet._forward_impl = MethodType(
           _forward_impl,self.resnet)
        self.conv1 = torch_geometric.nn.GATConv(512,256)
        self.dropout = nn.Dropout(0.5)
        self.conv2 = torch_geometric.nn.GATConv(256,
           num_classes)
    def forward_features(self,X):
        # B * 512 * 7 * 7
        features = self.resnet(X)
```

```
#         print("ResNet-18 Features Shape:", features.
   shape)
#         print("Features: ", features)
        return features

    def convert_graph(self, x):
        b,c,h,w = x.shape
        device = x.device
        k = self.k
        x = torch.permute(x,(0,2,3,1)).reshape((b,h*w,c)
           )
        y = torch.cdist(x,x,2)  # [b, hw, hw]
        _,idx = torch.topk(y,k,-1)
        source = torch.arange(h*w,device=device).
           repeat_interleave(k).repeat(b)
        target = torch.flatten(idx)
        step = torch.arange(b,device=device).
           repeat_interleave(h*w*k)*h*w
        adj = torch.row_stack([source,target]) + step
        return x.reshape((b*h*w,c)), adj.long()

    def froward_gcn(self,X,adj):
        X = self.conv1(X,adj)
        X = self.dropout(X)
        X = self.conv2(X,adj)
        # (B*7*7, num_classes)
        return X

    def forward(self,X):
```

```
        batch = X.shape[0]
        X = self.forward_features(X)
        X,adj = self.convert_graph(X)
        X = self.froward_gcn(X,adj)
        X = torch.reshape(X,(batch, -1, self.num_classes
           ))
        X = torch.mean(X,1)
        return X
```

A.12 Function for the Training Loss- Test Accuracy Graph

```
def display_model_performance(loss_array, acc_array):
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(loss_array)
    plt.title('Training Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')

    plt.subplot(1, 2, 2)
    plt.plot(acc_array)
    plt.title('test Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')

    plt.tight_layout()
```

```
    plt.show()


def display_loss_accuracy(loss_array, acc_array):
    for epoch, (loss, acc) in enumerate(zip(loss_array,
       acc_array), 1):
        print(f"Epoch {epoch} - Loss: {loss:.4f} -
           Accuracy: {acc:.4f}")
```

A.13 Function for Receiver Operating Characteristic Curve

```
def plot_roc_curve(model_name, model, test_loader,
   num_classes):
    print(f"Model Name: {model_name}")
    model.eval()
    device = next(model.parameters()).device
    y_true = []
    y_scores = []

    with torch.no_grad():
        for data in test_loader:
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.
               to(device)

            # Convert inputs to HalfTensor
            inputs = inputs.type(torch.cuda.HalfTensor)
```

```
            y_true.extend(labels.cpu().numpy())
            outputs = model(inputs)
            y_scores.extend(outputs.cpu().numpy())

y_true = label_binarize(y_true, classes=np.arange(
    num_classes))
y_scores = np.array(y_scores)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(len(class_names)):
    fpr[i], tpr[i], _ = roc_curve(y_true[:, i],
        y_scores[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
plt.figure(figsize=(8, 6))
for i in range(len(class_names)):
    plt.plot(fpr[i], tpr[i], label=f'{class_names[i
        ]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for {} Model'.format(model_name
    ))
```

```
    plt.legend(loc="lower right")
    plt.show()



@torch.no_grad()
def test_with_predictions(model, test_loader):
    model.eval()
    model.to(device)
    model.type(dtype)

    all_labels = []
    all_predictions = []

    for data in test_loader:
        images, labels = data
        images = images.type(dtype).to(device)

        outputs = model(images)
        _, predicted = torch.max(outputs.data, dim=1)

        all_labels.extend(labels.numpy())
        all_predictions.extend(predicted.cpu().numpy())

    return all_labels, all_predictions
```

A.14 Instantiating ResNet18 Model and Visualization

```
model_Resnet18 = GraphCNN_Resnet18(k=8,num_classes=4)


loss_array_Resnet18, acc_array_Resnet18 = train(
   model_Resnet18, 100, train_loader, test_loader, lr
   =0.001)


display_loss_accuracy(loss_array_Resnet18,
   acc_array_Resnet18)

display_model_performance(loss_array_Resnet18,
   acc_array_Resnet18)

acc_Resnet18 = test(model_Resnet18, test_loader)


print("Test Accuracy: ", acc_Resnet18)

# Evaluate the model and get labels and predictions
true_labels, pred_labels = test_with_predictions(
   model_Resnet18, test_loader)

# Compute confusion matrix
conf_matrix_Resnet18 = confusion_matrix(true_labels,
   pred_labels)
```

```
# Compute classification report
class_report_Resnet18 = classification_report(
   true_labels, pred_labels)


print("Confusion Matrix:")
print(conf_matrix_Resnet18)

print("\nClassification Report:")
print(class_report_Resnet18)

# Plot confusion matrix heatmap
plt.figure(figsize=(10,8))
plt.title("Confusion Matrix Heatmap - Resnet-18")
sns.heatmap(conf_matrix_Resnet18, annot=True, fmt="d",
            xticklabels=train_dataset.classes,
               yticklabels=train_dataset.classes)
plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

class_names = ['glioma_tumor', 'meningioma_tumor', '
   no_tumor', 'pituitary_tumor']
plot_roc_curve("GNN-ResNet18", model_Resnet18,
```

```
    test_loader, num_classes=4)
```

A.15 Instantiating DenseNet121 Model and Visualization

```
model_Densenet121 = GraphCNN_Densenet121(k=8,num_classes
    =4)


loss_array_Densenet121, acc_array_Densenet121 = train(
    model_Densenet121, 100, train_loader, test_loader, lr
    =0.0001)


display_loss_accuracy(loss_array_Densenet121,
    acc_array_Densenet121)

display_model_performance(loss_array_Densenet121,
    acc_array_Densenet121)

acc_Densenet121 = test(model_Densenet121, test_loader)


print("Test Accuracy: ", acc_Densenet121)

# Evaluate the model and get labels and predictions
true_labels, pred_labels = test_with_predictions(
    model_Resnet18, test_loader)
```

```
# Compute confusion matrix
conf_matrix_Densenet121 = confusion_matrix(true_labels,
   pred_labels)

# Compute classification report
class_report_Densenet121 = classification_report(
   true_labels, pred_labels)

print("Confusion Matrix:")
print(conf_matrix_Densenet121)

print("\nClassification Report:")
print(class_report_Densenet121)

# Plot confusion matrix heatmap
plt.figure(figsize=(10,8))
plt.title("Confusion Matrix Heatmap - Densenet-121")
sns.heatmap(conf_matrix_Densenet121, annot=True, fmt="d
   ",
            xticklabels=train_dataset.classes,
               yticklabels=train_dataset.classes)
plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

```
class_names = ['glioma_tumor', 'meningioma_tumor', '
   no_tumor', 'pituitary_tumor']
plot_roc_curve("GNN-Densenet121", model_Densenet121,
   test_loader, num_classes=4)
```

A.16 Instantiating ResNet18 Model and Visualization

```
model_VGG16 = GraphCNN_VGG16(k=8,num_classes=4)


loss_array_VGG16, acc_array_VGG16 = train(model_VGG16,
   100, train_loader, test_loader, lr=0.001)


display_loss_accuracy(loss_array_VGG16, acc_array_VGG16)



display_model_performance(loss_array_VGG16,
   acc_array_VGG16)

acc_VGG16 = test(model_VGG16, test_loader)


print("Test Accuracy: ", acc_VGG16)
```

```
# Evaluate the model and get labels and predictions
true_labels, pred_labels = test_with_predictions(
   model_Resnet18, test_loader)

# Compute confusion matrix
conf_matrix_VGG16 = confusion_matrix(true_labels,
   pred_labels)

# Compute classification report
class_report_VGG16 = classification_report(true_labels,
   pred_labels)

print("Confusion Matrix:")
print(conf_matrix_VGG16)

print("\nClassification Report:")
print(class_report_VGG16)

# Plot confusion matrix heatmap
plt.figure(figsize=(10, 8))
plt.title("Confusion Matrix Heatmap - VGG16")
sns.heatmap(conf_matrix_VGG16, annot=True, fmt="d",
            xticklabels=train_dataset.classes,
               yticklabels=train_dataset.classes)
plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.xticks(rotation=45, ha="right")
plt.yticks(rotation=0)
```

```
plt.tight_layout()
plt.show()

class_names = ['glioma_tumor', 'meningioma_tumor', '
   no_tumor', 'pituitary_tumor']
plot_roc_curve("GNN-VGG16", model_VGG16, test_loader,
   num_classes=4)
```

A.17 Accuracies Table

```
# Test accuracies obtained for each model
accuracies = [
    ("GraphCNN_VGG16", acc_VGG16),
    ("GraphCNN_Densenet121", acc_Densenet121),
    ("GraphCNN_Resnet18", acc_Resnet18),
]

# Print the accuracies in a visually appealing table (
   fancy_grid format)
table_str = tabulate(accuracies, headers=["Model", "Test
    Accuracy"], tablefmt="fancy_grid")

# Display the table
print(table_str)


train_dataset.classes
```

A.18 Predictions

```
class_names = ['glioma_tumor', 'meningioma_tumor', '
   no_tumor', 'pituitary_tumor']
import torch
from PIL import Image
import torchvision.transforms as transforms

# Add the dataset before running this cell (explained
   above)
load_path = "/kaggle/input/data-set/models/
   model_VGG_complete.pth"
model_VGG = torch.load(load_path)
model_VGG.eval()

from PIL import Image
import matplotlib.pyplot as plt

# Load the image
image_path = "/kaggle/input/brain-tumor-classification-
   mri/Training/glioma_tumor/gg (3).jpg"
image = Image.open(image_path)

# Display the original image
plt.figure()
plt.imshow(image)
plt.show()
from PIL import Image
```

```
import torchvision.transforms as transforms

#  Load the image
image = Image.open(image_path)

# Apply the same transformations used during training
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5,
       0.5))
])

image_tensor = data_transform(image)

image_tensor = image_tensor.unsqueeze(0)

# Ensure model is in evaluation mode and on the
   appropriate device
model_VGG.eval()
model_VGG.to(device)

# Get the model's prediction (no gradients needed)
with torch.no_grad():
    output = model_VGG(image_tensor.type(dtype).to(
       device))
    predicted_class_index = int(torch.argmax(output))
```

```
# Access the corresponding class name
predicted_class_name = class_names[predicted_class_index
   ]

print("Predicted Class:", predicted_class_name)
```

A.19 Feature Visualization

```
import torch
import torchvision.transforms as transforms
from PIL import Image

def preprocess_image(image_path):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5,
            0.5))
    ])
    image = Image.open(image_path)
    image = transform(image).unsqueeze(0)  # Add batch
       dimension
    image = image.type(torch.cuda.HalfTensor)  # Convert
        to half precision
    return image

def extract_features(model, image_path):
```

```
    image_tensor = preprocess_image(image_path)
    model.eval()
    with torch.no_grad():
        features = model.forward_features(image_tensor)
    return features

image_path = "/kaggle/input/brain-tumor-classification-
   mri/Testing/glioma_tumor/image(100).jpg"

# Extract features using VGG16 model
vgg16_features = extract_features(model_VGG16,
   image_path)
VGG16_Features_Shape = vgg16_features.shape
print("VGG16 Features :", vgg16_features)

# Extract features using DenseNet121 model
densenet121_features = extract_features(
   model_Densenet121, image_path)
DenseNet121_Features_Shape = densenet121_features.shape
print("DenseNet121 Features :", densenet121_features)



# Extract features using ResNet18 model
resnet18_features = extract_features(model_Resnet18,
   image_path)
ResNet18_Features_Shape = resnet18_features.shape
print("ResNet18 Features :", resnet18_features)
```

```
# Test accuracies obtained for each model
Features = [
    ("ResNet18 Features", ResNet18_Features_Shape),
    ("DenseNet121 Features", DenseNet121_Features_Shape)
       ,
    ("VGG16 Features", VGG16_Features_Shape),
]

# Print the accuracies in a visually appealing table (
   fancy_grid format)
table_Features = tabulate(Features, headers=["Model", "
   Test Accuracy"], tablefmt="fancy_grid")

# Display the table
print(table_Features)

def preprocess_image_visualize(image_path):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5,
            0.5))
    ])
    image = Image.open(image_path)
    image = transform(image).unsqueeze(0).to(torch.
       float16).cuda()  # Convert to half precision and
       move to GPU
    return image
```

```
def extract_features_visualize(model, image_path):
    model.eval()
    with torch.no_grad():
        image_tensor = preprocess_image_visualize(
           image_path)
        features = model.forward_features(image_tensor)
    return features

def reduce_features(features, n_components=2):
    # Flatten the features to (n_samples, n_features)
    flattened_features = features.squeeze().cpu().numpy
       ()
    flattened_features = flattened_features.reshape(
       flattened_features.shape[0], -1)

    pca = PCA(n_components=n_components)
    reduced_features = pca.fit_transform(
       flattened_features)
    return reduced_features

import torch
import torchvision.transforms as transforms
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Extract features using VGG16 model
vgg16_features_visualize = extract_features_visualize(
```

```
    model_VGG16, image_path)
reduced_vgg16_features = reduce_features(
    vgg16_features_visualize)


# Extract features using DenseNet121 model
densenet121_features_visualize =
    extract_features_visualize(model_Densenet121,
    image_path)
reduced_densenet121_features = reduce_features(
    densenet121_features_visualize)

# Extract features using ResNet18 model
resnet18_features_visualize = extract_features_visualize
    (model_Resnet18, image_path)
reduced_resnet18_features = reduce_features(
    resnet18_features_visualize)

# Plotting
plt.figure(figsize=(12, 5))

plt.subplot(1, 3, 1)
plt.scatter(reduced_vgg16_features[:, 0],
    reduced_vgg16_features[:, 1])
plt.title('VGG16 Features')

plt.subplot(1, 3, 2)
plt.scatter(reduced_densenet121_features[:, 0],
    reduced_densenet121_features[:, 1])
```

```
plt.title('DenseNet121 Features')

plt.subplot(1, 3, 3)
plt.scatter(reduced_resnet18_features[:, 0],
   reduced_resnet18_features[:, 1])
plt.title('ResNet18 Features')

plt.tight_layout()
plt.show()
```