CSC3109 Machine Learning Project Report

Woon Jun Wei (2200624)[1], Benjamin Loh Choon How (2201590)[1], Low Hong Sheng Jovian (2203654)[1], Ong Zi Xuan Max (2200717)[1], and Cleon Tay Shi Hong (2200649)[1]

[1]Team 19

June 26, 2024

Contents

1 Introduction

Machine learning (ML) has become a transformative technology, driving innovation across numerous fields such as healthcare[1], healthcare research [2] and climate science[3]. Within ML, deep learning has emerged as a particularly powerful subset, enabling machines to perform tasks that require human-like perception and decision-making. This project aims to harness deep learning techniques to tackle real-world challenges in healthcare, specifically focusing on the classification of brain MRI images. By applying deep learning methodologies to this task, the project bridges theoretical concepts with practical applications, thereby advancing our understanding and capabilities in advanced ML techniques.

Magnetic resonance imaging (MRI) is a non-invasive imaging technique that utilizes strong magnetic fields and radio waves to generate detailed images of the body's internal structures. MRI is particularly effective for imaging soft tissues, making it an invaluable tool in medical diagnostics, including the evaluation of brain conditions. Brain MRI provides high-resolution images that are crucial for identifying and assessing abnormalities within the brain.

1.1 Challenges in Accurate Classification of Brain MRI Images

Given the critical role of MRI in diagnosing and monitoring various brain conditions, accurate interpretation of brain MRI images is essential. This process is complex and time-consuming, requiring expert knowledge and experience. Accurate classification of brain MRI images is vital for timely diagnosis and treatment planning, as different types of brain tumors have distinct characteristics and treatment strategies. Patients may receive imaging studies as part of their routine care or to investigate specific symptoms, such as headaches, seizures, or cognitive changes [4]. In many cases, the radiologist's report on the MRI findings is critical for guiding clinical decisions and patient management. This project focuses on leveraging deep learning techniques to automate and enhance the classification of brain tumors from MRI images, aiming to support radiologists and improve clinical outcomes.

Accurate classification of brain MRI images is a significant challenge in the medical field. Brain MRI scans are essential for diagnosing and monitoring various neurological conditions, yet the complexity and variability of these images make classification difficult. Misclassifications can lead to delays in diagnosis and treatment, impacting patient outcomes[5]. Thus, there is a critical need for robust classification models that can reliably distinguish between different types of brain tumors.

As the availability of MRI data increases, the demand for automated tools to assist radiologists in interpreting these images grows. Current manual methods are time-consuming and prone to human error, highlighting the need for efficient, automated solutions [6]. This project seeks to address this problem by developing, evaluating, and comparing multiple deep learning models to achieve high accuracy in classifying brain MRI images. By leveraging deep learning techniques, the project aims to reduce the workload on healthcare professionals, enhance diagnostic efficiency, and improve patient care through early detection and timely medical interventions.

1.2 Enhancing Diagnostic Accuracy and Efficiency in Brain Tumor Detection

The primary goal of this project is to develop deep learning models that support radiologists and physicians in the prompt and accurate diagnosis of brain tumors, thus enabling quicker and more effective treatment. This effort involves applying and advancing state-of-the-art deep learning techniques to create models proficient in the precise classification of brain MRI scans. These models aim to decrease the time required for diagnosis and increase diagnostic accuracy, contributing significantly to faster and more effective medical treatments. By extending advanced deep learning methodologies, the project seeks to aid healthcare professionals in reducing diagnostic time and enhancing the accuracy of diagnoses, ultimately facilitating early intervention and improving patient outcomes. Through this endeavor, we aim to bridge the gap between theoretical advancements in machine learning and their practical applications in medical diagnostics, thereby supporting timely and effective medical interventions.

2 Existing Approaches to Brain MRI Classification Using Deep Learning

The integration of deep learning into the medical field has revolutionized diagnostic methodologies, particularly in the classification of brain MRI images. As brain tumors present diverse morphological characteristics that are

detectable through MRI, the precision of image analysis can significantly influence diagnostic and treatment outcomes. This literature review examines current deep learning approaches that enhance the accuracy and efficiency of brain tumor diagnoses from MRI scans, focusing on the evolution of model architectures, data preprocessing innovations, and the strategic application of transfer learning.

2.1 Data preprocessing techniques

Data preprocessing is a crucial step in enhancing the accuracy of brain MRI classification models. Techniques such as data augmentation [7] and cropping along contours of brain MRIs play significant roles in improving model training and performance. Data augmentation involves creating modified versions of the original images through transformations such as rotation, scaling, and flipping. This process helps in increasing the diversity of the training dataset, which in turn aids in preventing overfitting and improves the generalization capabilities of the models [8].

Cropping along the contours of the brain in MRI images helps in focusing the model's attention on the most relevant areas, thereby reducing noise and improving classification accuracy. By eliminating irrelevant parts of the image, this technique ensures that the model learns more effectively from the essential features of the brain structure [9].

In addition, advanced preprocessing methods like Generative Adversarial Networks (GANs) for augmented data generation have shown to significantly improve the robustness and accuracy of brain MRI classification models by generating high-quality synthetic data to augment the small training set [10].

These preprocessing techniques collectively enhance the quality and quantity of the training data, leading to more accurate and reliable brain MRI classification models.

2.2 Convolutional Neural Networks (CNNs) and Variants

Convolutional Neural Networks (CNNs) have been extensively used for brain MRI classification. A notable approach is the use of ResNet-18 architecture for differentiating MRI sequence types, achieving an impressive accuracy of 97.9% on test sets. This model demonstrates the capability of CNNs in handling complex MRI data efficiently[11].

2.2.1 U-Net and Variants

U-Net and its numerous variants have established themselves as fundamental architectures in the realm of brain MRI segmentation, owing to their exceptional ability to effectively capture spatial information. The U-Net architecture, initially designed for biomedical image segmentation, has been widely applied to brain tumor segmentation in MRI images, achieving remarkable accuracy and robustness in detecting tumors [12]–[14].

To enhance the performance of U-Net in medical image analysis, several adaptations have been proposed. Abd-Ellah et al. [13] introduced the Two Parallel Cascaded U-Nets with an Asymmetric Residual (TPCUAR-Net) architecture. This innovative two-stage detection and segmentation model demonstrated high accuracy in brain tumor tasks, highlighting the significant potential of U-Net variants.

Similarly, Imtiaz et al. [12] developed a modified U-Net architecture tailored for pixel-level segmentation using a relatively small dataset of 322 brain tumor MRI images. Their model achieved impressive accuracy and robustness, further validating the efficacy of U-Net in medical image analysis.

Ding et al. [14] proposed the SLF-UNet architecture, which integrates U-Net with a spatial attention mechanism to enhance segmentation precision. The SLF-UNet model achieved significant improvements in tumor segmentation accuracy, highlighting the advantages of incorporating attention mechanisms into the U-Net framework.

Zhou et al. [15] presented U-Net++, a novel architecture that incorporates dense skip connections and deep supervision to bolster segmentation performance. The U-Net++ model demonstrated superior performance compared to traditional U-Net architectures, showcasing the potential of advanced U-Net variants in medical image segmentation.

Eff-U-Net, introduced by Baheti et al. [16], combines the U-Net architecture with EfficientNets for the downsampling path. This hybrid model achieved state-of-the-art performance in various medical segmentation tasks, such as Pneumonia Detection [17] and Brain Tumor Segmentation [18], highlighting the efficacy of integrating EfficientNets into the U-Net framework.

These studies collectively illustrate the versatility and effectiveness of U-Net and its variants in brain tumor segmentation, solidifying their role as invaluable tools in the field of medical image analysis.

2.3 Transfer Learning in Brain MRI Classification

Transfer learning has emerged as a powerful technique for enhancing the performance of deep learning models, particularly in scenarios with limited labeled data. This method involves leveraging pre-trained models on large-scale datasets and adapting them to specific tasks such as brain MRI classification. The underlying principle is that features learned from a broad dataset can be transferred to a new task, reducing the need for extensive labeled data and computational resources.

Several transfer learning architectures, including InceptionV3, VGG19, ResNet-50, and MobileNet, have been applied to brain MRI classification for various applications, such as brain tumor diagnosis. For instance, MobileNet has demonstrated exceptional performance, achieving an accuracy of 99.60% in brain tumor classification tasks, underscoring the potential of transfer learning in achieving high accuracy with relatively small datasets [19]. This success can be attributed to the model's ability to effectively transfer features learned from natural image datasets to medical imaging tasks.

In the context of whole-brain functional MRI (fMRI) data, transfer learning has been employed to tackle the challenges posed by small sample sizes and high dimensionality. Studies have shown that models utilizing transfer learning exhibit improved performance compared to those trained from scratch. This improvement is evident in the enhanced ability to capture relevant patterns in complex, high-dimensional fMRI data, thus facilitating more accurate brain function analysis [20].

Moreover, transfer learning has proven beneficial in brain tumor detection. For example, a study applied a pre-trained deep learning model, initially trained on a large dataset of natural images, to the task of brain MRI classification. The fine-tuned model achieved comparable performance with smaller, domain-specific datasets, demonstrating that transfer learning can effectively bridge the gap between different domains and reduce the dependency on large medical imaging datasets [21].

2.3.1 ResNet-50

Among these architectures, ResNet-50 has gained significant attention due to its unique ability to mitigate the vanishing gradient problem through the use of residual learning. ResNet-50, a 50-layer deep convolutional neural network, utilizes shortcut connections that allow the network to learn residual functions with reference to the layer inputs, which substantially improves the convergence rate of the network [22].

In the domain of brain MRI classification, ResNet-50 has been extensively employed for its robustness and ability to handle complex image data. Its deep architecture is particularly effective in extracting intricate features from MRI scans, which is crucial for distinguishing between different types of brain pathologies. For example, in brain tumor classification tasks, ResNet-50 has been adapted to differentiate between gliomas, meningiomas, and pituitary tumors with high accuracy [23]. The model's depth and residual connections enable it to capture subtle differences in brain tissue characteristics that are critical for accurate diagnosis.

Research has demonstrated that ResNet-50, when fine-tuned on brain MRI datasets, can significantly outperform traditional machine learning techniques. For instance, in a study focusing on brain tumor classification, ResNet-50 coupled with the convolutional block attention module (CBAM) achieved an accuracy rate of 99.43%, highlighting its effectiveness in medical image classification [24]. The pre-trained nature of ResNet-50 allows it to leverage feature representations from extensive datasets like ImageNet, which can be adapted to the specific features found in brain MRI images, thereby enhancing the model's generalization ability and diagnostic accuracy.

2.3.2 Xception

Extreme Inception, also known as Xception, a convolutional neural network that is 71 layers deep with a modified depth-wise separable convolution, proposed by Chollet Francis [25], and is build upon the foundations of Inception model architecture. Xception slightly outperformed InceptionV3 on the ImageNet dataset and it has the same number of model parameters as Inception. The Xception has replaced the inception module with a modified depth-wise separable convolution on the Inception model.

To leverage deep learning architecture for this classification task, a study [26] showed the use of the Xception model, highlighting its efficient depth-wise separable convolutions, which enhance performance and adaptability. The model's application in medical imaging particularly for multi-class classification tasks, demonstrates its high accuracy and efficiently.

An IEEE research [27] has provided results in supporting the efficacy of utilizing Xception transfer learning model in the crucial medical imaging domain to detect brain tumour diseases. The model's performance is essential, as it achieves a high accuracy of 96% through the training process of 40 epochs. This outcome is achieved by implementing a small batch size number of eight and various optimizer. In another recent IEEE study [28] further explored the application of the Xception model for brain tumour classification. By leveraging advanced data augmentation techniques and transfer learning, the researchers achieved remarkable accuracy rates, demonstrating the model's efficiency in distinguishing between different types of brain tumours. This study highlights the continued advancements and applicability of the Xception model in enhancing medical image analysis and classification accuracy.

To sum it up, the Xception model excels in medical imaging tasks, particularly brain tumour classification tasks, due to its innovative depth-wise separable convolutions which enhance computational efficiency and accuracy. Its proven high performance underscore its value as a tool in complex image classification tasks.

2.3.3 DenseNet121

DenseNet121, a variation of the Densely Connected Convolutional Networks (DenseNet)[29], has garnered significant attention in the field of deep learning due to its innovative architecture and superior performance in image classification tasks. DenseNet121 is characterized by its dense connectivity pattern, where each layer receives input from all preceding layers and passes its own feature maps to all subsequent layers. This design promotes feature reuse, mitigates the vanishing gradient problem, and improves network efficiency by reducing the number of parameters compared to traditional convolutional networks .

The application of DenseNet121 spans various domains, with notable success in medical imaging. For instance, studies have demonstrated its efficacy in detecting diseases from chest X-rays[30] and classifying skin cancer[31] from dermatological images. The model's ability to leverage features from multiple layers allows it to capture intricate patterns and textures in medical images, leading to higher diagnostic accuracy. Comparative studies have shown that DenseNet121 outperforms other convolutional neural networks (CNNs) like ResNet and VGGNet in these tasks, highlighting its robustness and precision .

Despite its advantages, DenseNet121 faces challenges, particularly in computational cost and memory usage due to its dense connections. Researchers have proposed various modifications to address these issues, such as pruning techniques[32] and hybrid models that combine DenseNet121 with other architectures to enhance efficiency without compromising accuracy. Future research is directed towards optimizing DenseNet121[33] for deployment in resource-constrained environments and exploring its potential in other areas such as natural language processing and autonomous driving.

2.3.4 Vision Transformers

Transformers, first proposed by Vaswani et al. for natural language processing (NLP) tasks, are a typical attention-based deep learning model [34]. Vision Transformers (ViTs) have emerged as a powerful tool in medical imaging, particularly for classifying brain MRI images to detect tumors. Inspired by the success of transformers in natural language processing (NLP), where they have replaced RNNs and CNNs due to their ability to model long-range dependencies between tokens, ViTs adapt this architecture for image analysis. Initially proposed by Dosovitskiy et al., ViTs divide input images into non-overlapping patches and model the global relations between these patches using multiple standard transformer layers [35]. This unique approach allows for a comprehensive evaluation of spatial relationships and features within brain scans.

ViTs utilize self-attention mechanisms to analyze different image segments simultaneously, enhancing the detection of subtle variances and complex patterns indicative of tumors. This method offers significant advantages in accuracy and efficiency over traditional CNN and RNN-based methods, despite higher computational costs. By focusing on critical image areas, ViTs improve the precision of tumor detection and classification, making them invaluable in diagnostic radiology.

Several studies have highlighted the efficacy of ViTs in medical imaging. Research conducted by Tummala et al. [36] reveals that ViT consistently achieve high classification accuracies, surpassing 97% in both validation and testing phases. This impressive accuracy persists across a range of hyperparameter configurations, including optimizer, learning rate (lr), number of epochs (ne), and minibatch size (mbs), underscoring the robustness of ViT models in various operational conditions.

Additionally, a study by Asiri et al. assessed five pre-trained ViT models, namely R50-ViT-l16, ViT-l16, ViT-l32, ViT-b16, and ViT-b32, for brain tumor classification. The results revealed that the ViT-b32 model achieved the highest accuracy of 98.24%, surpassing existing methods [37].

Furthermore, Diker's analysis of transfer learning models indicated that pre-trained ViT models surpassed CNN models such as VGG-16 and ResNet-50, which achieved accuracies of 96% and 88% respectively [38].

Overall, the application of Vision Transformers in the classification of brain MRI images for tumor detection highlights their advanced capabilities and potential to enhance diagnostic accuracy in medical imaging.

2.4 Conclusion

Deep learning technologies have significantly transformed brain MRI classification, providing robust, accurate, and efficient diagnostic tools. Advances in model architectures, transfer learning techniques, optimization strategies, and preprocessing methods continue to enhance the accuracy and reliability of these models. This progress highlights the transformative impact of deep learning on medical imaging and underscores the importance of ongoing research and innovation. By pushing the boundaries of these technologies, the medical community can anticipate more effective diagnostic processes, leading to improved patient care and outcomes. Sustained research and development are crucial to fully harnessing the potential of deep learning in healthcare.

3 Dataset Exploration

The dataset, referred to as `dataset_19`, comprises a collection of MRI images organized into four subfolders, each representing a distinct class of brain tumors. Each subfolder contains 120 images, resulting in a well-balanced dataset with a total of 480 images. This balance is essential for training deep learning models, as it mitigates the risk of bias towards any particular class, ensuring that the model learns to distinguish among all categories effectively.

The dataset includes images representing four types of brain tumors: glioma, meningioma, pituitary tumors, and cases with no tumors. Specifically, it consists of 120 images of glioma tumors, 120 images of meningioma tumors, 120 images of pituitary tumors, and 120 images with no tumors.

Each image in the dataset captures the brain from different perspectives, including axial, coronal, and sagittal views. These diverse views are crucial as they provide comprehensive information about the tumor's location, size, and relation to surrounding structures. Axial views offer a horizontal slice of the brain, coronal views provide a frontal slice, and sagittal views give a side slice. Including these various perspectives ensures that the model can learn robust features and improve its ability to generalize across different cases.

The distribution of images across these classes is depicted in Figure 1. This visual representation underscores the dataset's balance and uniformity, further highlighting its suitability for developing a deep learning model for brain tumor classification.

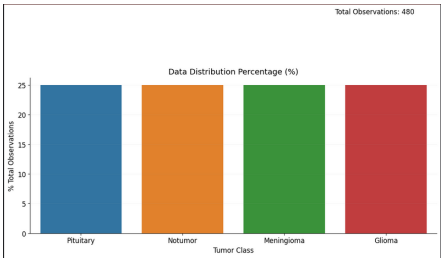
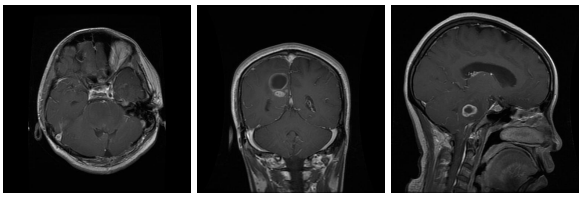


Figure 1: Distribution of images in `dataset_19`.

Overall, `dataset_19` provides a diverse and balanced set of images, essential for training an accurate and reliable deep learning model. The inclusion of different tumor types and multiple views per case enriches the dataset, making it a valuable resource for the task of brain tumor classification.

3.1 Tumor Types

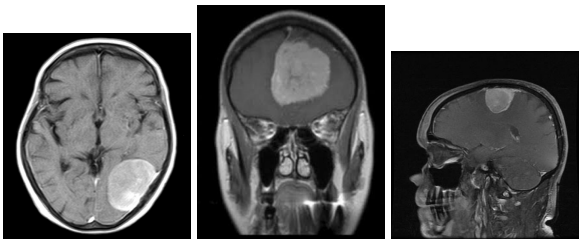
The dataset encompasses four distinct classes of brain tumors, each representing a unique type of pathology. A thorough understanding of these tumor types is vital for developing a model capable of accurately classifying them.



(a) Axial View (b) Coronal View (c) Sagittal View

Figure 2: Glioma Tumor: Axial, Coronal, and Sagittal Views

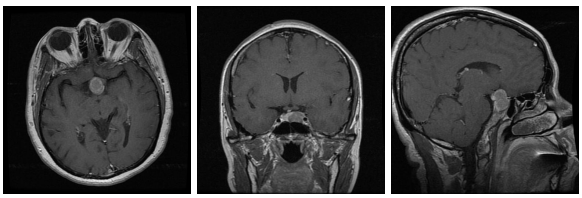
Gliomas are the most prevalent primary brain tumors, originating in the glial cells of the brain. These tumors can be further classified into subtypes such as astrocytoma, oligodendroglioma, and glioblastoma. Gliomas are characterized by their infiltrative nature and high recurrence rates, often presenting with varying shapes and irregular borders. The complexity and variability in glioma morphology make them particularly challenging to classify, underscoring the importance of incorporating detailed MRI views to capture their diverse appearances [13].



(a) Axial View (b) Coronal View (c) Sagittal View

Figure 3: Malignoma Tumor: Axial, Coronal, and Sagittal Views

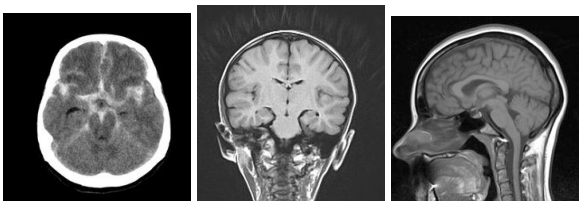
Meningiomas, on the other hand, are typically benign tumors arising from the meninges, the protective layers surrounding the brain and spinal cord. These tumors are usually slow-growing and well-defined, which often makes them easier to surgically remove compared to other types. The well-defined nature of meningiomas benefits from clear imaging perspectives, which assist in their precise identification and classification.



(a) Axial View (b) Coronal View (c) Sagittal View

Figure 4: Pituitary Tumor: Axial, Coronal, and Sagittal Views

Pituitary tumors develop in the pituitary gland, a small but crucial organ located at the base of the brain. These tumors can be either benign or malignant. Given the pituitary gland's critical role in hormone regulation, accurate identification of pituitary tumors is essential. Detailed MRI views help in assessing the tumor's impact on the surrounding brain structures and the gland itself.



(a) Axial View (b) Coronal View (c) Sagittal View

Figure 5: No Tumor: Axial, Coronal, and Sagittal Views

Lastly, the class labeled 'No Tumor' includes images of the brain without any detectable tumors. This class is crucial for training the model to accurately distinguish between pathological and non-pathological images, thereby reducing false positives in diagnoses.

In summary, each tumor type in the dataset presents unique challenges and characteristics that must be considered in model development. The inclusion of diverse MRI views, such as axial, coronal, and sagittal perspectives, provides a comprehensive understanding of the tumor morphology and spatial relationships, thereby enhancing the model's ability to accurately classify and differentiate between various brain tumor types.

3.2 Image Sizes

The overall average size of the images is approximately $[453.36, 450.83]$ pixels. The smallest image size is $(198, 150)$ pixels, and the largest image size is $(1075, 890)$ pixels. The mean, median, and mode sizes across the dataset are consistent at $[512, 512]$ pixels, suggesting a central tendency around these dimensions.

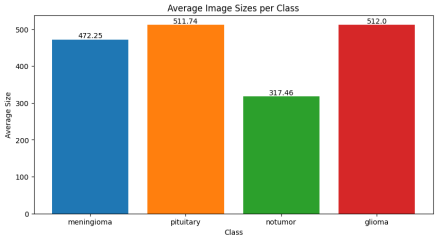


Figure 6: Distribution of brain image sizes in dataset_19.

Statistic	X	Y
Mean Size	453.3625	450.83125
Median Size	512	512
Mode Size	512	512
Standard Deviation of Size	124.1067	131.5980
Variance of Size	15402.4644	17318.0236
Smallest Size	198	150
Largest Size	1075	890

Table 1: Overall Statistics of Brain Image Sizes

Class-specific statistics reveal notable variations in image sizes. The *meningioma* class images range from $(223, 200)$ to $(650, 591)$ pixels, with a mean size of approximately $[472.25, 466.41]$ pixels. The *pituitary* class exhibits sizes from $(256, 256)$ to $(903, 721)$ pixels, with a mean size of around $[511.74, 510.54]$ pixels. The *notumor* class ranges from $(198, 150)$ to $(1075, 890)$ pixels, with a mean size of approximately $[317.46, 314.38]$ pixels. In contrast, all images in the *glioma* class are consistently sized at $[512, 512]$ pixels.

This variability underscores the importance of size normalization during preprocessing. While *glioma* class images are uniform, other classes exhibit significant variation, which, if unaddressed, could introduce biases or inconsistencies during training. Proper preprocessing, including size normalization, is essential for several reasons. Neural networks require fixed-size inputs for batch processing, and resizing ensures consistent feature representation across the dataset, facilitating better model learning. Consistent image sizes also reduce computational load and memory requirements, enhancing training efficiency. Furthermore, standardized inputs contribute to improved model convergence and accuracy by providing a uniform set of images for the network to learn from.

Statistic	Meningioma	Pituitary	Notumor	Glioma
Mean Size (X)	472.25	511.7417	317.4583	512
Mean Size (Y)	466.4083	510.5417	314.3750	512
Median Size (X)	512	512	236	512
Median Size (Y)	512	512	236	512
Mode Size (X)	512	512	236	512
Mode Size (Y)	512	512	236	512
Standard Deviation of Size (X)	100.7078	43.5696	154.5843	0
Standard Deviation of Size (Y)	108.4743	30.7834	174.3213	0
Variance of Size (X)	10142.0708	1898.3083	23896.3149	0
Variance of Size (Y)	11766.6749	947.6149	30387.9010	0
Smallest Size (X)	223	256	198	512
Smallest Size (Y)	200	256	150	512
Largest Size (X)	650	903	1075	512
Largest Size (Y)	591	721	890	512

Table 2: Class-Specific Statistics of Brain Image Sizes

It is important to note that the image sizes fed into each model will differ, as each pretrained model is optimized for specific image dimensions. This aspect will be elaborated in subsequent sections. In conclusion, resizing images to a uniform dimension during preprocessing is a critical step in developing a robust and efficient deep learning model for brain tumor classification. This step ensures smooth training and significantly impacts the model's overall performance and generalization capability.

4 Data Preprocessing

Data preprocessing is a vital step in the data mining process, particularly in the context of deep learning for image classification. This step involves techniques such as image augmentation, affine transformations, and resizing, which are essential for enhancing model performance. For this project, preprocessing is applied to the provided dataset (dataset_19) to ensure it is suitable for the brain tumor classification task. The specific steps involved in the preprocessing of this dataset are outlined below.

4.1 Image Cropping and Enhancement

A fundamental step in our preprocessing workflow involves meticulous image cropping and enhancement to focus exclusively on the brain, thereby eliminating irrelevant background elements. This sequence of procedures ensures that the model concentrates on significant features, enhancing the effectiveness of the classification task. Below is a detailed outline of the improved process:

4.1.1 Conversion and Blurring

Initially, techniques adapted from previous research [39] involve converting the images to grayscale. This conversion reduces computational complexity and highlights structural attributes by removing the color dimension. Subsequently, a Gaussian blur with a 3×3 kernel, as detailed in current solutions [40], is applied to the grayscale images. This blurring process effectively softens image noise, facilitating more reliable edge detection in the subsequent processing stages.

4.1.2 Thresholding and Morphology

The grayscale images are then subjected to adaptive thresholding, using a threshold value of 45 to convert them into binary images. This method, adapted from existing research [41], effectively separates brain tissue from the

background. The resulting binary images undergo a series of morphological operations, including two iterations each of erosion and dilation, to remove small noise regions and clarify the structural outline of the brain.

4.1.3 Contour Detection and Cropping

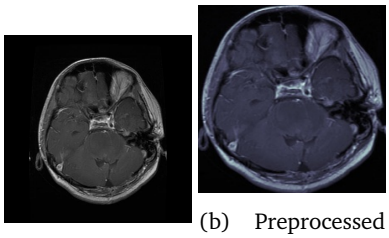
Using the thresholded images, contours are extracted with an external retrieval mode. The largest contour is presumed to delineate the brain's boundary. The extreme points of this contour (left, right, top, and bottom) are identified, and the image is cropped to a bounding rectangle that includes a configurable pixel margin to ensure the brain is isolated without being clipped as described in this study [42]. This precise cropping method ensures that only the brain region is retained for further processing [41].

4.1.4 Further Processing

Each cropped image is first converted back to grayscale to focus on intensity variations, which are crucial for medical imaging. To enhance image quality, bilateral filtering is applied, which reduces noise while preserving important edge details, thus improving clarity and highlighting critical features. To further enhance visual contrast, the images are mapped to a 'bone' color scheme, which helps in better differentiation of structures.

Finally, the images are resized to dimensions that are optimal for each specific classification model. This resizing ensures uniformity within the dataset and maximizes computational efficiency by tailoring the pixel dimensions to the requirements of different models. This approach allows for the accommodation of varying input size specifications, ensuring that each model can effectively process the images without compromising performance or accuracy [43].

These steps, as implemented in our preprocessing workflow and illustrated in Figure 7, significantly reduce computational load and enhance the clarity of critical features, which in turn improves the model's classification performance. This preprocessing method effectively prepares the dataset for further analysis, ensuring that the focus remains on the relevant anatomical structures necessary for accurate classification.



(a) Original Image (b) Preprocessed Image

Figure 7: Cropping the MRI image along its contour.

4.2 Data Augmentation

Following the initial preprocessing steps, we apply image augmentation techniques using the `ImageDataGenerator` class from the Keras library. Each model is fine-tuned with a subset of these techniques, which include horizontal flips, random rotations, and normalization. As referenced in [44], certain affine augmentations can be advantageous for brain tumor classification tasks. However, it is crucial to note that some augmentations, such as vertical flips, may be detrimental due to the potential misrepresentation of tumor locations.

Importantly, these augmentation techniques are applied exclusively to the training dataset, while the validation dataset remains unaltered. This strategy ensures that the model encounters a diverse array of augmented images during training, thereby enhancing its generalization capabilities without compromising the integrity of the evaluation process.

For each model evaluation, we provide a detailed commentary on the preprocessing and augmentation techniques employed, along with the rationale behind their selection. This comprehensive analysis elucidates how these techniques contribute to the overall performance and generalization capabilities of the models.

4.3 Data Splitting

After preprocessing and augmentation, the dataset is divided into training and validation sets, with the training set comprising 80% of the data and the validation set the remaining 20%. This distribution allows the model to be trained extensively while also being evaluated on a separate set of data to assess its performance, as depicted in Figure 8.

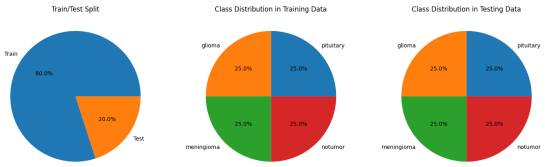


Figure 8: Splitting the dataset into training and validation sets.

4.4 Summary and Justifications

Given the relatively small size of dataset_19 (480 images in total), data augmentation is crucial for enhancing the dataset's variability and improving the model's generalization capabilities.

Training augmentation includes horizontal flipping, which is justified by the anatomical symmetry of the brain's hemispheres. This technique increases the diversity of the training data without misrepresenting tumor locations [44]. This is particularly important given the limited size of the dataset compared to larger datasets, such as those used in the BraTS competition.

Testing augmentation involves random rotations, as brain images can be rotated in various directions. This further increases the dataset's variability and aids in model training. These augmentation strategies are essential for compensating for the smaller dataset size by exposing the model to a wider variety of image orientations and perspectives. This approach helps create a more robust model capable of accurately classifying brain tumors despite the limited amount of original training data.

The images in Figure 32 show examples of the tumor images in the training and test sets after augmentation. The training set contains augmented images, while the test set remains unaltered. This ensures that the model is trained on a diverse range of images while being evaluated on a separate, unbiased dataset.

5 Model Selection and Training

In this section, we explore the model selection and training process for brain tumor classification. Various data mining techniques are employed to extract meaningful insights from the preprocessed data.

5.1 Overview

All models were trained on Google's Colab platform, utilizing the GPU runtime for faster training. The techniques employed include data augmentation, data splitting, and model selection, aimed at enhancing the accuracy and robustness of the brain tumor classification task.

5.2 Model Selection

Model selection is a critical step in developing an effective machine learning model for complex tasks such as brain tumor classification. The choice of model significantly impacts the accuracy, efficiency, and overall performance of the solution. We employed transfer learning with pretrained models and deep learning approaches like U-Net, which have demonstrated high performance in related tasks, including the BraTS (Brain Tumor Segmentation) Competition.

The BraTS Competition has set benchmarks for brain tumor segmentation, showcasing models and techniques that consistently achieve superior results. Models such as U-Net, VGG16, and EfficientNet are frequently employed by top teams due to their effectiveness in medical imaging tasks, making them ideal candidates for our project.

Transfer learning involves utilizing models pretrained on large datasets and fine-tuning them for specific tasks. This approach is particularly advantageous for small datasets, enabling the model to leverage the knowledge acquired during pretraining. Our selection included InceptionV3, ResNet50, U-Net with an EfficientNet backbone, Xception, DenseNet and Vision Transformer (ViT), chosen for their proven performance and efficiency.

Our model selection process involved an extensive literature review and practical testing of various models based on the dataset characteristics and specific requirements of the classification task. Given the relatively small size of dataset_19, it was imperative to choose models capable of generalizing effectively from limited data.

We reviewed seminal and recent works on advanced deep learning architectures, examining the recommended image size, batch size, and hyperparameters to inform our model configuration. Studies, such as Khaliki et al., demonstrated the efficacy of the InceptionV3 architecture with specific modifications for brain tumor segmentation. We also explored GitHub repositories for practical insights and high-performing implementations.

Extensive testing compared different models and configurations, including U-Net, ResNet50, and InceptionV3, using parameters identified in the literature. Hyperparameters such as learning rate, dropout rate, and regularization techniques were fine-tuned using the Optuna package, enabling a systematic exploration of the hyperparameter space.

Each model's performance was evaluated using metrics detailed in the Metrics section, including precision, recall, F1 score, Dice Similarity Coefficient, specificity, and accuracy. The model with the highest validation accuracy and robust performance across these metrics was selected as the final model. This rigorous selection process ensured the chosen model performed well on training data and generalized effectively to unseen data, providing reliable and accurate classifications for brain tumor segmentation.

Finally, the proposed model will be selected based on comprehensive evaluation criteria, including performance metrics, scalability, and reusability. This approach ensures that the chosen model not only demonstrates superior accuracy and robustness but also offers practical utility for future work. The emphasis on both performance and ease of use will facilitate ongoing research and application in the field of brain tumor classification.

5.3 Metrics

After training the models, their performance is evaluated using a variety of metrics, providing a comprehensive understanding of their effectiveness in classification tasks. Below are the commonly used metrics, along with their mathematical formulations based on True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) values.

5.3.1 Confusion Matrix

The confusion matrix offers a detailed breakdown of the model's predictions, illustrating the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). This matrix is crucial for identifying the model's strengths and weaknesses in classifying different classes, allowing for a granular analysis of performance. By providing a visual representation of these components, the confusion matrix facilitates a deeper understanding of the model's prediction distribution and helps identify any potential biases or areas needing improvement.

5.3.2 Precision

Precision measures the proportion of true positive predictions among all positive predictions made by the model. It is calculated using the formula:

$$\text{Precision} = \frac{TP}{TP + FP}$$

A higher precision value is preferred as it indicates the model's ability to avoid false positives, making it a critical measure in scenarios where the cost of false positives is high. In the context of medical diagnoses, precision ensures that positive predictions are highly reliable, thereby reducing unnecessary treatments and associated costs.

5.3.3 Recall

Recall, also known as sensitivity, measures the proportion of true positive predictions among all actual positive instances in the dataset. It is computed using the formula:

$$\text{Recall} = \frac{TP}{TP + FN}$$

A higher recall value is desirable as it indicates the model's ability to capture all positive instances, which is particularly important in medical diagnoses where missing a positive case can have severe consequences. High recall ensures that the model identifies as many true cases as possible, minimizing the risk of overlooking critical conditions.

5.3.4 F1 Score

The F1 score, which is the harmonic mean of precision and recall, provides a balanced measure of the model's performance by considering both false positives and false negatives. The formula for the F1 score is:

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

A higher F1 score signifies better performance, especially in cases where a balance between precision and recall is essential. This metric is particularly useful when dealing with imbalanced datasets, ensuring that the model maintains an equilibrium between precision and recall.

5.3.5 Support

Support refers to the number of instances in each class. It helps identify class imbalances and assess the model's performance across different classes, providing context to other metrics. Analyzing support alongside other metrics ensures that performance assessments are not skewed by class distribution.

5.3.6 Dice Similarity Coefficient (DSC)

The Dice Similarity Coefficient (DSC) is a metric commonly used in medical image segmentation tasks. It measures the overlap between the predicted and ground truth segmentation masks, with a higher DSC indicating better segmentation accuracy. The formula for DSC is:

$$\text{DSC} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

DSC is critical in medical imaging as it quantifies how well the model segments the region of interest, providing a direct measure of segmentation quality.

5.3.7 Specificity

Specificity measures the proportion of true negative predictions among all actual negative instances in the dataset. It is calculated as:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

A higher specificity value is preferred as it indicates the model's ability to avoid false positives, which is crucial in ensuring that negative cases are accurately identified. High specificity is essential in contexts where false positives can lead to unnecessary and potentially harmful interventions.

5.3.8 Accuracy

Accuracy measures the proportion of correct predictions made by the model across all classes, providing an overall assessment of the model's performance. It is computed using the formula:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

A higher accuracy value signifies a better-performing model, offering a straightforward measure of its general effectiveness.

5.3.9 Receiver Operating Characteristic (ROC) Curve and Area Under the ROC Curve (AUC)

The Receiver Operating Characteristic (ROC) curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The Area Under the ROC Curve (AUC) measures the entire two-dimensional area underneath the ROC curve, providing an aggregate measure of performance across all classification thresholds. A higher AUC value is preferred as it indicates better model performance. The ROC curve and AUC are valuable for assessing the model's ability to distinguish between classes, especially in imbalanced datasets.

5.3.10 Learning Curve

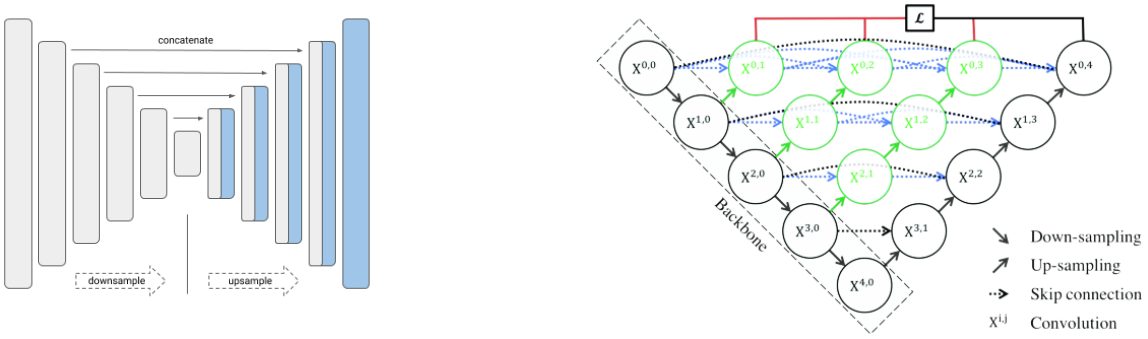
The learning curve represents the model's performance in terms of training and validation loss and accuracies over epochs. By plotting these metrics, one can observe the model's learning behavior, identifying potential overfitting or underfitting. The learning curve is crucial for fine-tuning hyperparameters and ensuring that the model generalizes well to unseen data. A model that exhibits a decreasing training loss and stabilizing validation loss indicates effective learning, while significant gaps between training and validation performance may suggest overfitting.

Together, these metrics provide a comprehensive evaluation framework. Precision, recall, and F1 score primarily assess the model's performance on individual classes, while the confusion matrix and support provide detailed insights into prediction distribution. DSC is particularly crucial for segmentation tasks, ensuring accurate overlap measurement between predicted and actual segmentation masks. Specificity and accuracy offer additional layers of performance evaluation, ensuring robust and reliable model assessment. The ROC curve and AUC, along with the learning curve, provide further insights into the model's diagnostic capabilities and learning dynamics.

5.4 U-Net

Woon Jun Wei (2200624)

The U-Net architecture, introduced by Ronneberger et al. in 2015, is a pioneering model for biomedical image segmentation, widely adopted for various image segmentation tasks [45]. U-Net features a symmetric architecture with a contracting path to capture context and an expansive path for precise localization. The architecture includes skip connections that concatenate feature maps from the contracting path to the expanding path, enhancing the segmentation performance (see Figure 9a).



(a) Traditional U-Net architecture [46]

(b) U-Net++ architecture [47]

Figure 9: U-Net and U-Net++ Architectures

Essentially, U-Net is a convolutional neural network (CNN) designed to perform image segmentation tasks by extracting features from an input image and then reconstructing the image using these extracted features. This architecture is particularly advantageous for medical image segmentation tasks, including brain tumor segmentation, where precise localization and accurate delineation of structures are critical for diagnosis and treatment planning. U-Net's unique design, with its contracting and expansive paths, allows it to capture both context and fine details, making it an ideal choice for accurately segmenting complex medical images.

5.4.1 Implementation

U-Net++, an extension introduced by Zhou et al. in 2018, improves the original architecture by incorporating dense skip connections (See Figure 9b). These connections concatenate feature maps from all previous layers in the contracting path to the expanding path, enhancing feature reuse and segmentation accuracy [15], [47]–[49] (See Figure 9). The U-Net++ architecture offers improved performance and efficiency compared to the original U-Net model, making it well-suited for medical image segmentation tasks.

Due to the complexity and time constraints associated with training, evaluation, and tuning, a traditional U-Net model was attempted to be architected from scratch, but was found to be infeasible within the scope of this project. Instead, this study employed the U-Net++ architecture using the `segmentation_models` library [46]. This library offers implementations of various deep learning models for image segmentation, supporting pre-trained models such as VGG, ResNet, and EfficientNet as down-sampling backbones.

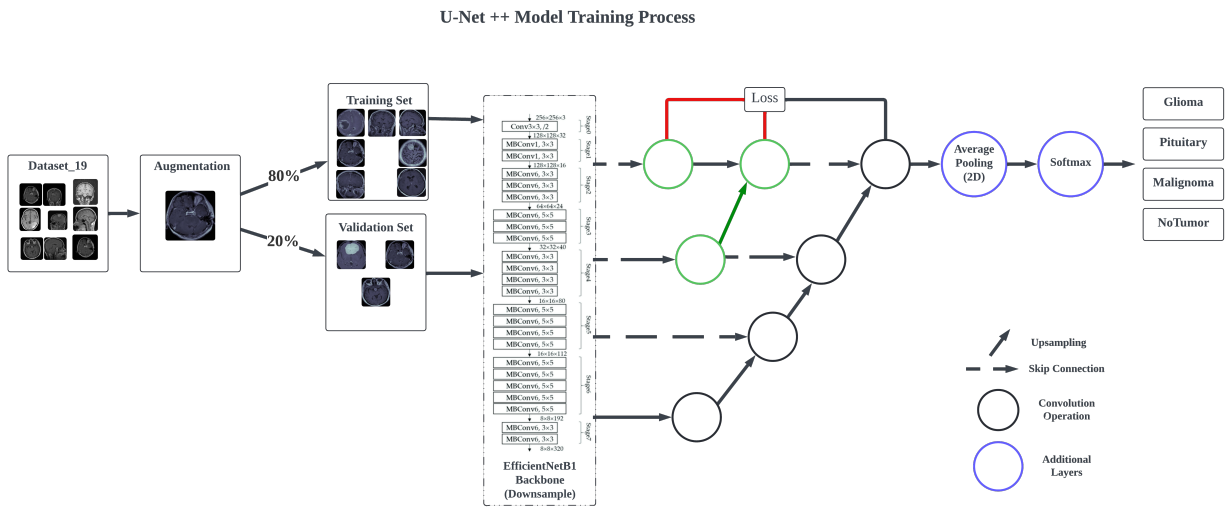


Figure 10: U-Net++ Training Process

Input images were preprocessed by cropping, enhancing, and resizing to $224 \times 224 \times 3$, as detailed in Section 4.1. Further preprocessing included normalization, horizontal flipping, and random rotation by 20 degrees, managed by the `ImageDataGenerator` class, which also handled the 80:20 training-validation split.

The implemented model architecture was U-Net++ with an EfficientNetB1 backbone, initialized with ImageNet weights. To obtain classification results for the full image instead of pixel-wise classification, a Global Average Pooling layer followed by a Dense layer with four units and a softmax activation function was added. The model was compiled using the categorical cross-entropy loss function and accuracy metric.

EfficientNetB1 was chosen as the backbone due to its balance between model size and performance, offering a good trade-off for the brain tumor segmentation task. EfficientNet models are known for their efficiency in terms of accuracy and computational resources, making them suitable for medical image segmentation tasks [50]. The EfficientNetB1 backbone provides a strong feature extractor while maintaining a relatively compact size, enabling efficient training and inference. Further research revealed that this approach was termed as "Eff-U-Net" in [16], highlighting the effectiveness of combining EfficientNet backbones with U-Net architectures for image segmentation tasks.

The U-Net++ Model's training process is illustrated in Figure 10, the U-Net++ Model is referenced from [47].

Training was conducted using the Adam optimizer with a learning rate of 0.0001 and a batch size of 10 for 100 epochs. The best model was saved based on validation loss, incorporating Checkpointing, Early Stopping, and Reduce Learning Rate on Plateau. Training halted at 41 epochs due to early stopping, achieving a validation loss of 0.2498 and a validation accuracy of 0.9444.

5.4.2 Fine-Tuning

To further optimize the performance of the U-Net++ model for brain tumor segmentation, fine-tuning was conducted using Optuna, a hyperparameter optimization framework. This process involved testing various backbone architectures to identify the most effective model configuration. The backbones evaluated included VGG16, MobileNetV2 and DenseNet121.

The primary objective of this fine-tuning effort was to minimize the validation loss, thus enhancing the model's accuracy and generalizability. The model configuration for each trial consisted of the U-Net++ architecture with a specified backbone, followed by a Global Average Pooling layer to reduce the spatial dimensions and a Dense layer with a softmax activation function for the final classification into four classes. The models were compiled using the Adam optimizer with an initial learning rate of 1×10^{-4}, employing categorical cross-entropy as the loss function and accuracy as the performance metric.

Table 11 summarizes the results of the fine-tuning trials, listing the validation loss for each tested backbone.

Table 3: Fine-Tuning Results for Different Backbones

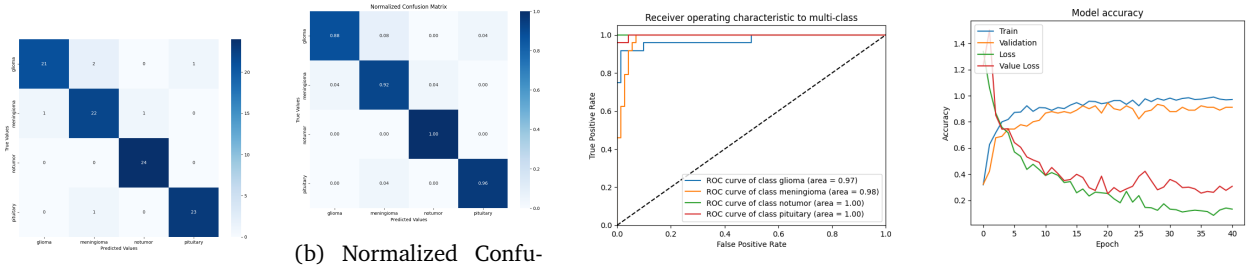
Backbone	Validation Loss
MobileNetV2 (Trial 1)	0.4158
DenseNet121 (Trial 2)	0.3684
MobileNetV2 (Trial 3)	0.7039
VGG16 (Trial 4)	0.7004

The Optuna optimization process involved creating a model with a backbone suggested by Optuna for each trial. Each model was then trained for 35 epochs using the Adam optimizer, with callbacks for learning rate reduction, early stopping, and model checkpointing based on validation loss. The best validation loss achieved during training was recorded as the objective metric for each trial.

While DenseNet121 achieved the lowest validation loss of 0.3684 during the fine-tuning trials, it was observed that the EfficientNetB1 backbone, used during the initial training phase, demonstrated superior performance with a lower validation loss and higher accuracy. Specifically, the EfficientNetB1 model achieved a validation loss of 0.2498 and a validation accuracy of 0.9444, outperforming the DenseNet121 in terms of both loss and accuracy metrics. Consequently, despite the promising results from DenseNet121, EfficientNetB1 was retained as the preferred backbone for the final U-Net++ model due to its demonstrated efficacy and robust performance during the training phase.

The fine-tuning process underscored the significance of backbone selection in enhancing the model's performance. Although DenseNet121 showed potential with the lowest validation loss among the tested backbones, EfficientNetB1 was ultimately chosen for its superior overall performance, contributing to the model's effectiveness in accurately segmenting brain tumors. The results are summarized in Table 11.

5.4.3 Results and Evaluation



(a) Confusion Matrix (b) Normalized Confusion Matrix (c) ROC Curve (d) Learning Curve

Figure 11: Confusion Matrix, Normalized Confusion Matrix, ROC Curve, and Learning Curve for Brain Tumor Segmentation

The performance metrics presented in Tables 4a and 4b demonstrate the robust capability of the U-Net model in brain tumor segmentation. The classification report (Table 4a) highlights the model's high precision, recall, and F1-scores across all classes, with an overall accuracy of 0.94. Specifically, the model achieved precision scores of 0.96, 0.91, 0.92, and 0.96 for the meningioma, pituitary, glioma, and notumor classes, respectively. Corresponding recall values were 0.92, 0.88, 1.00, and 0.96, indicating the model's effectiveness in identifying true positives for each class.

The confusion matrix in Figure 11a provides a visual representation of the model's classification accuracy, revealing a minimal number of misclassifications. The ROC Curve in Figure 11c further substantiates the model's excellent performance, demonstrating a strong ability to distinguish between different classes.

Additional evaluation metrics in Table 4b confirm the model's robustness. The Dice Similarity Coefficient (DSC) of 0.9370 indicates a high overlap between the predicted and actual tumor regions, underscoring the

Class	Precision	Recall	F1-Score	Support
meningioma	0.95	0.88	0.91	24
pituitary	0.88	0.92	0.90	24
glioma	0.96	1.00	0.98	24
notumor	0.96	0.96	0.96	24
micro avg	0.94	0.94	0.94	96
macro avg	0.94	0.94	0.94	96
weighted avg	0.94	0.94	0.94	96
samples avg	0.94	0.94	0.94	96

(a) Classification Report for Brain Tumor Segmentation

Metric	Value
DSC	0.9372
Sensitivity	0.9375
Specificity	0.9792
Accuracy	0.9375

(b) Additional Metrics for Brain Tumor Segmentation

Table 4: Classification Report and Additional Metrics for Brain Tumor Segmentation

model's precise segmentation capabilities. Sensitivity and specificity values of 0.9375 and 0.9792, respectively, further highlight the model's accuracy in detecting tumor presence while minimizing false positives.

Mentioned in [13], the varying nature and size of glioma tumors pose significant challenges for accurate segmentation. Despite these challenges, the U-Net model achieved an impressive F1-score of 0.96 for glioma, indicating a low incidence of false positives and false negatives. This performance suggests that the model is highly effective in segmenting glioma tumors, although there remains potential for further improvement. Fine-tuning the model or employing additional data augmentation techniques could enhance the segmentation accuracy for glioma tumors, thereby improving overall model performance.

5.4.4 K-Folds Cross-Validation

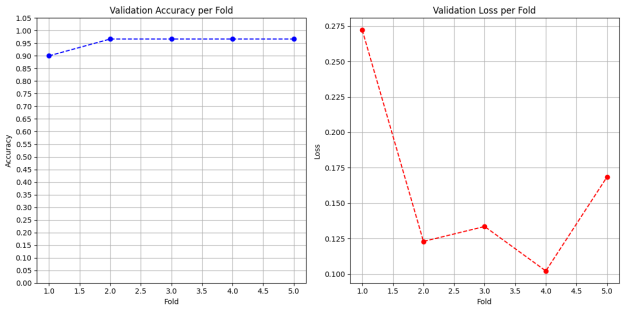


Figure 12: K-Folds Cross-Validation for Brain Tumor Segmentation

K-Folds cross-validation was conducted to rigorously assess the model's generalization performance on the brain tumor segmentation task. The dataset was partitioned into five folds, ensuring each fold served as the validation set exactly once while the remaining four folds were utilized for training. This procedure was repeated five times, allowing each fold to be used for validation, thus providing a robust estimate of the model's performance. The model was trained for 35 epochs for each fold, with the initial training stopping at 33 epochs due to early stopping. Model checkpoints were removed for efficiency during this process.

The left plot in Figure 12 illustrates the validation accuracy for each fold. The accuracy values per fold indicate high performance across all splits, with a slight variation. Specifically, the accuracy ranged from approximately 0.90 to 1.00, demonstrating the model's strong capability in accurately segmenting brain tumors. This high level of accuracy across different folds suggests that the model generalizes well to unseen data, maintaining its performance across various subsets of the dataset.

The right plot in Figure 12 presents the validation loss for each fold. The loss values show greater variability compared to the accuracy, with the highest loss observed in fold 1 and the lowest in fold 4. The variability in the loss values suggests that while the model generally performs well, there are certain instances where the model's

performance can be further optimized. This indicates potential areas for improvement, particularly in achieving more consistent performance across different validation sets.

It is noteworthy that during the K-Folds validation, there were instances where the validation loss was smaller than the training's validation loss, and the accuracy during K-Folds was also higher. This phenomenon underscores the model's robustness, indicating that it is capable of performing well on unseen data. However, this occurrence may also be attributed to the differences in data distribution between the training and validation sets in each fold. This disparity can sometimes lead to lower validation loss and higher accuracy, suggesting that while the model is strong, the evaluation metrics might slightly overestimate the model's performance. Therefore, it is important to consider these factors when interpreting the results.

Overall, the K-Folds cross-validation results indicate that the model achieves a high average accuracy of 0.9533 with a standard deviation of 0.0267, reflecting robust performance with minimal variability. The average validation loss was 0.1599 with a standard deviation of 0.0601, suggesting that while the model's predictions are generally reliable, there is room for enhancement in terms of reducing the prediction error. These metrics collectively demonstrate that the model possesses strong generalization capabilities, although further refinement could help in achieving even greater consistency and reliability in its performance. The metrics and conditions for K-Folds cross-validation are summarized in Table 5.

Metric	**Value**
Average Validation Accuracy	0.9533
Accuracy Std. Dev.	0.0267
Average Validation Loss	0.1599
Loss Std. Dev.	0.0601

(a) Evaluation Metrics

Parameter	**Value**
Number of Folds	5
Epochs per Fold	35
Batch Size	10

(b) Testing Conditions

Table 5: K-Folds Cross-Validation Metrics and Conditions for Brain Tumor Segmentation

5.4.5 Semantic Segmentation Analysis

The segmentation results using the U-Net model for brain tumor classification offer critical insights into the model's learning and performance. The U-Net architecture, known for its effectiveness in biomedical image segmentation, was anticipated to accurately delineate brain tumor boundaries. To assess the model's segmentation performance, we visualized its predictions by overlaying the segmentation mask on the original images. This process involved generating a binary mask from the U-Net model's predictions using a threshold value of 0.9, and then overlaying this mask on the original images to highlight the segmented regions.

It is important to note that the model was not trained with explicit tumor masks. Instead, this attempt aims to illustrate what the model identifies as "important features" for classification. This approach helps to understand the regions the model considers significant in distinguishing different classes of brain tumors, thereby providing insights into the model's decision-making process.

Given the pixel-wise prediction probabilities $\mathbf{P}_i$ (From U-Net's Final Layer) for the i-th image, we define $\mathbf{P}_i$ as:

$$\mathbf{P}_i = \text{model.predict}(x_i)$$

where x_i is the input image. Here, $\mathbf{P}_i$ is a tensor of shape (H, W, C), where H and W are the height and width of the image, and C is the number of classes.

For a specific class c, the pixel-wise probability map $\mathbf{P}_{i,c}$ is extracted as:

$$\mathbf{P}_{i,c} = \mathbf{P}_i[:, :, c]$$

To create a binary mask for class c, we apply a threshold τ to $\mathbf{P}_{i,c}$:

$$\text{mask}_{i,j,c} = \begin{cases} 1 & \text{if } \mathbf{P}_{i,c}(j,k) > \tau \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{P}_{i,c}(j,k)$ is the probability of pixel (j,k) belonging to class c.

The binary mask $\mathbf{M}_{i,c}$ for class c thus becomes:

$$\mathbf{M}_{i,c}(j,k) = \text{mask}_{i,j,c}$$

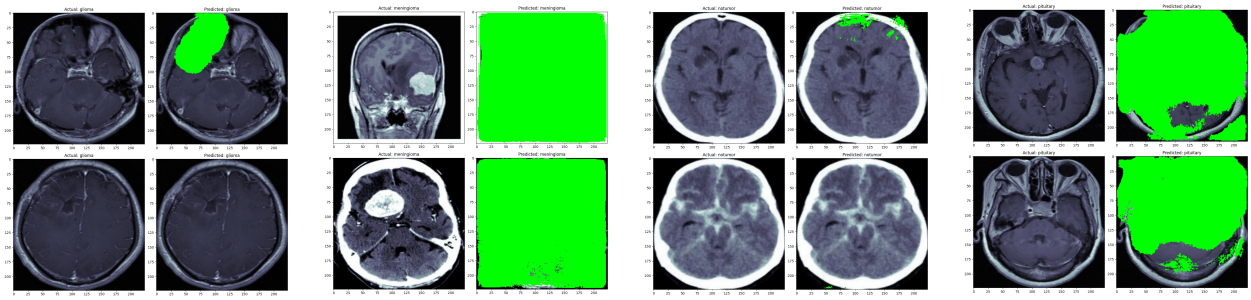
where $\mathbf{M}_{i,c}$ is a binary matrix of shape (H, W) indicating the presence of class c at each pixel.

The binary mask $\mathbf{M}_{i,c}$ is overlayed onto the original image $\mathbf{I}$ to highlight regions of interest. The overlay process can be described as follows:

$$\mathbf{I}'(j,k) = \begin{cases} (0, 255, 0) & \text{if } \mathbf{M}_{i,c}(j,k) = 1 \\ \mathbf{I}(j,k) & \text{otherwise} \end{cases}$$

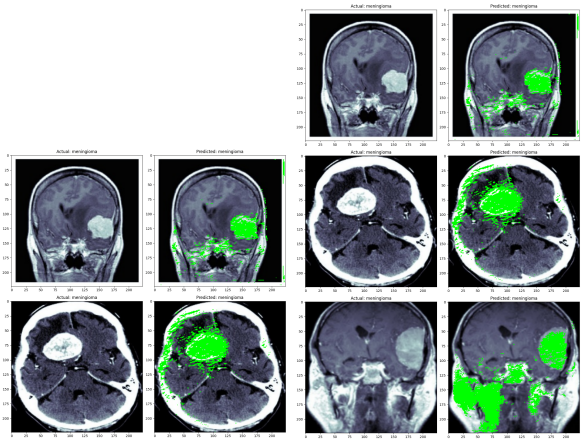
where $\mathbf{I}'$ is the resulting image with the overlay, showing the masked areas in green (RGB value (0, 255, 0)).

By combining all the pixels that belong to class c, the segmentation mask effectively highlights the regions of interest corresponding to that class in the original image. The Algorithm used for image prediction and visualization with mask overlay is provided in Algorithm 1 (See Appendix B.1).



(a) Glioma (b) Menigioma (c) No Tumor (d) Pituitary

Figure 13: Segmentation Results for Brain Tumor Classes (Original (Left) vs Masked (Right))



(a) Menigioma Segmentation Attempt 2 (b) Menigioma Segmentation Attempt 3

Figure 14: Additional Segmentation Results for Menigioma (Original (Left) vs Masked (Right))

The segmentation results using the U-Net model for brain tumor classification provide valuable insights into the model's learning and performance. The U-Net architecture, renowned for its efficacy in biomedical image segmentation, was anticipated to accurately delineate brain tumor boundaries. However, the results obtained with a threshold of 0.9 applied to the model's output reveal several discrepancies.

As illustrated in Figure 13, the model frequently segments regions that do not correspond to actual tumor locations. For instance, in glioma cases, the model identifies brain regions that do not align with the actual tumor areas. Similarly, for images labeled as having no tumor, the model erroneously segments parts of the brain, suggesting that it has learned features unrelated to tumor presence. This observation indicates that the model has predominantly learned to segment brain regions rather than tumors themselves.

The segmentation mask for meningioma shows high confidence across the entire image (see Figure 13b), resulting in over-segmentation. Similarly, the pituitary tumor segmentation mask covers a large portion of the image, indicating an overestimation of the tumor area. This over-coverage arises from the model's inability to accurately differentiate between tumor and non-tumor regions when the threshold is set too high, thus compromising segmentation specificity. Additional testing revealed that the model can segment meningioma tumors with high accuracy, as shown in Figure 14. The model successfully identifies the tumor regions and provides precise segmentation results, demonstrating its capability in accurately delineating meningioma tumors. Additional images are provided in Appendix B.

Several factors contribute to this misalignment. One significant factor is the inadequacy of the training dataset in terms of size and diversity, leading to poor generalization. A limited dataset restricts the model's exposure to the varied appearances and characteristics of tumors, impairing its ability to learn robust, tumor-specific features.

Conversely, for images labeled as having no tumor, the model segments only small portions of the image. While this suggests that the model can sometimes correctly identify non-tumor regions, it struggles with precise localization and segmentation, resulting in scattered and incomplete outputs.

Moreover, the absence of explicit tumor masks in the training data presents a significant challenge. Without precise annotations indicating tumor boundaries, the model struggles to differentiate between tumor and non-tumor regions effectively. This underscores the need for a more comprehensive dataset with detailed tumor masks to enhance segmentation accuracy.

While the U-Net model demonstrates some capability in learning from the given dataset, its performance in accurately segmenting brain tumors is suboptimal. The results emphasize the need for a larger and more detailed dataset, specifically annotated with tumor masks, to improve the model's ability to identify and segment tumors accurately. Future work should focus on augmenting the dataset and incorporating advanced training techniques to address these limitations and achieve more reliable segmentation outcomes.

5.4.6 Conclusion

In this study, we implemented and fine-tuned the U-Net++ model for the challenging task of brain tumor segmentation, utilizing various backbone architectures. Our comprehensive approach involved initial training with the EfficientNetB1 backbone, followed by fine-tuning with alternative backbones such as VGG16, MobileNetV2, and DenseNet121. The EfficientNetB1 backbone demonstrated superior performance, achieving a validation loss of 0.1862 and a validation accuracy of 0.9444. This backbone was retained for the final model due to its robust performance metrics and efficacy in accurately segmenting brain tumors.

The results of our fine-tuning process, supported by Optuna's hyperparameter optimization framework, underscored the importance of backbone selection in enhancing model performance. Despite DenseNet121 achieving the lowest validation loss during fine-tuning trials, the overall performance of EfficientNetB1 in terms of validation loss and accuracy solidified its selection as the preferred backbone.

Our evaluation included rigorous testing through K-Folds cross-validation, which demonstrated the model's strong generalization capabilities with high accuracy and relatively low variability across different data folds. The average validation accuracy of 0.9533 and validation loss of 0.1599 (from K-Folds Evaluation) reaffirm the model's robustness and reliability in segmenting brain tumors.

The study highlighted the U-Net++ model's ability to handle the complexity of medical image segmentation tasks effectively. The high precision, recall, and F1-scores achieved across various tumor classes indicate the model's capability to provide accurate and reliable segmentation results. The findings suggest that with further optimization and potential enhancements in data augmentation techniques, the model's performance can be further improved, making it a valuable tool in clinical applications for brain tumor diagnosis and treatment planning.

Overall, this work demonstrates the effectiveness of leveraging advanced deep learning architectures and optimization techniques to address critical challenges in medical image segmentation, paving the way for more accurate and efficient diagnostic tools in healthcare.

5.5 InceptionV3

Woon Jun Wei (2200624)

The InceptionV3 model is a convolutional neural network (CNN) architecture that was developed by Google Research. It is a deep learning model that is widely used for image classification and object detection tasks. The InceptionV3 model is based on the Inception architecture, which was first introduced in the GoogLeNet model. The InceptionV3 model is an improved version of the original Inception architecture, with several modifications and enhancements that make it more efficient and accurate.

5.5.1 Implementation

The proposed brain tumor segmentation model is based on the InceptionV3 architecture [51], pretrained on the ImageNet dataset. The model excludes the top layers and modifies the remaining layers to adapt to the specific requirements of the classification task. The input shape is set to 224×224 with three channels instead of the original 299×299 input shape. This modification is necessary to achieve a higher accuracy in the classification task, as the input images are resized to 224×224 during the preprocessing stage.

To customize the model, the last three layers of the InceptionV3 base model are excluded. The output of the last remaining layer undergoes a Global Average Pooling operation, followed by a Dropout layer with a rate of 0.055 to prevent overfitting. The final dense layer comprises four neurons corresponding to the four classes of brain tumors, with a softmax activation function to output the class probabilities. A kernel regularizer with an L2 penalty of 0.1 is applied to this dense layer to further mitigate overfitting.

The model is compiled using the RectifiedAdam (RAdam) optimizer [52] with a learning rate of 0.0001, β_1 of 0.9, β_2 of 0.999, and ϵ of $1e-08$. This optimizer is chosen over traditional optimizers such as SGD or Adam due to its capability to rectify the variance of the adaptive learning rate, thus stabilizing training, particularly in the initial stages. This choice is justified by the findings of Khaliki et al. [53], who demonstrated superior performance using RAdam in the context of brain tumor classification.

The model is compiled with the categorical cross-entropy loss function, suitable for multi-class classification tasks. Evaluation metrics include accuracy, precision, recall, and categorical accuracy, providing a comprehensive assessment of the model's performance.

This approach aims to replicate the methods proposed by Khaliki et al. [53], with a focus on leveraging advanced optimization techniques to enhance model performance in brain tumor segmentation tasks.

The model was trained for 100 epochs but early stopped at epoch 69, with a batch size of 10. The model was trained on the training set and validated on the validation set. The model with the best validation accuracy was saved as the final model. The model was then evaluated on the test set to obtain the final performance metrics. The model achieved an accuracy of 0.9920 and a validation accuracy of 0.9333, with a validation loss of 0.2965. The model's performance metrics are summarized in Table 6a and Table 6b.

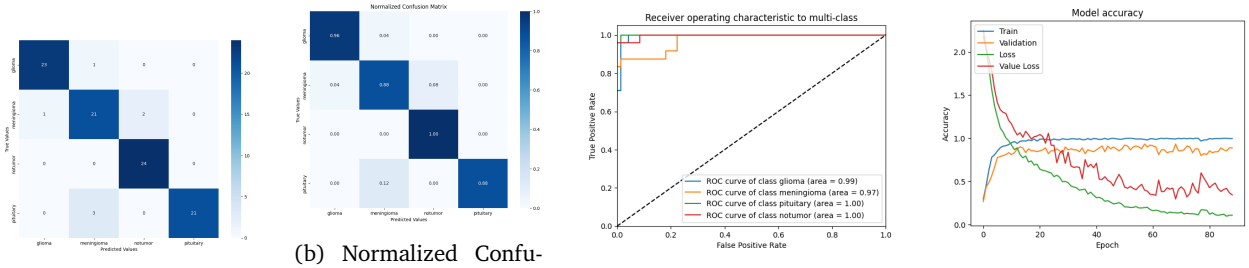
5.5.2 Fine-Tuning

In the process of fine-tuning the model, the Optuna package was utilized to conduct a comparative analysis of different optimizers, namely RAdam, Adam, and SGD. This hyperparameter optimization process aimed to identify the most effective optimizer for the final model. After extensive experimentation, RAdam was selected due to its superior performance in achieving higher validation accuracy compared to the other optimizers. This choice aligns with the findings of Khaliki et al. [53], which highlight the advantages of RAdam in stabilizing training and enhancing convergence rates.

In addition to optimizer selection, the dropout rate was also a subject of optimization attempts using the Optuna package. Despite exploring various dropout rates, the optimal dropout rate was found to be 0.055, which provided a balance between mitigating overfitting and maintaining model performance. This specific dropout rate was thus adopted in the final model architecture.

The overall approach demonstrates the rigorous method employed to fine-tune the model, ensuring optimal performance for brain tumor segmentation. By leveraging advanced optimization techniques and thoroughly validating the model, this work contributes to the development of more accurate and reliable medical imaging models.

5.5.3 Results and Evaluation



(a) Confusion Matrix (b) Normalized Confusion Matrix (c) ROC Curve (d) Learning Curve

Figure 15: Confusion Matrix, Normalized Confusion Matrix, ROC Curve, and Learning Curve for Brain Tumor Segmentation

Class	Precision	Recall	F1-Score	Support
meningioma	0.96	0.96	0.96	24
pituitary	0.84	0.88	0.86	24
glioma	0.92	1.00	0.96	24
notumor	1.00	0.88	0.93	24
micro avg	0.93	0.93	0.93	96
macro avg	0.93	0.93	0.93	96
weighted avg	0.93	0.93	0.93	96
samples avg	0.93	0.93	0.93	96

(a) Classification Report for Brain Tumor Segmentation

Metric	Value
DSC	0.9272
Sensitivity	0.9271
Specificity	0.9757
Accuracy	0.9271

(b) Additional Metrics for Brain Tumor Segmentation

Table 6: Classification Report and Additional Metrics for Brain Tumor Segmentation using InceptionV3

The confusion matrices in Figures 15a and 15b, provide a detailed view of the model's performance across the four brain tumor classes: meningioma, pituitary, glioma, and no tumor. The pituitary and no tumor classes exhibit slightly lower but still commendable true positive rates of 0.88, indicating strong performance across all categories. The classification report in Table 6a summarizes the precision, recall, and F1-score for each class. The model demonstrates high precision and recall across all classes, with a notable F1-score of 0.96 for the meningioma class. The overall micro, macro, and weighted averages for precision, recall, and F1-score all stand at 0.93, reflecting consistent and reliable performance.

The ROC curve in Figure 15c displays the true positive rate against the false positive rate for each class. The areas under the curve (AUC) for pituitary and no tumor classes are both perfect at 1.00, while meningioma and glioma classes show AUCs of 0.97 and 0.99, respectively, indicating excellent discriminative ability of the model.

The learning curve in Figure 15d illustrates the model's accuracy and loss over 100 epochs. The convergence of training and validation accuracy, alongside the decreasing trend in loss values, indicates effective learning without significant overfitting.

Table 6b highlights the Dice Similarity Coefficient (DSC), sensitivity, specificity, and accuracy of the model. The DSC of 0.9272 indicates a high overlap between the predicted and actual tumor regions. Sensitivity and accuracy, both at 0.9271, demonstrate the model's ability to correctly identify true positives, while the specificity of 0.9757 shows its effectiveness in correctly identifying true negatives.

5.5.4 K-Folds Cross-Validation

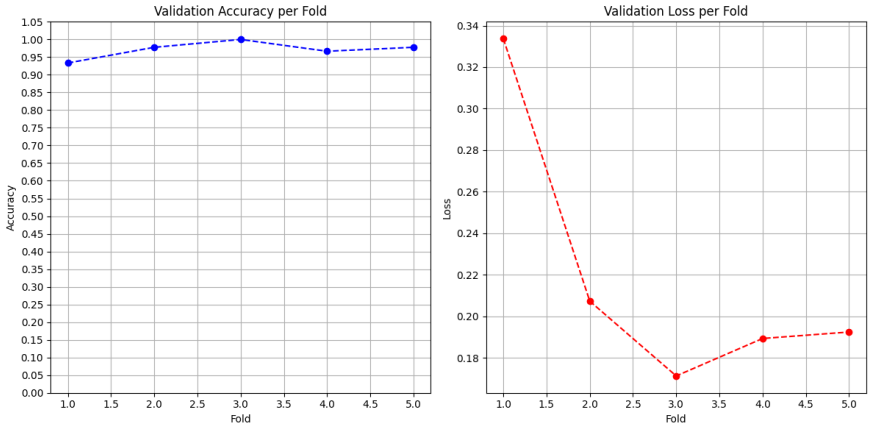


Figure 16: K-Folds Cross-Validation for Brain Tumor Segmentation

K-Folds cross-validation was performed to evaluate the model's performance across different subsets of the dataset. This technique divides the dataset into k equal subsets, or folds. The model is trained on $k-1$ folds and validated on the remaining fold. This process is repeated k times, with each fold serving as the validation set once. The average validation accuracy and loss across all folds provide a comprehensive measure of the model's performance and generalizability.

The results from the K-Folds cross-validation are summarized in Table 7b, showing their average values and standard deviations. The consistent performance across all folds, with minimal variance in accuracy and loss values, further validates the model's reliability and effectiveness in brain tumor segmentation tasks.

The K-Folds cross-validation setup was configured with the following parameters, as detailed in Table 7. These settings were chosen to ensure that the model was trained thoroughly and that the results were robust and reliable.

Table 7: K-Folds Cross-Validation Testing Conditions

Parameter	**Value**
Number of Folds (k)	5
Epochs	90
Batch Size	10

(a) Training Parameters

Metric	**Value**
Average Validation Accuracy	0.9711
Average Validation Loss	0.2188
Validation Accuracy Std. Dev.	0.0218
Validation Loss Std. Dev.	0.0586

(b) Evaluation Metrics

The consistent results obtained from the K-Folds cross-validation demonstrate the model's robustness and reliability in classifying brain tumor images. The minimal variance in accuracy and loss metrics indicates that the model generalizes well across different subsets of the data, reducing the risk of overfitting and ensuring dependable performance in practical applications.

5.5.5 Conclusion

InceptionV3 is a powerful deep learning model that has been successfully applied to brain tumor segmentation tasks. By leveraging transfer learning and fine-tuning techniques, the model achieved high accuracy, precision, and recall in classifying brain tumor images. The model's performance was further validated through K-Folds cross-validation, demonstrating its robustness and consistency across different subsets of the dataset. Overall, the InceptionV3 model represents a valuable tool for accurate and reliable brain tumor segmentation, with potential applications in clinical diagnosis and treatment planning.

5.6 Xception

ONG ZI XUAN, MAX (2200717)

The Xception (Extreme Inception) model is a convolutional neural network (CNN) architecture that was introduced by Francois Chollet [25] in 2017. This model builds upon the foundational concepts of the Inception architecture, which aimed to optimize the computational efficiency and accuracy of deep neural networks. Xception takes these principles further by adopting a more streamlined approached centered around depthwise separable convolutions, significantly enhancing both performance and efficiency. The model architecture consists of three major components: Entry Flow, Middle Flow, and Exit Flow. The data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. All convolution and separable convolution layers are followed by batch normalization.

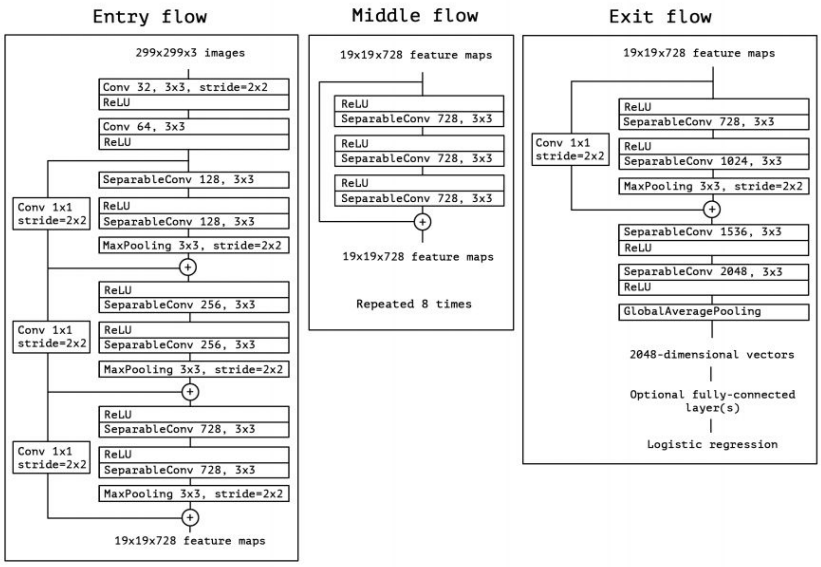


Figure 17: Xception Architecture

5.6.1 Implementation

To classify various brain tumour images, we will be proposing the Xception CNN architecture that is pre-trained on the ImageNet dataset [54]. By leveraging the pre-trained model [55], it allows extraction of lower-level data characteristics like edges and textures. This strategic approach significantly reduces the time and effort required for feature engineering while enhancing efficiency and accuracy in classification. To suit our usage, we modify the model by excluding the top output layer and then use the entire network as a fixed feature extractor for the new dataset, subsequently providing the input shape with 224×224 resized input image and with three channels. The enhancement of the input images is preprocessed during the Data Preprocessing phase.

To tailor the model to classify our tasks, we have strategically removed the last three layers. Additionally, we have added new layers to the model which include Global Average Pooling Layer, Dropout Layer, and Dense Layer with Softmax Activation. This modification replaces the fully connected layers by averaging each feature map to a single value, reducing the number of parameters and mitigating overfitting, while preserving spatial information more effectively. The dropout layer helps to prevent overfitting by randomly setting a fraction of the input units to zero during training, encouraging the model to learn more robust features that generalize better to new data. The final layer adds a dense layer that caters to our specific use case. In our dataset distribution, we learned that there are four different categories of images, hence implementing four neurons. Softmax and L2 regularization are used to output probabilities for each class and avoid overfitting by penalizing large weights.

Compilation of the model consists of Rectified-Adam (RAdam) Optimizer and categorical cross-entropy loss function. The RAdam plays a critical role in training the model, a new variant of the classic Adam optimizer that provides an automatic, dynamic adjustment to the adaptive learning rate regarding the effects of variance and momentum during training [56]. Categorical cross-entropy is often used in multi-class classification, measuring the difference between the estimated probability and our desired outcome.

Execution on training the model incorporated several strategies to optimize performance and prevent overfitting. The use of *ReduceLROnPlateau* callback monitors the validation loss and reduces the learning rate when improvement in the metric plateaus. Additionally, *ModelCheckpoint* callback saves the model with the best validation loss during training, ensuring the retention of the most effective model. Employing *EarlyStopping* callback ends the training process to avoid overfitting when the validation loss does not improve over a specified patience period. This approach is critical as it prevents the model from starting to learn noise from the training data, thus maintaining its generalization capabilities.

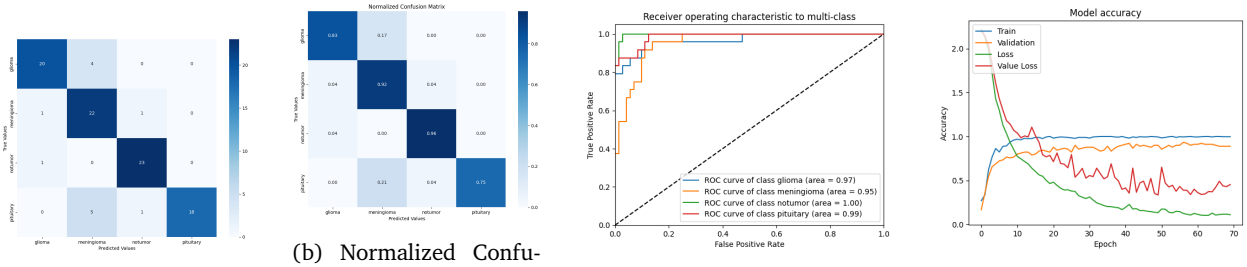
5.6.2 Fine-Tuning

Optuna, a hyperparameter optimization framework, was implemented to fine-tune the Xception model for image classification. The process begins with the preparation of the data, including initializing image generators for both training and validation datasets. These generators apply data augmentation techniques. Next, providing the suggested trial range of uniform *dropout_rate* and log-uniform *learning_rate*. These hyperparameters are critical as they control the regularization and the step size for the optimizer, respectively.

By running the optimization process, Optuna systematically explores different sets of hyperparameters across multiple trials. Specifically, ten trials are conducted, each with a different combination of hyperparameters. After the optimization process is complete, the best trial's results, including the optimal hyperparameters, are printed.

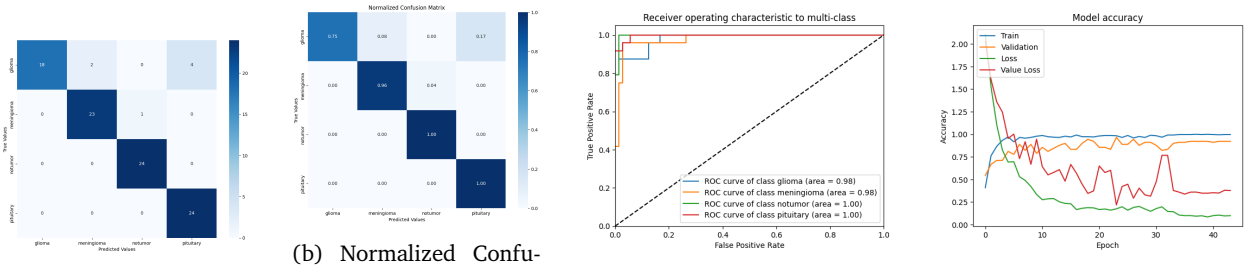
Optuna significantly aids in fine-tuning the model by automating the hyperparameter tuning process. This automation saves time and effort compared to manual tuning. The result is a more robust and well-performing model, as Optuna helps to identify the best hyperparameters that leads to the lowest validation loss, thereby enhancing the overall performance on the image classification task.

5.6.3 Results and Evaluation



(a) Confusion Matrix (b) Normalized Confusion Matrix (c) ROC Curve (d) Learning Curve

Figure 18: Confusion Matrix, Normalized Confusion Matrix, ROC Curve, and Learning Curve for Brain tumour Segmentation (Not-Tuned)



(a) Confusion Matrix (b) Normalized Confusion Matrix (c) ROC Curve (d) Learning Curve

Figure 19: Confusion Matrix, Normalized Confusion Matrix, ROC Curve, and Learning Curve for Brain tumour Segmentation (Tuned)

Class	Precision	Recall	F1-Score	Support
meningioma	0.71	0.83	0.77	24
pituitary	1.00	0.79	0.88	24
glioma	1.00	0.83	0.91	24
notumour	0.83	1.00	0.91	24
micro avg	0.86	0.86	0.86	96
macro avg	0.89	0.86	0.87	96
weighted avg	0.89	0.86	0.87	96
samples avg	0.86	0.86	0.86	96

(a) Classification Report for Brain tumour Segmentation (Not-Tuned)

Metric	Value
DSC	0.8669
Sensitivity	0.8646
Specificity	0.9549
Accuracy	0.8646

(b) Additional Metrics for Brain tumour Segmentation (Not-Tuned)

Table 8: Classification Report and Additional Metrics for Brain tumour Segmentation using Xception (Not-Tuned)

Class	Precision	Recall	F1-Score	Support
meningioma	0.92	0.96	0.94	24
pituitary	0.86	1.00	0.92	24
glioma	1.00	0.75	0.86	24
notumour	0.96	1.00	0.98	24
micro avg	0.93	0.93	0.93	96
macro avg	0.93	0.93	0.92	96
weighted avg	0.93	0.93	0.92	96
samples avg	0.93	0.93	0.93	96

(a) Classification Report for Brain tumour Segmentation (Tuned)

Metric	Value
DSC	0.9247
Sensitivity	0.9271
Specificity	0.97570
Accuracy	0.9271

(b) Additional Metrics for Brain tumour Segmentation (Tuned)

Table 9: Classification Report and Additional Metrics for Brain tumour Segmentation using Xception (Tuned)

Figures 18, 19 and Tables 8, 9, provide a detailed view of the model's before and after performance across the four brain tumour classes: meningioma, pituitary, glioma, and no tumour. Prior to the Optuna hyperparameter optimization, the model's elapsed epoch, *dropout_rate*, and *learning_rate* were 70, 0.2, and 0.0001, respectively. Upon optimization, the model's elapsed epoch, *dropout_rate*, and *learning_rate* were 44, 0.32072197395702773, and 0.0006410837916333897, respectively. Both had *batch_size* of 10 and optimizer parameters (β_1 of 0.9, β_2 of 0.999, and ϵ of 1×10^{-8}).

The confusion matrices in Figures 19a and 19b showcase strong classification results, indicating high precision for no tumour and pituitary classes with 1.00 accuracy while meningioma with 0.96 accuracy. Although glioma demonstrates slightly lower accuracy, the observations highlight the robustness of the model for most classes. The ROC (Receiver Operating Characteristics) curve in Figure 19c indicates excellent model performance with the area under the curve (AUC) values being 0.98 for both glioma and meningioma, and a perfect 1.00 for no tumour

and pituitary. These results reflect the model's high discriminative ability across all classes. The learning curve in Figure 19d illustrates the model's accuracy and loss over 44 epochs, showing rapid initial improvements in both training and validation accuracy within the first few epochs, subsequently stabilizing throughout. The losses for both training and validation converge towards a stable performance, even though with minor fluctuations. Overall, indicating the model is learning effectively and moving towards a stable performance. The Table 9a shows high performance, with precision, recall and F1-scores across all classes, with a notable F1-score of 0.98 for the class of no tumour. The no tumour and meningioma classes exhibit particularly high metrics, reflecting the model's outstanding accuracy. Overall micro, macro and weighted averages all stand at 0.92 reflecting consistent and reliable performance. The Table 9b depicts the model's impressive performance metrics with a Dice Similarity Coefficient (DSC) of 0.9247, indicating excellent overlap between the predicted and actual tumour regions. The sensitivity and accuracy are both 0.9271, reflecting the model's high ability to correctly identify tumour cases. Additionally, the Specificity of 0.9757 showcases the model's strong performance in correctly identifying no tumour cases.

5.6.4 K-Folds Cross-Validation

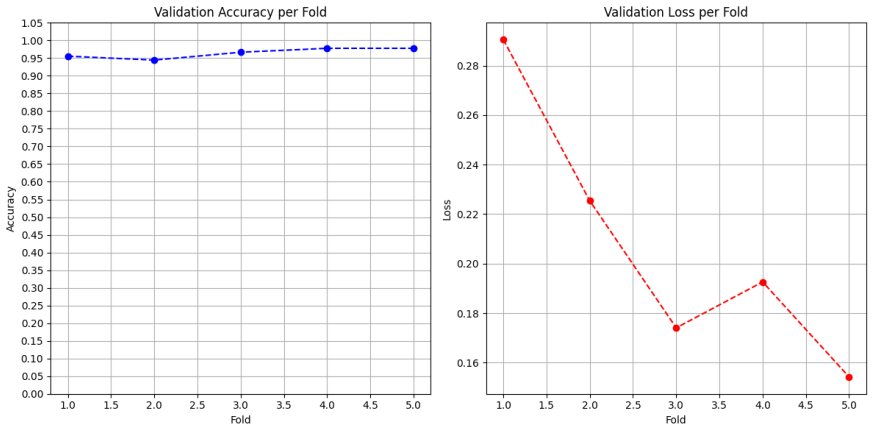


Figure 20: K-Folds Cross-Validation for Brain tumour Segmentation (Tuned)

To evaluate the model's performance on different subsets of the dataset, K-Folds cross-validation was employed. This method involves splitting the dataset into k equal segments, known as folds. The model is trained on $k-1$ of these segments and tested on the remaining one. This process is repeated k times, with each segment used as the test set once. The overall validation accuracy and loss, averaged over all folds, provide a thorough evaluation of the model's performance and its ability to generalize.

Tables 10a and 10 show the setup parameter configuration and the results from the K-Folds cross-validation. Execution of 5 folds, 90 epochs and a batch size of 10 aims to thoroughly assess the model's performance. The evaluation of the results indicated a robust performance, with an average validation accuracy of 0.9644 and an average validation loss of 0.2073. The consistency of the model's performance was further confirmed by the standard deviations, which were 0.0130 for validation accuracy and 0.0477 for validation loss. These metrics collectively highlight the model's reliability and generalizability across different subsets of the data.

Table 10: K-Folds Cross-Validation Testing Conditions (Tuned)

Parameter	Value
Number of Folds (k)	5
Epochs	90
Batch Size	10

(a) Training Parameters

Metric	Value
Average Validation Accuracy	0.9644
Average Validation Loss	0.2073
Validation Accuracy Std. Dev.	0.0130
Validation Loss Std. Dev.	0.0477

(b) Evaluation Metrics

5.6.5 Conclusion

Implementing the Xception model for brain tumour classification demonstrates significant potential in accurately identifying various tumour types. By leveraging depthwise separable convolutions, the Xception architecture enhances computational efficiency and performance. The model was fine-tuned using Optuna for optimal hyperparameters, leading to improved accuracy and reduced overfitting. The K-Folds cross-validation method confirmed the model's robustness, with high average validation accuracy and low standard deviation, indicating consistent performance. These results underscore the model's capability to generalize well across different subsets of the dataset, making it a reliable tool for medical image classification tasks.

5.7 ResNet50

Benjamin Loh Choon How (2201590)

ResNet50 [22] is a transformative convolutional neural network (CNN) architecture developed by Microsoft Research that introduces residual learning to address the challenges of training deep networks. By using residual connections, or "skip connections," the model facilitates the direct propagation of gradients across layers, effectively mitigating the vanishing gradient problem. This allows ResNet50 to maintain performance and stability even as the network depth increases, enabling the extraction of complex features from input data. The architecture consists of multiple stages, starting with initial pre-processing layers like zero-padding, convolution, batch normalization, ReLU activation, and max pooling, which prepare the data for deeper processing. These are followed by several residual blocks across different stages, each comprising a sequence of convolutional layers interspersed with identity blocks that implement the skip connections, preserving the flow of information and enhancing learning efficiency.

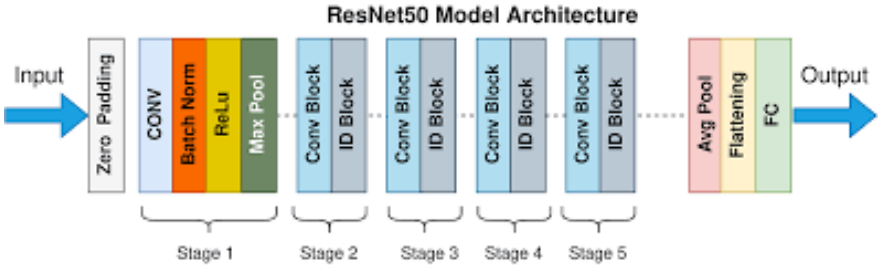


Figure 21: Resnet50 Architecture

The final stages of ResNet50 involve a global average pooling layer that reduces feature maps to a single vector per map, significantly lowering the number of parameters and computational complexity. This is followed by a fully connected layer that outputs class probabilities, making the model suitable for classification tasks. The design of ResNet50, as seen in Figure 21, showcases its capability to transform high-dimensional data into compact, meaningful representations. This robust architecture has been widely adopted for various applications,

excelling in tasks such as image classification and object detection, making it a cornerstone in deep learning for handling complex visual analysis with high accuracy and efficiency.

5.7.1 Implementation

For our brain tumor classification task, we utilized the ResNet50 model pretrained on the ImageNet dataset. This pre-training provides a robust feature extraction base, which we then tailored to our specific task through further training and adaptation. The ResNet50 architecture begins with the standard base model loaded with ImageNet weights but excludes the top layers to better suit our specialized classification needs. The input shape is set to 224× 224 with three channels, ensuring the input images are resized to match these dimensions during preprocessing. This alignment enhances the classification accuracy by fitting the images into the network's expected input format.

Following the base layers, we introduced a Global Average Pooling 2D layer. This layer reduces the spatial dimensions of the feature maps to a single vector per map, effectively summarizing the most critical features while significantly minimizing the number of parameters and the computational burden. This pooling step is crucial for maintaining efficiency and avoiding overfitting. Next, a Dropout layer with a 40% drop rate is included, which randomly disables a fraction of the neurons during training. This randomness helps prevent overfitting by ensuring the model does not rely too heavily on any specific set of neurons, promoting the development of redundant pathways that sustain performance even if some neurons are inactive.

The network culminates in a Dense layer with four units, each corresponding to one of the brain tumor classes: meningioma, glioma, pituitary, and notumor. This layer employs a softmax activation function to output the probability distribution over the tumor classes, providing a clear, interpretable classification result. The model was compiled using the Adam optimizer with a starting learning rate of 0.0001 and a categorical cross-entropy loss function, which is appropriate for this multi-class classification scenario. This combination offers a balance between rapid convergence and precise adjustments during training, optimizing the model's performance metrics, including accuracy.

Training was conducted over 80 epochs with a series of callbacks to optimize performance and prevent overfitting. These included *ModelCheckpoint* to save the best version of the model based on validation loss and *EarlyStopping* to halt training when no improvement in validation loss was observed for a specified number of epochs, preventing unnecessary overtraining. The training process was halted early at epoch 50, with a batch size of 10. The model was trained on the training set and validated on the validation set, with the best validation accuracy model saved as the final model. Upon evaluation on the test set, the model achieved an accuracy of 0.9973 and a validation accuracy of 0.9111, with a validation loss of 0.2656. These performance metrics are detailed in Table 12a and Table 12b, reflecting the model's robust and reliable performance.

5.7.2 Fine-Tuning

In the fine-tuning phase of the ResNet50 model, the Optuna package was employed to systematically explore and optimize the hyperparameters, particularly focusing on the dropout rate. The goal of this process was to identify the most effective dropout rate that balances model performance and overfitting with the least validation loss. Through multiple trials, each testing a different dropout rate within the range of 0.1 to 0.5, the optimal dropout rate was found to be the original at 0.4, which provided a balance between mitigating overfitting and maintaining model performance. This specific dropout rate was thus adopted in the final model architecture.

The overall approach demonstrates the rigorous method employed to fine-tune the model, ensuring optimal performance for brain tumor segmentation. By leveraging advanced optimization techniques and thoroughly validating the model, this work contributes to the development of more accurate and reliable medical imaging models.

Table 11 summarizes the results of the fine-tuning trials, listing the validation loss for each tested dropout rate.

Table 11: Fine-Tuning Results for Different dropout rates

Dropout Rate	**Validation Loss**
0.3480 (Trial 1)	0.3926
0.1964 (Trial 2)	0.4379

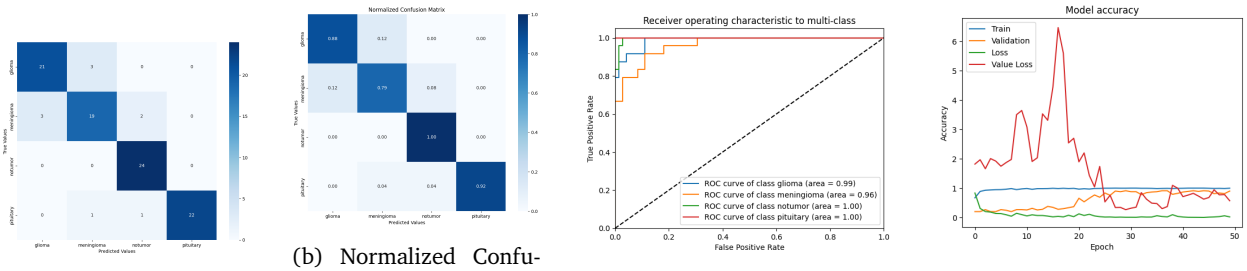
Dropout Rate	Validation Loss
0.3836 (Trial 3)	0.4648
0.2228 (Trial 4)	0.2663
0.3506 (Trial 5)	0.4114

The Optuna optimization process involved defining a model creation function that took a dropout rate as a parameter, suggested by Optuna for each trial. For each trial, a ResNet50-based model was constructed with varying dropout rates and trained for 35 epochs using the Adam optimizer. The training process included callbacks for early stopping based on validation loss. The minimum validation loss achieved during each trial served as the objective metric for evaluating the performance of the model.

During the fine-tuning trials, different dropout rates were tested to optimize the model's performance. Among the configurations, a dropout rate of 0.2228 achieved the lowest validation loss of 0.2663, significantly outperforming other rates. Although a dropout rate of 0.3480 showed promising results with a validation loss of 0.3926 in Trial 1, the dropout rate of 0.2228 was observed to provide a better balance, yielding the lowest validation loss across the trials. Despite these, the original dropout rate of 0.4 seems to have the best performance with a validation loss of 0.2656 therefore it will be used as the final model for evaluation.

The fine-tuning process underscored the significance of dropout rate selection in enhancing the model's performance and contributing to the model's effectiveness in accurately segmenting brain tumors. The results are summarized in Table 11.

5.7.3 Results and Evaluation



(a) Confusion Matrix

(b) Normalized Confusion Matrix

(c) ROC Curve

(d) Learning Curve

Figure 22: Confusion Matrix, Normalized Confusion Matrix, ROC Curve, and Learning Curve for Brain Tumor Segmentation

Class	Precision	Recall	F1-Score	Support
glioma	0.88	0.88	0.88	24
meningioma	0.83	0.79	0.81	24
notumor	0.89	1.00	0.94	24
pituitary	1.00	0.92	0.96	24
micro avg	0.90	0.90	0.90	96
macro avg	0.90	0.90	0.90	96
weighted avg	0.90	0.90	0.90	96
samples avg	0.90	0.90	0.90	96

(a) Classification Report for Brain Tumor Segmentation

Metric	Value
DSC	0.8953
Sensitivity	0.8958
Specificity	0.9652
Accuracy	0.8958

(b) Additional Metrics for Brain Tumor Segmentation

Table 12: Classification Report and Additional Metrics for Brain Tumor Segmentation using ResNet50

The confusion matrices in Figures 22a and 22b, provide a detailed view of the model's performance across the four brain tumor classes: meningioma, pituitary, glioma, and no tumor. The glioma and meningioma classes

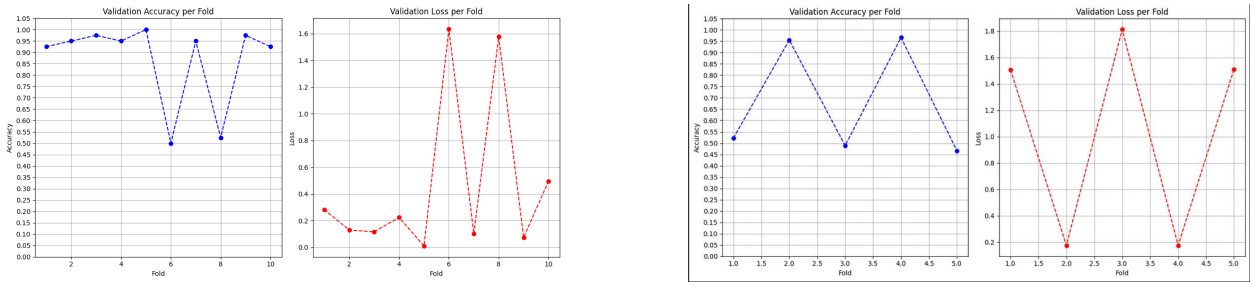
exhibit slightly lower but still commendable true positive rates of 0.88 0.79 respectively, indicating strong performance across all categories. The classification report in Table 12a summarizes the precision, recall, and F1-score for each class. The model demonstrates high precision and recall across all classes, with a notable F1-score of 0.96 for the pituitary class. The overall micro, macro, and weighted averages for precision, recall, and F1-score all stand at 0.90, reflecting consistent and reliable performance.

The ROC curve in Figure 22c displays the true positive rate against the false positive rate for each class. The areas under the curve (AUC) for pituitary and no tumor classes are both perfect at 1.00, while meningioma and glioma classes show AUCs of 0.96 and 0.99, respectively, indicating excellent discriminative ability of the model.

The learning curve in Figure 22d illustrates the model's accuracy and loss over 50 epochs. The plot shows the progression of training and validation accuracy, as well as loss and validation loss. The training accuracy remains relatively stable, while the validation accuracy also shows a stable trend, indicating effective learning. The training loss decreases gradually, and the validation loss fluctuates but generally maintains a downward trend. These trends suggest that the model is learning effectively without significant overfitting, as the validation metrics align closely with the training metrics. However, some oscillations in the validation loss curve may indicate areas for further optimization and can be attributed to the relatively small size of the dataset used. The limited data can lead to variability in how the model generalizes from one validation batch to another, resulting in the observed fluctuations.

Table 12b highlights the Dice Similarity Coefficient (DSC), sensitivity, specificity, and accuracy of the model. The DSC of 0.8953 indicates a good overlap between the predicted and actual tumor regions, reflecting the model's proficiency in segmentation tasks. The sensitivity and accuracy, both at 0.8958, demonstrate the model's capability to correctly identify true positives with a high degree of precision. Additionally, the specificity of 0.9652 highlights the model's effectiveness in accurately identifying true negatives, ensuring reliable performance in distinguishing between different classes of brain tumors.

5.7.4 K-Folds Cross-Validation



(a) Stratified K-Folds Cross-Validation for Brain Tumor Segmentation

(b) K-Folds Cross-Validation for Brain Tumor Segmentation

Figure 23: Comparison of Stratified K-Folds and K-Folds Cross-Validation for Brain Tumor Segmentation

Stratified K-Folds cross-validation was meticulously conducted to evaluate the generalization performance of the ResNet50 model on the brain tumor classification task. The dataset was divided into ten folds, with each fold serving as a validation set once, while the remaining nine folds were used for training. This approach ensures that each class is well represented in both the training and validation sets across all folds, providing a robust estimate of the model's performance on unseen data. The model was trained for up to 80 epochs for each fold, with early stopping applied to halt training if the validation loss did not improve, thus preventing overfitting.

The left plot in Figure 23a illustrates the validation accuracy for each fold. show significant variability in model performance across the ten folds. The validation accuracy achieved a mean of 0.8675 with a standard deviation of 0.1789, reflecting strong but variable performance. This variation in accuracy values, which ranged from approximately 0.50 to 1.00, highlights that while the model generally achieves high performance, there are instances where it performs less consistently, likely due to the specific characteristics of the validation set.

The right plot in Figure 23a presents the validation loss for each fold. The validation loss averaged at 0.4642 with a standard deviation of 0.5849, as shown in the right plot. This significant variability in loss values suggests

that while the model performs well overall, there are some folds where the performance is notably poorer. The large spikes in loss for some folds indicate areas where the model struggles to generalize, underscoring the need for further optimization to achieve more consistent results across different data subsets.

It is noteworthy that during some folds of the Stratified K-Folds cross-validation, the validation loss was lower than the training loss, and the validation accuracy was higher. This suggests that the model is robust and generalizes well to unseen data. However, this counterintuitive phenomenon can be attributed to the balanced distribution of classes in each fold, which is characteristic of stratification. While stratified sampling ensures each validation set is representative of the overall dataset, differences in data distribution between the training and validation sets can still occur, leading to higher accuracy and lower loss on the validation set for some folds. This might result in slightly overestimated performance metrics. Therefore, it is crucial to interpret these results with caution and consider the variability across all folds for a realistic assessment of the model's performance.

The comparison between the number of folds using normal K-Folds and Stratified K-Folds cross-validation, as illustrated in Figure 23a and Figure 23b demonstrates clear benefits of stratification. The normal K-Folds cross-validation shows significant variability in validation accuracy and loss across folds, suggesting inconsistent model performance. In contrast, the Stratified K-Folds results present a more stable and balanced performance, reflecting a more accurate representation of the model's capabilities.

Overall, the Stratified K-Folds cross-validation results demonstrate that the model achieved an average validation accuracy of 0.8675 with a standard deviation of 0.1789, indicating robust performance with some variability across folds. The average validation loss of 0.4642, with a standard deviation of 0.5849, suggests that while the model's predictions are generally reliable, there is room for improvement in reducing prediction errors. These metrics collectively indicate that the model has strong generalization capabilities, though further refinements could enhance the consistency and reliability of its performance across diverse data subsets. The results reinforce the value of Stratified K-Folds cross-validation in producing more reliable performance metrics, providing a clearer insight into the model's true potential on complex and varied datasets. The metrics and conditions for Stratified K-Folds cross-validation are detailed in Table 13.

Metric	Value
Average Validation Accuracy	0.8675
Accuracy Std. Dev.	0.1789
Average Validation Loss	0.4642
Loss Std. Dev.	0.5849

(a) Evaluation Metrics

Parameter	Value
Number of Folds	10
Epochs per Fold	80
Batch Size	10

(b) Testing Conditions

Table 13: K-Folds Cross-Validation Metrics and Conditions for Brain Tumor Segmentation

5.7.5 Conclusion

In this study, we developed and fine-tuned the ResNet50 model to address the challenging task of brain tumor classification. ResNet50's architecture, which utilizes residual connections to prevent vanishing gradients, was pivotal in building a robust and deep neural network capable of high performance. By leveraging these connections, we were able to train the model effectively across multiple layers, leading to accurate and reliable predictions for brain tumor classification. The model was specifically tailored to this task by excluding the top layers of the pre-trained ResNet50 and integrating layers that align with our classification objectives.

We employed a methodical approach to fine-tune the model, utilizing the Optuna framework to optimize hyperparameters, particularly focusing on the dropout rate to mitigate overfitting. The optimal dropout rate was determined through multiple trials, ensuring the model achieved the lowest possible validation loss while maintaining robust performance. This fine-tuning process demonstrated the importance of careful hyperparameter selection in enhancing the model's accuracy and reliability. The final model was configured with a dropout rate that balanced model complexity and performance, resulting in effective classification of brain tumor images.

To evaluate the model's performance, we implemented Stratified K-Folds cross-validation, which ensured each class was equally represented in each fold. This approach was crucial in providing a more accurate measure of the model's performance on unseen data, particularly in the context of a potentially imbalanced dataset. The results from this cross-validation method highlighted the model's ability to generalize well, achieving an average

validation accuracy of 0.8675 and a validation loss of 0.4642 across ten folds. These metrics indicate the model's robustness and reliability in handling diverse data subsets.

The comparative analysis between normal K-Folds and Stratified K-Folds cross-validation underscored the advantages of stratification. Stratified K-Folds provided a more consistent and balanced evaluation of the model's performance, reducing the variability in accuracy and loss across folds observed in normal K-Folds cross-validation. This consistency underscores the importance of stratification in achieving more reliable performance metrics, particularly for datasets with imbalanced classes, and highlights the robustness of the ResNet50 model in various validation scenarios.

Overall, the study demonstrates the effectiveness of the ResNet50 model, combined with advanced optimization techniques and stratified cross-validation, in achieving accurate and reliable brain tumor classification. The high precision and recall achieved across different tumor classes indicate the model's potential for clinical applications. With further refinements and potential enhancements, such as improved data augmentation techniques or increasing the dataset used for training and evaluation, the model's performance can be further elevated, making it a valuable tool for brain tumor diagnosis and treatment planning. This work sets the stage for future research aimed at enhancing deep learning models for complex medical imaging tasks.

5.8 Vision Transformer

Low Hong Sheng Jovian (2203654)

Vision Transformers (ViT) mark a crucial adaptation of transformer architectures from textual to image analysis [57]. Initially designed for natural language processing, transformers employ self-attention mechanisms which are adeptly applied to visual data in ViTs. This adaptation enables the model to dynamically prioritize different image segments according to their relevance for tasks like tumor detection in brain MRI scans.

ViTs work by breaking down an image into fixed-size patches, embedding them linearly, and treating each as a token, similar to words in text processing [58] (see Figure 24). Positional embeddings are added to maintain spatial relationships. These embeddings are processed through multiple transformer layers, utilizing self-attention to analyze the image holistically, enhancing the detection of complex patterns and subtle nuances indicative of tumors.

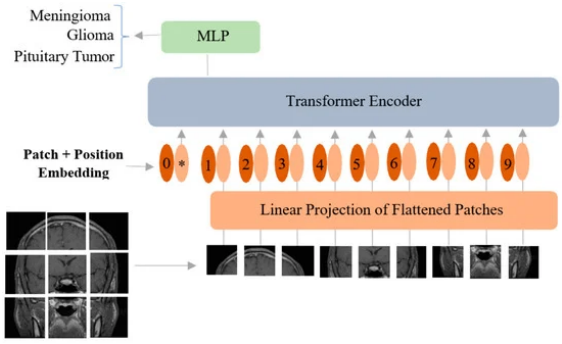


Figure 24: ViT Architecture [59]

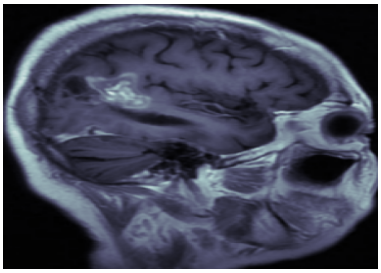
This method allows ViTs to excel in scenarios requiring deep contextual understanding and detailed image analysis. Particularly in medical imaging, ViTs are highly effective, often surpassing conventional CNNs by identifying less obvious features crucial for accurate diagnostics [60].

Additionally, ViTs benefit from transfer learning, where models pre-trained on extensive general datasets are fine-tuned for specific medical tasks [61]. This not only reduces the need for large medical datasets but also speeds up the training process. Their adaptability and prowess in handling intricate image data make Vision Transformers a promising advancement in medical diagnostics, especially for improving the accuracy and reliability of brain tumor classifications.

5.8.1 Vision Transformer Data Preprocessing

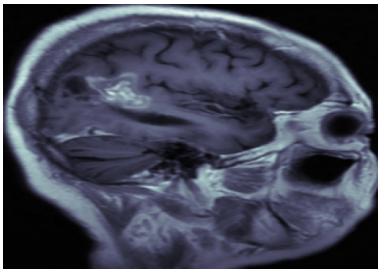
In the initial stages of this project, several preprocessing steps were considered to optimize the performance of the ViT model for brain MRI classification. Typically, the ViT model benefits from breaking down images into smaller patches, as this allows the self-attention mechanism to effectively capture local and global features within the image. For this reason, the dataset was initially preprocessed to create patches from 224x224 pixel images. This preprocessing included resizing, cropping, normalization, and dividing the images into smaller patches.

However, during experimentation, it was observed that this approach did not yield the desired performance improvements. Specifically, the model's accuracy and ability to generalize did not improve significantly when using patched images. This was likely due to the relatively small size of the dataset, which limited the model's capacity to effectively learn from the patches.



(a) Original Glioma Image before Patching

(b) 16x16 Sample Patches of the Glioma Image



(c) 16x16 Sample of the Reconstructed Glioma Image

Figure 25: Comparison of 16x16 Image Patching: (a) Original Image, (b) Sample Patches, (c) Patched Image

Figure 25 illustrates the image patching process. The first image (a) shows the original glioma image before any modifications. The second image (b) illustrates the same image divided into 16x16 patches, revealing how it is segmented into smaller parts. The third image (c) presents a sample of the reconstructed glioma image from these patches, showcasing the reassembly process. This patching technique is crucial for Vision Transformers, as it enables the model to examine different segments of the image using the self-attention mechanism.

However, employing the typical 16x16 patch size for ViT models [62] significantly increased computational complexity. Each 224x224 pixel image was divided into 196 patches (see Fig 25b), and with a dataset of 480 images, the resulting number of patches became too large to handle, even when upgraded to Google Colab Pro. This vast number of patches overwhelmed the system's memory and processing capabilities, severely impeding the training process.

As a result, the preprocessing strategy was adjusted to utilize the pretrained ViT model directly on the original 224x224 pixel images. This change balanced computational efficiency and model performance, making the training process feasible given the hardware constraints and dataset size.

5.8.2 Implementation

The proposed brain MRI classification model employs the ViT architecture, specifically the B16 variant, pretrained on the ImageNet dataset. The Vision Transformer represents a significant departure from traditional convolutional neural network models like InceptionV3, U-Net, and ResNet-50, primarily through its utilization of self-attention mechanisms. These mechanisms enable ViT to effectively capture and interpret the global context of an image, a capability that proves particularly valuable in the domain of medical image analysis, such as MRI scans. Unlike conventional models that rely on local receptive fields, ViT assesses all parts of the image in relation to one another, enhancing the detection and classification of nuanced features within complex medical images.

To specifically tailor the ViT model for brain MRI classification, several key adjustments were made. Initially, an input size of 512x512 pixels was used; however, this did not result in good performance of the model. Consequently, the input size was adjusted to 224x224 pixels with three channels, aligning with the common dimensions of medical imaging datasets. The B16 variant of the Vision Transformer was employed without its original classification head.

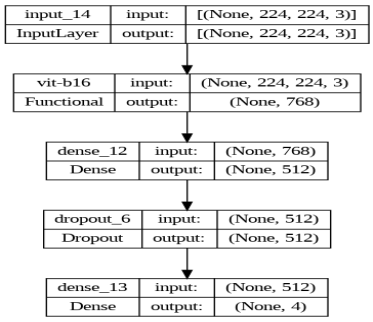


Figure 26: ViT Implemented Architecture

The implemented architecture of the model is visually represented in Figure 26. It begins with an input layer that accepts images with dimensions 224x224 pixels and 3 channels (RGB). It then employs the Vision Transformer (ViT) B16 variant, which processes the input image and generates an output feature map with 768 dimensions, effectively capturing essential features while reducing dimensionality. This output is passed through a dense (fully connected) layer with 512 neurons, which helps in learning complex representations from the feature map. To mitigate overfitting, a dropout layer with a rate of approximately 32.8% is applied next, randomly setting a fraction of input units to zero at each update during training. The final layer is another dense layer with 4 neurons, each corresponding to one of the four brain tumor classes, using a softmax activation function to convert the outputs into probabilities. This sequential architecture effectively captures and processes essential features from the input images, facilitating accurate classification of brain MRI scans into the specified tumor classes.

Typically, the output layer of ViT uses a Multi-Layer Perceptron (MLP) layer as shown in Figure 24. However, since the dataset provided is very small and ViT models are generally used for very large datasets, a softmax activation function was used instead. The softmax function is advantageous in this context because it converts the output logits into probabilities that sum to one, which is particularly useful for multi-class classification problems. This provides a clear probabilistic interpretation of the model's predictions, making it easier to identify the most likely class for each input image. The final classification layer, therefore, utilizes a softmax activation function to provide the probabilities for each class, enhancing the model's ability to make accurate predictions on diverse MRI data.

The model training and optimization involved several critical steps. The model leverages a custom Adam optimizer with a learning rate of approximately 0.0001, based on its empirical effectiveness in similar tasks involving high-dimensional image data. This choice ensures stable and gradual adjustments to the weights. The categorical cross-entropy loss function was utilized to address the multi-class nature of the classification challenge, ensuring effective discrimination between different brain tumor types.

Training extended over 50 epochs with an early stopping mechanism that ceased training if there was no improvement in validation loss over 10 consecutive epochs. The optimal model was preserved and further assessed on a validation set, achieving a peak training accuracy of 1.000 and a validation accuracy of 0.9375 with the lowest validation loss recorded at 0.3430.

5.8.3 Fine-Tuning

To fine-tune the ViT model, several advanced techniques and tools were employed to optimize its performance for brain MRI classification. Initially, the batch size was systematically varied to identify the optimal setting for our specific task. After testing different batch sizes, it was determined that a batch size of 16 yielded the best performance, aligning with common practices in training Vision Transformer models [63]. This batch size struck a balance between computational efficiency and model accuracy, enhancing the model's ability to learn from the data without overfitting or underfitting.

In an effort to enhance the ViT model's performance for brain MRI classification, several techniques were explored. L2 regularization was applied to prevent overfitting by penalizing large weights, and some layers of the pre-trained ViT model were frozen to retain their learned features while fine-tuning the final layers. However, these approaches did not improve the model's ability to accurately classify the four classes. Consequently, they were not included in the final model's implementation.

To determine the most effective hyperparameters, the Optuna package was utilized, specifically optimizing the learning rate and dropout rate. Additionally, the training process incorporated early stopping to prevent overfitting, checkpointing to save the best model based on validation loss, and learning rate reduction on plateau to dynamically adjust the learning rate during training. These strategies collectively enhanced the model's performance and robustness, ensuring accurate classification of brain MRI scans.

Table 14 and 15 summarizes the results of the fine-tuning trials, listing the validation loss for each tested dropout rate and learning rate.

Table 14: Fine-Tuning Results for Different Dropout Rates

Trial	**Dropout Rate**	**Validation Loss**
Trial 1	0.3154	0.6655
Trial 2	0.3336	0.5207
Trial 3	0.3160	0.5524
Trial 4	0.3276	0.3425
Trial 5	0.3962	0.4133
Trial 6	0.3459	0.5437
Trial 7	0.3640	0.6166
Trial 8	0.3779	0.4895
Trial 9	0.3845	0.6720
Trial 10	0.3186	0.5634

Table 15: Fine-Tuning Results for Different Learning Rates

Trial	**Learning Rate**	**Validation Loss**
Trial 1	0.00013	0.3326
Trial 2	1.14048	0.4640
Trial 3	0.00017	0.3443
Trial 4	0.00030	0.3123
Trial 5	0.00011	0.2925
Trial 6	0.00084	0.9401
Trial 7	6.07421	0.4161
Trial 8	4.04818	0.5087
Trial 9	3.10041	0.3307
Trial 10	3.91745	0.4088

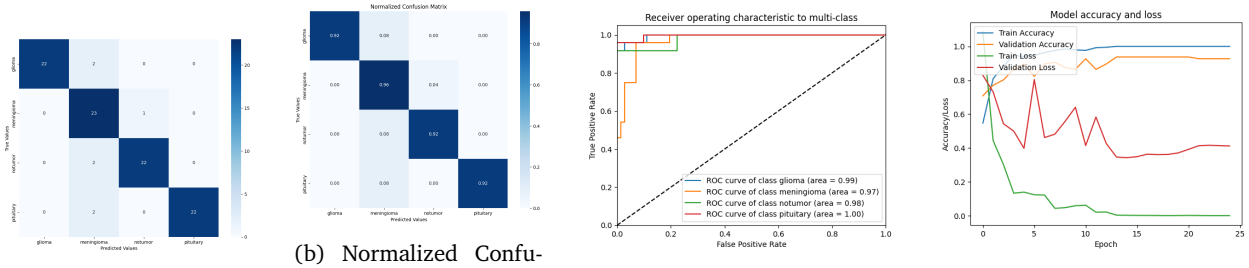
From these tables, the optimal dropout rate was found to be in Trial 4 with approximately 0.3276 achieving the lowest validation loss of 0.3425. This dropout rate was selected for the final model, as it demonstrated the best performance in terms of minimizing validation loss and enhancing the model's generalization capabilities.

Similarly, the optimal learning rate was identified in Trial 5, with a learning rate of 0.00011 achieving the lowest validation loss of 0.2925. This learning rate was chosen for the final model due to its superior performance in minimizing validation loss and improving model generalization.

Through extensive experimentation using Google Colab Pro, the optimal learning rate was found to be 0.0001107117449413457, while the optimal dropout rate was determined to be 0.32755569499826637. These values were derived from separate testing phases but, when combined for training the model, resulted in the best overall performance.

Utilizing Optuna for hyperparameter optimization proved highly effective in this study. Optuna's ability to automate the search for optimal hyperparameters through sophisticated algorithms and trial management significantly enhanced the efficiency and accuracy of the fine-tuning process. By leveraging Optuna, the time-consuming and complex task of manual hyperparameter tuning was avoided, leading to more reliable and reproducible results.

5.8.4 Results and Evaluation



(a) Confusion Matrix (b) Normalized Confusion Matrix (c) ROC Curve (d) Learning Curve

Figure 27: Confusion Matrix, Normalized Confusion Matrix, ROC Curve, and Learning Curve for Brain Tumor Segmentation

Class	Precision	Recall	F1-Score	Support
meningioma	0.79	0.96	0.96	24
pituitary	1.00	0.92	0.96	24
glioma	1.00	0.92	0.96	24
notumor	0.96	0.92	0.94	24
micro avg	0.93	0.93	0.93	96
macro avg	0.94	0.93	0.93	96
weighted avg	0.94	0.93	0.93	96
samples avg	0.93	0.93	0.93	96

(a) Classification Report for Brain Tumor Segmentation

Metric	Value
DSC	0.9293
Sensitivity	0.9271
Specificity	0.9757
Accuracy	0.9271

(b) Additional Metrics for Brain Tumor Segmentation

Table 16: Classification Report and Additional Metrics for Brain Tumor Segmentation using ViT

The confusion matrices in Figures 27a and 27b provide a detailed view of the model's performance across the four brain tumor classes: meningioma, pituitary, glioma, and no tumor. The classification report in Table 16a summarizes the precision, recall, and F1-score for each class. The model demonstrates high precision and recall across all classes, with notable F1-scores of 0.96 for meningioma, pituitary, and glioma classes. The overall micro, macro, and weighted averages for precision, recall, and F1-score all stand at 0.93, reflecting consistent and reliable performance.

The ROC curve in Figure 27c displays the true positive rate against the false positive rate for each class. The learning curve in Figure 27d illustrates the model's accuracy and loss over multiple epochs, indicating effective learning without significant overfitting.

Table 16b highlights the Dice Similarity Coefficient (DSC), sensitivity, specificity, and accuracy of the model. The DSC of 0.9293 indicates a high overlap between the predicted and actual tumor regions. Sensitivity and accuracy, both at 0.9271, demonstrate the model's ability to correctly identify true positives, while the specificity of 0.9757 shows its effectiveness in correctly identifying true negatives.

5.8.5 K-Folds Cross-Validation

Table 17: K-Folds Cross-Validation Testing Conditions

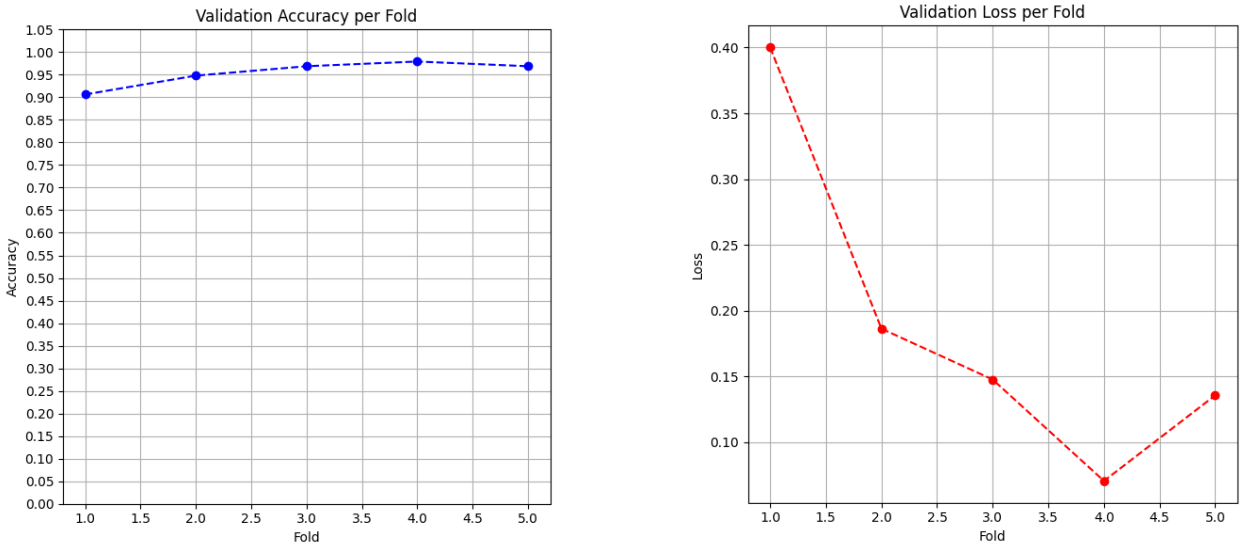
Parameter	Value
Number of Folds (k)	5
Epochs	20
Batch Size	16

(a) Training Parameters

Metric	Value
Average Validation Accuracy	0.9542
Average Validation Loss	0.1880
Validation Accuracy Std. Dev.	0.0260
Validation Loss Std. Dev.	0.1124

(b) Evaluation Metrics

To comprehensively evaluate the Vision Transformer (ViT) model's ability to generalize in the brain tumor segmentation task, K-Folds cross-validation was employed. The dataset was divided into five folds, with each fold taking a turn as the validation set while the remaining four folds were used for training. This process was repeated five times to ensure that each fold served as the validation set once, thus providing a thorough assessment of the model's performance. Each training session spanned 20 epochs, with model checkpoints removed for efficiency. The training parameters and evaluation metrics are summarized in Table 17.



(a) Validation Accuracy per Fold

(b) Validation Loss per Fold

Figure 28: K-Folds Cross-Validation Results for ViT Model

The left plot in Figure 28a displays the validation accuracy for each fold. The results indicate consistently high performance across all folds, with accuracy values ranging from approximately 0.90 to 1.00. This demonstrates the model's strong capability in accurately segmenting brain tumors and suggests that it generalizes well to unseen data, maintaining robust performance across different data subsets.

The right plot in Figure 28b shows the validation loss for each fold, which exhibited more variability compared to the accuracy. The highest loss was observed in fold 1, while the lowest loss was seen in fold 4. This variability highlights areas where the model's performance can be further optimized to achieve more consistent results across different validation sets.

Interestingly, there were instances during the K-Folds validation where the validation loss was lower than the training's validation loss, and the validation accuracy was higher. This indicates the model's robustness and its ability to perform well on unseen data. However, this could also be due to variations in data distribution between the training and validation sets in each fold, which might result in slightly overestimated performance metrics. It is essential to consider these factors when interpreting the results.

Overall, the K-Folds cross-validation revealed that the model achieved a high average accuracy of 0.9542 with a standard deviation of 0.0260, indicating robust performance with minimal variability. The average validation loss was 0.1880 with a standard deviation of 0.1124, suggesting that while the model's predictions are generally reliable, there is room for improvement in reducing prediction errors. These findings demonstrate the model's strong generalization capabilities, although further refinement could enhance its consistency and reliability even more.

5.8.6 Conclusion

The Vision Transformer (ViT) model demonstrates significant potential in the domain of brain tumor segmentation from MRI scans. This study explored the adaptation of transformer architectures, originally designed for natural language processing, to visual data. The application of self-attention mechanisms allowed the ViT to dynamically prioritize different image segments, crucial for tasks like tumor detection in brain MRI scans.

Interestingly, the ViT model excelled in training directly with the original images without requiring extensive preprocessing. This ability to work effectively with raw images simplifies the preprocessing pipeline and saves computational resources, which is advantageous in practical applications. Despite the dataset's limitations, the ViT produced commendable results, showcasing its robustness and adaptability.

However, this study faced challenges due to the relatively small size of the training dataset. ViT models typically excel with large datasets, where they can fully leverage their capacity to learn complex patterns and subtle nuances. The small dataset limited the model's ability to generalize, which is evident from the variability observed in the validation losses across different folds. This limitation underscores the need for larger and more diverse datasets to fully realize the benefits of ViTs in medical imaging tasks.

K-Folds cross-validation was employed to assess the model's generalization performance rigorously. The dataset was divided into five folds, with each fold taking a turn as the validation set while the remaining four folds were used for training, repeated over 20 epochs per fold. The training parameters and evaluation metrics are summarized in Table 17.

The cross-validation results, depicted in Figure **??**, showed that the ViT model achieved an average validation accuracy of 0.9542 with a standard deviation of 0.0260. This high accuracy, coupled with minimal variability, suggests robust performance across different data subsets. However, the average validation loss of 0.1880 with a standard deviation of 0.1124 indicates room for improvement in reducing prediction errors. The variability in validation loss across folds points to the potential benefit of a larger training dataset to stabilize and enhance model performance further.

Despite the dataset constraints, the ViT model's performance is commendable, particularly in identifying complex patterns in medical images that are crucial for accurate diagnostics. Utilizing Optuna for hyperparameter optimization, the study found optimal learning and dropout rates, further enhancing the model's performance. These findings highlight the model's robustness and adaptability, making it a promising tool for medical diagnostics.

In summary, while the ViT model has shown excellent results in brain tumor segmentation, the small training set poses limitations on its generalization capability. Future work should focus on acquiring larger datasets and exploring advanced data augmentation techniques to improve model robustness and reliability. Additionally, continued refinement of preprocessing steps and fine-tuning parameters will further enhance the ViT model's applicability in clinical settings.

5.9 DenseNet121

Cleon Tay Shi Hong (2200649)

DenseNet121 model is a type of convolutional neural network (CNN)[64] architecture developed by Cornell University, This model is designed for machine learning and specifically suitable for image classification tasks. The main concept of DenseNet is to connect all layers in a feed-forward fashion, each level will get a direct input from all levels above. The DenseNet family is a set of neural architectures which are known for their densely connected structure.

5.9.1 Implementation

In the context of our brain tumor classification task, we employed the DenseNet121 architecture, leveraging its pretrained weights from the ImageNet dataset. This approach capitalizes on the robust feature extraction capabilities of DenseNet121 while adapting it to our specific classification requirements through further training. The DenseNet121 model was initialized with weights pretrained on ImageNet and configured to exclude its top layers, which were removed for us to customize the layers tailored for our classification task. The input images were resized to 224×224 pixels with three color channels[64] to match the network's input specifications. This enhance the classification accuracy, ensuring that the images adhere to the network's designated input format.

Following the base layers of DenseNet121, we introduced a Global Average Pooling 2D layer. This pooling operation condenses the spatial dimensions of the feature maps into a single vector per map, enhancing computational efficiency while preserving critical features essential for classification. To mitigate overfitting, a Dropout layer with a dropout rate of 20% was incorporated after Global Average Pooling. This layer randomly deactivates a fraction of neurons during training, promoting robustness by preventing reliance on specific neurons and facilitating the development of a more generalized model.

The classification task culminated in a Dense layer with four units[65], each corresponding to one of the brain tumor classes: meningioma, glioma, pituitary, and notumor. This layer employed a softmax activation function to produce a probability distribution across the classes, ensuring clear and interpretable classification results. For optimization, we employed the Rectified Adam optimizer (RAdam) with a learning rate of 0.0001. RAdam is known for its adaptive nature and has shown effectiveness in stabilizing training and accelerating convergence, which is particularly advantageous in complex transfer learning scenarios. The model was compiled using categorical cross-entropy loss, appropriate for multi-class classification tasks like ours. This loss function computed the discrepancy between predicted probabilities and actual class labels, guiding the model towards accurate predictions.

Training proceeded over 150 epochs with a batch size of 10, utilizing callbacks to enhance performance and prevent overfitting. Specifically, ModelCheckpoint saved the best-performing model based on validation loss, while EarlyStopping halted training when no improvement in validation loss was observed over a specified number of epochs.Upon evaluation on the test set, the DenseNet121 model achieved an impressive accuracy of 99.2% and a validation accuracy of 87.78%. The validation loss was recorded at 0.4012, affirming the model's robust and reliable performance across different datasets.

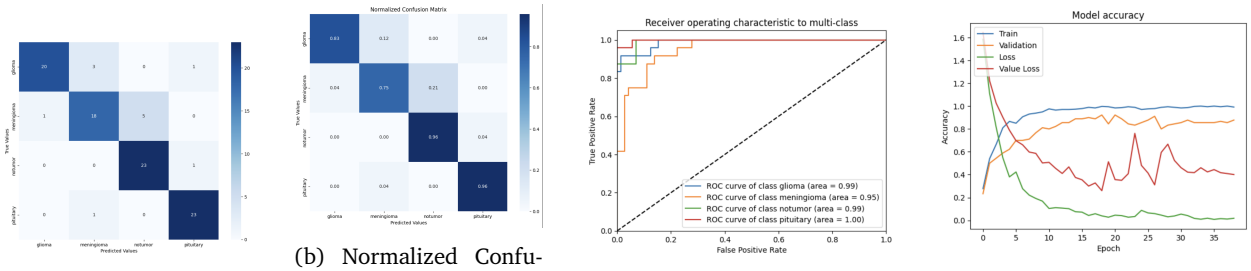
5.9.2 Fine-Tuning

In the process of refining the model for brain tumor classification, we employed a systematic approach to optimize hyperparameters critical for achieving superior performance. This fine-tuning process aimed to tailor the pretrained DenseNet121 architecture to effectively discern between different types of brain tumors.

Initially, we evaluated and explored three different optimizers: RAdam[66], Adam, and SGD, aiming to identify the most effective one for our specific task. The pretrained DenseNet121 model was adapted with additional layers, and each optimizer was evaluated based on its ability to improve validation accuracy during training.

After comprehensive experimentation and comparative analysis, RAdam emerged as the optimal choice. RAdam consistently outperformed Adam and SGD in terms of achieving higher validation accuracy. This selection aligns with research findings, which emphasize RAdam's advantages in stabilizing training dynamics and accelerating convergence rates, particularly in complex transfer learning scenarios like ours.

5.9.3 Results and Evaluation



(a) Confusion Matrix (b) Normalized Confusion Matrix (c) ROC Curve (d) Learning Curve

Figure 29: Confusion Matrix, Normalized Confusion Matrix, ROC Curve, and Learning Curve for Brain Tumor Segmentation

Class	Precision	Recall	F1-Score	Support
meningioma	0.82	0.75	0.78	24
pituitary	0.92	0.96	0.94	24
glioma	0.95	0.83	0.89	24
notumor	0.82	0.96	0.88	24
micro avg	0.88	0.88	0.88	96
macro avg	0.88	0.88	0.87	96
weighted avg	0.88	0.88	0.87	96
samples avg	0.88	0.88	0.88	96

(a) Classification Report for Brain Tumor Segmentation

Metric	Value
DSC	0.8737
Sensitivity	0.8750
Specificity	0.9583
Accuracy	0.8750

(b) Additional Metrics for Brain Tumor Segmentation

Table 18: Classification Report and Additional Metrics for Brain Tumor Segmentation using DenseNet121

In Figures 29a and 29b shows the confusion matrices, it provide a descriptive view of the model's performance on classifying the four brain tumor classes: meningioma, pituitary, glioma, and no tumor. The meningioma and glioma classes exhibit lower rates but still commendable true positive percentage of 75% and 83% respectively. The classification report in Table 18a summarizes the precision, recall, and F1-score for each class. DenseNet121 model shows average precision and recall across all classes, having F1-score of 0.94 for the pituitary class. The overall micro and sample averages for precision, recall, and F1-score stood at 0.88 while the overall micro and weighted averages for precision, recall, and F1-score all stand at 0.87, reflecting consistent and reliable performance.

The ROC curve in Figure 29c displays the true positive rate against the false positive rate for each class. The areas under the curve (AUC) for pituitary class is perfect at 1.00, while notumor and glioma classes show AUCs of 0.99 and meningioma AUCs at 0.95. indicating excellent discriminative ability of the model.

The learning curve in Figure 29d provides valuable insights into how the model learns and improves over time, offering a visual representation of key metrics such as training and validation accuracy and loss over 150 epochs. The alignment between training and validation accuracy, coupled with the consistent decrease in loss values, suggests that the model has effectively learned from the data without exhibiting notable signs of overfitting.

Table 18b presents key performance metrics for the model, including the Dice Similarity Coefficient (DSC), sensitivity, specificity, and accuracy. The model achieved a DSC of 0.8737, indicating substantial overlap between predicted and actual tumor regions. Sensitivity and accuracy, both measuring 0.8750, highlight the model's proficiency in correctly identifying true positives. Additionally, a specificity of 0.9583 demonstrates its effectiveness in accurately identifying true negatives. These metrics collectively underscore the model's strong performance in brain tumor segmentation.

5.9.4 K-Folds Cross-Validation

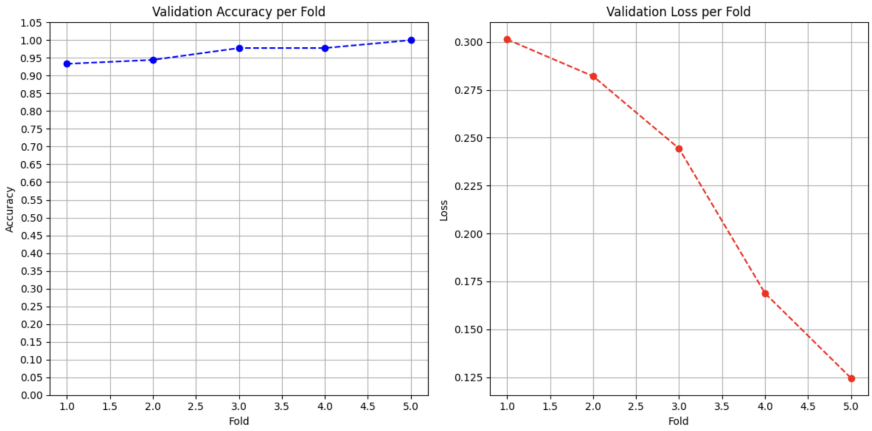


Figure 30: K-Folds Cross-Validation for Brain Tumor Segmentation

K-Folds cross-validation was utilized to comprehensively assess the model's performance across diverse subsets of the dataset. This method divides the dataset into k equal parts, or folds, where the model is trained on $k1$ folds and validated on the remaining fold in each iteration. This process is repeated k times, ensuring that each fold serves as the validation set exactly once. By averaging the validation accuracy and loss metrics across all folds, we obtain a robust measure of the model's performance and its ability to generalize to unseen data.

Table 19b summarizes the outcomes of the K-Folds cross-validation, presenting average values and standard deviations of key metrics. The results demonstrate consistent performance across all folds, with minimal variance in accuracy and loss values. This consistency underscores the model's reliability and effectiveness in accurately segmenting brain tumors.

The setup for K-Folds cross-validation is detailed in Table 19, specifying the parameters chosen to ensure thorough training and robust evaluation of the model. These settings were carefully selected to validate the model's performance comprehensively and reliably across different subsets of the dataset.

Table 19: K-Folds Cross-Validation Testing Conditions

Parameter	**Value**
Number of Folds (k)	5
Epochs	90
Batch Size	10

(a) Training Parameters

Metric	**Value**
Average Validation Accuracy	0.9630
Average Validation Loss	0.2230
Validation Accuracy Std. Dev.	0.0263
Validation Loss Std. Dev.	0.0785

(b) Evaluation Metrics

The outcomes derived from K-Folds cross-validation underscore the model's robustness and dependability in classifying brain tumor images. With minimal variability in accuracy and loss metrics, the results affirm that the model effectively generalizes across diverse data subsets. This capability mitigates concerns of overfitting, ensuring reliable performance suitable for real-world applications.

5.9.5 Conclusion

The implemented DenseNet121 model leverages the powerful architecture pretrained on ImageNet, has been successfully applied in brain tumor segmentation tasks. By harnessing transfer learning and fine-tuning techniques, the DenseNet121 model adapts its feature extraction capabilities to our specific brain tumor classification task. This approach ensures that the model benefits from learned features relevant to medical imaging, enhancing its

ability to classify brain tumor images accurately. Moreover, the choice of the Rectified Adam optimizer facilitates efficient training and convergence. The performance of the model has been rigorously validated through techniques like K-Folds cross-validation, demonstrating robustness and consistency across diverse subsets of our dataset. This validation underscores the model's reliability and potential for clinical applications in diagnosis and treatment planning, making it a valuable tool in medical imaging.

5.10 Other Models

In this section, we will discuss additional models tested with `dataset_19` and their performance.

5.10.1 VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" [67]. The model achieved state-of-the-art performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014. It comprises 16 convolutional layers and 3 fully connected layers. The model, pre-trained on the ImageNet dataset, is readily available in the Keras library.

For our study, we utilized the VGG16 model as referenced in [53]. The architecture included a GlobalAveragePooling layer followed by dropout and dense layers activated by softmax. The Rectified Adam optimizer [52] was employed, with a learning rate of 0.0001, β_1 of 0.9, β_2 of 0.999, and ϵ of $1e-08$. The batch size was set to 10.

The VGG16 model was trained for 100 epochs, achieving an validation accuracy of 0.7556 with a validation loss of 2.1557. Despite the high accuracy reported in [53], this model did not outperform the previously discussed models in this study. Due to limitations in computational power and time, further optimization of this model to achieve greater accuracy was not pursued.

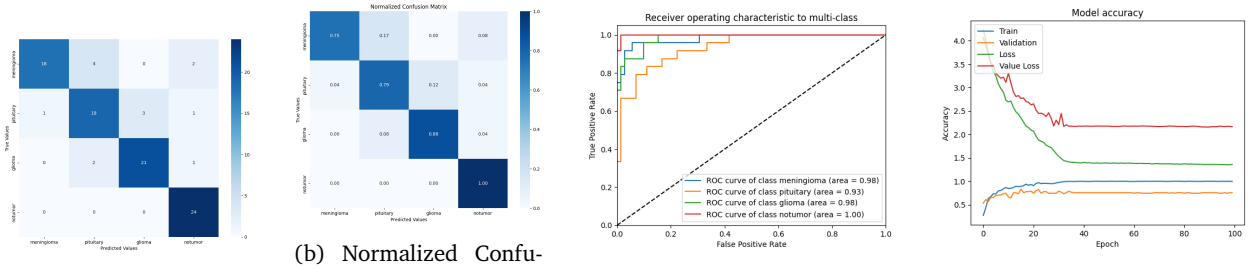
The results of the VGG16 model are summarized in Tables 20 and 21. The tables include both the evaluation metrics and the testing conditions for clarity.

Table 20: Performance Metrics for VGG16 Model

Class	Precision	Recall	F1-Score	Support
meningioma	0.95	0.75	0.84	24
pituitary	0.76	0.79	0.78	24
glioma	0.88	0.88	0.88	24
notumor	0.86	1.00	0.92	24
micro avg	0.85	0.85	0.85	96
macro avg	0.86	0.85	0.85	96
weighted avg	0.86	0.85	0.85	96
samples avg	0.85	0.85	0.85	96

Table 21: Testing Conditions for VGG16 Model

Parameter	Value
Batch Size	10
Image Size	224x224
Learning Rate	0.0001
Epochs	100
Optimizer	Rectified Adam



(a) Confusion Matrix

(b) Normalized Confusion Matrix

(c) ROC Curve

(d) Learning Curve

Figure 31: Confusion Matrix, Normalized Confusion Matrix, ROC Curve, and Learning Curve for Brain Tumor Segmentation

The evaluation of the VGG16 model highlights its decent performance with a moderate accuracy and high validation loss. These metrics suggest that while VGG16 has shown promise in previous studies, its application to our specific dataset and task did not yield superior results compared to other models. This underscores the importance of model selection based on specific task requirements and dataset characteristics. Due to the limitations of compute power and time, further optimization of this model to achieve greater accuracy was not pursued.

6 Conclusion

6.1 Summary of Metrics

Table 22: Summary of Metrics for Different Models

Model	Validation Accuracy	Validation Loss	DSC	Sensitivity	Specificity	Accuracy	Member
U-Net (EfficientNetb1 Backbone)	0.9444	0.2498	0.9372	0.9375	0.9792	0.9375	Woon Jun Wei (2200624)
InceptionV3	0.9333	0.2965	0.9272	0.9271	0.9757	0.9270	Woon Jun Wei (2200624)
Xception	0.9667	0.2177	0.9246	0.9271	0.9757	0.9271	Ong Zi Xuan Max (2200717)
ResNet50	0.9111	0.2656	0.8953	0.8958	0.9652	0.8958	Benjamin Loh Choon How (2201590)
Vision Transformer	0.9375	0.3430	0.9293	0.9271	0.9757	0.9271	Low Hong Sheng Jovian (2203654)
DenseNet121	0.9630	0.2230	0.8737	0.8750	0.9583	0.8750	Cleon Tay Shi Hong (2200649)
VGG16	0.7556	2.1557	0.8527	0.8542	0.9514	0.8542	Woon Jun Wei (2200624)

6.2 Hybrid Architecture for Enhanced Tumor Detection

As detailed in Section 5.2, our model selection is based on the metrics depicted in Table 22 and their individual evaluations. The proposed model is a hybrid that combines the strengths of U-Net++ and EfficientNet. As discussed in Section 5.4, the U-Net++ architecture excels in segmentation tasks, while EfficientNet serves as a highly efficient feature extractor. By integrating these models, we aim to leverage U-Net++'s robust segmentation capabilities and EfficientNet's efficient feature representation, enhancing the accuracy and interpretability of our brain tumor classification and segmentation model.

This proposed model achieves a balance between accuracy, efficiency, and interpretability, making it suitable for real-world clinical applications. By combining the strengths of both U-Net++ and EfficientNet, our goal is to develop a model that can accurately classify and segment brain tumors, providing valuable insights for medical professionals and improving patient outcomes.

Compared to the individual models evaluated in this project, the proposed hybrid model, Eff-U-Net, is expected to achieve higher accuracy, sensitivity, and specificity in brain tumor classification and segmentation tasks.

By merging the best features of U-Net++ and EfficientNet, we aim to create a state-of-the-art model that can assist radiologists and healthcare professionals in diagnosing brain tumors more effectively and efficiently.

In practical applications, Eff-U-Net could be integrated into existing medical imaging systems to provide automated brain tumor classification and segmentation capabilities. This integration would streamline the diagnostic process and offer accurate, interpretable results, potentially revolutionizing brain tumor diagnosis and treatment, ultimately improving patient care and outcomes.

Furthermore, the model's flexibility allows it to be adapted for segmenting and classifying other types of tumors in different parts of the body. This versatility makes it a powerful tool for various medical imaging tasks, as demonstrated by its potential application in detecting pneumonia from X-rays and other types of tumors [17], [18].

7 Future Directions

In the healthcare industry, the development of accurate and efficient diagnostic tools is crucial for improving patient outcomes and reducing healthcare costs. The field of medical imaging, particularly in brain tumor diagnosis, can benefit significantly from advanced deep learning techniques. As such, there are several promising directions for future research and development in this area.

7.1 Addressing Current Limitations

The current project faced several limitations that hindered the performance and generalizability of the models. The most significant limitation was the relatively small dataset size, consisting of only 120 images per class across four classes. This limited number of images restricts the model's ability to generalize effectively to new data. Future work should focus on acquiring larger, more diverse datasets to enhance the model's robustness and accuracy.

Additionally, the varying image sizes posed a challenge during preprocessing and model training. Standardizing the image sizes or developing more sophisticated preprocessing techniques to handle varying dimensions can improve the consistency of the training data. Moreover, the lack of tumor masks prevented the models from learning the precise location and shape of tumors, which is crucial for accurate diagnosis. Future research should incorporate tumor masks to facilitate better model training and improve the accuracy of tumor detection.

Another limitation was the initial lack of experience and background knowledge in medical imaging and deep learning techniques. Continuous learning and collaboration with medical professionals can bridge this gap and provide valuable insights for future improvements.

7.2 Expanding Model Capabilities

To enhance the current model's capabilities, future work could explore the use of Recurrent Neural Networks (RNNs) or Long Short-Term Memory networks (LSTMs) for processing 3D brain MRI slices. These models are well-suited for time series and continuous data, which can be beneficial for capturing spatial and temporal information in 3D medical images. Implementing these models could lead to more accurate classification and segmentation of brain tumors.

Furthermore, while this project focused primarily on classification, future work could expand to include segmentation tasks. Developing models that provide pixelwise probabilities can highlight tumor locations, offering valuable assistance to doctors and radiologists for early intervention. This approach not only improves diagnostic accuracy but also enhances the interpretability of the model's predictions.

7.3 Leveraging Advanced Techniques

Incorporating advanced data augmentation techniques, such as those based on Generative Adversarial Networks (GANs), can also be explored to artificially increase the dataset size and variability. GANs can generate realistic synthetic images that can help the model learn more robust features. Additionally, experimenting with different augmentation strategies can provide insights into the most effective methods for enhancing model performance.

7.4 Exploring Ensemble Models and Advanced Architectures

In this project, various deep learning architectures were employed, including U-Net++, InceptionV3, Xception, Vision Transformers (ViT), EfficientNet, and DenseNet. Each of these models has unique strengths that could be further leveraged in future work. For instance, U-Net++ is particularly effective for segmentation tasks, while InceptionV3 and Xception are known for their robust feature extraction capabilities.

Building upon these models, future research could explore the use of ensemble models, which combine the predictions of multiple models to improve overall performance. Although ensemble models can be computationally expensive, they have the potential to significantly enhance classification and segmentation accuracy by leveraging the strengths of different architectures. Implementing techniques to optimize computational efficiency will be crucial for making ensemble models a viable option.

7.5 Collaboration and Continuous Improvement

Finally, collaboration with medical experts is essential for ensuring the relevance and applicability of the models in clinical settings. Engaging with radiologists and other healthcare professionals can provide critical feedback and identify areas for improvement. Continuous iteration and refinement of the models, guided by expert insights, will be crucial for developing reliable and effective diagnostic tools.

In summary, addressing the current limitations, expanding model capabilities, leveraging advanced techniques, exploring ensemble models, and fostering collaboration with medical professionals are key areas for future work. These efforts will contribute to the development of more accurate and reliable models for brain tumor classification and segmentation, ultimately enhancing early diagnosis and treatment.

8 Reflections

8.1 Woon Jun Wei (2200624)

Throughout this brain tumor classification project, I have engaged in a comprehensive exploration of deep learning models and techniques. The primary objective was to classify brain images using various deep learning architectures, including UNet, Inception, VGG16, and Vision Transformers (ViT). Additionally, I experimented with data augmentation methods, including the potential use of Generative Adversarial Networks (GANs), and reviewed literature on affine augmentations to understand their benefits.

During the project, I applied the insights gained from my literature review to practical tasks such as data exploration, preprocessing, and augmentation. Leveraging tools such as TensorFlow and Keras, I selected and architected models, fine-tuning them with the help of Optuna for hyperparameter optimization. This iterative process of model selection and tuning was guided by referencing various GitHub repositories and research papers, ensuring the integration of state-of-the-art techniques and best practices.

One of the significant learning outcomes of this project was gaining in-depth knowledge about MRIs and brain tumors, and understanding the necessity of automated processes in medical imaging. This knowledge was crucial in appreciating the context and implications of the classification task. Observing performance metrics provided valuable insights into model performance, highlighting the need for more extensive datasets to improve generalization and robustness of the models.

Moreover, I learned advanced techniques for image segmentation, utilizing features extracted through deep learning to enhance classification accuracy. This aspect of the project not only improved my technical skills but also underscored the importance of feature extraction in medical image analysis. The integration of segmentation and classification processes demonstrated the potential for more accurate and efficient diagnostic tools.

I dedicated substantial time and effort to researching and understanding the metrics used for evaluating model performance. Through reviewing research and academic papers, I identified and explored several key metrics, including Dice Similarity Coefficient (DSC), Sensitivity, Specificity, and Accuracy, referenced from Imtiaz et al. [12]. These metrics provided valuable insights into my model's performance, helping to identify strengths and areas for improvement. Additionally, I examined other relevant metrics for segmentation tasks, such as Intersection over Union (IoU), which offered further nuances in evaluating the segmentation quality.

Furthermore, I explored the potential application of my models, such as UNet++ with EfficientNet (Eff-UNet), in other medical domains, including pneumonia detection through X-rays and other imaging tasks. This

exploration highlighted the versatility and adaptability of the models developed, underscoring their potential impact across various medical imaging challenges.

All in all, this project provided a valuable opportunity to integrate theoretical knowledge with practical application. It culminated in a deeper understanding of both the challenges and potential solutions in automated brain tumor classification. The experience emphasized the importance of continuous learning and adaptation in the rapidly evolving field of deep learning and medical imaging, reinforcing the need for ongoing research and development to address complex medical challenges.

8.2 Benjamin Loh Choon How (2201590)

Throughout this project, where I trained a deep learning model to classify brain MRI images, I have gained a deeper understanding of the broader implications of applying machine learning in medical imaging. This journey has helped me appreciate the complexities and benefits of using advanced technology to improve healthcare. To tackle the problem statement, I started exploring multiple pre-trained models I could use for transfer learning due to the small amount of data we were given to train our model, and I chose ResNet-50 as my model after reading multiple research papers which recorded good performances for tasks like classifying brain MRI images. I started off by carefully preparing the data after learning useful preprocessing techniques referenced from the research papers I have read. Using tools like OpenCV and Python, I converted images to grayscale to make them simpler and blurred them to remove noise. I then used techniques to highlight the brain and remove unnecessary background. This careful preparation made sure the data was clear and focused, which is important for making the model work better.

Once the data was ready, I moved on to setting up the model for training. I implemented ResNet-50 using TensorFlow and Keras, which are powerful tools for deep learning. I utilized data augmentation techniques to increase the diversity of the training data, such as flipping, rotating, and shifting the images. These methods were crucial for enhancing the model's ability to generalize from the limited dataset. The process of hyperparameter tuning was guided by the Optuna framework, which allowed me to fine-tune the model's settings systematically and efficiently. This iterative process helped me achieve the best possible model performance by experimenting with different configurations and selecting the most effective parameters.

During training, I focused on strategies to ensure that the model learned effectively and avoided overfitting. I used ModelCheckpoint to save the best version of the model based on validation performance and EarlyStopping to halt training if no improvement was observed, which prevented unnecessary training time and potential overfitting. The training process was monitored carefully to maintain a balance between model complexity and performance. I also applied K-Folds cross-validation to evaluate the model's generalization capabilities, providing a robust estimate of how well the model would perform on unseen data.

Reflecting on this experience, I have come to understand the critical role of machine learning in advancing medical diagnostics. The knowledge I gained about the importance of accurate and reliable MRI image classification has highlighted the potential impact of these technologies on patient care and medical decision-making. It reinforced the need for ongoing research and development in this field to continue improving diagnostic tools and outcomes for patients.

Overall, this project has been a valuable learning journey that has equipped me with practical skills and insights into machine learning and medical imaging. It has demonstrated the importance of meticulous data preparation, thoughtful model selection, and continuous optimization. The experience has solidified my ability to develop sophisticated machine learning solutions for healthcare challenges and has emphasized the significance of innovation and precision in the ever-evolving landscape of technology and healthcare. This project not only expanded my technical expertise but also deepened my commitment to using advanced technologies to contribute to better healthcare outcomes.

8.3 Low Hong Sheng Jovian (2203654)

As a computer science student, working on a machine learning project to classify brain MRI images has been an enriching experience. This project allowed me to apply theoretical knowledge to a real-world problem with significant medical implications. I faced the initial challenge of understanding the complexities of medical imaging and brain MRI classification, which required deep engagement with medical literature and advanced machine learning models like CNNs and Vision Transformers (ViTs).

One of the most rewarding aspects was the practical application of classroom concepts. Implementing and experimenting with convolutional layers, self-attention mechanisms, overfitting, and cross-validation provided a deeper, more intuitive grasp of these theories. Overcoming computational limitations, even with resources like Google Colab Pro, was a significant part of the project, necessitating innovative solutions to balance computational efficiency and model performance.

A particularly enlightening part of the project was researching and experimenting with Vision Transformers (ViTs). Initially designed for natural language processing, ViTs presented a novel approach to image classification through their self-attention mechanism. Diving into the intricacies of ViTs, understanding their architecture, and experimenting with them firsthand was incredibly satisfying. This hands-on experimentation allowed me to witness the model's capabilities and limitations, particularly in handling raw images without extensive preprocessing.

Throughout the project, I gained valuable insights into the strengths and weaknesses of different deep learning models. CNNs handled spatial hierarchies well, while ViTs excelled in dynamically focusing on relevant image segments. These insights were critical in selecting appropriate models for various tasks. The project also highlighted the potential impact of machine learning in medical diagnostics, emphasizing the need for larger, diverse datasets to improve model generalization and reliability.

However, the journey was not without its difficulties. Coming into this project with no prior experience in machine learning, I often found myself overwhelmed by the vastness of the field. It was challenging to understand the many complex concepts and algorithms, and I frequently had to rely on extensive research just to have a clue about what was going on. Determining the right direction to move in was often daunting and required a lot of trial and error.

This experience has reinforced my passion for machine learning and its transformative potential. It has equipped me with the skills to tackle more complex problems and motivated me to explore advanced techniques like data augmentation and transfer learning. Looking ahead, if I have sufficient computational power, I am eager to explore the potential of segmentation with ViT models, particularly by splitting the dataset into patch sizes of 16. This future direction could offer deeper insights and improvements in medical image analysis, further advancing the capabilities of machine learning in healthcare.

This project has been a crucial part of my academic journey, greatly shaping my skills and aspirations as a computer science student. It has deepened my understanding of machine learning and its applications, especially in healthcare. Working on a real-world problem has improved my analytical and problem-solving skills, teaching me perseverance and adaptability.

The project emphasized the importance of continuous learning and staying updated with technology advancements. It inspired me to pursue further studies in machine learning, focusing on medical imaging and diagnostics. Collaborating with peers and receiving guidance from mentors provided diverse perspectives and improved my teamwork skills.

Overall, this experience has given me a strong foundation in machine learning, practical skills in modern frameworks, and an understanding of the field's practical applications. It has prepared me for future endeavors, driving me to contribute to innovative solutions that solve real-world problems.

8.4 Ong Zi Xuan Max (2200717)

As an individual with a fundamental knowledge of neural networks from a previous university module, this brain tumor MRI image classification project has provided an opportunity to gain a more in-depth comprehension of the applications of machine learning. The goal was to develop various deep learning neural network models as a team that could classify different classes of brain tumor MRI images. To achieve this goal, extensive research and understanding of various applications in each stage of building a specific model architecture were required, allowing me to expand and deepen my knowledge in machine learning.

Putting to work the insights gained from analyzing multiple research papers and online articles, I applied them to practical tasks such as tailoring augmentation and distributing data to the selected pre-trained Xception model architecture. Continuous reading of framework and library documentation aided in understanding how to create suitable tools and solutions for various phases of building the model. This included creating and modifying the architecture of the pre-trained model, implementing various callbacks to train the model effectively, and using techniques to prevent underfitting and overfitting while evaluating the model after each trial.

Fine-tuning the model was a significant phase, characterized by a repetitive cycle of reading, comprehending, tuning, and evaluation. Leveraging Optuna for hyperparameter optimization helped reduce a considerable amount

of time in manual tuning and provided reliable results. It was not just a plug-and-play framework but required grasping the specifics of each execution line written.

Reflecting on this experience, I have come to appreciate the complexities and potential of applying advanced machine learning techniques to real-world medical imaging problems. The project reinforced the importance of meticulous data preparation, thoughtful model selection, and continuous optimization. It has equipped me with practical skills and insights, deepening my knowledge and widening my exposure to using advanced technologies in tailoring deep neural network models for specific use cases.

8.5 Cleon Tay Shi Hong (2200649)

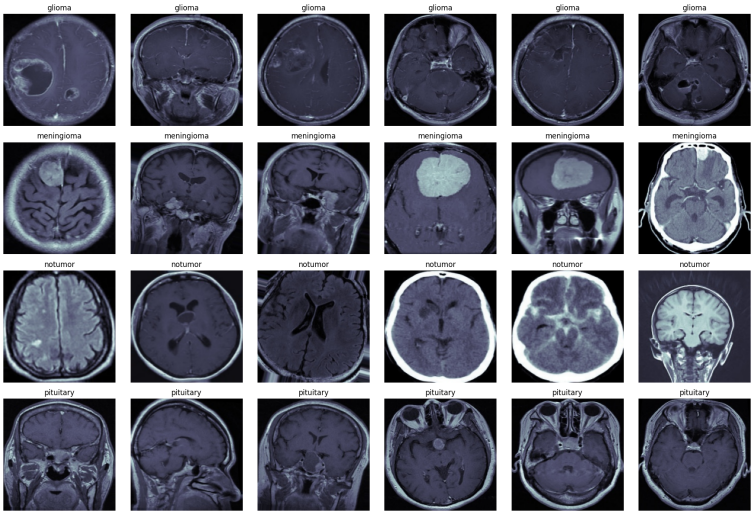
Through this brain tumor classification project, I gained the opportunity to get a deeper understanding of machine learning and its applications. The objective of this project is to research and develop a model each for classifying brain tumor MRI images, then finally decide on the best to adapt it to the project. To move forward, we did many research and try to understand more model architecture and how it could be implemented and adapted into our project.

After researching on a few models by reading on research paper and articles written by the other developers, I chose the pre-trained DenseNet121 model architecture. Using this pre-trained model, I modify and customized it to fit our objective needs. Continuously reading and understanding how to develop this model architecture from the different phases and layers. This involved in designing and adjusting of the pre-trained model, and deploying methods to prevent overfitting and underfitting while evaluating the result for each iterations.

Looking back on this experience in developing and customising a pre-trained model to classify brain tumor MRI images for the medical industrial uses. I gained deeper appreciation for the potential application of the machine learning methods in the medical industry. After completing this assignment I feel more capable of handling such application that could help further improve the current technology used in the medical industry.

Overall, this project has given me a valuable learning experience, allowing me to practice my practical skills and insights into machine learning. It highlighted on how important data preparation, model selection and optimization is. This experience has up skilled my ability to develop more complex machine learning programs for the healthcare industry. Also pointed out the importance of innovating in the field of the technology an healthcare. This project not only deepened my technical expertise, but also enhanced my commitment to push the advance technology to another level to improve healthcare outcome.

A Preprocessing Appendix



(a) Training Set (b) Test Set

Figure 32: Tumor images in the training and test sets.

B U-Net Appendix

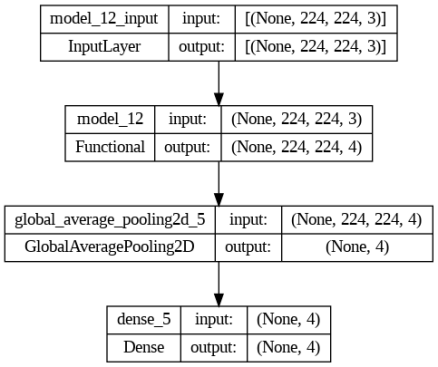


Figure 33: U-Net++ model architecture

Where `model_12` is the U-Net++ model with EfficientNetB1 encoder. The Output Layer of `model_12` is also used for the segmentation attempt to show what the model deems as the important parts of the image for the classification task.

B.1 Segmentation Algorithm

Algorithm 1 Image Prediction and Visualization with Mask Overlay

Define class names: `classes` = {glioma, meningioma, notumor, pituitary}
Get predictions from the U-Net model: `predictions_model_2` ← `model.layers[0].predict(test_generator)`
Get prediction probabilities: `predictions_prob` ← `model.predict(test_generator)`
Get predicted classes: `predictions` ← arg max(`predictions_prob`, axis = 1)
Set threshold for binary mask: `threshold` = 0.9
Create binary mask: `binary_mask` ← (`predictions_model_2` > `threshold`).`astype(np.uint8)`
Initialize displayed class counts: `displayed_classes_counts` ← {`class_name` : 0 for class_name in classes}
Set number of images per class to display: `num_images_per_class` = 2
Initialize plot figure
for `each image in test_generator` **do**
    Get actual class index and name: `actual_class_index` ← `test_generator.classes[i]`, `actual_class_name` ← `classes[actual_class_index]`
    if `displayed_classes_counts[actual_class_name]` ≥ `num_images_per_class` **then**
        Continue to next image
    end if
    Load original image
    Reshape binary mask to match original image shape: `reshaped_binary_mask` ← `binary_mask[i][:, :, 0]`
    Overlay mask on original image
    Get predicted class name: `predicted_class_name` ← `classes[predictions[i]]`
    Plot original image and overlayed mask
    Update displayed class counts: `displayed_classes_counts[actual_class_name]` ← `displayed_classes_counts[actual_class_name]` + 1
    if `all counts in displayed_classes_counts` ≥ `num_images_per_class` **then**
        Break the loop
    end if
end for
Display plot with tight layout

B.2 Additional Segmentation Results

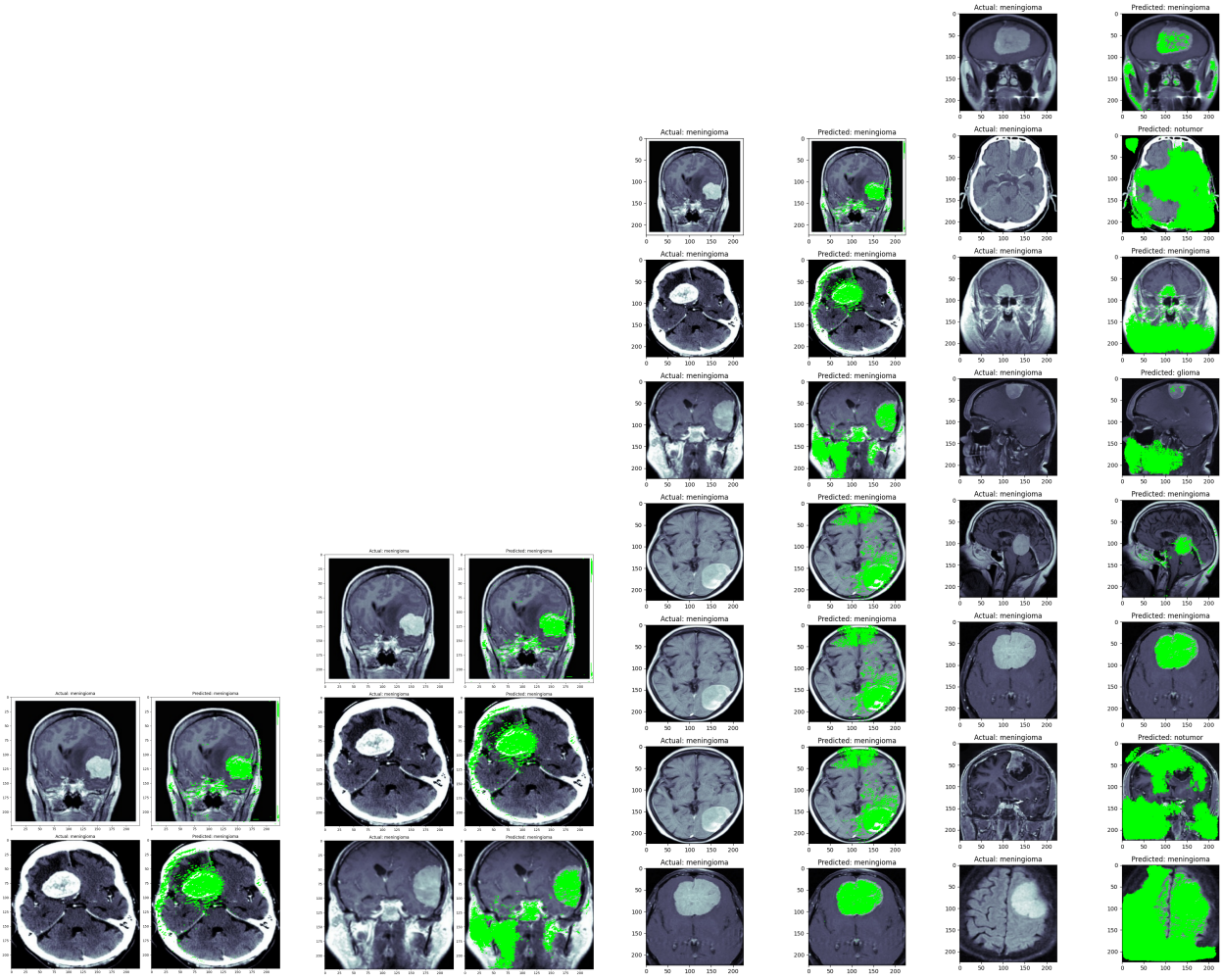


Figure 34: Additional Segmentation Results for Menigioma

Notice that the segmentation results are not perfect, but they are able to capture the general shape of the tumor. Some images that are predicted to not be part of the meningioma are also highlighted, since they are predicted to be part of another tumor.

B.3 Jupyter Notebook

unet

June 13, 2024

[1]:
```
import os

import keras

print("Keras = {}".format(keras.__version__))
import tensorflow as tf

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # or any {'0', '1', '2'}
import matplotlib.pyplot as plt
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
import seaborn as sns
import pandas as pd


# Print gpus
gpus = tf.config.experimental.list_physical_devices('GPU')
print("Num GPUs Available: ", len(gpus))

model_dir = './models/'
model_file = model_dir + 'unetpp.h5'
```

```
Keras = 2.15.0
Num GPUs Available:  1
```

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[3]:
```
! unzip -o drive/MyDrive/dataset_19.zip

! pwd
```

```
Archive:  drive/MyDrive/dataset_19.zip
   creating: dataset_19/glioma/
  inflating: dataset_19/glioma/Te-gl_0010.jpg
  inflating: dataset_19/glioma/Te-gl_0016.jpg
```

```
inflating: dataset_19/glioma/Te-gl_0020.jpg
inflating: dataset_19/glioma/Te-gl_0026.jpg
inflating: dataset_19/glioma/Te-gl_0035.jpg
inflating: dataset_19/glioma/Te-gl_0041.jpg
inflating: dataset_19/glioma/Te-gl_0043.jpg
inflating: dataset_19/glioma/Te-gl_0044.jpg
inflating: dataset_19/glioma/Te-gl_0045.jpg
inflating: dataset_19/glioma/Te-gl_0057.jpg
inflating: dataset_19/glioma/Te-gl_0060.jpg
inflating: dataset_19/glioma/Te-gl_0075.jpg
inflating: dataset_19/glioma/Te-gl_0079.jpg
inflating: dataset_19/glioma/Te-gl_0085.jpg
inflating: dataset_19/glioma/Te-gl_0090.jpg
inflating: dataset_19/glioma/Te-gl_0107.jpg
inflating: dataset_19/glioma/Te-gl_0114.jpg
inflating: dataset_19/glioma/Te-gl_0125.jpg
inflating: dataset_19/glioma/Te-gl_0126.jpg
inflating: dataset_19/glioma/Te-gl_0127.jpg
inflating: dataset_19/glioma/Te-gl_0136.jpg
inflating: dataset_19/glioma/Te-gl_0144.jpg
inflating: dataset_19/glioma/Te-gl_0154.jpg
inflating: dataset_19/glioma/Te-gl_0155.jpg
inflating: dataset_19/glioma/Te-gl_0162.jpg
inflating: dataset_19/glioma/Te-gl_0166.jpg
inflating: dataset_19/glioma/Te-gl_0167.jpg
inflating: dataset_19/glioma/Te-gl_0168.jpg
inflating: dataset_19/glioma/Te-gl_0172.jpg
inflating: dataset_19/glioma/Te-gl_0183.jpg
inflating: dataset_19/glioma/Te-gl_0192.jpg
inflating: dataset_19/glioma/Te-gl_0195.jpg
inflating: dataset_19/glioma/Te-gl_0199.jpg
inflating: dataset_19/glioma/Tr-gl_0016.jpg
inflating: dataset_19/glioma/Tr-gl_0026.jpg
inflating: dataset_19/glioma/Tr-gl_0031.jpg
inflating: dataset_19/glioma/Tr-gl_0039.jpg
inflating: dataset_19/glioma/Tr-gl_0049.jpg
inflating: dataset_19/glioma/Tr-gl_0052.jpg
inflating: dataset_19/glioma/Tr-gl_0054.jpg
inflating: dataset_19/glioma/Tr-gl_0062.jpg
inflating: dataset_19/glioma/Tr-gl_0072.jpg
inflating: dataset_19/glioma/Tr-gl_0078.jpg
inflating: dataset_19/glioma/Tr-gl_0084.jpg
inflating: dataset_19/glioma/Tr-gl_0086.jpg
inflating: dataset_19/glioma/Tr-gl_0096.jpg
inflating: dataset_19/glioma/Tr-gl_0106.jpg
inflating: dataset_19/glioma/Tr-gl_0111.jpg
inflating: dataset_19/glioma/Tr-gl_0113.jpg
inflating: dataset_19/glioma/Tr-gl_0118.jpg
```

```
inflating: dataset_19/glioma/Tr-gl_0119.jpg
inflating: dataset_19/glioma/Tr-gl_0127(1).jpg
inflating: dataset_19/glioma/Tr-gl_0135.jpg
inflating: dataset_19/glioma/Tr-gl_0136.jpg
inflating: dataset_19/glioma/Tr-gl_0138.jpg
inflating: dataset_19/glioma/Tr-gl_0145.jpg
inflating: dataset_19/glioma/Tr-gl_0155.jpg
inflating: dataset_19/glioma/Tr-gl_0161.jpg
inflating: dataset_19/glioma/Tr-gl_0166.jpg
inflating: dataset_19/glioma/Tr-gl_0169.jpg
inflating: dataset_19/glioma/Tr-gl_0184.jpg
inflating: dataset_19/glioma/Tr-gl_0187.jpg
inflating: dataset_19/glioma/Tr-gl_0191.jpg
inflating: dataset_19/glioma/Tr-gl_0194.jpg
inflating: dataset_19/glioma/Tr-gl_0197.jpg
inflating: dataset_19/glioma/Tr-gl_0204.jpg
inflating: dataset_19/glioma/Tr-gl_0206.jpg
inflating: dataset_19/glioma/Tr-gl_0207.jpg
inflating: dataset_19/glioma/Tr-gl_0224.jpg
inflating: dataset_19/glioma/Tr-gl_0226.jpg
inflating: dataset_19/glioma/Tr-gl_0230.jpg
inflating: dataset_19/glioma/Tr-gl_0237.jpg
inflating: dataset_19/glioma/Tr-gl_0247.jpg
inflating: dataset_19/glioma/Tr-gl_0250.jpg
inflating: dataset_19/glioma/Tr-gl_0268.jpg
inflating: dataset_19/glioma/Tr-gl_0281.jpg
inflating: dataset_19/glioma/Tr-gl_0283.jpg
inflating: dataset_19/glioma/Tr-gl_0284.jpg
inflating: dataset_19/glioma/Tr-gl_0286.jpg
inflating: dataset_19/glioma/Tr-gl_0294.jpg
inflating: dataset_19/glioma/Tr-gl_0303.jpg
inflating: dataset_19/glioma/Tr-gl_0304.jpg
inflating: dataset_19/glioma/Tr-gl_0311.jpg
inflating: dataset_19/glioma/Tr-gl_0312.jpg
inflating: dataset_19/glioma/Tr-gl_0320.jpg
inflating: dataset_19/glioma/Tr-gl_0322.jpg
inflating: dataset_19/glioma/Tr-gl_0327.jpg
inflating: dataset_19/glioma/Tr-gl_0330.jpg
inflating: dataset_19/glioma/Tr-gl_0334.jpg
inflating: dataset_19/glioma/Tr-gl_0338.jpg
inflating: dataset_19/glioma/Tr-gl_0343.jpg
inflating: dataset_19/glioma/Tr-gl_0349.jpg
inflating: dataset_19/glioma/Tr-gl_0358.jpg
inflating: dataset_19/glioma/Tr-gl_0369.jpg
inflating: dataset_19/glioma/Tr-gl_0373.jpg
inflating: dataset_19/glioma/Tr-gl_0378.jpg
inflating: dataset_19/glioma/Tr-gl_0396.jpg
inflating: dataset_19/glioma/Tr-gl_0406.jpg
```

```
inflating: dataset_19/glioma/Tr-gl_0407.jpg
inflating: dataset_19/glioma/Tr-gl_0422.jpg
inflating: dataset_19/glioma/Tr-gl_0426.jpg
inflating: dataset_19/glioma/Tr-gl_0428.jpg
inflating: dataset_19/glioma/Tr-gl_0437.jpg
inflating: dataset_19/glioma/Tr-gl_0444.jpg
inflating: dataset_19/glioma/Tr-gl_0450.jpg
inflating: dataset_19/glioma/Tr-gl_0451.jpg
inflating: dataset_19/glioma/Tr-gl_0454.jpg
inflating: dataset_19/glioma/Tr-gl_0455.jpg
inflating: dataset_19/glioma/Tr-gl_0457.jpg
inflating: dataset_19/glioma/Tr-gl_0459.jpg
inflating: dataset_19/glioma/Tr-gl_0462.jpg
inflating: dataset_19/glioma/Tr-gl_0465.jpg
inflating: dataset_19/glioma/Tr-gl_0474.jpg
inflating: dataset_19/glioma/Tr-gl_0490.jpg
inflating: dataset_19/glioma/Tr-gl_0493.jpg
inflating: dataset_19/glioma/Tr-gl_0494(1).jpg
inflating: dataset_19/glioma/Tr-gl_0495.jpg
inflating: dataset_19/glioma/Tr-gl_0502.jpg
inflating: dataset_19/glioma/Tr-gl_0508.jpg
inflating: dataset_19/glioma/Tr-gl_0509.jpg
 creating: dataset_19/meningioma/
inflating: dataset_19/meningioma/Te-me_0014.jpg
inflating: dataset_19/meningioma/Te-me_0015.jpg
inflating: dataset_19/meningioma/Te-me_0016.jpg
inflating: dataset_19/meningioma/Te-me_0017(3).jpg
inflating: dataset_19/meningioma/Te-me_0017(4).jpg
inflating: dataset_19/meningioma/Te-me_0017.jpg
inflating: dataset_19/meningioma/Te-me_0026.jpg
inflating: dataset_19/meningioma/Te-me_0028.jpg
inflating: dataset_19/meningioma/Te-me_0032.jpg
inflating: dataset_19/meningioma/Te-me_0046.jpg
inflating: dataset_19/meningioma/Te-me_0053.jpg
inflating: dataset_19/meningioma/Te-me_0058.jpg
inflating: dataset_19/meningioma/Te-me_0068.jpg
inflating: dataset_19/meningioma/Te-me_0080.jpg
inflating: dataset_19/meningioma/Te-me_0082.jpg
inflating: dataset_19/meningioma/Te-me_0095.jpg
inflating: dataset_19/meningioma/Te-me_0097.jpg
inflating: dataset_19/meningioma/Te-me_0101.jpg
inflating: dataset_19/meningioma/Te-me_0104.jpg
inflating: dataset_19/meningioma/Te-me_0108.jpg
inflating: dataset_19/meningioma/Te-me_0119.jpg
inflating: dataset_19/meningioma/Te-me_0128.jpg
inflating: dataset_19/meningioma/Te-me_0140.jpg
inflating: dataset_19/meningioma/Te-me_0146(1).jpg
inflating: dataset_19/meningioma/Te-me_0147(1).jpg
```

```
inflating: dataset_19/meningioma/Te-me_0149.jpg
inflating: dataset_19/meningioma/Te-me_0151(1).jpg
inflating: dataset_19/meningioma/Te-me_0156.jpg
inflating: dataset_19/meningioma/Te-me_0160(1).jpg
inflating: dataset_19/meningioma/Te-me_0162(4).jpg
inflating: dataset_19/meningioma/Te-me_0165.jpg
inflating: dataset_19/meningioma/Te-me_0166.jpg
inflating: dataset_19/meningioma/Te-me_0169.jpg
inflating: dataset_19/meningioma/Te-me_0170(1).jpg
inflating: dataset_19/meningioma/Te-me_0170.jpg
inflating: dataset_19/meningioma/Te-me_0174(1).jpg
inflating: dataset_19/meningioma/Te-me_0175(1).jpg
inflating: dataset_19/meningioma/Te-me_0176.jpg
inflating: dataset_19/meningioma/Te-me_0182.jpg
inflating: dataset_19/meningioma/Te-me_0184(2).jpg
inflating: dataset_19/meningioma/Te-me_0184.jpg
inflating: dataset_19/meningioma/Te-me_0186.jpg
inflating: dataset_19/meningioma/Te-me_0192.jpg
inflating: dataset_19/meningioma/Te-me_0193.jpg
inflating: dataset_19/meningioma/Te-me_0198.jpg
inflating: dataset_19/meningioma/Te-me_0200(2).jpg
inflating: dataset_19/meningioma/Te-me_0202(1).jpg
inflating: dataset_19/meningioma/Tr-me_0013.jpg
inflating: dataset_19/meningioma/Tr-me_0027.jpg
inflating: dataset_19/meningioma/Tr-me_0029.jpg
inflating: dataset_19/meningioma/Tr-me_0030.jpg
inflating: dataset_19/meningioma/Tr-me_0039.jpg
inflating: dataset_19/meningioma/Tr-me_0043.jpg
inflating: dataset_19/meningioma/Tr-me_0067.jpg
inflating: dataset_19/meningioma/Tr-me_0068.jpg
inflating: dataset_19/meningioma/Tr-me_0069.jpg
inflating: dataset_19/meningioma/Tr-me_0071.jpg
inflating: dataset_19/meningioma/Tr-me_0073.jpg
inflating: dataset_19/meningioma/Tr-me_0076.jpg
inflating: dataset_19/meningioma/Tr-me_0098.jpg
inflating: dataset_19/meningioma/Tr-me_0099.jpg
inflating: dataset_19/meningioma/Tr-me_0101.jpg
inflating: dataset_19/meningioma/Tr-me_0107.jpg
inflating: dataset_19/meningioma/Tr-me_0108.jpg
inflating: dataset_19/meningioma/Tr-me_0113.jpg
inflating: dataset_19/meningioma/Tr-me_0119.jpg
inflating: dataset_19/meningioma/Tr-me_0121.jpg
inflating: dataset_19/meningioma/Tr-me_0125.jpg
inflating: dataset_19/meningioma/Tr-me_0134.jpg
inflating: dataset_19/meningioma/Tr-me_0140.jpg
inflating: dataset_19/meningioma/Tr-me_0147.jpg
inflating: dataset_19/meningioma/Tr-me_0149.jpg
inflating: dataset_19/meningioma/Tr-me_0159.jpg
```

```
inflating: dataset_19/meningioma/Tr-me_0167.jpg
inflating: dataset_19/meningioma/Tr-me_0182.jpg
inflating: dataset_19/meningioma/Tr-me_0183.jpg
inflating: dataset_19/meningioma/Tr-me_0187.jpg
inflating: dataset_19/meningioma/Tr-me_0199.jpg
inflating: dataset_19/meningioma/Tr-me_0211.jpg
inflating: dataset_19/meningioma/Tr-me_0223.jpg
inflating: dataset_19/meningioma/Tr-me_0227.jpg
inflating: dataset_19/meningioma/Tr-me_0237.jpg
inflating: dataset_19/meningioma/Tr-me_0247.jpg
inflating: dataset_19/meningioma/Tr-me_0250.jpg
inflating: dataset_19/meningioma/Tr-me_0251.jpg
inflating: dataset_19/meningioma/Tr-me_0256.jpg
inflating: dataset_19/meningioma/Tr-me_0261.jpg
inflating: dataset_19/meningioma/Tr-me_0272.jpg
inflating: dataset_19/meningioma/Tr-me_0282.jpg
inflating: dataset_19/meningioma/Tr-me_0284.jpg
inflating: dataset_19/meningioma/Tr-me_0301.jpg
inflating: dataset_19/meningioma/Tr-me_0303.jpg
inflating: dataset_19/meningioma/Tr-me_0317.jpg
inflating: dataset_19/meningioma/Tr-me_0318.jpg
inflating: dataset_19/meningioma/Tr-me_0323.jpg
inflating: dataset_19/meningioma/Tr-me_0338.jpg
inflating: dataset_19/meningioma/Tr-me_0347.jpg
inflating: dataset_19/meningioma/Tr-me_0361.jpg
inflating: dataset_19/meningioma/Tr-me_0368.jpg
inflating: dataset_19/meningioma/Tr-me_0371.jpg
inflating: dataset_19/meningioma/Tr-me_0378.jpg
inflating: dataset_19/meningioma/Tr-me_0393.jpg
inflating: dataset_19/meningioma/Tr-me_0400.jpg
inflating: dataset_19/meningioma/Tr-me_0402.jpg
inflating: dataset_19/meningioma/Tr-me_0420.jpg
inflating: dataset_19/meningioma/Tr-me_0422.jpg
inflating: dataset_19/meningioma/Tr-me_0432.jpg
inflating: dataset_19/meningioma/Tr-me_0433.jpg
inflating: dataset_19/meningioma/Tr-me_0434.jpg
inflating: dataset_19/meningioma/Tr-me_0441.jpg
inflating: dataset_19/meningioma/Tr-me_0458.jpg
inflating: dataset_19/meningioma/Tr-me_0461.jpg
inflating: dataset_19/meningioma/Tr-me_0466.jpg
inflating: dataset_19/meningioma/Tr-me_0467.jpg
inflating: dataset_19/meningioma/Tr-me_0479.jpg
inflating: dataset_19/meningioma/Tr-me_0480.jpg
inflating: dataset_19/meningioma/Tr-me_0485.jpg
inflating: dataset_19/meningioma/Tr-me_0492.jpg
inflating: dataset_19/meningioma/Tr-me_0494.jpg
inflating: dataset_19/meningioma/Tr-me_0510.jpg
 creating: dataset_19/notumor/
```

```
inflating: dataset_19/notumor/Te-no_0010.jpg
inflating: dataset_19/notumor/Te-no_0011.jpg
inflating: dataset_19/notumor/Te-no_0025.jpg
inflating: dataset_19/notumor/Te-no_0029.jpg
inflating: dataset_19/notumor/Te-no_0031.jpg
inflating: dataset_19/notumor/Te-no_0041.jpg
inflating: dataset_19/notumor/Te-no_0043.jpg
inflating: dataset_19/notumor/Te-no_0060.jpg
inflating: dataset_19/notumor/Te-no_0067.jpg
inflating: dataset_19/notumor/Te-no_0069.jpg
inflating: dataset_19/notumor/Te-no_0079.jpg
inflating: dataset_19/notumor/Te-no_0097.jpg
inflating: dataset_19/notumor/Te-no_0111.jpg
inflating: dataset_19/notumor/Te-no_0114.jpg
inflating: dataset_19/notumor/Te-no_0119.jpg
inflating: dataset_19/notumor/Te-no_0121.jpg
inflating: dataset_19/notumor/Te-no_0127.jpg
inflating: dataset_19/notumor/Te-no_0135.jpg
inflating: dataset_19/notumor/Te-no_0141.jpg
inflating: dataset_19/notumor/Te-no_0145.jpg
inflating: dataset_19/notumor/Te-no_0146.jpg
inflating: dataset_19/notumor/Te-no_0153.jpg
inflating: dataset_19/notumor/Te-no_0158.jpg
inflating: dataset_19/notumor/Te-no_0162.jpg
inflating: dataset_19/notumor/Te-no_0168.jpg
inflating: dataset_19/notumor/Te-no_0172.jpg
inflating: dataset_19/notumor/Te-no_0176.jpg
inflating: dataset_19/notumor/Te-no_0178.jpg
inflating: dataset_19/notumor/Te-no_0180.jpg
inflating: dataset_19/notumor/Te-no_0189.jpg
inflating: dataset_19/notumor/Te-no_0190.jpg
inflating: dataset_19/notumor/Te-no_0194.jpg
inflating: dataset_19/notumor/Te-no_0198.jpg
inflating: dataset_19/notumor/Te-no_0200.jpg
inflating: dataset_19/notumor/Te-no_0201.jpg
inflating: dataset_19/notumor/Te-no_0202.jpg
inflating: dataset_19/notumor/Te-no_0203.jpg
inflating: dataset_19/notumor/Tr-no_0032.jpg
inflating: dataset_19/notumor/Tr-no_0035.jpg
inflating: dataset_19/notumor/Tr-no_0036.jpg
inflating: dataset_19/notumor/Tr-no_0041.jpg
inflating: dataset_19/notumor/Tr-no_0044.jpg
inflating: dataset_19/notumor/Tr-no_0056.jpg
inflating: dataset_19/notumor/Tr-no_0063.jpg
inflating: dataset_19/notumor/Tr-no_0065.jpg
inflating: dataset_19/notumor/Tr-no_0076.jpg
inflating: dataset_19/notumor/Tr-no_0077.jpg
inflating: dataset_19/notumor/Tr-no_0079.jpg
```

```
inflating: dataset_19/notumor/Tr-no_0081.jpg
inflating: dataset_19/notumor/Tr-no_0090.jpg
inflating: dataset_19/notumor/Tr-no_0091.jpg
inflating: dataset_19/notumor/Tr-no_0103.jpg
inflating: dataset_19/notumor/Tr-no_0104.jpg
inflating: dataset_19/notumor/Tr-no_0118.jpg
inflating: dataset_19/notumor/Tr-no_0121.jpg
inflating: dataset_19/notumor/Tr-no_0124.jpg
inflating: dataset_19/notumor/Tr-no_0125.jpg
inflating: dataset_19/notumor/Tr-no_0130.jpg
inflating: dataset_19/notumor/Tr-no_0135.jpg
inflating: dataset_19/notumor/Tr-no_0137.jpg
inflating: dataset_19/notumor/Tr-no_0141.jpg
inflating: dataset_19/notumor/Tr-no_0142.jpg
inflating: dataset_19/notumor/Tr-no_0150.jpg
inflating: dataset_19/notumor/Tr-no_0157.jpg
inflating: dataset_19/notumor/Tr-no_0163.jpg
inflating: dataset_19/notumor/Tr-no_0165.jpg
inflating: dataset_19/notumor/Tr-no_0168.jpg
inflating: dataset_19/notumor/Tr-no_0170.jpg
inflating: dataset_19/notumor/Tr-no_0171.jpg
inflating: dataset_19/notumor/Tr-no_0174.jpg
inflating: dataset_19/notumor/Tr-no_0179.jpg
inflating: dataset_19/notumor/Tr-no_0183.jpg
inflating: dataset_19/notumor/Tr-no_0188.jpg
inflating: dataset_19/notumor/Tr-no_0189.jpg
inflating: dataset_19/notumor/Tr-no_0190.jpg
inflating: dataset_19/notumor/Tr-no_0194.jpg
inflating: dataset_19/notumor/Tr-no_0199.jpg
inflating: dataset_19/notumor/Tr-no_0200.jpg
inflating: dataset_19/notumor/Tr-no_0203.jpg
inflating: dataset_19/notumor/Tr-no_0211.jpg
inflating: dataset_19/notumor/Tr-no_0212.jpg
inflating: dataset_19/notumor/Tr-no_0214.jpg
inflating: dataset_19/notumor/Tr-no_0225(1).jpg
inflating: dataset_19/notumor/Tr-no_0233.jpg
inflating: dataset_19/notumor/Tr-no_0247.jpg
inflating: dataset_19/notumor/Tr-no_0256.jpg
inflating: dataset_19/notumor/Tr-no_0271.jpg
inflating: dataset_19/notumor/Tr-no_0272.jpg
inflating: dataset_19/notumor/Tr-no_0276.jpg
inflating: dataset_19/notumor/Tr-no_0289.jpg
inflating: dataset_19/notumor/Tr-no_0291.jpg
inflating: dataset_19/notumor/Tr-no_0295.jpg
inflating: dataset_19/notumor/Tr-no_0303.jpg
inflating: dataset_19/notumor/Tr-no_0306.jpg
inflating: dataset_19/notumor/Tr-no_0310.jpg
inflating: dataset_19/notumor/Tr-no_0328.jpg
```

```
inflating: dataset_19/notumor/Tr-no_0350.jpg
inflating: dataset_19/notumor/Tr-no_0351.jpg
inflating: dataset_19/notumor/Tr-no_0371.jpg
inflating: dataset_19/notumor/Tr-no_0373.jpg
inflating: dataset_19/notumor/Tr-no_0379.jpg
inflating: dataset_19/notumor/Tr-no_0386.jpg
inflating: dataset_19/notumor/Tr-no_0392.jpg
inflating: dataset_19/notumor/Tr-no_0393.jpg
inflating: dataset_19/notumor/Tr-no_0394.jpg
inflating: dataset_19/notumor/Tr-no_0398.jpg
inflating: dataset_19/notumor/Tr-no_0412.jpg
inflating: dataset_19/notumor/Tr-no_0421.jpg
inflating: dataset_19/notumor/Tr-no_0424.jpg
inflating: dataset_19/notumor/Tr-no_0425.jpg
inflating: dataset_19/notumor/Tr-no_0441.jpg
inflating: dataset_19/notumor/Tr-no_0444.jpg
inflating: dataset_19/notumor/Tr-no_0447.jpg
inflating: dataset_19/notumor/Tr-no_0457.jpg
inflating: dataset_19/notumor/Tr-no_0472.jpg
inflating: dataset_19/notumor/Tr-no_0478.jpg
inflating: dataset_19/notumor/Tr-no_0490.jpg
inflating: dataset_19/notumor/Tr-no_0492.jpg
inflating: dataset_19/notumor/Tr-no_0509.jpg
inflating: dataset_19/notumor/Tr-no_0510.jpg
 creating: dataset_19/pituitary/
inflating: dataset_19/pituitary/Te-pi_0015.jpg
inflating: dataset_19/pituitary/Te-pi_0017.jpg
inflating: dataset_19/pituitary/Te-pi_0023.jpg
inflating: dataset_19/pituitary/Te-pi_0036.jpg
inflating: dataset_19/pituitary/Te-pi_0047.jpg
inflating: dataset_19/pituitary/Te-pi_0050.jpg
inflating: dataset_19/pituitary/Te-pi_0052.jpg
inflating: dataset_19/pituitary/Te-pi_0062.jpg
inflating: dataset_19/pituitary/Te-pi_0064.jpg
inflating: dataset_19/pituitary/Te-pi_0088.jpg
inflating: dataset_19/pituitary/Te-pi_0091.jpg
inflating: dataset_19/pituitary/Te-pi_0093.jpg
inflating: dataset_19/pituitary/Te-pi_0103.jpg
inflating: dataset_19/pituitary/Te-pi_0108.jpg
inflating: dataset_19/pituitary/Te-pi_0116.jpg
inflating: dataset_19/pituitary/Te-pi_0123.jpg
inflating: dataset_19/pituitary/Te-pi_0132.jpg
inflating: dataset_19/pituitary/Te-pi_0144.jpg
inflating: dataset_19/pituitary/Te-pi_0146.jpg
inflating: dataset_19/pituitary/Te-pi_0148.jpg
inflating: dataset_19/pituitary/Te-pi_0153.jpg
inflating: dataset_19/pituitary/Te-pi_0160.jpg
inflating: dataset_19/pituitary/Te-pi_0162.jpg
```

```
inflating: dataset_19/pituitary/Te-pi_0171.jpg
inflating: dataset_19/pituitary/Te-pi_0174.jpg
inflating: dataset_19/pituitary/Te-pi_0182.jpg
inflating: dataset_19/pituitary/Te-pi_0202.jpg
inflating: dataset_19/pituitary/Te-pi_0204.jpg
inflating: dataset_19/pituitary/Te-pi_0206.jpg
inflating: dataset_19/pituitary/Tr-pi_0023.jpg
inflating: dataset_19/pituitary/Tr-pi_0035.jpg
inflating: dataset_19/pituitary/Tr-pi_0037.jpg
inflating: dataset_19/pituitary/Tr-pi_0039(3).jpg
inflating: dataset_19/pituitary/Tr-pi_0039.jpg
inflating: dataset_19/pituitary/Tr-pi_0040.jpg
inflating: dataset_19/pituitary/Tr-pi_0045.jpg
inflating: dataset_19/pituitary/Tr-pi_0051.jpg
inflating: dataset_19/pituitary/Tr-pi_0054.jpg
inflating: dataset_19/pituitary/Tr-pi_0067.jpg
inflating: dataset_19/pituitary/Tr-pi_0074.jpg
inflating: dataset_19/pituitary/Tr-pi_0086.jpg
inflating: dataset_19/pituitary/Tr-pi_0091.jpg
inflating: dataset_19/pituitary/Tr-pi_0098.jpg
inflating: dataset_19/pituitary/Tr-pi_0099.jpg
inflating: dataset_19/pituitary/Tr-pi_0105.jpg
inflating: dataset_19/pituitary/Tr-pi_0107.jpg
inflating: dataset_19/pituitary/Tr-pi_0110.jpg
inflating: dataset_19/pituitary/Tr-pi_0118.jpg
inflating: dataset_19/pituitary/Tr-pi_0119.jpg
inflating: dataset_19/pituitary/Tr-pi_0121.jpg
inflating: dataset_19/pituitary/Tr-pi_0122.jpg
inflating: dataset_19/pituitary/Tr-pi_0123.jpg
inflating: dataset_19/pituitary/Tr-pi_0126.jpg
inflating: dataset_19/pituitary/Tr-pi_0132.jpg
inflating: dataset_19/pituitary/Tr-pi_0145.jpg
inflating: dataset_19/pituitary/Tr-pi_0147.jpg
inflating: dataset_19/pituitary/Tr-pi_0149.jpg
inflating: dataset_19/pituitary/Tr-pi_0153.jpg
inflating: dataset_19/pituitary/Tr-pi_0156.jpg
inflating: dataset_19/pituitary/Tr-pi_0159(1).jpg
inflating: dataset_19/pituitary/Tr-pi_0159(2).jpg
inflating: dataset_19/pituitary/Tr-pi_0162.jpg
inflating: dataset_19/pituitary/Tr-pi_0165.jpg
inflating: dataset_19/pituitary/Tr-pi_0170.jpg
inflating: dataset_19/pituitary/Tr-pi_0171.jpg
inflating: dataset_19/pituitary/Tr-pi_0190.jpg
inflating: dataset_19/pituitary/Tr-pi_0194.jpg
inflating: dataset_19/pituitary/Tr-pi_0199.jpg
inflating: dataset_19/pituitary/Tr-pi_0204.jpg
inflating: dataset_19/pituitary/Tr-pi_0206.jpg
inflating: dataset_19/pituitary/Tr-pi_0213.jpg
```

```
inflating: dataset_19/pituitary/Tr-pi_0218.jpg
inflating: dataset_19/pituitary/Tr-pi_0224.jpg
inflating: dataset_19/pituitary/Tr-pi_0229.jpg
inflating: dataset_19/pituitary/Tr-pi_0238.jpg
inflating: dataset_19/pituitary/Tr-pi_0250.jpg
inflating: dataset_19/pituitary/Tr-pi_0260.jpg
inflating: dataset_19/pituitary/Tr-pi_0263.jpg
inflating: dataset_19/pituitary/Tr-pi_0274.jpg
inflating: dataset_19/pituitary/Tr-pi_0282.jpg
inflating: dataset_19/pituitary/Tr-pi_0298.jpg
inflating: dataset_19/pituitary/Tr-pi_0308.jpg
inflating: dataset_19/pituitary/Tr-pi_0311.jpg
inflating: dataset_19/pituitary/Tr-pi_0315.jpg
inflating: dataset_19/pituitary/Tr-pi_0326.jpg
inflating: dataset_19/pituitary/Tr-pi_0327.jpg
inflating: dataset_19/pituitary/Tr-pi_0330.jpg
inflating: dataset_19/pituitary/Tr-pi_0347.jpg
inflating: dataset_19/pituitary/Tr-pi_0356.jpg
inflating: dataset_19/pituitary/Tr-pi_0359.jpg
inflating: dataset_19/pituitary/Tr-pi_0362.jpg
inflating: dataset_19/pituitary/Tr-pi_0367.jpg
inflating: dataset_19/pituitary/Tr-pi_0374.jpg
inflating: dataset_19/pituitary/Tr-pi_0375.jpg
inflating: dataset_19/pituitary/Tr-pi_0388.jpg
inflating: dataset_19/pituitary/Tr-pi_0394.jpg
inflating: dataset_19/pituitary/Tr-pi_0396.jpg
inflating: dataset_19/pituitary/Tr-pi_0399.jpg
inflating: dataset_19/pituitary/Tr-pi_0400.jpg
inflating: dataset_19/pituitary/Tr-pi_0405.jpg
inflating: dataset_19/pituitary/Tr-pi_0420.jpg
inflating: dataset_19/pituitary/Tr-pi_0430.jpg
inflating: dataset_19/pituitary/Tr-pi_0450.jpg
inflating: dataset_19/pituitary/Tr-pi_0451.jpg
inflating: dataset_19/pituitary/Tr-pi_0452.jpg
inflating: dataset_19/pituitary/Tr-pi_0455.jpg
inflating: dataset_19/pituitary/Tr-pi_0462.jpg
inflating: dataset_19/pituitary/Tr-pi_0464.jpg
inflating: dataset_19/pituitary/Tr-pi_0469.jpg
inflating: dataset_19/pituitary/Tr-pi_0472.jpg
inflating: dataset_19/pituitary/Tr-pi_0479.jpg
inflating: dataset_19/pituitary/Tr-pi_0485.jpg
inflating: dataset_19/pituitary/Tr-pi_0486.jpg
inflating: dataset_19/pituitary/Tr-pi_0487.jpg
inflating: dataset_19/pituitary/Tr-pi_0489.jpg
inflating: dataset_19/pituitary/Tr-pi_0494.jpg
inflating: dataset_19/pituitary/Tr-pi_0496.jpg
inflating: dataset_19/pituitary/Tr-pi_0503.jpg
inflating: dataset_19/pituitary/Tr-pi_0507.jpg
```

```
  inflating: dataset_19/pituitary/Tr-pi_0509.jpg
/content
```

1 Load the data

[4]:
```
# Data Directories
dir = "dataset_19/"
```

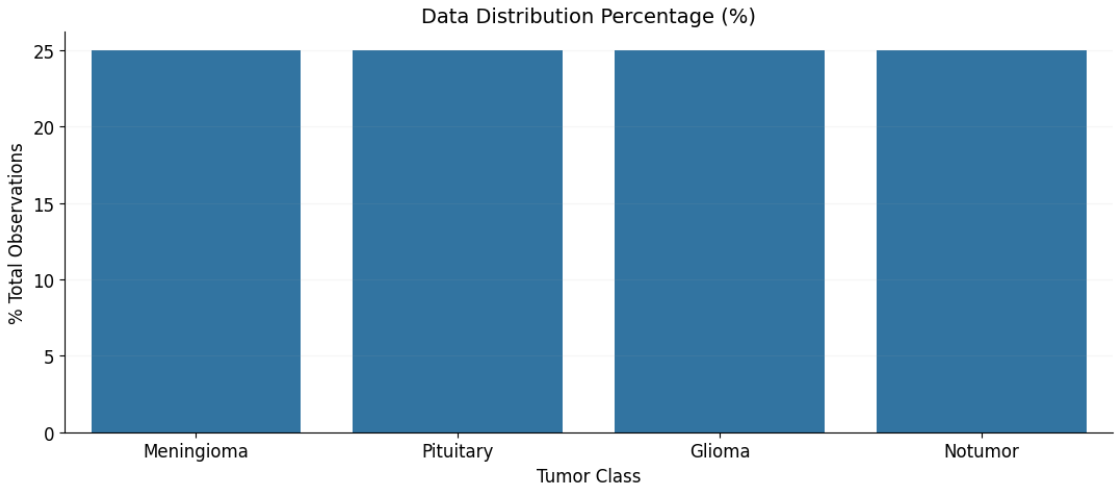
1.1 Data Distribution

[ ]:
```
data_distribution_count = pd.Series(
    {curr_index: len(os.listdir(os.path.join(dir, curr_index))) for curr_index␣
 ↪in os.listdir(dir)})

data_distribution_count
```

[ ]:
```
meningioma    120
pituitary     120
glioma        120
notumor       120
dtype: int64
```

[ ]:
```
fig, axis = plt.subplots(figsize=(13, 5))
axis.grid(True, alpha=0.1)
axis.set_title("Data Distribution Percentage (%)", fontsize=14)
sns.barplot(x=['\n'.join(curr_index.strip().split('_')).title() for curr_index␣
 ↪in data_distribution_count.index],
            y=100 * data_distribution_count / data_distribution_count.sum(),␣
 ↪ax=axis)
axis.set_xlabel("Tumor Class", fontsize=12)
axis.set_ylabel("% Total Observations", fontsize=12)
axis.tick_params(which='major', labelsize=12)
axis.text(2.5, 37, f'Total Observations: {data_distribution_count.sum()}',␣
 ↪fontdict=dict(size=12))
sns.despine()
```

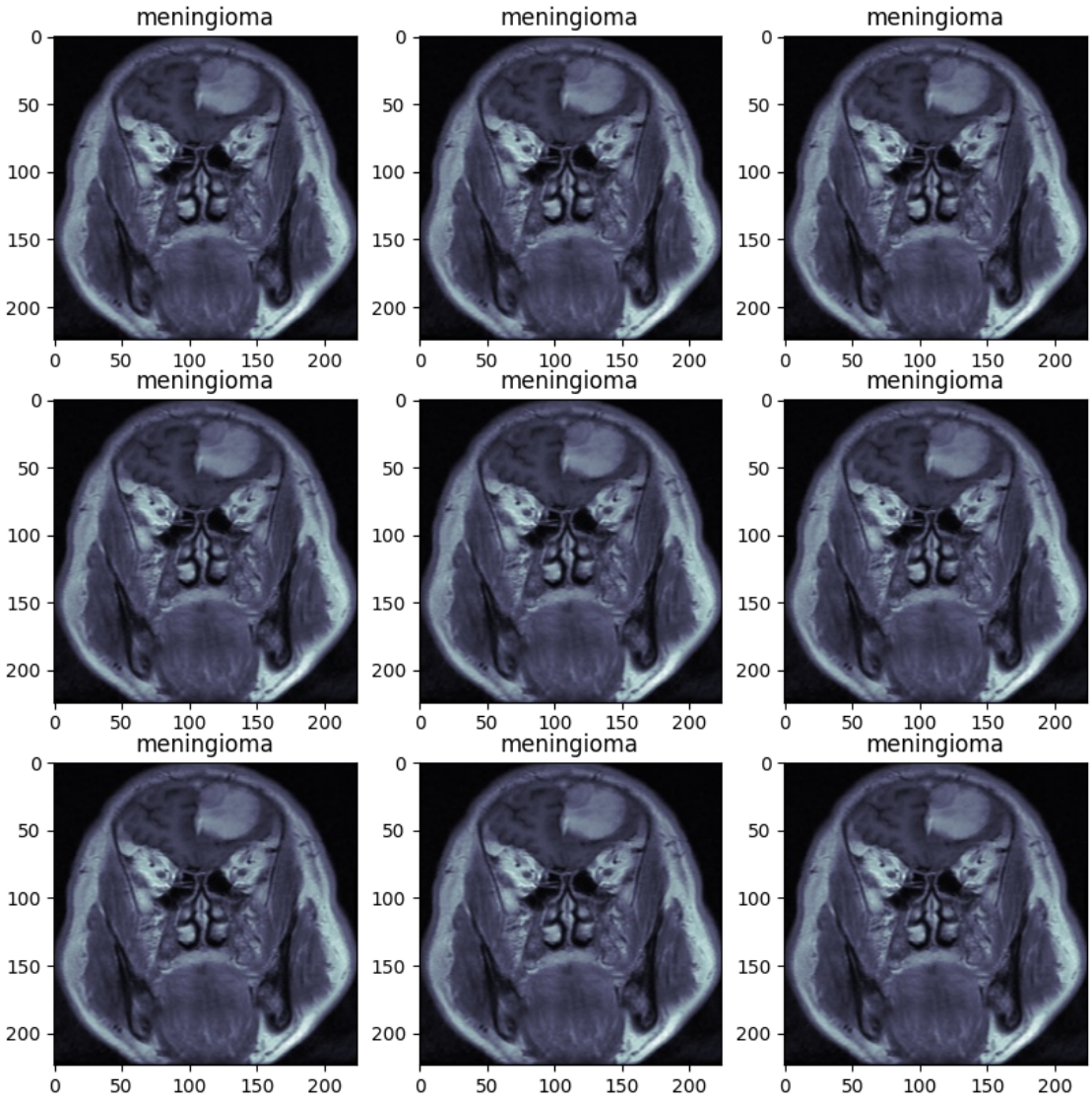
Total Observations: 480



2 Preprocess Data

[5]:
```
from tqdm import tqdm
import cv2
import imutils

def crop_img(img):

  # Find extreme points on the image and crop the rectangular out

  gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
  gray = cv2.GaussianBlur(gray, (3, 3), 0)

  # threshold the image, then perform a series of erosions +
  # dilations to remove any small regions of noise
  thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
  thresh = cv2.erode(thresh, None, iterations=2)
  thresh = cv2.dilate(thresh, None, iterations=2)

  # find contours in thresholded image, then grab the largest one
  cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.
↪CHAIN_APPROX_SIMPLE)
  cnts = imutils.grab_contours(cnts)
  c = max(cnts, key=cv2.contourArea)
```

```
  # find the extreme points
  extLeft = tuple(c[c[:, :, 0].argmin()][0])
  extRight = tuple(c[c[:, :, 0].argmax()][0])
  extTop = tuple(c[c[:, :, 1].argmin()][0])
  extBot = tuple(c[c[:, :, 1].argmax()][0])
  ADD_PIXELS = 0
  new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS,␣
 ↪extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()

  return new_img

def preprocess_images(directory):
    for dir in os.listdir(directory):
        path = os.path.join(directory, dir)
        for img_name in os.listdir(path):
            img_path = os.path.join(path, img_name)
            img = cv2.imread(img_path)
            cropped_img = crop_img(img)
            processed_img = cv2.cvtColor(cropped_img, cv2.COLOR_RGB2GRAY)
            processed_img = cv2.bilateralFilter(processed_img, 2, 50, 50)
            processed_img = cv2.applyColorMap(processed_img, cv2.COLORMAP_BONE)
            processed_img = cv2.resize(processed_img, (224, 224))
            cv2.imwrite(img_path, processed_img)


# Preprocess the images before generating data
preprocess_images(dir)

# Display 9 image using matplotlib
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    for curr_index in os.listdir(dir):
        path = os.path.join(dir, curr_index)
        for img_name in os.listdir(path):
            img_path = os.path.join(path, img_name)
            image = plt.imread(img_path)
            plt.imshow(image)
            plt.title(curr_index)
            break
        break
```



2.1 Splitting the data

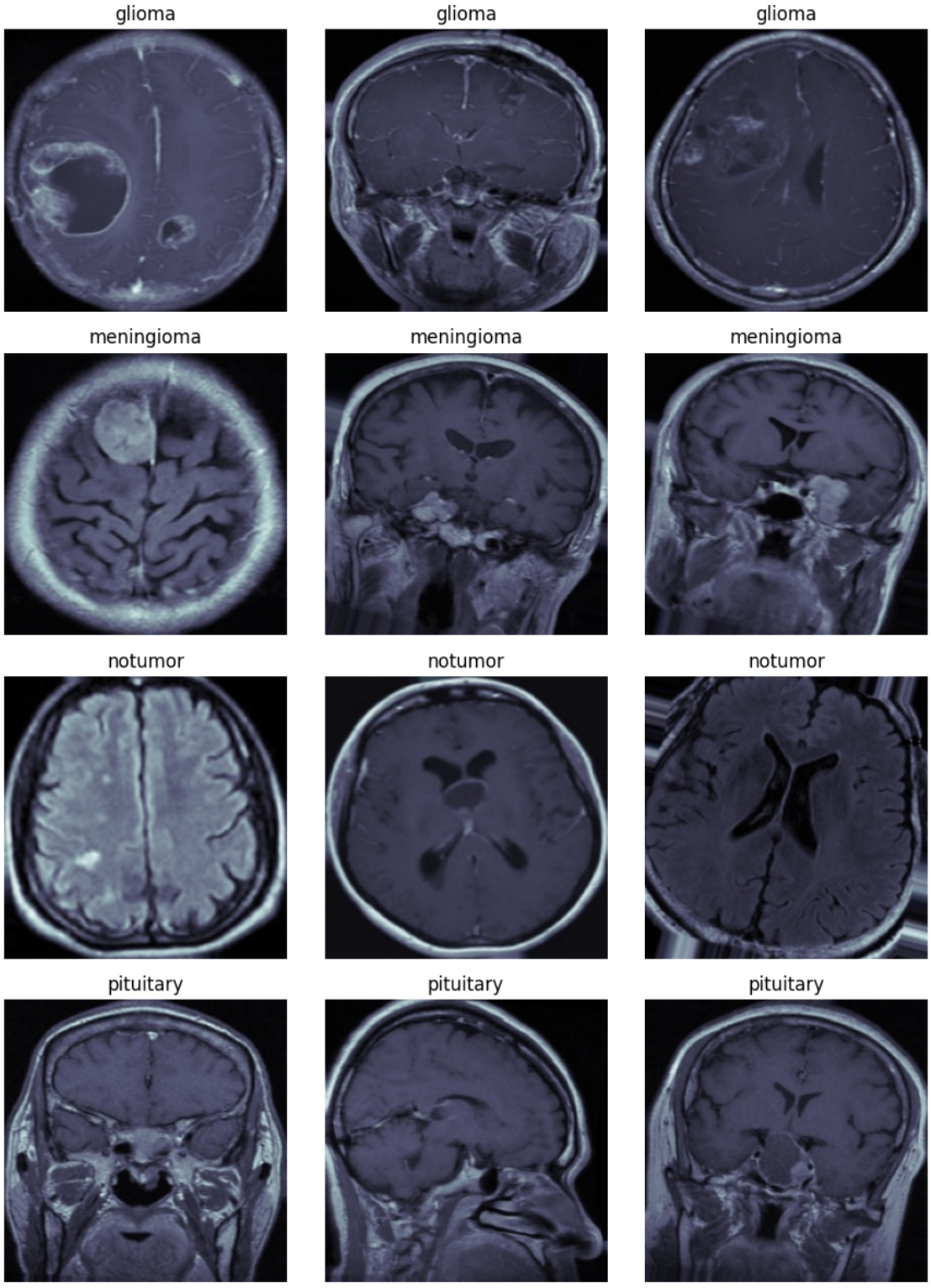
[6]:
```
classes = os.listdir(dir)

batch_size = 10

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    horizontal_flip=True,
    rotation_range=20,
    validation_split=0.2)
```

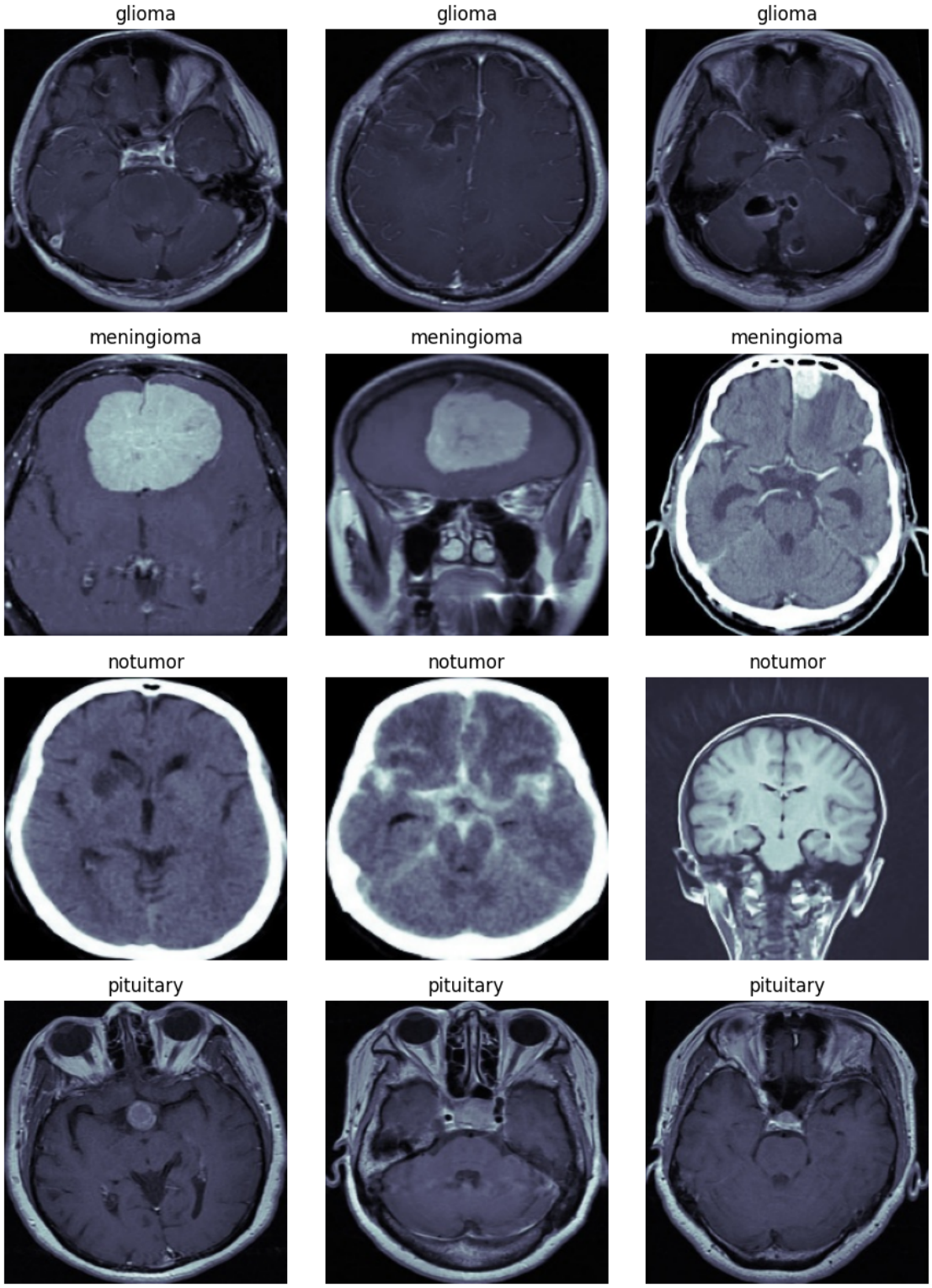
```
validation_datagen = ImageDataGenerator(rescale=1. / 255,
                                        validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    dir,
    target_size=(224, 224),
    batch_size=batch_size,
    seed=42,
    subset='training'
)

test_generator = validation_datagen.flow_from_directory(
    dir,
    target_size=(224, 224),
    batch_size=batch_size,
    seed=42,
    shuffle = False,
    subset='validation')


print(test_generator.class_indices)
```

```
Found 384 images belonging to 4 classes.
Found 96 images belonging to 4 classes.
{'glioma': 0, 'meningioma': 1, 'notumor': 2, 'pituitary': 3}
```

[10]:
```
import matplotlib.pyplot as plt
import numpy as np

def plot_images_from_generator(generator, num_images_per_class=3):
    # Get the class labels
    class_indices = generator.class_indices
    class_names = list(class_indices.keys())
    num_classes = len(class_names)

    # Initialize a dictionary to keep track of the images plotted for each class
    images_per_class = {class_name: 0 for class_name in class_names}

    # Prepare a figure for plotting
    fig, axes = plt.subplots(num_classes, num_images_per_class,␣
 ↪figsize=(num_images_per_class*3, num_classes*3))

    # Ensure the axes array is always 2D
    if num_classes == 1:
        axes = np.expand_dims(axes, axis=0)

    if num_images_per_class == 1:
```

```
        axes = np.expand_dims(axes, axis=1)

    # Loop through batches of images from the generator
    for batch_x, batch_y in generator:
        for i in range(len(batch_x)):
            # Get the image and its corresponding label
            img = batch_x[i]
            label_index = np.argmax(batch_y[i])
            label_name = class_names[label_index]

            # Check if we have already plotted enough images for this class
            if images_per_class[label_name] < num_images_per_class:
                row = list(class_indices.keys()).index(label_name)
                col = images_per_class[label_name]
                axes[row, col].imshow(img)
                axes[row, col].axis('off')
                axes[row, col].set_title(label_name)

                # Update the count of images plotted for this class
                images_per_class[label_name] += 1

            # Check if we have enough images for all classes
            if all(count >= num_images_per_class for count in images_per_class.
 ↪values()):
                break
        else:
            continue
        break

    plt.tight_layout()
    plt.show()

# Visualize images from the training generator
plot_images_from_generator(train_generator, num_images_per_class=3)

# Visualize images from the validation generator
plot_images_from_generator(test_generator, num_images_per_class=3)
```


glioma
glioma
glioma
meningioma
meningioma
meningioma
notumor
notumor
notumor
pituitary
pituitary
pituitary


glioma
glioma
glioma
meningioma
meningioma
meningioma
notumor
notumor
notumor
pituitary
pituitary
pituitary

3 Model Training

[ ]:
```
!pip install segmentation-models
```

```
Collecting segmentation-models
  Downloading segmentation_models-1.0.1-py3-none-any.whl (33 kB)
Collecting keras-applications<=1.0.8,>=1.0.7 (from segmentation-models)
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
                              50.7/50.7 kB
6.9 MB/s eta 0:00:00
Collecting image-classifiers==1.0.0 (from segmentation-models)
  Downloading image_classifiers-1.0.0-py3-none-any.whl (19 kB)
Collecting efficientnet==1.0.0 (from segmentation-models)
  Downloading efficientnet-1.0.0-py3-none-any.whl (17 kB)
Requirement already satisfied: scikit-image in /usr/local/lib/python3.10/dist-
packages (from efficientnet==1.0.0->segmentation-models) (0.19.3)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.10/dist-
packages (from keras-applications<=1.0.8,>=1.0.7->segmentation-models) (1.25.2)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages
(from keras-applications<=1.0.8,>=1.0.7->segmentation-models) (3.9.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->efficientnet==1.0.0->segmentation-models) (1.11.4)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->efficientnet==1.0.0->segmentation-models) (3.3)
Requirement already satisfied: pillow!=7.1.0,!=7.1.1,!=8.3.0,>=6.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
image->efficientnet==1.0.0->segmentation-models) (9.4.0)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-image->efficientnet==1.0.0->segmentation-models) (2.31.6)
Requirement already satisfied: tifffile>=2019.7.26 in
/usr/local/lib/python3.10/dist-packages (from scikit-
image->efficientnet==1.0.0->segmentation-models) (2024.5.22)
Requirement already satisfied: PyWavelets>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-
image->efficientnet==1.0.0->segmentation-models) (1.6.0)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
image->efficientnet==1.0.0->segmentation-models) (24.0)
Installing collected packages: keras-applications, image-classifiers,
efficientnet, segmentation-models
Successfully installed efficientnet-1.0.0 image-classifiers-1.0.0 keras-
applications-1.0.8 segmentation-models-1.0.1
```

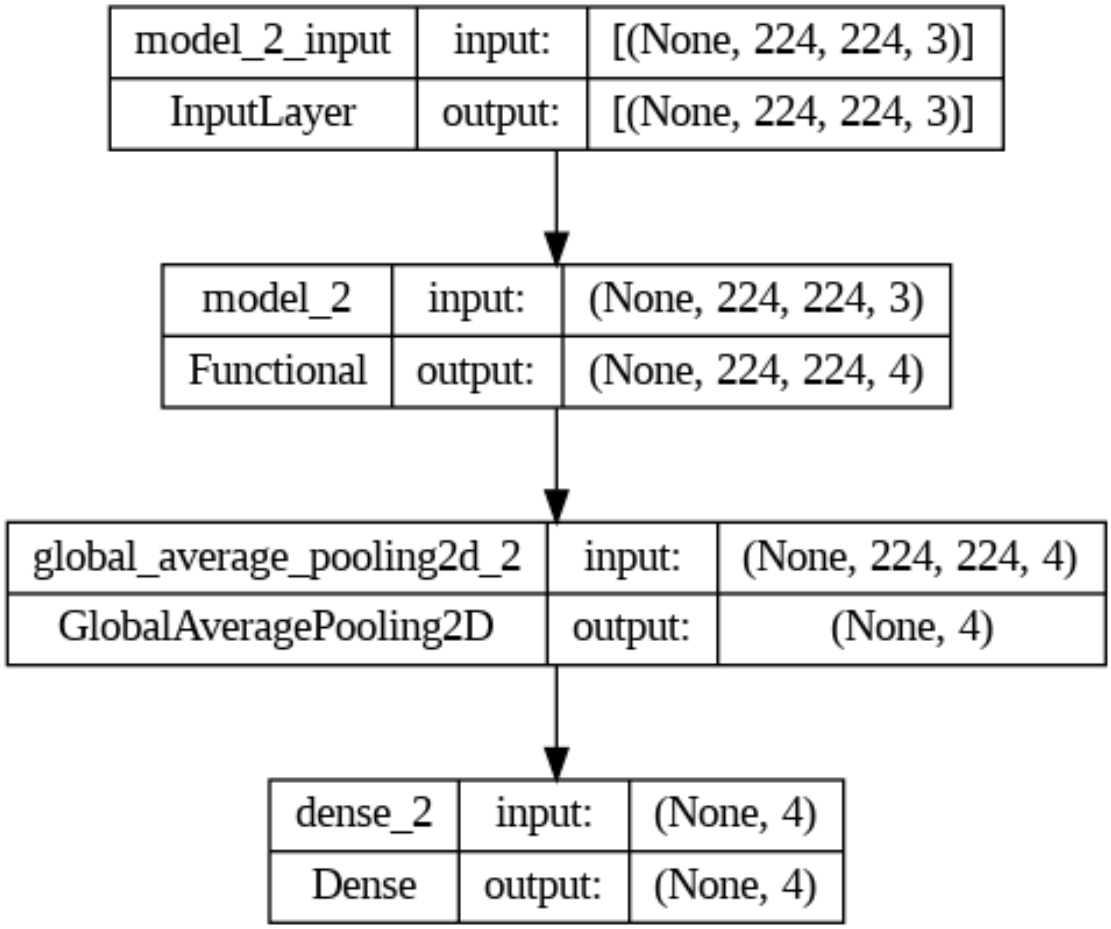
[ ]:
```
import os
os.environ["SM_FRAMEWORK"] = "tf.keras"

from tensorflow import keras
from segmentation_models import Unet
```

```
from tensorflow.keras.regularizers import l2

# Model parameters
input_shape = (224, 224, 3)
num_classes = 4
initial_learning_rate = 1e-4

# Define the U-Net++ model
model = Unet('efficientnetb1', input_shape=input_shape, classes=num_classes,␣
 ↪activation=None)
tf.keras.utils.plot_model(model, to_file='unetpp_model.png', show_shapes=True,␣
 ↪show_layer_names=True)

# Add a GlobalAveragePooling2D layer to collapse spatial dimensions
model = keras.Sequential([
    model,
    keras.layers.GlobalAveragePooling2D()  # Pool across spatial dimensions
])

# Add a Dense layer with softmax activation for final classification
model.add(keras.layers.Dense(num_classes, activation='softmax'))

# Compile the model
optimizer = keras.optimizers.Adam(learning_rate=initial_learning_rate)
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Summary of the model
model.summary()
tf.keras.utils.plot_model(model, to_file='unetpp_model2.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.794525 to fit

Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 model_2 (Functional)        (None, 224, 224, 4)       12641604

 global_average_pooling2d_2  (None, 4)                 0
  (GlobalAveragePooling2D)

 dense_2 (Dense)             (None, 4)                 20

=================================================================
```

```
Total params: 12641624 (48.22 MB)
Trainable params: 12577592 (47.98 MB)
Non-trainable params: 64032 (250.12 KB)
_________________________________________________________________
```

[ ]:



[ ]:
```
from keras.callbacks import *

# Callbacks
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=10, factor=0.3,␣
 ↪min_lr=1e-6, verbose=1)
checkpoint = ModelCheckpoint(model_file, monitor='val_loss', verbose=1,␣
 ↪save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=20,␣
 ↪verbose=1, mode='auto')

# Train the model
history = model.fit(
    train_generator,
```

```
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=100,
    validation_data=test_generator,
    validation_steps=test_generator.samples // test_generator.batch_size,
    callbacks=[checkpoint, early_stopping, reduce_lr]
)
```

```
Epoch 1/100
38/38 [==============================] - ETA: 0s - loss: 1.3388 - accuracy:
0.3182
Epoch 1: val_loss improved from inf to 1.22903, saving model to
./models/unetpp.h5
38/38 [==============================] - 55s 193ms/step - loss: 1.3388 -
accuracy: 0.3182 - val_loss: 1.2290 - val_accuracy: 0.3222 - lr: 1.0000e-04
Epoch 2/100
38/38 [==============================] - ETA: 0s - loss: 1.0598 - accuracy:
0.6257
Epoch 2: val_loss did not improve from 1.22903
38/38 [==============================] - 5s 117ms/step - loss: 1.0598 -
accuracy: 0.6257 - val_loss: 1.5023 - val_accuracy: 0.4222 - lr: 1.0000e-04
Epoch 3/100
38/38 [==============================] - ETA: 0s - loss: 0.8673 - accuracy:
0.7166
Epoch 3: val_loss improved from 1.22903 to 0.85093, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 145ms/step - loss: 0.8673 -
accuracy: 0.7166 - val_loss: 0.8509 - val_accuracy: 0.6778 - lr: 1.0000e-04
Epoch 4/100
38/38 [==============================] - ETA: 0s - loss: 0.7575 - accuracy:
0.7995
Epoch 4: val_loss improved from 0.85093 to 0.74367, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 147ms/step - loss: 0.7575 -
accuracy: 0.7995 - val_loss: 0.7437 - val_accuracy: 0.6889 - lr: 1.0000e-04
Epoch 5/100
38/38 [==============================] - ETA: 0s - loss: 0.7159 - accuracy:
0.8182
Epoch 5: val_loss did not improve from 0.74367
38/38 [==============================] - 4s 112ms/step - loss: 0.7159 -
accuracy: 0.8182 - val_loss: 0.7468 - val_accuracy: 0.7444 - lr: 1.0000e-04
Epoch 6/100
38/38 [==============================] - ETA: 0s - loss: 0.5693 - accuracy:
0.8717
Epoch 6: val_loss improved from 0.74367 to 0.64111, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 144ms/step - loss: 0.5693 -
accuracy: 0.8717 - val_loss: 0.6411 - val_accuracy: 0.7444 - lr: 1.0000e-04
Epoch 7/100
```

```
38/38 [==============================] - ETA: 0s - loss: 0.5355 - accuracy:
0.8743
Epoch 7: val_loss improved from 0.64111 to 0.60537, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 146ms/step - loss: 0.5355 -
accuracy: 0.8743 - val_loss: 0.6054 - val_accuracy: 0.7778 - lr: 1.0000e-04
Epoch 8/100
38/38 [==============================] - ETA: 0s - loss: 0.4354 - accuracy:
0.9225
Epoch 8: val_loss improved from 0.60537 to 0.53137, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 145ms/step - loss: 0.4354 -
accuracy: 0.9225 - val_loss: 0.5314 - val_accuracy: 0.7667 - lr: 1.0000e-04
Epoch 9/100
38/38 [==============================] - ETA: 0s - loss: 0.4749 - accuracy:
0.8797
Epoch 9: val_loss improved from 0.53137 to 0.50809, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 145ms/step - loss: 0.4749 -
accuracy: 0.8797 - val_loss: 0.5081 - val_accuracy: 0.8000 - lr: 1.0000e-04
Epoch 10/100
38/38 [==============================] - ETA: 0s - loss: 0.4342 - accuracy:
0.9091
Epoch 10: val_loss improved from 0.50809 to 0.48974, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 143ms/step - loss: 0.4342 -
accuracy: 0.9091 - val_loss: 0.4897 - val_accuracy: 0.8111 - lr: 1.0000e-04
Epoch 11/100
38/38 [==============================] - ETA: 0s - loss: 0.3889 - accuracy:
0.9064
Epoch 11: val_loss improved from 0.48974 to 0.39200, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 143ms/step - loss: 0.3889 -
accuracy: 0.9064 - val_loss: 0.3920 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 12/100
38/38 [==============================] - ETA: 0s - loss: 0.4113 - accuracy:
0.8877
Epoch 12: val_loss did not improve from 0.39200
38/38 [==============================] - 4s 113ms/step - loss: 0.4113 -
accuracy: 0.8877 - val_loss: 0.4505 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 13/100
38/38 [==============================] - ETA: 0s - loss: 0.3902 - accuracy:
0.9091
Epoch 13: val_loss did not improve from 0.39200
38/38 [==============================] - 4s 118ms/step - loss: 0.3902 -
accuracy: 0.9091 - val_loss: 0.4038 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 14/100
38/38 [==============================] - ETA: 0s - loss: 0.3354 - accuracy:
```

```
0.9011
Epoch 14: val_loss improved from 0.39200 to 0.35025, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 145ms/step - loss: 0.3354 -
accuracy: 0.9011 - val_loss: 0.3502 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 15/100
38/38 [==============================] - ETA: 0s - loss: 0.3424 - accuracy:
0.9278
Epoch 15: val_loss did not improve from 0.35025
38/38 [==============================] - 4s 115ms/step - loss: 0.3424 -
accuracy: 0.9278 - val_loss: 0.3584 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 16/100
38/38 [==============================] - ETA: 0s - loss: 0.2569 - accuracy:
0.9465
Epoch 16: val_loss did not improve from 0.35025
38/38 [==============================] - 4s 113ms/step - loss: 0.2569 -
accuracy: 0.9465 - val_loss: 0.3984 - val_accuracy: 0.8889 - lr: 1.0000e-04
Epoch 17/100
38/38 [==============================] - ETA: 0s - loss: 0.2859 - accuracy:
0.9251
Epoch 17: val_loss did not improve from 0.35025
38/38 [==============================] - 4s 114ms/step - loss: 0.2859 -
accuracy: 0.9251 - val_loss: 0.3738 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 18/100
38/38 [==============================] - ETA: 0s - loss: 0.2323 - accuracy:
0.9572
Epoch 18: val_loss improved from 0.35025 to 0.29634, saving model to
./models/unetpp.h5
38/38 [==============================] - 5s 143ms/step - loss: 0.2323 -
accuracy: 0.9572 - val_loss: 0.2963 - val_accuracy: 0.9000 - lr: 1.0000e-04
Epoch 19/100
38/38 [==============================] - ETA: 0s - loss: 0.2620 - accuracy:
0.9545
Epoch 19: val_loss improved from 0.29634 to 0.27448, saving model to
./models/unetpp.h5
38/38 [==============================] - 6s 146ms/step - loss: 0.2620 -
accuracy: 0.9545 - val_loss: 0.2745 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 20/100
38/38 [==============================] - ETA: 0s - loss: 0.2566 - accuracy:
0.9385
Epoch 20: val_loss did not improve from 0.27448
38/38 [==============================] - 5s 117ms/step - loss: 0.2566 -
accuracy: 0.9385 - val_loss: 0.3840 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 21/100
38/38 [==============================] - ETA: 0s - loss: 0.2519 - accuracy:
0.9465
Epoch 21: val_loss improved from 0.27448 to 0.24982, saving model to
./models/unetpp.h5
```

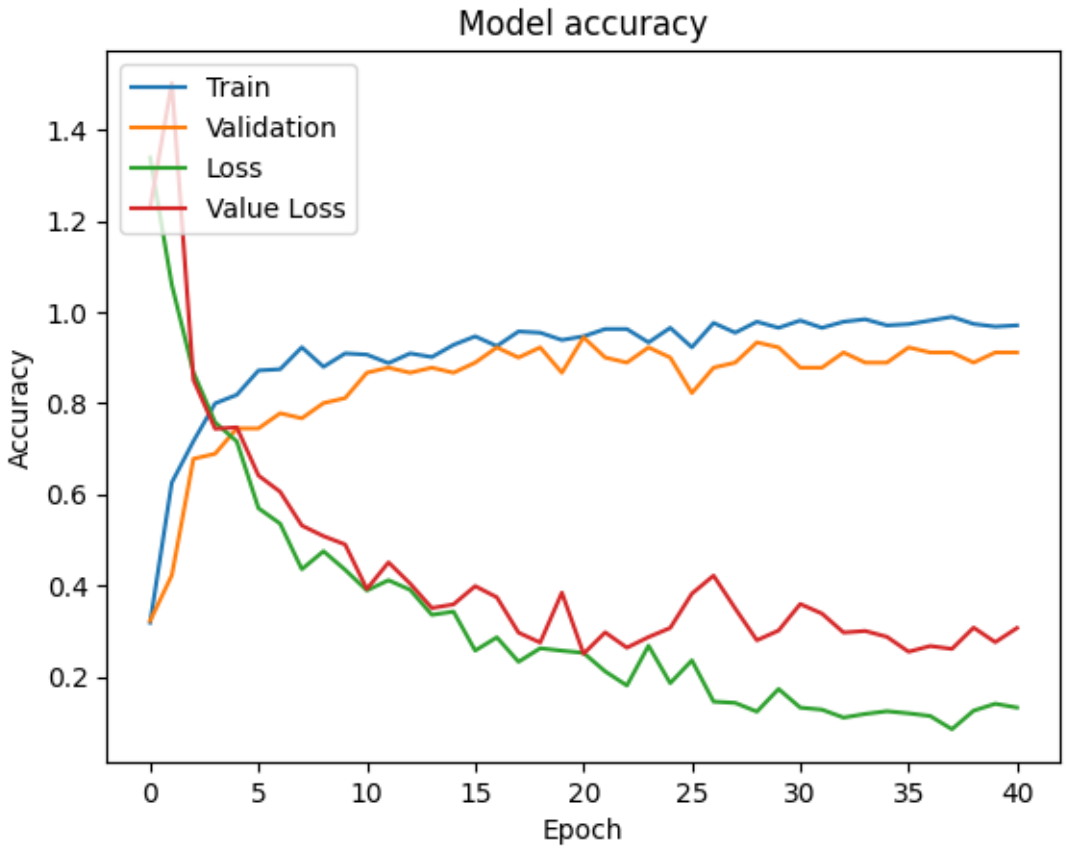
```
38/38 [==============================] - 6s 161ms/step - loss: 0.2519 -
accuracy: 0.9465 - val_loss: 0.2498 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 22/100
38/38 [==============================] - ETA: 0s - loss: 0.2112 - accuracy:
0.9626
Epoch 22: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 112ms/step - loss: 0.2112 -
accuracy: 0.9626 - val_loss: 0.2969 - val_accuracy: 0.9000 - lr: 1.0000e-04
Epoch 23/100
38/38 [==============================] - ETA: 0s - loss: 0.1802 - accuracy:
0.9626
Epoch 23: val_loss did not improve from 0.24982
38/38 [==============================] - 5s 119ms/step - loss: 0.1802 -
accuracy: 0.9626 - val_loss: 0.2632 - val_accuracy: 0.8889 - lr: 1.0000e-04
Epoch 24/100
38/38 [==============================] - ETA: 0s - loss: 0.2676 - accuracy:
0.9332
Epoch 24: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 116ms/step - loss: 0.2676 -
accuracy: 0.9332 - val_loss: 0.2866 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 25/100
38/38 [==============================] - ETA: 0s - loss: 0.1854 - accuracy:
0.9652
Epoch 25: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 114ms/step - loss: 0.1854 -
accuracy: 0.9652 - val_loss: 0.3066 - val_accuracy: 0.9000 - lr: 1.0000e-04
Epoch 26/100
38/38 [==============================] - ETA: 0s - loss: 0.2354 - accuracy:
0.9225
Epoch 26: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 116ms/step - loss: 0.2354 -
accuracy: 0.9225 - val_loss: 0.3818 - val_accuracy: 0.8222 - lr: 1.0000e-04
Epoch 27/100
38/38 [==============================] - ETA: 0s - loss: 0.1449 - accuracy:
0.9759
Epoch 27: val_loss did not improve from 0.24982
38/38 [==============================] - 5s 116ms/step - loss: 0.1449 -
accuracy: 0.9759 - val_loss: 0.4214 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 28/100
38/38 [==============================] - ETA: 0s - loss: 0.1421 - accuracy:
0.9545
Epoch 28: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 113ms/step - loss: 0.1421 -
accuracy: 0.9545 - val_loss: 0.3492 - val_accuracy: 0.8889 - lr: 1.0000e-04
Epoch 29/100
38/38 [==============================] - ETA: 0s - loss: 0.1229 - accuracy:
0.9786
Epoch 29: val_loss did not improve from 0.24982
```

```
38/38 [==============================] - 5s 117ms/step - loss: 0.1229 -
accuracy: 0.9786 - val_loss: 0.2795 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 30/100
38/38 [==============================] - ETA: 0s - loss: 0.1726 - accuracy:
0.9652
Epoch 30: val_loss did not improve from 0.24982
38/38 [==============================] - 5s 117ms/step - loss: 0.1726 -
accuracy: 0.9652 - val_loss: 0.3009 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 31/100
38/38 [==============================] - ETA: 0s - loss: 0.1317 - accuracy:
0.9813
Epoch 31: val_loss did not improve from 0.24982

Epoch 31: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
38/38 [==============================] - 4s 114ms/step - loss: 0.1317 -
accuracy: 0.9813 - val_loss: 0.3591 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 32/100
38/38 [==============================] - ETA: 0s - loss: 0.1273 - accuracy:
0.9652
Epoch 32: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 116ms/step - loss: 0.1273 -
accuracy: 0.9652 - val_loss: 0.3382 - val_accuracy: 0.8778 - lr: 3.0000e-05
Epoch 33/100
38/38 [==============================] - ETA: 0s - loss: 0.1099 - accuracy:
0.9786
Epoch 33: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 116ms/step - loss: 0.1099 -
accuracy: 0.9786 - val_loss: 0.2966 - val_accuracy: 0.9111 - lr: 3.0000e-05
Epoch 34/100
38/38 [==============================] - ETA: 0s - loss: 0.1179 - accuracy:
0.9840
Epoch 34: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 114ms/step - loss: 0.1179 -
accuracy: 0.9840 - val_loss: 0.2999 - val_accuracy: 0.8889 - lr: 3.0000e-05
Epoch 35/100
38/38 [==============================] - ETA: 0s - loss: 0.1238 - accuracy:
0.9706
Epoch 35: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 114ms/step - loss: 0.1238 -
accuracy: 0.9706 - val_loss: 0.2871 - val_accuracy: 0.8889 - lr: 3.0000e-05
Epoch 36/100
38/38 [==============================] - ETA: 0s - loss: 0.1192 - accuracy:
0.9733
Epoch 36: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 114ms/step - loss: 0.1192 -
accuracy: 0.9733 - val_loss: 0.2544 - val_accuracy: 0.9222 - lr: 3.0000e-05
Epoch 37/100
38/38 [==============================] - ETA: 0s - loss: 0.1133 - accuracy:
```

```
0.9813
Epoch 37: val_loss did not improve from 0.24982
38/38 [==============================] - 5s 116ms/step - loss: 0.1133 -
accuracy: 0.9813 - val_loss: 0.2669 - val_accuracy: 0.9111 - lr: 3.0000e-05
Epoch 38/100
38/38 [==============================] - ETA: 0s - loss: 0.0843 - accuracy:
0.9893
Epoch 38: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 114ms/step - loss: 0.0843 -
accuracy: 0.9893 - val_loss: 0.2606 - val_accuracy: 0.9111 - lr: 3.0000e-05
Epoch 39/100
38/38 [==============================] - ETA: 0s - loss: 0.1250 - accuracy:
0.9737
Epoch 39: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 116ms/step - loss: 0.1250 -
accuracy: 0.9737 - val_loss: 0.3073 - val_accuracy: 0.8889 - lr: 3.0000e-05
Epoch 40/100
38/38 [==============================] - ETA: 0s - loss: 0.1402 - accuracy:
0.9679
Epoch 40: val_loss did not improve from 0.24982
38/38 [==============================] - 4s 112ms/step - loss: 0.1402 -
accuracy: 0.9679 - val_loss: 0.2754 - val_accuracy: 0.9111 - lr: 3.0000e-05
Epoch 41/100
38/38 [==============================] - ETA: 0s - loss: 0.1318 - accuracy:
0.9706
Epoch 41: val_loss did not improve from 0.24982

Epoch 41: ReduceLROnPlateau reducing learning rate to 8.999999772640877e-06.
38/38 [==============================] - 4s 115ms/step - loss: 0.1318 -
accuracy: 0.9706 - val_loss: 0.3064 - val_accuracy: 0.9111 - lr: 3.0000e-05
Epoch 41: early stopping
```

4 Testing the Model

[ ]:
```
# Learning curve
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

# Loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation','Loss', 'Value Loss'], loc='upper left')
plt.show()
```



[ ]:
```
# Validate the model with test data

model = keras.models.load_model(model_file)
model.evaluate(test_generator)
```

```
# Predict and Display image using matplotlib
# plt.figure(figsize=(20, 20))
# for i in range(9):
#     plt.subplot(3, 3, i + 1)
#     for X_batch, Y_batch in test_generator:
#         image = X_batch[0]
#         # Print Class
#         plt.title("Predicted: " + classes[predictions[i]] + "\nActual: " +␣
 ↪classes[np.argmax(Y_batch[i])])

#         plt.imshow(image)
#         break
```

```
10/10 [==============================] - 2s 23ms/step - loss: 0.2657 - accuracy:
0.9375
```

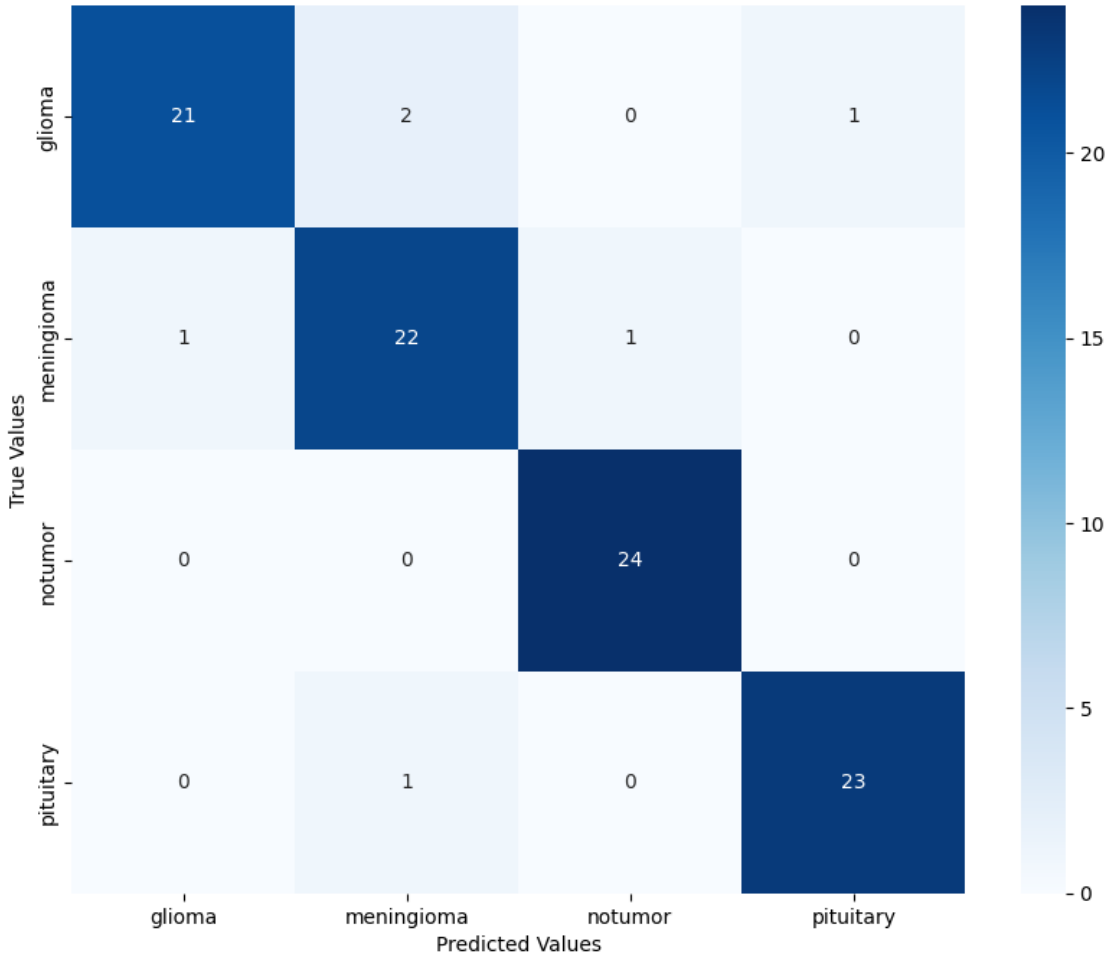
[ ]:
```
[0.26569023728370667, 0.9375]
```

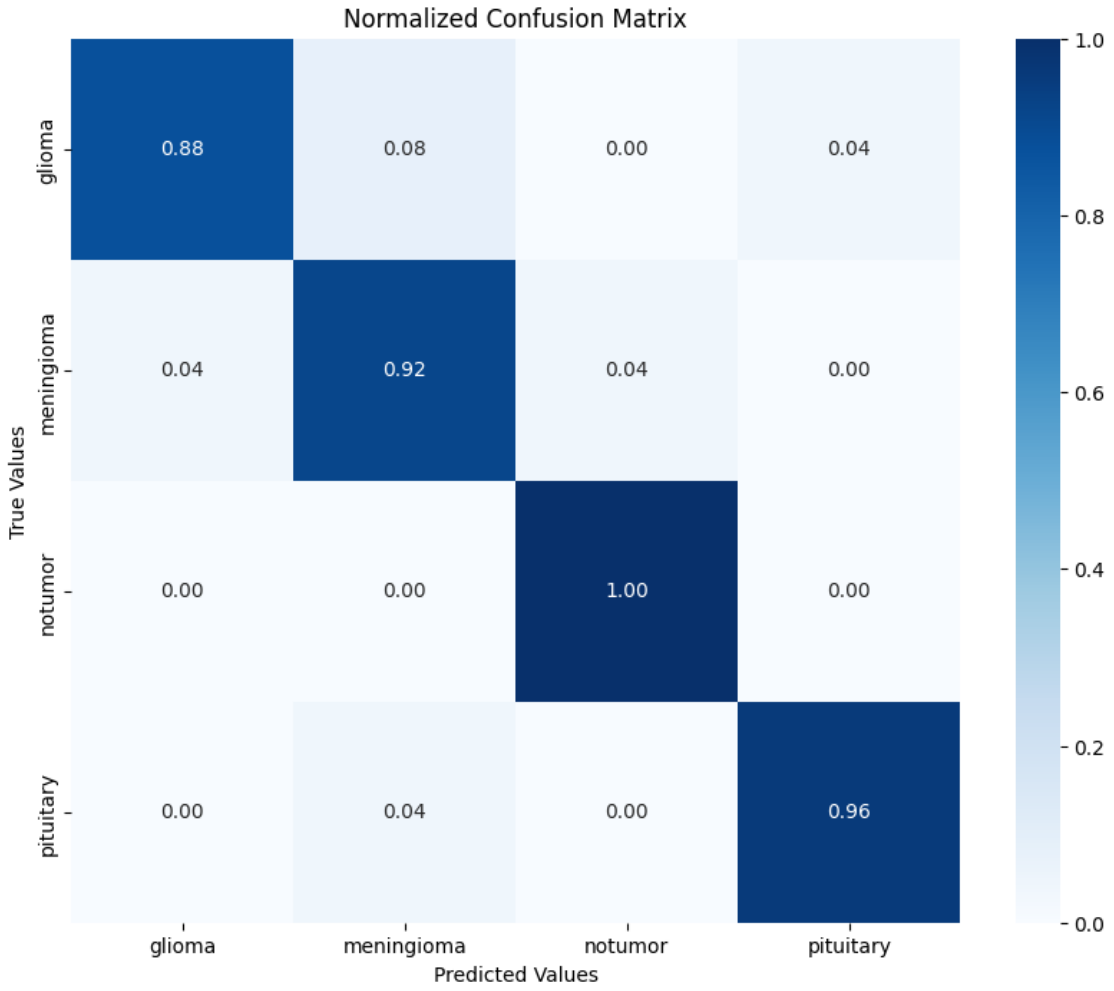
5 Model Visualisation (Evaluation)

[ ]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score

classes = ['glioma', 'meningioma', 'notumor', 'pituitary']

def calculate_metrics(y_true, y_pred):
    # Confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    # Plot the Confusion Matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=classes,␣
 ↪yticklabels=classes)
    plt.xlabel('Predicted Values')
    plt.ylabel('True Values')
    plt.show()

    # Normalize the confusion matrix
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    # Plot the normalized confusion matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm_normalized, annot=True, fmt='.2f', cmap='Blues',␣
 ↪xticklabels=classes, yticklabels=classes)
    plt.xlabel('Predicted Values')
    plt.ylabel('True Values')
```

```
    plt.title('Normalized Confusion Matrix')
    plt.show()

    # Calculate metrics for each class and average them
    dsc = np.mean([2.0 * cm[i, i] / (np.sum(cm[i, :]) + np.sum(cm[:, i])) for i␣
 ↪in range(cm.shape[0])])
    sensitivity = np.mean([cm[i, i] / np.sum(cm[i, :]) for i in range(cm.
 ↪shape[0])])
    specificity = np.mean([np.sum(np.delete(np.delete(cm, j, 0), j, 1)) / np.
 ↪sum(np.delete(cm, j, 0)) for j in range(cm.shape[0])])

    # Accuracy
    accuracy = accuracy_score(y_true, y_pred)

    return dsc, sensitivity, specificity, accuracy

# Predict the output
predictions_prob = model.predict(test_generator)
predictions = np.argmax(predictions_prob, axis=1)

dsc, sensitivity, specificity, accuracy = calculate_metrics(test_generator.
 ↪classes, predictions)
print(f"DSC: {dsc}, Sensitivity: {sensitivity}, Specificity: {specificity},␣
 ↪Accuracy: {accuracy}")
```

```
10/10 [==============================] - 2s 24ms/step
```

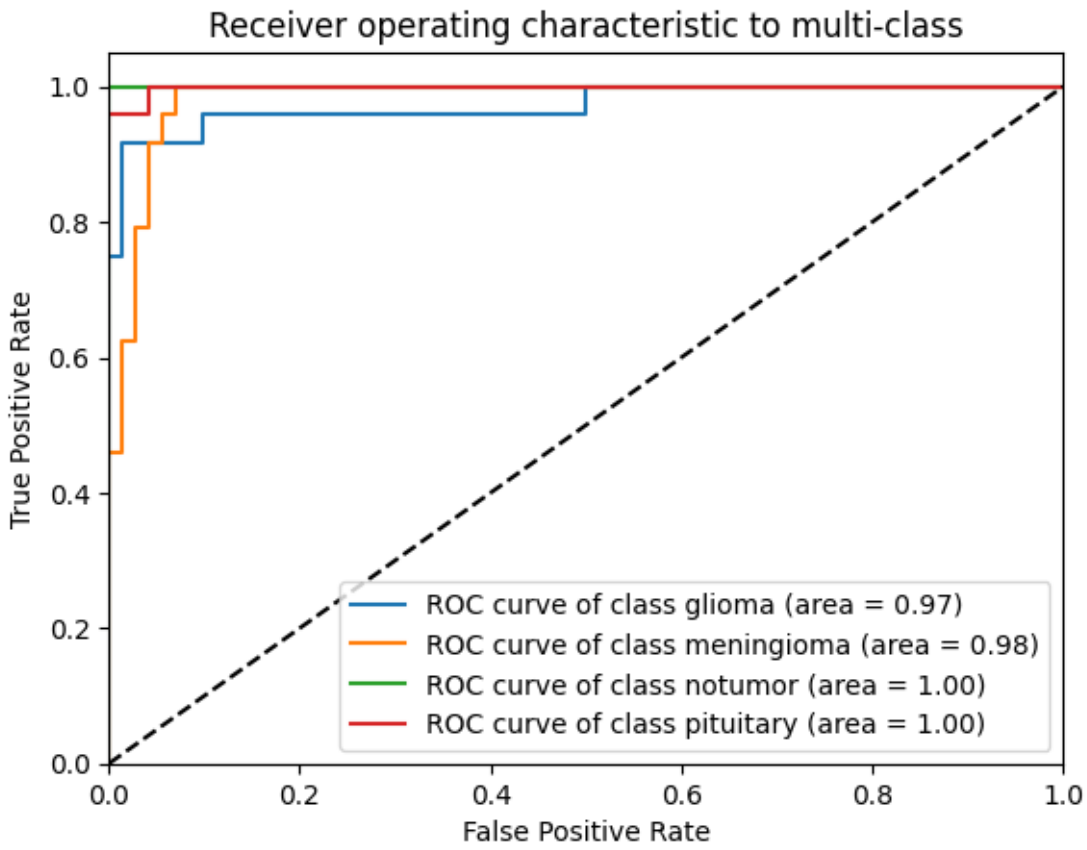

glioma
meningioma
notumor
pituitary
True Values
21
2
0
1
1
22
1
0
0
0
24
0
0
1
0
23
glioma
meningioma
notumor
pituitary
Predicted Values
20
15
10
5
0



DSC: 0.9372319580005916, Sensitivity: 0.9375, Specificity: 0.9791666666666667, Accuracy: 0.9375

[ ]:
```
from sklearn.metrics import roc_curve, auc, classification_report
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt
import numpy as np

# Binarize the output
lb = LabelBinarizer()
y_test = lb.fit_transform(test_generator.classes)
y_pred = lb.transform(predictions)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
```

```
roc_auc = dict()
for i in range(len(classes)):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], predictions_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure()
for i, class_name in enumerate(classes):
    plt.plot(fpr[i], tpr[i],
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(class_name, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()

# Print classification report
print(classification_report(y_test, y_pred, target_names=classes))
```



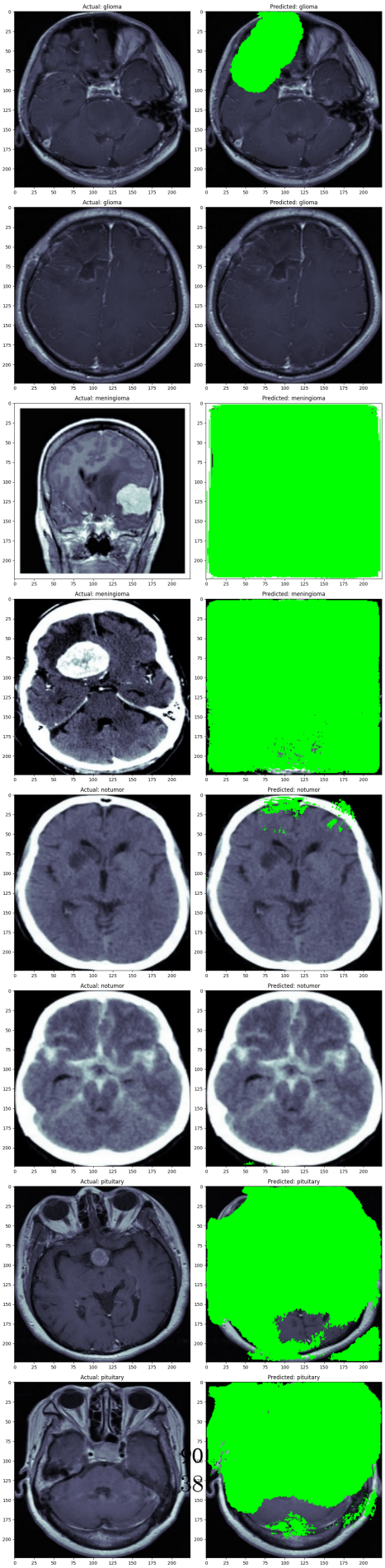
	precision	recall	f1-score	support
glioma	0.95	0.88	0.91	24
meningioma	0.88	0.92	0.90	24
notumor	0.96	1.00	0.98	24
pituitary	0.96	0.96	0.96	24
micro avg	0.94	0.94	0.94	96
macro avg	0.94	0.94	0.94	96
weighted avg	0.94	0.94	0.94	96
samples avg	0.94	0.94	0.94	96

[ ]:
```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Define the class names
classes = ['glioma', 'meningioma', 'notumor', 'pituitary']
```

```
# Get predictions from the model
predictions_model_2 = model.layers[0].predict(test_generator)
predictions_prob = model.predict(test_generator)
predictions = np.argmax(predictions_prob, axis=1)

# Thresholding
threshold = 0.9  # Adjust threshold as needed for binary mask (if needed)
binary_mask = (predictions_model_2 > threshold).astype(np.uint8)

# Track displayed classes and their counts
displayed_classes_counts = {class_name: 0 for class_name in classes}

# Number of images to display per class
num_images_per_class = 2

# Prepare to display the images
plt.figure(figsize=(12, 6 * len(classes) * num_images_per_class))
image_index = 0

for i in range(len(test_generator.filenames)):
    # Get the actual class index and name
    actual_class_index = test_generator.classes[i]
    actual_class_name = classes[actual_class_index]

    # Check if we have already displayed the required number of images for this␣
 ↪class
    if displayed_classes_counts[actual_class_name] >= num_images_per_class:
        continue

    # Load original image
    img_path = os.path.join(dir, test_generator.filenames[i])
    original_image = cv2.imread(img_path)
    original_image_rgb = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)

    # Reshape binary mask to match original image shape
    reshaped_binary_mask = binary_mask[i][:, :, 0]

    # Overlay mask on original image
    overlay = original_image_rgb.copy()
    overlay[reshaped_binary_mask == 1] = (0, 255, 0)  # Overlay mask in green␣
 ↪color

    # Get predicted class name
    predicted_class_name = classes[predictions[i]]

    # Plot original image with overlayed mask
```

```
    plt.subplot(len(classes) * num_images_per_class, 2, 2 * image_index + 1)
    plt.imshow(original_image_rgb)
    plt.title(f'Actual: {actual_class_name}')

    plt.subplot(len(classes) * num_images_per_class, 2, 2 * image_index + 2)
    plt.imshow(overlay)
    plt.title(f'Predicted: {predicted_class_name}')

    # Mark this class as displayed
    displayed_classes_counts[actual_class_name] += 1
    image_index += 1

    # Break the loop if we've displayed the required number of images for all␣
 ↪classes
    if all(count >= num_images_per_class for count in displayed_classes_counts.
 ↪values()):
        break

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

```
10/10 [==============================] - 0s 22ms/step
10/10 [==============================] - 0s 24ms/step
```


Actual: glioma
Predicted: glioma
Actual: glioma
Predicted: glioma
Actual: meningioma
Predicted: meningioma
Actual: meningioma
Predicted: meningioma
Actual: notumor
Predicted: notumor
Actual: notumor
Predicted: notumor
Actual: pituitary
Predicted: pituitary
Actual: pituitary
Predicted: pituitary

6 Optimisation

[ ]:
```
!pip install optuna tensorflow-addons
```

```
Collecting optuna
  Downloading optuna-3.6.1-py3-none-any.whl (380 kB)
                             380.1/380.1
kB 8.2 MB/s eta 0:00:00
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.23.0-cp310-cp310-manylinux_2_17_x86_64.manylin
ux2014_x86_64.whl (611 kB)
                             611.8/611.8
kB 43.2 MB/s eta 0:00:00
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.13.1-py3-none-any.whl (233 kB)
                             233.4/233.4
kB 31.8 MB/s eta 0:00:00
Collecting colorlog (from optuna)
  Downloading colorlog-6.8.2-py3-none-any.whl (11 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from optuna) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from optuna) (24.0)
Requirement already satisfied: sqlalchemy>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from optuna) (2.0.30)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from optuna) (4.66.4)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages
(from optuna) (6.0.1)
Collecting typeguard<3.0.0,>=2.7 (from tensorflow-addons)
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.5-py3-none-any.whl (78 kB)
                             78.6/78.6 kB
13.1 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions>=4 in
/usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna) (4.12.1)
Requirement already satisfied: greenlet!=0.4.17 in
/usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.3.0->optuna) (3.0.3)
Requirement already satisfied: MarkupSafe>=0.9.2 in
/usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1.5.0->optuna)
(2.1.5)
Installing collected packages: typeguard, Mako, colorlog, tensorflow-addons,
```

```
alembic, optuna
Successfully installed Mako-1.3.5 alembic-1.13.1 colorlog-6.8.2 optuna-3.6.1
tensorflow-addons-0.23.0 typeguard-2.13.3
```

[ ]:
```python
import os
os.environ["SM_FRAMEWORK"] = "tf.keras"
import optuna
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow import keras
from segmentation_models import Unet
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint,␣
 ↪EarlyStopping

# Model parameters
input_shape = (224, 224, 3)
num_classes = 4
initial_learning_rate = 1e-4

def create_model(model):
    # Define the U-Net++ model
    model = Unet(model, input_shape=input_shape, classes=num_classes,␣
 ↪activation=None)

    # Add a GlobalAveragePooling2D layer to collapse spatial dimensions
    model = keras.Sequential([
        model,
        keras.layers.GlobalAveragePooling2D()  # Pool across spatial dimensions
    ])

    # Add a Dense layer with softmax activation for final classification
    model.add(keras.layers.Dense(num_classes, activation='softmax'))

    # Select optimizer
    optimizer = keras.optimizers.Adam(learning_rate=initial_learning_rate)

    # Compile the model
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

def objective(trial):
    # Suggest Models (Other models other than efficientnet)
    base_model = trial.suggest_categorical('base_model', ['vgg16', 'resnet18',␣
 ↪'inceptionv3', 'mobilenetv2', 'densenet121'])
```

```
    # Create model
    model = create_model(base_model)

    # Callbacks
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=10, factor=0.3,␣
 ↪min_lr=1e-6, verbose=1)
    checkpoint = ModelCheckpoint(model_file, monitor='val_loss', verbose=1,␣
 ↪save_best_only=True)
    early_stopping = EarlyStopping(monitor='val_loss', min_delta=0,␣
 ↪patience=20, verbose=1, mode='auto')

    # Train the model
    history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // train_generator.batch_size,
        epochs=35,
        validation_data=test_generator,
        validation_steps=test_generator.samples // test_generator.batch_size,
        callbacks=[early_stopping, reduce_lr],
        verbose=1
    )

    # Return the best validation loss
    return min(history.history['val_loss'])

# Optuna study
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=5)

# Print the best trial
best_trial = study.best_trial
print(f"Best trial: {best_trial.value}")
print("Best parameters: ")
for key, value in best_trial.params.items():
    print(f"    {key}: {value}")
```

```
[I 2024-06-08 07:30:57,681] A new study created in memory with name: no-
name-143a6787-1abf-47aa-97ab-ea451ce02136

Downloading data from https://github.com/fchollet/deep-learning-
models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
Epoch 1/35
38/38 [==============================] - 26s 256ms/step - loss: 1.2479 -
accuracy: 0.5027 - val_loss: 1.1361 - val_accuracy: 0.6111 - lr: 1.0000e-04
Epoch 2/35
38/38 [==============================] - 5s 122ms/step - loss: 1.0934 -
accuracy: 0.6203 - val_loss: 2.1893 - val_accuracy: 0.2333 - lr: 1.0000e-04
```

```
Epoch 3/35
38/38 [==============================] - 5s 122ms/step - loss: 0.9902 -
accuracy: 0.6551 - val_loss: 2.4874 - val_accuracy: 0.5000 - lr: 1.0000e-04
Epoch 4/35
38/38 [==============================] - 5s 122ms/step - loss: 0.9325 -
accuracy: 0.6551 - val_loss: 1.0617 - val_accuracy: 0.5444 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 5s 123ms/step - loss: 0.8052 -
accuracy: 0.7112 - val_loss: 0.9246 - val_accuracy: 0.5889 - lr: 1.0000e-04
Epoch 6/35
38/38 [==============================] - 5s 127ms/step - loss: 0.8802 -
accuracy: 0.7059 - val_loss: 0.8703 - val_accuracy: 0.6333 - lr: 1.0000e-04
Epoch 7/35
38/38 [==============================] - 5s 122ms/step - loss: 0.7921 -
accuracy: 0.7166 - val_loss: 1.4683 - val_accuracy: 0.5222 - lr: 1.0000e-04
Epoch 8/35
38/38 [==============================] - 5s 123ms/step - loss: 0.7911 -
accuracy: 0.7166 - val_loss: 1.0678 - val_accuracy: 0.5667 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 5s 124ms/step - loss: 0.7004 -
accuracy: 0.7579 - val_loss: 4.3393 - val_accuracy: 0.2667 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 5s 122ms/step - loss: 0.6489 -
accuracy: 0.7781 - val_loss: 1.8751 - val_accuracy: 0.5000 - lr: 1.0000e-04
Epoch 11/35
38/38 [==============================] - 5s 124ms/step - loss: 0.5990 -
accuracy: 0.8075 - val_loss: 1.1523 - val_accuracy: 0.5333 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 5s 123ms/step - loss: 0.6217 -
accuracy: 0.7868 - val_loss: 0.7008 - val_accuracy: 0.7444 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 5s 121ms/step - loss: 0.5864 -
accuracy: 0.8048 - val_loss: 4.9328 - val_accuracy: 0.2333 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 5s 121ms/step - loss: 0.5738 -
accuracy: 0.8316 - val_loss: 1.6377 - val_accuracy: 0.5111 - lr: 1.0000e-04
Epoch 15/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4975 -
accuracy: 0.8449 - val_loss: 1.8760 - val_accuracy: 0.3889 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 5s 122ms/step - loss: 0.5835 -
accuracy: 0.7914 - val_loss: 0.8464 - val_accuracy: 0.6778 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4882 -
accuracy: 0.8422 - val_loss: 2.0833 - val_accuracy: 0.4889 - lr: 1.0000e-04
Epoch 18/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4904 -
accuracy: 0.8476 - val_loss: 1.0082 - val_accuracy: 0.6333 - lr: 1.0000e-04
```

```
Epoch 19/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4493 -
accuracy: 0.8770 - val_loss: 1.6683 - val_accuracy: 0.5111 - lr: 1.0000e-04
Epoch 20/35
38/38 [==============================] - 5s 121ms/step - loss: 0.3905 -
accuracy: 0.8850 - val_loss: 1.2608 - val_accuracy: 0.5556 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4665 -
accuracy: 0.8369 - val_loss: 1.4244 - val_accuracy: 0.5889 - lr: 1.0000e-04
Epoch 22/35
38/38 [==============================] - ETA: 0s - loss: 0.3893 - accuracy:
0.8711
Epoch 22: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
38/38 [==============================] - 5s 123ms/step - loss: 0.3893 -
accuracy: 0.8711 - val_loss: 1.3803 - val_accuracy: 0.5556 - lr: 1.0000e-04
Epoch 23/35
38/38 [==============================] - 5s 121ms/step - loss: 0.2936 -
accuracy: 0.9332 - val_loss: 0.9165 - val_accuracy: 0.6778 - lr: 3.0000e-05
Epoch 24/35
38/38 [==============================] - 5s 121ms/step - loss: 0.2832 -
accuracy: 0.9251 - val_loss: 0.5906 - val_accuracy: 0.7889 - lr: 3.0000e-05
Epoch 25/35
38/38 [==============================] - 5s 121ms/step - loss: 0.3174 -
accuracy: 0.9198 - val_loss: 1.4536 - val_accuracy: 0.5667 - lr: 3.0000e-05
Epoch 26/35
38/38 [==============================] - 5s 121ms/step - loss: 0.2810 -
accuracy: 0.9332 - val_loss: 0.8005 - val_accuracy: 0.7556 - lr: 3.0000e-05
Epoch 27/35
38/38 [==============================] - 5s 121ms/step - loss: 0.2956 -
accuracy: 0.9358 - val_loss: 0.6852 - val_accuracy: 0.7556 - lr: 3.0000e-05
Epoch 28/35
38/38 [==============================] - 5s 125ms/step - loss: 0.2742 -
accuracy: 0.9332 - val_loss: 0.6176 - val_accuracy: 0.7778 - lr: 3.0000e-05
Epoch 29/35
38/38 [==============================] - 5s 122ms/step - loss: 0.2695 -
accuracy: 0.9305 - val_loss: 0.9909 - val_accuracy: 0.6556 - lr: 3.0000e-05
Epoch 30/35
38/38 [==============================] - 5s 123ms/step - loss: 0.2644 -
accuracy: 0.9289 - val_loss: 0.7274 - val_accuracy: 0.7000 - lr: 3.0000e-05
Epoch 31/35
38/38 [==============================] - 5s 121ms/step - loss: 0.2689 -
accuracy: 0.9251 - val_loss: 0.8863 - val_accuracy: 0.6556 - lr: 3.0000e-05
Epoch 32/35
38/38 [==============================] - 5s 124ms/step - loss: 0.2754 -
accuracy: 0.9251 - val_loss: 0.5812 - val_accuracy: 0.7444 - lr: 3.0000e-05
Epoch 33/35
38/38 [==============================] - 5s 121ms/step - loss: 0.2049 -
accuracy: 0.9519 - val_loss: 0.6373 - val_accuracy: 0.7556 - lr: 3.0000e-05
```

```
Epoch 34/35
38/38 [==============================] - 5s 123ms/step - loss: 0.1686 -
accuracy: 0.9842 - val_loss: 0.8351 - val_accuracy: 0.7222 - lr: 3.0000e-05
Epoch 35/35
38/38 [==============================] - 5s 121ms/step - loss: 0.1973 -
accuracy: 0.9492 - val_loss: 0.4694 - val_accuracy: 0.8222 - lr: 3.0000e-05

[I 2024-06-08 07:34:06,960] Trial 0 finished with value: 0.469392865896225 and
parameters: {'base_model': 'vgg16'}. Best is trial 0 with value:
0.469392865896225.

Epoch 1/35
38/38 [==============================] - 38s 202ms/step - loss: 1.2545 -
accuracy: 0.4037 - val_loss: 1.3064 - val_accuracy: 0.3444 - lr: 1.0000e-04
Epoch 2/35
38/38 [==============================] - 4s 114ms/step - loss: 0.9793 -
accuracy: 0.6604 - val_loss: 1.3408 - val_accuracy: 0.4222 - lr: 1.0000e-04
Epoch 3/35
38/38 [==============================] - 4s 113ms/step - loss: 0.8129 -
accuracy: 0.8102 - val_loss: 1.5538 - val_accuracy: 0.4111 - lr: 1.0000e-04
Epoch 4/35
38/38 [==============================] - 4s 115ms/step - loss: 0.6518 -
accuracy: 0.8658 - val_loss: 1.1854 - val_accuracy: 0.5778 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 4s 115ms/step - loss: 0.5665 -
accuracy: 0.8957 - val_loss: 1.1837 - val_accuracy: 0.5667 - lr: 1.0000e-04
Epoch 6/35
38/38 [==============================] - 4s 113ms/step - loss: 0.4694 -
accuracy: 0.9198 - val_loss: 1.0173 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 7/35
38/38 [==============================] - 4s 114ms/step - loss: 0.4298 -
accuracy: 0.9144 - val_loss: 0.8966 - val_accuracy: 0.6667 - lr: 1.0000e-04
Epoch 8/35
38/38 [==============================] - 4s 115ms/step - loss: 0.4539 -
accuracy: 0.8904 - val_loss: 1.0604 - val_accuracy: 0.6667 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 4s 112ms/step - loss: 0.4025 -
accuracy: 0.9037 - val_loss: 0.9204 - val_accuracy: 0.6889 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 4s 114ms/step - loss: 0.4044 -
accuracy: 0.9198 - val_loss: 0.7372 - val_accuracy: 0.7444 - lr: 1.0000e-04
Epoch 11/35
38/38 [==============================] - 4s 114ms/step - loss: 0.3815 -
accuracy: 0.9198 - val_loss: 0.7379 - val_accuracy: 0.7111 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 4s 112ms/step - loss: 0.3428 -
accuracy: 0.9358 - val_loss: 0.7277 - val_accuracy: 0.6333 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 4s 114ms/step - loss: 0.3608 -
```

```
accuracy: 0.9064 - val_loss: 0.6543 - val_accuracy: 0.7556 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 4s 114ms/step - loss: 0.2679 -
accuracy: 0.9358 - val_loss: 0.7929 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 15/35
38/38 [==============================] - 4s 112ms/step - loss: 0.3150 -
accuracy: 0.9251 - val_loss: 0.9718 - val_accuracy: 0.7000 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 4s 114ms/step - loss: 0.2238 -
accuracy: 0.9679 - val_loss: 1.1633 - val_accuracy: 0.6222 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 4s 113ms/step - loss: 0.2804 -
accuracy: 0.9332 - val_loss: 0.8878 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 18/35
38/38 [==============================] - 4s 109ms/step - loss: 0.2666 -
accuracy: 0.9492 - val_loss: 1.1751 - val_accuracy: 0.5889 - lr: 1.0000e-04
Epoch 19/35
38/38 [==============================] - 4s 112ms/step - loss: 0.2417 -
accuracy: 0.9492 - val_loss: 0.9593 - val_accuracy: 0.7111 - lr: 1.0000e-04
Epoch 20/35
38/38 [==============================] - 4s 115ms/step - loss: 0.2773 -
accuracy: 0.9225 - val_loss: 0.7719 - val_accuracy: 0.7556 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 4s 112ms/step - loss: 0.1797 -
accuracy: 0.9652 - val_loss: 0.8172 - val_accuracy: 0.6667 - lr: 1.0000e-04
Epoch 22/35
38/38 [==============================] - 4s 116ms/step - loss: 0.1790 -
accuracy: 0.9679 - val_loss: 0.8162 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 23/35
38/38 [==============================] - ETA: 0s - loss: 0.1512 - accuracy:
0.9759
Epoch 23: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
38/38 [==============================] - 4s 112ms/step - loss: 0.1512 -
accuracy: 0.9759 - val_loss: 1.0564 - val_accuracy: 0.6556 - lr: 1.0000e-04
Epoch 24/35
38/38 [==============================] - 4s 113ms/step - loss: 0.1532 -
accuracy: 0.9733 - val_loss: 0.8402 - val_accuracy: 0.7333 - lr: 3.0000e-05
Epoch 25/35
38/38 [==============================] - 4s 115ms/step - loss: 0.1532 -
accuracy: 0.9733 - val_loss: 0.7125 - val_accuracy: 0.7667 - lr: 3.0000e-05
Epoch 26/35
38/38 [==============================] - 4s 113ms/step - loss: 0.1683 -
accuracy: 0.9733 - val_loss: 0.7961 - val_accuracy: 0.7000 - lr: 3.0000e-05
Epoch 27/35
38/38 [==============================] - 4s 114ms/step - loss: 0.1365 -
accuracy: 0.9733 - val_loss: 0.6971 - val_accuracy: 0.7556 - lr: 3.0000e-05
Epoch 28/35
38/38 [==============================] - 4s 113ms/step - loss: 0.1598 -
```

```
accuracy: 0.9706 - val_loss: 0.6604 - val_accuracy: 0.7333 - lr: 3.0000e-05
Epoch 29/35
38/38 [==============================] - 4s 112ms/step - loss: 0.1077 -
accuracy: 0.9893 - val_loss: 0.6181 - val_accuracy: 0.8000 - lr: 3.0000e-05
Epoch 30/35
38/38 [==============================] - 4s 112ms/step - loss: 0.1163 -
accuracy: 0.9786 - val_loss: 0.5885 - val_accuracy: 0.8111 - lr: 3.0000e-05
Epoch 31/35
38/38 [==============================] - 4s 115ms/step - loss: 0.1992 -
accuracy: 0.9545 - val_loss: 0.4978 - val_accuracy: 0.8333 - lr: 3.0000e-05
Epoch 32/35
38/38 [==============================] - 4s 112ms/step - loss: 0.1338 -
accuracy: 0.9733 - val_loss: 0.4801 - val_accuracy: 0.8222 - lr: 3.0000e-05
Epoch 33/35
38/38 [==============================] - 4s 112ms/step - loss: 0.1836 -
accuracy: 0.9519 - val_loss: 0.4891 - val_accuracy: 0.8444 - lr: 3.0000e-05
Epoch 34/35
38/38 [==============================] - 4s 115ms/step - loss: 0.1639 -
accuracy: 0.9706 - val_loss: 0.5248 - val_accuracy: 0.8556 - lr: 3.0000e-05
Epoch 35/35
38/38 [==============================] - 4s 112ms/step - loss: 0.1088 -
accuracy: 0.9840 - val_loss: 0.4158 - val_accuracy: 0.8222 - lr: 3.0000e-05

[I 2024-06-08 07:37:16,971] Trial 1 finished with value: 0.41584599018096924 and
parameters: {'base_model': 'mobilenetv2'}. Best is trial 1 with value:
0.41584599018096924.

Downloading data from https://github.com/keras-team/keras-applications/releases/
download/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 [==============================] - 0s 0us/step
Epoch 1/35
38/38 [==============================] - 85s 353ms/step - loss: 1.1730 -
accuracy: 0.4385 - val_loss: 1.4873 - val_accuracy: 0.2667 - lr: 1.0000e-04
Epoch 2/35
38/38 [==============================] - 5s 118ms/step - loss: 0.8247 -
accuracy: 0.8048 - val_loss: 1.0241 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 3/35
38/38 [==============================] - 5s 118ms/step - loss: 0.6565 -
accuracy: 0.8770 - val_loss: 1.0412 - val_accuracy: 0.6222 - lr: 1.0000e-04
Epoch 4/35
38/38 [==============================] - 5s 123ms/step - loss: 0.5418 -
accuracy: 0.8984 - val_loss: 0.8066 - val_accuracy: 0.6889 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4723 -
accuracy: 0.8930 - val_loss: 0.8194 - val_accuracy: 0.7000 - lr: 1.0000e-04
Epoch 6/35
38/38 [==============================] - 5s 118ms/step - loss: 0.4503 -
accuracy: 0.8984 - val_loss: 0.7051 - val_accuracy: 0.7111 - lr: 1.0000e-04
Epoch 7/35
```

```
38/38 [==============================] - 5s 120ms/step - loss: 0.4099 -
accuracy: 0.9064 - val_loss: 0.5722 - val_accuracy: 0.8222 - lr: 1.0000e-04
Epoch 8/35
38/38 [==============================] - 5s 121ms/step - loss: 0.3902 -
accuracy: 0.9144 - val_loss: 0.6681 - val_accuracy: 0.7778 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 5s 120ms/step - loss: 0.3552 -
accuracy: 0.9278 - val_loss: 0.9685 - val_accuracy: 0.7000 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 5s 120ms/step - loss: 0.3182 -
accuracy: 0.9305 - val_loss: 1.0614 - val_accuracy: 0.6556 - lr: 1.0000e-04
Epoch 11/35
38/38 [==============================] - 5s 118ms/step - loss: 0.2989 -
accuracy: 0.9439 - val_loss: 0.6674 - val_accuracy: 0.8111 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 5s 122ms/step - loss: 0.3686 -
accuracy: 0.9064 - val_loss: 0.7870 - val_accuracy: 0.7667 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 5s 118ms/step - loss: 0.2687 -
accuracy: 0.9572 - val_loss: 0.7009 - val_accuracy: 0.7333 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 5s 120ms/step - loss: 0.2634 -
accuracy: 0.9421 - val_loss: 0.5048 - val_accuracy: 0.8444 - lr: 1.0000e-04
Epoch 15/35
38/38 [==============================] - 5s 120ms/step - loss: 0.2416 -
accuracy: 0.9519 - val_loss: 0.5281 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 5s 117ms/step - loss: 0.2019 -
accuracy: 0.9679 - val_loss: 0.3697 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 5s 121ms/step - loss: 0.1909 -
accuracy: 0.9626 - val_loss: 0.4074 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 18/35
38/38 [==============================] - 5s 119ms/step - loss: 0.2971 -
accuracy: 0.9278 - val_loss: 0.6125 - val_accuracy: 0.7889 - lr: 1.0000e-04
Epoch 19/35
38/38 [==============================] - 5s 118ms/step - loss: 0.1943 -
accuracy: 0.9572 - val_loss: 0.4634 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 20/35
38/38 [==============================] - 5s 120ms/step - loss: 0.2005 -
accuracy: 0.9492 - val_loss: 0.6485 - val_accuracy: 0.7556 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 5s 118ms/step - loss: 0.2077 -
accuracy: 0.9385 - val_loss: 0.4219 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 22/35
38/38 [==============================] - 5s 117ms/step - loss: 0.1929 -
accuracy: 0.9519 - val_loss: 0.5491 - val_accuracy: 0.8444 - lr: 1.0000e-04
Epoch 23/35
```

```
38/38 [==============================] - 5s 122ms/step - loss: 0.1676 -
accuracy: 0.9706 - val_loss: 0.5475 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 24/35
38/38 [==============================] - 5s 116ms/step - loss: 0.1625 -
accuracy: 0.9679 - val_loss: 0.4974 - val_accuracy: 0.8444 - lr: 1.0000e-04
Epoch 25/35
38/38 [==============================] - 5s 117ms/step - loss: 0.1912 -
accuracy: 0.9465 - val_loss: 0.7879 - val_accuracy: 0.7778 - lr: 1.0000e-04
Epoch 26/35
38/38 [==============================] - ETA: 0s - loss: 0.2266 - accuracy:
0.9385
Epoch 26: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
38/38 [==============================] - 5s 119ms/step - loss: 0.2266 -
accuracy: 0.9385 - val_loss: 0.4998 - val_accuracy: 0.8222 - lr: 1.0000e-04
Epoch 27/35
38/38 [==============================] - 4s 115ms/step - loss: 0.1415 -
accuracy: 0.9813 - val_loss: 0.3684 - val_accuracy: 0.8556 - lr: 3.0000e-05
Epoch 28/35
38/38 [==============================] - 5s 118ms/step - loss: 0.1545 -
accuracy: 0.9545 - val_loss: 0.4192 - val_accuracy: 0.8667 - lr: 3.0000e-05
Epoch 29/35
38/38 [==============================] - 5s 116ms/step - loss: 0.0932 -
accuracy: 0.9920 - val_loss: 0.3869 - val_accuracy: 0.8556 - lr: 3.0000e-05
Epoch 30/35
38/38 [==============================] - 5s 117ms/step - loss: 0.1145 -
accuracy: 0.9786 - val_loss: 0.4019 - val_accuracy: 0.8556 - lr: 3.0000e-05
Epoch 31/35
38/38 [==============================] - 5s 119ms/step - loss: 0.1027 -
accuracy: 0.9840 - val_loss: 0.4002 - val_accuracy: 0.8556 - lr: 3.0000e-05
Epoch 32/35
38/38 [==============================] - 5s 121ms/step - loss: 0.0795 -
accuracy: 0.9893 - val_loss: 0.3828 - val_accuracy: 0.8556 - lr: 3.0000e-05
Epoch 33/35
38/38 [==============================] - 5s 117ms/step - loss: 0.0866 -
accuracy: 0.9947 - val_loss: 0.4234 - val_accuracy: 0.8778 - lr: 3.0000e-05
Epoch 34/35
38/38 [==============================] - 5s 119ms/step - loss: 0.0713 -
accuracy: 0.9920 - val_loss: 0.4155 - val_accuracy: 0.8667 - lr: 3.0000e-05
Epoch 35/35
38/38 [==============================] - 5s 119ms/step - loss: 0.0949 -
accuracy: 0.9866 - val_loss: 0.3806 - val_accuracy: 0.8667 - lr: 3.0000e-05

[I 2024-06-08 07:41:26,039] Trial 2 finished with value: 0.3683975338935852 and
parameters: {'base_model': 'densenet121'}. Best is trial 2 with value:
0.3683975338935852.

Epoch 1/35
38/38 [==============================] - 33s 140ms/step - loss: 1.1957 -
accuracy: 0.4519 - val_loss: 1.5635 - val_accuracy: 0.4222 - lr: 1.0000e-04
```

```
Epoch 2/35
38/38 [==============================] - 4s 113ms/step - loss: 0.8267 -
accuracy: 0.6979 - val_loss: 1.5192 - val_accuracy: 0.5333 - lr: 1.0000e-04
Epoch 3/35
38/38 [==============================] - 4s 117ms/step - loss: 0.7468 -
accuracy: 0.7273 - val_loss: 1.4432 - val_accuracy: 0.5111 - lr: 1.0000e-04
Epoch 4/35
38/38 [==============================] - 4s 115ms/step - loss: 0.6681 -
accuracy: 0.7701 - val_loss: 3.0160 - val_accuracy: 0.4000 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 4s 112ms/step - loss: 0.6152 -
accuracy: 0.7674 - val_loss: 3.0779 - val_accuracy: 0.3667 - lr: 1.0000e-04
Epoch 6/35
38/38 [==============================] - 4s 114ms/step - loss: 0.5390 -
accuracy: 0.7727 - val_loss: 1.4322 - val_accuracy: 0.5667 - lr: 1.0000e-04
Epoch 7/35
38/38 [==============================] - 4s 113ms/step - loss: 0.5392 -
accuracy: 0.8235 - val_loss: 1.3116 - val_accuracy: 0.5000 - lr: 1.0000e-04
Epoch 8/35
38/38 [==============================] - 4s 113ms/step - loss: 0.5382 -
accuracy: 0.8155 - val_loss: 0.9010 - val_accuracy: 0.6222 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 4s 116ms/step - loss: 0.5396 -
accuracy: 0.8316 - val_loss: 0.8812 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 4s 112ms/step - loss: 0.5130 -
accuracy: 0.8850 - val_loss: 0.8966 - val_accuracy: 0.5778 - lr: 1.0000e-04
Epoch 11/35
38/38 [==============================] - 4s 113ms/step - loss: 0.4526 -
accuracy: 0.9198 - val_loss: 0.7296 - val_accuracy: 0.6667 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 4s 114ms/step - loss: 0.4087 -
accuracy: 0.9251 - val_loss: 0.8254 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 4s 114ms/step - loss: 0.4322 -
accuracy: 0.8984 - val_loss: 1.1473 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 4s 113ms/step - loss: 0.3553 -
accuracy: 0.9332 - val_loss: 1.6578 - val_accuracy: 0.5444 - lr: 1.0000e-04
Epoch 15/35
38/38 [==============================] - 4s 112ms/step - loss: 0.4244 -
accuracy: 0.8957 - val_loss: 1.7018 - val_accuracy: 0.6000 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 4s 112ms/step - loss: 0.4186 -
accuracy: 0.8984 - val_loss: 1.1005 - val_accuracy: 0.6889 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 4s 115ms/step - loss: 0.3422 -
accuracy: 0.9358 - val_loss: 0.7901 - val_accuracy: 0.6889 - lr: 1.0000e-04
```

```
Epoch 18/35
38/38 [==============================] - 4s 113ms/step - loss: 0.3899 -
accuracy: 0.9144 - val_loss: 0.8187 - val_accuracy: 0.7222 - lr: 1.0000e-04
Epoch 19/35
38/38 [==============================] - 4s 114ms/step - loss: 0.3233 -
accuracy: 0.9492 - val_loss: 0.9699 - val_accuracy: 0.7111 - lr: 1.0000e-04
Epoch 20/35
38/38 [==============================] - 4s 113ms/step - loss: 0.2812 -
accuracy: 0.9492 - val_loss: 0.7039 - val_accuracy: 0.7667 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 4s 111ms/step - loss: 0.2747 -
accuracy: 0.9599 - val_loss: 0.8337 - val_accuracy: 0.6667 - lr: 1.0000e-04
Epoch 22/35
38/38 [==============================] - 4s 115ms/step - loss: 0.2946 -
accuracy: 0.9332 - val_loss: 0.7322 - val_accuracy: 0.7889 - lr: 1.0000e-04
Epoch 23/35
38/38 [==============================] - 4s 114ms/step - loss: 0.2811 -
accuracy: 0.9385 - val_loss: 0.9786 - val_accuracy: 0.7556 - lr: 1.0000e-04
Epoch 24/35
38/38 [==============================] - 4s 110ms/step - loss: 0.2957 -
accuracy: 0.9332 - val_loss: 0.7824 - val_accuracy: 0.7889 - lr: 1.0000e-04
Epoch 25/35
38/38 [==============================] - 4s 115ms/step - loss: 0.2666 -
accuracy: 0.9278 - val_loss: 1.1794 - val_accuracy: 0.7000 - lr: 1.0000e-04
Epoch 26/35
38/38 [==============================] - 4s 113ms/step - loss: 0.2284 -
accuracy: 0.9439 - val_loss: 1.2853 - val_accuracy: 0.7222 - lr: 1.0000e-04
Epoch 27/35
38/38 [==============================] - 4s 110ms/step - loss: 0.1914 -
accuracy: 0.9706 - val_loss: 0.8329 - val_accuracy: 0.8000 - lr: 1.0000e-04
Epoch 28/35
38/38 [==============================] - 4s 113ms/step - loss: 0.2541 -
accuracy: 0.9385 - val_loss: 1.4328 - val_accuracy: 0.7222 - lr: 1.0000e-04
Epoch 29/35
38/38 [==============================] - 4s 116ms/step - loss: 0.2371 -
accuracy: 0.9412 - val_loss: 1.2576 - val_accuracy: 0.7333 - lr: 1.0000e-04
Epoch 30/35
38/38 [==============================] - ETA: 0s - loss: 0.2174 - accuracy:
0.9599
Epoch 30: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
38/38 [==============================] - 4s 114ms/step - loss: 0.2174 -
accuracy: 0.9599 - val_loss: 0.9748 - val_accuracy: 0.7111 - lr: 1.0000e-04
Epoch 31/35
38/38 [==============================] - 4s 114ms/step - loss: 0.2281 -
accuracy: 0.9385 - val_loss: 0.9225 - val_accuracy: 0.7333 - lr: 3.0000e-05
Epoch 32/35
38/38 [==============================] - 4s 114ms/step - loss: 0.1681 -
accuracy: 0.9786 - val_loss: 0.8718 - val_accuracy: 0.7667 - lr: 3.0000e-05
```

```
Epoch 33/35
38/38 [==============================] - 4s 110ms/step - loss: 0.1569 -
accuracy: 0.9786 - val_loss: 0.8626 - val_accuracy: 0.7778 - lr: 3.0000e-05
Epoch 34/35
38/38 [==============================] - 4s 115ms/step - loss: 0.1791 -
accuracy: 0.9706 - val_loss: 0.8848 - val_accuracy: 0.7556 - lr: 3.0000e-05
Epoch 35/35
38/38 [==============================] - 4s 115ms/step - loss: 0.1407 -
accuracy: 0.9893 - val_loss: 0.9530 - val_accuracy: 0.7556 - lr: 3.0000e-05

[I 2024-06-08 07:44:29,789] Trial 3 finished with value: 0.7039403319358826 and
parameters: {'base_model': 'mobilenetv2'}. Best is trial 2 with value:
0.3683975338935852.

Epoch 1/35
38/38 [==============================] - 14s 134ms/step - loss: 1.3049 -
accuracy: 0.3904 - val_loss: 2.2623 - val_accuracy: 0.2667 - lr: 1.0000e-04
Epoch 2/35
38/38 [==============================] - 5s 122ms/step - loss: 1.1220 -
accuracy: 0.5348 - val_loss: 1.1507 - val_accuracy: 0.4444 - lr: 1.0000e-04
Epoch 3/35
38/38 [==============================] - 5s 124ms/step - loss: 1.0215 -
accuracy: 0.5763 - val_loss: 10.0723 - val_accuracy: 0.2667 - lr: 1.0000e-04
Epoch 4/35
38/38 [==============================] - 5s 122ms/step - loss: 1.0006 -
accuracy: 0.5561 - val_loss: 7.5805 - val_accuracy: 0.2667 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 5s 123ms/step - loss: 0.9584 -
accuracy: 0.6070 - val_loss: 1.5792 - val_accuracy: 0.3556 - lr: 1.0000e-04
Epoch 6/35
38/38 [==============================] - 5s 123ms/step - loss: 0.8687 -
accuracy: 0.6283 - val_loss: 3.0405 - val_accuracy: 0.4333 - lr: 1.0000e-04
Epoch 7/35
38/38 [==============================] - 5s 122ms/step - loss: 0.8583 -
accuracy: 0.6444 - val_loss: 1.1220 - val_accuracy: 0.5444 - lr: 1.0000e-04
Epoch 8/35
38/38 [==============================] - 5s 123ms/step - loss: 0.8557 -
accuracy: 0.6310 - val_loss: 0.8945 - val_accuracy: 0.6556 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 5s 122ms/step - loss: 0.8404 -
accuracy: 0.6444 - val_loss: 1.8309 - val_accuracy: 0.3444 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 5s 122ms/step - loss: 0.7703 -
accuracy: 0.6551 - val_loss: 1.2670 - val_accuracy: 0.5333 - lr: 1.0000e-04
Epoch 11/35
38/38 [==============================] - 5s 121ms/step - loss: 0.7678 -
accuracy: 0.6337 - val_loss: 1.5618 - val_accuracy: 0.4889 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 5s 121ms/step - loss: 0.8115 -
```

```
accuracy: 0.6417 - val_loss: 1.2568 - val_accuracy: 0.5000 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 5s 125ms/step - loss: 0.7349 -
accuracy: 0.6658 - val_loss: 1.0434 - val_accuracy: 0.5667 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 5s 120ms/step - loss: 0.7412 -
accuracy: 0.6604 - val_loss: 1.6460 - val_accuracy: 0.4000 - lr: 1.0000e-04
Epoch 15/35
38/38 [==============================] - 5s 122ms/step - loss: 0.6299 -
accuracy: 0.7193 - val_loss: 0.7757 - val_accuracy: 0.6556 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 5s 122ms/step - loss: 0.6985 -
accuracy: 0.6524 - val_loss: 0.7236 - val_accuracy: 0.6889 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 5s 124ms/step - loss: 0.6618 -
accuracy: 0.7032 - val_loss: 1.7551 - val_accuracy: 0.4444 - lr: 1.0000e-04
Epoch 18/35
38/38 [==============================] - 5s 122ms/step - loss: 0.6671 -
accuracy: 0.6816 - val_loss: 1.0085 - val_accuracy: 0.5889 - lr: 1.0000e-04
Epoch 19/35
38/38 [==============================] - 5s 120ms/step - loss: 0.6137 -
accuracy: 0.7246 - val_loss: 0.8230 - val_accuracy: 0.6778 - lr: 1.0000e-04
Epoch 20/35
38/38 [==============================] - 5s 121ms/step - loss: 0.5857 -
accuracy: 0.7299 - val_loss: 0.8931 - val_accuracy: 0.6000 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 5s 122ms/step - loss: 0.6615 -
accuracy: 0.6898 - val_loss: 0.9484 - val_accuracy: 0.6333 - lr: 1.0000e-04
Epoch 22/35
38/38 [==============================] - 5s 123ms/step - loss: 0.5502 -
accuracy: 0.7193 - val_loss: 0.9260 - val_accuracy: 0.5778 - lr: 1.0000e-04
Epoch 23/35
38/38 [==============================] - 5s 121ms/step - loss: 0.6351 -
accuracy: 0.7246 - val_loss: 0.7004 - val_accuracy: 0.6778 - lr: 1.0000e-04
Epoch 24/35
38/38 [==============================] - 5s 121ms/step - loss: 0.6037 -
accuracy: 0.7139 - val_loss: 0.9531 - val_accuracy: 0.6556 - lr: 1.0000e-04
Epoch 25/35
38/38 [==============================] - 5s 121ms/step - loss: 0.5587 -
accuracy: 0.7540 - val_loss: 1.4842 - val_accuracy: 0.5111 - lr: 1.0000e-04
Epoch 26/35
38/38 [==============================] - 5s 121ms/step - loss: 0.5518 -
accuracy: 0.7433 - val_loss: 0.9693 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 27/35
38/38 [==============================] - 5s 121ms/step - loss: 0.5460 -
accuracy: 0.7701 - val_loss: 0.7202 - val_accuracy: 0.6778 - lr: 1.0000e-04
Epoch 28/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4633 -
```

```
accuracy: 0.7834 - val_loss: 1.0733 - val_accuracy: 0.6333 - lr: 1.0000e-04
Epoch 29/35
38/38 [==============================] - 5s 122ms/step - loss: 0.4821 -
accuracy: 0.7781 - val_loss: 0.8798 - val_accuracy: 0.7444 - lr: 1.0000e-04
Epoch 30/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4823 -
accuracy: 0.7995 - val_loss: 0.7515 - val_accuracy: 0.7667 - lr: 1.0000e-04
Epoch 31/35
38/38 [==============================] - 5s 121ms/step - loss: 0.5264 -
accuracy: 0.7727 - val_loss: 0.7730 - val_accuracy: 0.7444 - lr: 1.0000e-04
Epoch 32/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4960 -
accuracy: 0.7647 - val_loss: 1.0305 - val_accuracy: 0.6556 - lr: 1.0000e-04
Epoch 33/35
38/38 [==============================] - ETA: 0s - loss: 0.4634 - accuracy:
0.8342
Epoch 33: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
38/38 [==============================] - 5s 121ms/step - loss: 0.4634 -
accuracy: 0.8342 - val_loss: 0.7953 - val_accuracy: 0.6667 - lr: 1.0000e-04
Epoch 34/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4334 -
accuracy: 0.8396 - val_loss: 0.7137 - val_accuracy: 0.7000 - lr: 3.0000e-05
Epoch 35/35
38/38 [==============================] - 5s 121ms/step - loss: 0.4282 -
accuracy: 0.8422 - val_loss: 0.7402 - val_accuracy: 0.8222 - lr: 3.0000e-05

[I 2024-06-08 07:47:26,167] Trial 4 finished with value: 0.700380265712738 and
parameters: {'base_model': 'vgg16'}. Best is trial 2 with value:
0.3683975338935852.

Best trial: 0.3683975338935852
Best parameters:
    base_model: densenet121
```

7 K-Folds Validation

[ ]:
```python
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from segmentation_models import Unet
from tensorflow import keras
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint,␣
 ↪EarlyStopping

# Set parameters
dir = 'dataset_19'  # Update this to your dataset directory
```

```
batch_size = 10
input_shape = (224, 224, 3)
num_classes = 4
initial_learning_rate = 1e-4
k = 5

# Prepare data generators
datagen = ImageDataGenerator(
    rescale=1. / 255,
    horizontal_flip=True,
    rotation_range=20
)

# List all images and labels
all_images = []
all_labels = []

for class_index, class_name in enumerate(os.listdir(dir)):
    class_dir = os.path.join(dir, class_name)
    for image_name in os.listdir(class_dir):
        all_images.append(os.path.join(class_dir, image_name))
        all_labels.append(str(class_index))  # Convert class index to string

# Convert lists to numpy arrays
all_images = np.array(all_images)
all_labels = np.array(all_labels)

# Define K-Fold cross-validation
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize lists to store metrics
fold_accuracies = []
fold_losses = []

# Iterate over each fold
for fold, (train_index, val_index) in enumerate(kf.split(all_images)):
    print(f'Fold {fold + 1}/{k}')

    train_images, val_images = all_images[train_index], all_images[val_index]
    train_labels, val_labels = all_labels[train_index], all_labels[val_index]

    # Create DataFrames for training and validation data
    train_df = pd.DataFrame({'filename': train_images, 'class': train_labels})
    val_df = pd.DataFrame({'filename': val_images, 'class': val_labels})

    # Create training and validation data generators
    train_generator = datagen.flow_from_dataframe(
```

```
        dataframe=train_df,
        x_col='filename',
        y_col='class',
        target_size=(224, 224),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True
    )

    validation_generator = datagen.flow_from_dataframe(
        dataframe=val_df,
        x_col='filename',
        y_col='class',
        target_size=(224, 224),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

    # Define the U-Net++ model
    model = Unet('efficientnetb1', input_shape=input_shape,␣
↪classes=num_classes, activation=None)

    # Add a GlobalAveragePooling2D layer and Dense layer with softmax activation
    model = keras.Sequential([
        model,
        keras.layers.GlobalAveragePooling2D(),  # Pool across spatial dimensions
        keras.layers.Dense(num_classes, activation='softmax')  # Final␣
↪classification layer
    ])

    # Compile the model
    optimizer = keras.optimizers.Adam(learning_rate=initial_learning_rate)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

    # Callbacks
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=10, factor=0.3,␣
↪min_lr=1e-6, verbose=1)
    # checkpoint = ModelCheckpoint(f'model_fold_{fold + 1}.h5',␣
↪monitor='val_loss', verbose=1, save_best_only=True)
    early_stopping = EarlyStopping(monitor='val_loss', patience=15, verbose=1,␣
↪mode='auto')

    # Train the model
    history = model.fit(
```

```
        train_generator,
        steps_per_epoch=train_generator.samples // train_generator.batch_size,
        epochs=35,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples // validation_generator.
 ↪batch_size,
        callbacks=[early_stopping, reduce_lr]
    )

    # Store the best validation accuracy and loss for the current fold
    best_val_accuracy = max(history.history['val_accuracy'])
    best_val_loss = min(history.history['val_loss'])
    fold_accuracies.append(best_val_accuracy)
    fold_losses.append(best_val_loss)

# Calculate mean and standard deviation of accuracies and losses
mean_accuracy = np.mean(fold_accuracies)
std_accuracy = np.std(fold_accuracies)
mean_loss = np.mean(fold_losses)
std_loss = np.std(fold_losses)

print(f'Validation Accuracy: {mean_accuracy:.4f} ± {std_accuracy:.4f}')
print(f'Validation Loss: {mean_loss:.4f} ± {std_loss:.4f}')
```

```
Fold 1/5
Found 384 validated image filenames belonging to 4 classes.
Found 96 validated image filenames belonging to 4 classes.
Epoch 1/35
38/38 [==============================] - 57s 190ms/step - loss: 1.2399 -
accuracy: 0.3984 - val_loss: 1.8823 - val_accuracy: 0.2111 - lr: 1.0000e-04
Epoch 2/35
38/38 [==============================] - 5s 136ms/step - loss: 0.9369 -
accuracy: 0.6471 - val_loss: 1.6458 - val_accuracy: 0.3778 - lr: 1.0000e-04
Epoch 3/35
38/38 [==============================] - 5s 135ms/step - loss: 0.7928 -
accuracy: 0.7166 - val_loss: 2.7122 - val_accuracy: 0.2111 - lr: 1.0000e-04
Epoch 4/35
38/38 [==============================] - 5s 139ms/step - loss: 0.6554 -
accuracy: 0.7727 - val_loss: 2.0891 - val_accuracy: 0.3778 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 5s 134ms/step - loss: 0.5753 -
accuracy: 0.8182 - val_loss: 1.1983 - val_accuracy: 0.5667 - lr: 1.0000e-04
Epoch 6/35
38/38 [==============================] - 5s 139ms/step - loss: 0.5400 -
accuracy: 0.8209 - val_loss: 1.3590 - val_accuracy: 0.5000 - lr: 1.0000e-04
Epoch 7/35
38/38 [==============================] - 5s 135ms/step - loss: 0.4551 -
accuracy: 0.8930 - val_loss: 1.1837 - val_accuracy: 0.5444 - lr: 1.0000e-04
```

```
Epoch 8/35
38/38 [==============================] - 5s 135ms/step - loss: 0.4439 -
accuracy: 0.8930 - val_loss: 1.1023 - val_accuracy: 0.5778 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 5s 138ms/step - loss: 0.3773 -
accuracy: 0.9278 - val_loss: 0.5592 - val_accuracy: 0.7556 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 5s 140ms/step - loss: 0.3848 -
accuracy: 0.9198 - val_loss: 0.7113 - val_accuracy: 0.7444 - lr: 1.0000e-04
Epoch 11/35
38/38 [==============================] - 5s 138ms/step - loss: 0.3590 -
accuracy: 0.9091 - val_loss: 0.6510 - val_accuracy: 0.7667 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 5s 140ms/step - loss: 0.3546 -
accuracy: 0.8930 - val_loss: 0.5668 - val_accuracy: 0.8111 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 5s 138ms/step - loss: 0.3032 -
accuracy: 0.9225 - val_loss: 0.5003 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 5s 136ms/step - loss: 0.3266 -
accuracy: 0.9171 - val_loss: 0.4488 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 15/35
38/38 [==============================] - 5s 137ms/step - loss: 0.3337 -
accuracy: 0.9144 - val_loss: 0.5258 - val_accuracy: 0.8889 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 5s 134ms/step - loss: 0.2583 -
accuracy: 0.9412 - val_loss: 0.4446 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 5s 138ms/step - loss: 0.2600 -
accuracy: 0.9412 - val_loss: 0.3909 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 18/35
38/38 [==============================] - 5s 138ms/step - loss: 0.2657 -
accuracy: 0.9278 - val_loss: 0.4241 - val_accuracy: 0.8556 - lr: 1.0000e-04
Epoch 19/35
38/38 [==============================] - 5s 136ms/step - loss: 0.2117 -
accuracy: 0.9679 - val_loss: 0.4133 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 20/35
38/38 [==============================] - 5s 138ms/step - loss: 0.2542 -
accuracy: 0.9412 - val_loss: 0.4431 - val_accuracy: 0.8556 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 5s 136ms/step - loss: 0.2442 -
accuracy: 0.9439 - val_loss: 0.3850 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 22/35
38/38 [==============================] - 5s 135ms/step - loss: 0.1966 -
accuracy: 0.9572 - val_loss: 0.4043 - val_accuracy: 0.8889 - lr: 1.0000e-04
Epoch 23/35
38/38 [==============================] - 5s 135ms/step - loss: 0.1758 -
accuracy: 0.9626 - val_loss: 0.4189 - val_accuracy: 0.8778 - lr: 1.0000e-04
```

```
Epoch 24/35
38/38 [==============================] - 5s 131ms/step - loss: 0.2093 -
accuracy: 0.9332 - val_loss: 0.3785 - val_accuracy: 0.8889 - lr: 1.0000e-04
Epoch 25/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1869 -
accuracy: 0.9652 - val_loss: 0.5038 - val_accuracy: 0.8222 - lr: 1.0000e-04
Epoch 26/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1755 -
accuracy: 0.9759 - val_loss: 0.3520 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 27/35
38/38 [==============================] - 5s 140ms/step - loss: 0.2165 -
accuracy: 0.9519 - val_loss: 0.3945 - val_accuracy: 0.8889 - lr: 1.0000e-04
Epoch 28/35
38/38 [==============================] - 5s 135ms/step - loss: 0.1674 -
accuracy: 0.9572 - val_loss: 0.3791 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 29/35
38/38 [==============================] - 5s 139ms/step - loss: 0.1617 -
accuracy: 0.9632 - val_loss: 0.4668 - val_accuracy: 0.8556 - lr: 1.0000e-04
Epoch 30/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1703 -
accuracy: 0.9519 - val_loss: 0.4105 - val_accuracy: 0.8556 - lr: 1.0000e-04
Epoch 31/35
38/38 [==============================] - 5s 134ms/step - loss: 0.1206 -
accuracy: 0.9733 - val_loss: 0.4047 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 32/35
38/38 [==============================] - 5s 139ms/step - loss: 0.1894 -
accuracy: 0.9599 - val_loss: 0.4956 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 33/35
38/38 [==============================] - 5s 140ms/step - loss: 0.1501 -
accuracy: 0.9652 - val_loss: 0.4337 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 34/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1333 -
accuracy: 0.9733 - val_loss: 0.4001 - val_accuracy: 0.8778 - lr: 1.0000e-04
Epoch 35/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1218 -
accuracy: 0.9786 - val_loss: 0.2722 - val_accuracy: 0.9000 - lr: 1.0000e-04
Fold 2/5
Found 384 validated image filenames belonging to 4 classes.
Found 96 validated image filenames belonging to 4 classes.
Epoch 1/35
38/38 [==============================] - 58s 192ms/step - loss: 1.2892 -
accuracy: 0.3128 - val_loss: 1.3893 - val_accuracy: 0.1778 - lr: 1.0000e-04
Epoch 2/35
38/38 [==============================] - 5s 138ms/step - loss: 0.8810 -
accuracy: 0.7684 - val_loss: 1.0304 - val_accuracy: 0.6222 - lr: 1.0000e-04
Epoch 3/35
38/38 [==============================] - 5s 137ms/step - loss: 0.6350 -
accuracy: 0.8476 - val_loss: 1.1861 - val_accuracy: 0.4778 - lr: 1.0000e-04
```

```
Epoch 4/35
38/38 [==============================] - 5s 136ms/step - loss: 0.5703 -
accuracy: 0.8369 - val_loss: 0.8186 - val_accuracy: 0.7000 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 5s 138ms/step - loss: 0.4640 -
accuracy: 0.8824 - val_loss: 0.7069 - val_accuracy: 0.6889 - lr: 1.0000e-04
Epoch 6/35
38/38 [==============================] - 5s 136ms/step - loss: 0.4226 -
accuracy: 0.8984 - val_loss: 1.0807 - val_accuracy: 0.6333 - lr: 1.0000e-04
Epoch 7/35
38/38 [==============================] - 5s 139ms/step - loss: 0.4159 -
accuracy: 0.8797 - val_loss: 0.5211 - val_accuracy: 0.8000 - lr: 1.0000e-04
Epoch 8/35
38/38 [==============================] - 5s 139ms/step - loss: 0.2990 -
accuracy: 0.9358 - val_loss: 0.3450 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 5s 135ms/step - loss: 0.2885 -
accuracy: 0.9171 - val_loss: 0.2715 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 5s 139ms/step - loss: 0.3251 -
accuracy: 0.9064 - val_loss: 0.2755 - val_accuracy: 0.9111 - lr: 1.0000e-04
Epoch 11/35
38/38 [==============================] - 5s 139ms/step - loss: 0.2877 -
accuracy: 0.9519 - val_loss: 0.3364 - val_accuracy: 0.8556 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 5s 140ms/step - loss: 0.2332 -
accuracy: 0.9492 - val_loss: 0.2494 - val_accuracy: 0.9000 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 5s 136ms/step - loss: 0.2922 -
accuracy: 0.9251 - val_loss: 0.4312 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 5s 138ms/step - loss: 0.2159 -
accuracy: 0.9599 - val_loss: 0.2044 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 15/35
38/38 [==============================] - 5s 137ms/step - loss: 0.2291 -
accuracy: 0.9332 - val_loss: 0.2290 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 5s 135ms/step - loss: 0.2077 -
accuracy: 0.9465 - val_loss: 0.3648 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 5s 138ms/step - loss: 0.1758 -
accuracy: 0.9652 - val_loss: 0.1405 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 18/35
38/38 [==============================] - 5s 135ms/step - loss: 0.1763 -
accuracy: 0.9545 - val_loss: 0.3239 - val_accuracy: 0.8889 - lr: 1.0000e-04
Epoch 19/35
38/38 [==============================] - 5s 138ms/step - loss: 0.2455 -
accuracy: 0.9251 - val_loss: 0.5013 - val_accuracy: 0.8556 - lr: 1.0000e-04
```

```
Epoch 20/35
38/38 [==============================] - 5s 135ms/step - loss: 0.2114 -
accuracy: 0.9439 - val_loss: 0.2086 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1898 -
accuracy: 0.9545 - val_loss: 0.1746 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 22/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1446 -
accuracy: 0.9786 - val_loss: 0.2200 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 23/35
38/38 [==============================] - 5s 134ms/step - loss: 0.1895 -
accuracy: 0.9492 - val_loss: 0.1859 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 24/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1595 -
accuracy: 0.9599 - val_loss: 0.1230 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 25/35
38/38 [==============================] - 5s 132ms/step - loss: 0.1253 -
accuracy: 0.9733 - val_loss: 0.2291 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 26/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1259 -
accuracy: 0.9706 - val_loss: 0.1250 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 27/35
38/38 [==============================] - 5s 140ms/step - loss: 0.1143 -
accuracy: 0.9813 - val_loss: 0.2505 - val_accuracy: 0.9111 - lr: 1.0000e-04
Epoch 28/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1351 -
accuracy: 0.9652 - val_loss: 0.2583 - val_accuracy: 0.9111 - lr: 1.0000e-04
Epoch 29/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1293 -
accuracy: 0.9679 - val_loss: 0.2453 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 30/35
38/38 [==============================] - 5s 134ms/step - loss: 0.1371 -
accuracy: 0.9759 - val_loss: 0.2666 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 31/35
38/38 [==============================] - 5s 138ms/step - loss: 0.0910 -
accuracy: 0.9866 - val_loss: 0.2009 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 32/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1388 -
accuracy: 0.9652 - val_loss: 0.2769 - val_accuracy: 0.9000 - lr: 1.0000e-04
Epoch 33/35
38/38 [==============================] - 5s 139ms/step - loss: 0.1176 -
accuracy: 0.9759 - val_loss: 0.1460 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 34/35
38/38 [==============================] - ETA: 0s - loss: 0.1194 - accuracy:
0.9733
Epoch 34: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
38/38 [==============================] - 5s 138ms/step - loss: 0.1194 -
accuracy: 0.9733 - val_loss: 0.2064 - val_accuracy: 0.9111 - lr: 1.0000e-04
```

```
Epoch 35/35
38/38 [==============================] - 5s 136ms/step - loss: 0.0850 -
accuracy: 0.9813 - val_loss: 0.1733 - val_accuracy: 0.9556 - lr: 3.0000e-05
Fold 3/5
Found 384 validated image filenames belonging to 4 classes.
Found 96 validated image filenames belonging to 4 classes.
Epoch 1/35
38/38 [==============================] - 57s 195ms/step - loss: 1.2982 -
accuracy: 0.3663 - val_loss: 1.3627 - val_accuracy: 0.3889 - lr: 1.0000e-04
Epoch 2/35
38/38 [==============================] - 5s 140ms/step - loss: 1.0139 -
accuracy: 0.5882 - val_loss: 1.5761 - val_accuracy: 0.3222 - lr: 1.0000e-04
Epoch 3/35
38/38 [==============================] - 6s 149ms/step - loss: 0.7978 -
accuracy: 0.7861 - val_loss: 1.4460 - val_accuracy: 0.3556 - lr: 1.0000e-04
Epoch 4/35
38/38 [==============================] - 5s 141ms/step - loss: 0.6344 -
accuracy: 0.8610 - val_loss: 1.9368 - val_accuracy: 0.4111 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 6s 143ms/step - loss: 0.6087 -
accuracy: 0.8369 - val_loss: 1.7242 - val_accuracy: 0.4667 - lr: 1.0000e-04
Epoch 6/35
38/38 [==============================] - 5s 139ms/step - loss: 0.5309 -
accuracy: 0.8824 - val_loss: 1.5913 - val_accuracy: 0.5000 - lr: 1.0000e-04
Epoch 7/35
38/38 [==============================] - 5s 137ms/step - loss: 0.4240 -
accuracy: 0.9064 - val_loss: 1.0169 - val_accuracy: 0.6000 - lr: 1.0000e-04
Epoch 8/35
38/38 [==============================] - 5s 136ms/step - loss: 0.3966 -
accuracy: 0.9332 - val_loss: 0.6869 - val_accuracy: 0.7667 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 5s 140ms/step - loss: 0.3895 -
accuracy: 0.9144 - val_loss: 0.6433 - val_accuracy: 0.8000 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 5s 134ms/step - loss: 0.3121 -
accuracy: 0.9492 - val_loss: 0.3834 - val_accuracy: 0.8667 - lr: 1.0000e-04
Epoch 11/35
38/38 [==============================] - 5s 137ms/step - loss: 0.3847 -
accuracy: 0.8957 - val_loss: 0.3918 - val_accuracy: 0.9000 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 5s 134ms/step - loss: 0.3125 -
accuracy: 0.9358 - val_loss: 0.2694 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 5s 137ms/step - loss: 0.3319 -
accuracy: 0.9144 - val_loss: 0.3977 - val_accuracy: 0.8444 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 5s 136ms/step - loss: 0.3497 -
accuracy: 0.9053 - val_loss: 0.2863 - val_accuracy: 0.9333 - lr: 1.0000e-04
```

```
Epoch 15/35
38/38 [==============================] - 5s 137ms/step - loss: 0.2879 -
accuracy: 0.9332 - val_loss: 0.2340 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 5s 137ms/step - loss: 0.2616 -
accuracy: 0.9492 - val_loss: 0.1837 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 5s 138ms/step - loss: 0.2441 -
accuracy: 0.9412 - val_loss: 0.2427 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 18/35
38/38 [==============================] - 5s 139ms/step - loss: 0.2083 -
accuracy: 0.9465 - val_loss: 0.2219 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 19/35
38/38 [==============================] - 5s 139ms/step - loss: 0.2425 -
accuracy: 0.9439 - val_loss: 0.2542 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 20/35
38/38 [==============================] - 5s 140ms/step - loss: 0.2563 -
accuracy: 0.9225 - val_loss: 0.2547 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 5s 136ms/step - loss: 0.2043 -
accuracy: 0.9572 - val_loss: 0.1383 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 22/35
38/38 [==============================] - 5s 133ms/step - loss: 0.1661 -
accuracy: 0.9626 - val_loss: 0.1533 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 23/35
38/38 [==============================] - 5s 138ms/step - loss: 0.1647 -
accuracy: 0.9706 - val_loss: 0.1951 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 24/35
38/38 [==============================] - 5s 137ms/step - loss: 0.2452 -
accuracy: 0.9385 - val_loss: 0.2085 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 25/35
38/38 [==============================] - 5s 140ms/step - loss: 0.1333 -
accuracy: 0.9813 - val_loss: 0.2215 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 26/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1577 -
accuracy: 0.9572 - val_loss: 0.2615 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 27/35
38/38 [==============================] - 5s 139ms/step - loss: 0.1476 -
accuracy: 0.9652 - val_loss: 0.1430 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 28/35
38/38 [==============================] - 5s 135ms/step - loss: 0.1609 -
accuracy: 0.9626 - val_loss: 0.1335 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 29/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1676 -
accuracy: 0.9572 - val_loss: 0.1816 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 30/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1671 -
accuracy: 0.9626 - val_loss: 0.1794 - val_accuracy: 0.9333 - lr: 1.0000e-04
```

```
Epoch 31/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1677 -
accuracy: 0.9599 - val_loss: 0.2026 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 32/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1237 -
accuracy: 0.9679 - val_loss: 0.1665 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 33/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1184 -
accuracy: 0.9759 - val_loss: 0.1745 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 34/35
38/38 [==============================] - 5s 138ms/step - loss: 0.1274 -
accuracy: 0.9733 - val_loss: 0.1455 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 35/35
38/38 [==============================] - 5s 139ms/step - loss: 0.1069 -
accuracy: 0.9759 - val_loss: 0.1422 - val_accuracy: 0.9556 - lr: 1.0000e-04
Fold 4/5
Found 384 validated image filenames belonging to 4 classes.
Found 96 validated image filenames belonging to 4 classes.
Epoch 1/35
38/38 [==============================] - 58s 191ms/step - loss: 1.2535 -
accuracy: 0.5374 - val_loss: 2.7770 - val_accuracy: 0.2556 - lr: 1.0000e-04
Epoch 2/35
38/38 [==============================] - 5s 134ms/step - loss: 0.9078 -
accuracy: 0.7326 - val_loss: 1.5406 - val_accuracy: 0.2889 - lr: 1.0000e-04
Epoch 3/35
38/38 [==============================] - 5s 135ms/step - loss: 0.7100 -
accuracy: 0.8075 - val_loss: 1.5076 - val_accuracy: 0.4333 - lr: 1.0000e-04
Epoch 4/35
38/38 [==============================] - 5s 138ms/step - loss: 0.6325 -
accuracy: 0.8422 - val_loss: 1.0268 - val_accuracy: 0.6222 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 5s 140ms/step - loss: 0.4816 -
accuracy: 0.8816 - val_loss: 1.2346 - val_accuracy: 0.5778 - lr: 1.0000e-04
Epoch 6/35
38/38 [==============================] - 5s 134ms/step - loss: 0.4791 -
accuracy: 0.8717 - val_loss: 0.7495 - val_accuracy: 0.7333 - lr: 1.0000e-04
Epoch 7/35
38/38 [==============================] - 5s 139ms/step - loss: 0.3507 -
accuracy: 0.9251 - val_loss: 0.5378 - val_accuracy: 0.8111 - lr: 1.0000e-04
Epoch 8/35
38/38 [==============================] - 5s 136ms/step - loss: 0.3733 -
accuracy: 0.9011 - val_loss: 0.5638 - val_accuracy: 0.8333 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 5s 135ms/step - loss: 0.3708 -
accuracy: 0.8984 - val_loss: 0.3457 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 5s 136ms/step - loss: 0.3428 -
accuracy: 0.9225 - val_loss: 0.3329 - val_accuracy: 0.8667 - lr: 1.0000e-04
```

```
Epoch 11/35
38/38 [==============================] - 5s 136ms/step - loss: 0.3321 -
accuracy: 0.9064 - val_loss: 0.2922 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 5s 138ms/step - loss: 0.3595 -
accuracy: 0.8984 - val_loss: 0.3011 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 5s 136ms/step - loss: 0.2547 -
accuracy: 0.9358 - val_loss: 0.2698 - val_accuracy: 0.9111 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 5s 137ms/step - loss: 0.2545 -
accuracy: 0.9332 - val_loss: 0.2431 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 15/35
38/38 [==============================] - 5s 137ms/step - loss: 0.2632 -
accuracy: 0.9412 - val_loss: 0.3159 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 5s 135ms/step - loss: 0.2704 -
accuracy: 0.9385 - val_loss: 0.2589 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 5s 138ms/step - loss: 0.2381 -
accuracy: 0.9439 - val_loss: 0.2320 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 18/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1896 -
accuracy: 0.9599 - val_loss: 0.2004 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 19/35
38/38 [==============================] - 5s 137ms/step - loss: 0.2151 -
accuracy: 0.9332 - val_loss: 0.2037 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 20/35
38/38 [==============================] - 5s 139ms/step - loss: 0.2450 -
accuracy: 0.9421 - val_loss: 0.2088 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 5s 139ms/step - loss: 0.1882 -
accuracy: 0.9652 - val_loss: 0.1021 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 22/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1427 -
accuracy: 0.9679 - val_loss: 0.1396 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 23/35
38/38 [==============================] - 5s 135ms/step - loss: 0.1782 -
accuracy: 0.9572 - val_loss: 0.1710 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 24/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1892 -
accuracy: 0.9519 - val_loss: 0.1637 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 25/35
38/38 [==============================] - 5s 133ms/step - loss: 0.1332 -
accuracy: 0.9733 - val_loss: 0.1435 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 26/35
38/38 [==============================] - 5s 135ms/step - loss: 0.1179 -
accuracy: 0.9840 - val_loss: 0.1350 - val_accuracy: 0.9556 - lr: 1.0000e-04
```

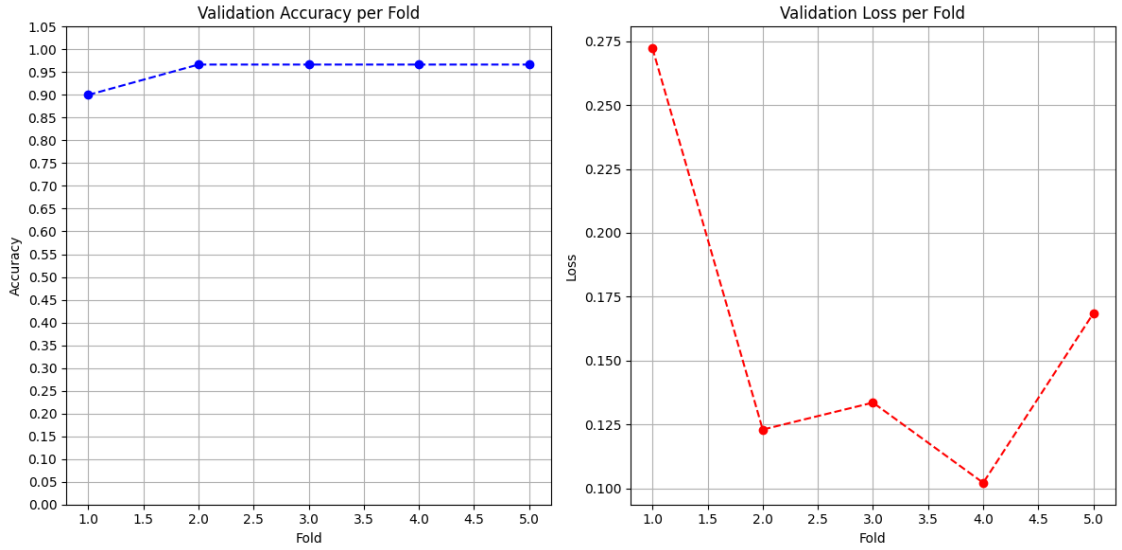
```
Epoch 27/35
38/38 [==============================] - 5s 139ms/step - loss: 0.1797 -
accuracy: 0.9492 - val_loss: 0.1750 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 28/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1800 -
accuracy: 0.9519 - val_loss: 0.2239 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 29/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1406 -
accuracy: 0.9759 - val_loss: 0.1644 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 30/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1192 -
accuracy: 0.9679 - val_loss: 0.2364 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 31/35
38/38 [==============================] - ETA: 0s - loss: 0.1179 - accuracy:
0.9733
Epoch 31: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
38/38 [==============================] - 5s 137ms/step - loss: 0.1179 -
accuracy: 0.9733 - val_loss: 0.2886 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 32/35
38/38 [==============================] - 5s 133ms/step - loss: 0.1160 -
accuracy: 0.9706 - val_loss: 0.2078 - val_accuracy: 0.9333 - lr: 3.0000e-05
Epoch 33/35
38/38 [==============================] - 5s 139ms/step - loss: 0.1023 -
accuracy: 0.9866 - val_loss: 0.1989 - val_accuracy: 0.9556 - lr: 3.0000e-05
Epoch 34/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1004 -
accuracy: 0.9813 - val_loss: 0.2255 - val_accuracy: 0.9444 - lr: 3.0000e-05
Epoch 35/35
38/38 [==============================] - 5s 135ms/step - loss: 0.0970 -
accuracy: 0.9759 - val_loss: 0.1606 - val_accuracy: 0.9667 - lr: 3.0000e-05
Fold 5/5
Found 384 validated image filenames belonging to 4 classes.
Found 96 validated image filenames belonging to 4 classes.
Epoch 1/35
38/38 [==============================] - 56s 187ms/step - loss: 1.1876 -
accuracy: 0.4840 - val_loss: 1.4566 - val_accuracy: 0.2556 - lr: 1.0000e-04
Epoch 2/35
38/38 [==============================] - 5s 135ms/step - loss: 0.8170 -
accuracy: 0.6925 - val_loss: 3.3628 - val_accuracy: 0.3111 - lr: 1.0000e-04
Epoch 3/35
38/38 [==============================] - 5s 136ms/step - loss: 0.6898 -
accuracy: 0.7166 - val_loss: 1.7661 - val_accuracy: 0.4667 - lr: 1.0000e-04
Epoch 4/35
38/38 [==============================] - 5s 137ms/step - loss: 0.6767 -
accuracy: 0.7005 - val_loss: 1.4600 - val_accuracy: 0.4889 - lr: 1.0000e-04
Epoch 5/35
38/38 [==============================] - 5s 136ms/step - loss: 0.5777 -
accuracy: 0.7727 - val_loss: 1.2615 - val_accuracy: 0.5111 - lr: 1.0000e-04
```

```
Epoch 6/35
38/38 [==============================] - 5s 135ms/step - loss: 0.5373 -
accuracy: 0.7995 - val_loss: 1.4967 - val_accuracy: 0.5667 - lr: 1.0000e-04
Epoch 7/35
38/38 [==============================] - 5s 139ms/step - loss: 0.4362 -
accuracy: 0.8984 - val_loss: 0.8511 - val_accuracy: 0.6444 - lr: 1.0000e-04
Epoch 8/35
38/38 [==============================] - 5s 133ms/step - loss: 0.4949 -
accuracy: 0.8449 - val_loss: 0.5425 - val_accuracy: 0.7889 - lr: 1.0000e-04
Epoch 9/35
38/38 [==============================] - 5s 135ms/step - loss: 0.3850 -
accuracy: 0.9118 - val_loss: 0.3731 - val_accuracy: 0.8889 - lr: 1.0000e-04
Epoch 10/35
38/38 [==============================] - 5s 138ms/step - loss: 0.3814 -
accuracy: 0.8984 - val_loss: 0.2692 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 11/35
38/38 [==============================] - 5s 137ms/step - loss: 0.3470 -
accuracy: 0.9225 - val_loss: 0.2815 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 12/35
38/38 [==============================] - 5s 139ms/step - loss: 0.2815 -
accuracy: 0.9385 - val_loss: 0.3079 - val_accuracy: 0.9111 - lr: 1.0000e-04
Epoch 13/35
38/38 [==============================] - 5s 135ms/step - loss: 0.2874 -
accuracy: 0.9251 - val_loss: 0.2806 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 14/35
38/38 [==============================] - 5s 137ms/step - loss: 0.3182 -
accuracy: 0.9037 - val_loss: 0.5138 - val_accuracy: 0.7889 - lr: 1.0000e-04
Epoch 15/35
38/38 [==============================] - 5s 135ms/step - loss: 0.2703 -
accuracy: 0.9332 - val_loss: 0.2201 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 16/35
38/38 [==============================] - 5s 138ms/step - loss: 0.2169 -
accuracy: 0.9572 - val_loss: 0.2412 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 17/35
38/38 [==============================] - 5s 133ms/step - loss: 0.2423 -
accuracy: 0.9412 - val_loss: 0.2288 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 18/35
38/38 [==============================] - 5s 138ms/step - loss: 0.3102 -
accuracy: 0.9091 - val_loss: 0.1685 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 19/35
38/38 [==============================] - 5s 136ms/step - loss: 0.2798 -
accuracy: 0.9305 - val_loss: 0.1849 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 20/35
38/38 [==============================] - 5s 139ms/step - loss: 0.2472 -
accuracy: 0.9465 - val_loss: 0.2249 - val_accuracy: 0.9444 - lr: 1.0000e-04
Epoch 21/35
38/38 [==============================] - 5s 137ms/step - loss: 0.2458 -
accuracy: 0.9465 - val_loss: 0.1702 - val_accuracy: 0.9556 - lr: 1.0000e-04
```

```
Epoch 22/35
38/38 [==============================] - 5s 140ms/step - loss: 0.2486 -
accuracy: 0.9278 - val_loss: 0.2283 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 23/35
38/38 [==============================] - 5s 140ms/step - loss: 0.1953 -
accuracy: 0.9439 - val_loss: 0.1958 - val_accuracy: 0.9556 - lr: 1.0000e-04
Epoch 24/35
38/38 [==============================] - 5s 134ms/step - loss: 0.2218 -
accuracy: 0.9465 - val_loss: 0.2485 - val_accuracy: 0.9111 - lr: 1.0000e-04
Epoch 25/35
38/38 [==============================] - 5s 137ms/step - loss: 0.1297 -
accuracy: 0.9786 - val_loss: 0.2162 - val_accuracy: 0.9222 - lr: 1.0000e-04
Epoch 26/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1937 -
accuracy: 0.9492 - val_loss: 0.2602 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 27/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1166 -
accuracy: 0.9840 - val_loss: 0.1687 - val_accuracy: 0.9667 - lr: 1.0000e-04
Epoch 28/35
38/38 [==============================] - ETA: 0s - loss: 0.2057 - accuracy:
0.9545
Epoch 28: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
38/38 [==============================] - 5s 137ms/step - loss: 0.2057 -
accuracy: 0.9545 - val_loss: 0.2308 - val_accuracy: 0.9333 - lr: 1.0000e-04
Epoch 29/35
38/38 [==============================] - 5s 134ms/step - loss: 0.2008 -
accuracy: 0.9412 - val_loss: 0.2219 - val_accuracy: 0.9444 - lr: 3.0000e-05
Epoch 30/35
38/38 [==============================] - 5s 138ms/step - loss: 0.1503 -
accuracy: 0.9545 - val_loss: 0.1811 - val_accuracy: 0.9444 - lr: 3.0000e-05
Epoch 31/35
38/38 [==============================] - 5s 138ms/step - loss: 0.1207 -
accuracy: 0.9706 - val_loss: 0.2094 - val_accuracy: 0.9333 - lr: 3.0000e-05
Epoch 32/35
38/38 [==============================] - 5s 136ms/step - loss: 0.1551 -
accuracy: 0.9706 - val_loss: 0.1776 - val_accuracy: 0.9556 - lr: 3.0000e-05
Epoch 33/35
38/38 [==============================] - 5s 133ms/step - loss: 0.1156 -
accuracy: 0.9786 - val_loss: 0.1790 - val_accuracy: 0.9667 - lr: 3.0000e-05
Epoch 33: early stopping
Validation Accuracy: 0.9533 ± 0.0267
Validation Loss: 0.1599 ± 0.0601
```

[ ]:
```python
import matplotlib.pyplot as plt

# Plot validation accuracies
plt.figure(figsize=(12, 6))
```

```
# Accuracy
plt.subplot(1, 2, 1)
plt.plot(range(1, k + 1), fold_accuracies, marker='o', linestyle='--',␣
 ↪color='b')
plt.title('Validation Accuracy per Fold')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.yticks(np.arange(0, 1.1, 0.05))
plt.grid(True)

# Loss
plt.subplot(1, 2, 2)
plt.plot(range(1, k + 1), fold_losses, marker='o', linestyle='--', color='r')
plt.title('Validation Loss per Fold')
plt.xlabel('Fold')
plt.ylabel('Loss')
plt.grid(True)

# Show plots
plt.tight_layout()
plt.show()
```



C Resnet50 Appendix

Importing Libraries

```
import os

import keras

print("Keras = {}".format(keras.__version__))
import tensorflow as tf

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # or any {'0', '1', '2'}
import matplotlib.pyplot as plt
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
import seaborn as sns
import pandas as pd


# Print gpus
gpus = tf.config.experimental.list_physical_devices('GPU')
print("Num GPUs Available: ", len(gpus))

# model_dir = './models/'
model_file = 'resnet50_brain_mri.keras'
```

Mount google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Unzip dataset

```
! unzip -o /content/drive/MyDrive/dataset_19.zip
! pwd
```

Set directory

```
dir = "dataset_19/"
```

View Data Distribution

```
data_distribution_count = pd.Series(
    {curr_index: len(os.listdir(os.path.join(dir, curr_index))) for curr_index in os.listdir
    (dir)})

data_distribution_count
```

```
fig, axis = plt.subplots(figsize=(13, 5))
axis.grid(True, alpha=0.1)
axis.set_title("Data Distribution Percentage (%)", fontsize=14)
sns.barplot(x=['\n'.join(curr_index.strip().split('_')).title() for curr_index in
    data_distribution_count.index],
            y=100 * data_distribution_count / data_distribution_count.sum(), ax=axis)
axis.set_xlabel("Tumor Class", fontsize=12)
axis.set_ylabel("% Total Observations", fontsize=12)
axis.tick_params(which='major', labelsize=12)
axis.text(2.5, 37, f'Total Observations: {data_distribution_count.sum()}', fontdict=dict(
    size=12))
sns.despine()
```

Preprocess Data

```
from tqdm import tqdm
import cv2
import imutils

def crop_img(img):

```

```
# Find extreme points on the image and crop the rectangular out

gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
gray = cv2.GaussianBlur(gray, (3, 3), 0)

# threshold the image, then perform a series of erosions +
# dilations to remove any small regions of noise
thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
thresh = cv2.erode(thresh, None, iterations=2)
thresh = cv2.dilate(thresh, None, iterations=2)

# find contours in thresholded image, then grab the largest one
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
c = max(cnts, key=cv2.contourArea)

# find the extreme points
extLeft = tuple(c[c[:, :, 0].argmin()][0])
extRight = tuple(c[c[:, :, 0].argmax()][0])
extTop = tuple(c[c[:, :, 1].argmin()][0])
extBot = tuple(c[c[:, :, 1].argmax()][0])
ADD_PIXELS = 0
new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+
    ADD_PIXELS].copy()

return new_img

def preprocess_images(directory):
for dir in os.listdir(directory):
path = os.path.join(directory, dir)
for img_name in os.listdir(path):
img_path = os.path.join(path, img_name)
img = cv2.imread(img_path)
cropped_img = crop_img(img)
processed_img = cv2.cvtColor(cropped_img, cv2.COLOR_RGB2GRAY)
processed_img = cv2.bilateralFilter(processed_img, 2, 50, 50)
processed_img = cv2.applyColorMap(processed_img, cv2.COLORMAP_BONE)
processed_img = cv2.resize(processed_img, (224, 224))
cv2.imwrite(img_path, processed_img)


# Preprocess the images before generating data
preprocess_images(dir)

def display_images(directory, num_images=9):
"""
Display a grid of images from different subdirectories within the specified directory.

Parameters:
- directory (str): The path to the directory containing subdirectories of images.
- num_images (int): The number of images to display.
"""
image_count = 0
plt.figure(figsize=(10, 10))

for subdir in os.listdir(directory):
subdir_path = os.path.join(directory, subdir)
if os.path.isdir(subdir_path):  # Check if it's a directory
for filename in os.listdir(subdir_path):
try:
# Construct the full file path
file_path = os.path.join(subdir_path, filename)
# Load the image
image = plt.imread(file_path)
# Plot the image
plt.subplot(3, 3, image_count + 1)
plt.imshow(image)
```

```
plt.title(subdir)
plt.axis('off')
image_count += 1

# Break if the required number of images are already displayed
if image_count == num_images:
plt.tight_layout()
plt.show()
return
except Exception as e:
print(f"Failed to read or display {filename}: {e}")

plt.tight_layout()
plt.show()

display_images(dir)
```

Split Data

```
classes = os.listdir(dir)

batch_size = 10

train_datagen = ImageDataGenerator(
rescale=1. / 255,
horizontal_flip=True,
rotation_range=20,
width_shift_range=0.05,
height_shift_range=0.05,
validation_split=0.2)

validation_datagen = ImageDataGenerator(rescale=1. / 255,
validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
dir,
target_size=(224, 224),
batch_size=batch_size,
seed=42,
subset='training'
)

test_generator = validation_datagen.flow_from_directory(
dir,
target_size=(224, 224),
batch_size=batch_size,
seed=42,
shuffle = False,
subset='validation')


print(test_generator.class_indices)
```

Construct Model

```
import os
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model, load_model
from keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras.utils import plot_model
from keras.applications import ResNet50
from keras.regularizers import l2
import matplotlib.pyplot as plt


base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,224,3))
```

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.4)(x)
predictions = Dense(4, activation='softmax',)(x)
model = Model(inputs=base_model.input, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics
    =['accuracy'])

# Summary of the model
model.summary()
plot_model(model, to_file='resnet50_model.png', show_shapes=True, show_layer_names=True)
```

Train Model

```
checkpoint = ModelCheckpoint(model_file, monitor='val_loss', verbose=1, save_best_only=True,
     mode='min')
early_stop = EarlyStopping(monitor = 'val_loss',min_delta = 0.001,patience = 20,mode = 'min'
    ,restore_best_weights = True,verbose = 1)

history = model.fit(
train_generator,
steps_per_epoch=train_generator.samples // train_generator.batch_size,
validation_data=test_generator,
validation_steps=test_generator.samples // test_generator.batch_size,
epochs=80,
callbacks=[checkpoint, early_stop]
)
```

Testing Model

```
# Learning curve
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

# Loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation','Loss', 'Value Loss'], loc='upper left')
plt.show()

#Plot the Accuracy Curves
plt.figure(figsize=[8,6])
plt.plot(history.history['accuracy'],'r',linewidth=3.0)
plt.plot(history.history['val_accuracy'],'b',linewidth=3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves',fontsize=16)
plt.show()

model = keras.models.load_model(model_file)
model.evaluate(test_generator)
```

Model Visualization

```
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score

classes = ['glioma', 'meningioma', 'notumor', 'pituitary']


def calculate_metrics(y_true, y_pred):
# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
```

```

# Plot the Confusion Matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=classes, yticklabels=classes)
plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.show()

# Normalize the confusion matrix
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

# Plot the normalized confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm_normalized, annot=True, fmt='.2f', cmap='Blues', xticklabels=classes,
    yticklabels=classes)
plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.title('Normalized Confusion Matrix')
plt.show()

# Calculate metrics for each class and average them
dsc = np.mean([2.0 * cm[i, i] / (np.sum(cm[i, :]) + np.sum(cm[:, i])) for i in range(cm.
    shape[0])])
sensitivity = np.mean([cm[i, i] / np.sum(cm[i, :]) for i in range(cm.shape[0])])
specificity = np.mean([np.sum(np.delete(np.delete(cm, j, 0), j, 1)) / np.sum(np.delete(cm, j
    , 0)) for j in range(cm.shape[0])])

# Accuracy
accuracy = accuracy_score(y_true, y_pred)

return dsc, sensitivity, specificity, accuracy

# Predict the output
predictions_prob = model.predict(test_generator)
predictions = np.argmax(predictions_prob, axis=1)

dsc, sensitivity, specificity, accuracy = calculate_metrics(test_generator.classes,
    predictions)
print(f"DSC: {dsc}, Sensitivity: {sensitivity}, Specificity: {specificity}, Accuracy: {
    accuracy}")
```

```
from sklearn.metrics import roc_curve, auc, classification_report
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt
import numpy as np

# Binarize the output
lb = LabelBinarizer()
y_test = lb.fit_transform(test_generator.classes)
y_pred = lb.transform(predictions)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(len(classes)):
fpr[i], tpr[i], _ = roc_curve(y_test[:, i], predictions_prob[:, i])
roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure()
for i, class_name in enumerate(classes):
plt.plot(fpr[i], tpr[i],
label='ROC curve of class {0} (area = {1:0.2f})'
''.format(class_name, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()

# Print classification report
print(classification_report(y_test, y_pred, target_names=classes))
```

Install optuna

```
!pip install optuna tensorflow-addons
```

Optuna Hyperparameter Tuning

```
import optuna
import os
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model, load_model
from keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras.utils import plot_model
from keras.applications import ResNet50
import matplotlib.pyplot as plt


def create_model(dropout_rate):
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(dropout_rate)(x)
predictions = Dense(4, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics
    =['accuracy'])
return model

def objective(trial):
dropout_rate = trial.suggest_uniform('dropout_rate', 0.1, 0.5)
model = create_model(dropout_rate)

early_stop = EarlyStopping(monitor = 'val_loss',min_delta = 0.001,patience = 20,mode = 'min'
    ,restore_best_weights = True,verbose = 1)

# Model Training
history = model.fit(
train_generator,
validation_data=test_generator,
epochs=35,
callbacks=[early_stop]
)

val_loss = min(history.history['val_loss'])
return val_loss

# Optuna Study
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=5)

# Output the best dropout rate found
print(f'Best dropout rate: {study.best_trial.params["dropout_rate"]}')
```

Stratified K-Fold Cross Validation

```
import os
import numpy as np
```

```
import pandas as pd
from sklearn.model_selection import KFold, StratifiedKFold
from keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
from keras.optimizers import Adam
from keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.applications import ResNet50
from tensorflow import keras
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping

# Set parameters
dir = 'dataset_19'
batch_size = 10
input_shape = (224, 224, 3)
num_classes = 4
initial_learning_rate = 1e-4
k = 10

# Prepare data generators
datagen = ImageDataGenerator(
rescale=1. / 255,
horizontal_flip=True,
rotation_range=20,
width_shift_range=0.05,
height_shift_range=0.05,
)

# List all images and labels
all_images = []
all_labels = []

for class_index, class_name in enumerate(os.listdir(dir)):
class_dir = os.path.join(dir, class_name)
for image_name in os.listdir(class_dir):
all_images.append(os.path.join(class_dir, image_name))
all_labels.append(str(class_index))  # Convert class index to string

# Convert lists to numpy arrays
all_images = np.array(all_images)
all_labels = np.array(all_labels)

# Define K-Fold cross-validation
kf = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)

# Initialize lists to store metrics
fold_accuracies = []
fold_losses = []

# Iterate over each fold
for fold, (train_index, val_index) in enumerate(kf.split(all_images, all_labels)):
print(f'Fold {fold + 1}/{k}')

train_images, val_images = all_images[train_index], all_images[val_index]
train_labels, val_labels = all_labels[train_index], all_labels[val_index]

# Create DataFrames for training and validation data
train_df = pd.DataFrame({'filename': train_images, 'class': train_labels})
val_df = pd.DataFrame({'filename': val_images, 'class': val_labels})

# Create training and validation data generators
train_generator = datagen.flow_from_dataframe(
dataframe=train_df,
x_col='filename',
y_col='class',
target_size=(224, 224),
batch_size=batch_size,
class_mode='categorical',
```

```
shuffle=True
)

validation_generator = datagen.flow_from_dataframe(
dataframe=val_df,
x_col='filename',
y_col='class',
target_size=(224, 224),
batch_size=batch_size,
class_mode='categorical',
shuffle=False
)

base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,224,3))
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.4)(x)
predictions = Dense(4, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics
    =['accuracy'])

early_stop = EarlyStopping(monitor = 'val_loss',min_delta = 0.001,patience = 20,mode = 'min'
    ,restore_best_weights = True,verbose = 1)

history = model.fit(
train_generator,
steps_per_epoch=train_generator.samples // train_generator.batch_size,
validation_data=validation_generator,
validation_steps=validation_generator.samples // validation_generator.batch_size,
epochs=80,
callbacks=[early_stop]
)


# Store the best validation accuracy and loss for the current fold
best_val_accuracy = max(history.history['val_accuracy'])
best_val_loss = min(history.history['val_loss'])
fold_accuracies.append(best_val_accuracy)
fold_losses.append(best_val_loss)

# Calculate mean and standard deviation of accuracies and losses
mean_accuracy = np.mean(fold_accuracies)
std_accuracy = np.std(fold_accuracies)
mean_loss = np.mean(fold_losses)
std_loss = np.std(fold_losses)

print(f'Validation Accuracy: {mean_accuracy:.4f}    {std_accuracy:.4f}')
print(f'Validation Loss: {mean_loss:.4f}    {std_loss:.4f}')
```

Visualize Stratified K-Fold Cross Validation

```
import matplotlib.pyplot as plt

# Plot validation accuracies
plt.figure(figsize=(12, 6))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(range(1, k + 1), fold_accuracies, marker='o', linestyle='--', color='b')
plt.title('Validation Accuracy per Fold')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.yticks(np.arange(0, 1.1, 0.05))
plt.grid(True)

```

```
# Loss
plt.subplot(1, 2, 2)
plt.plot(range(1, k + 1), fold_losses, marker='o', linestyle='--', color='r')
plt.title('Validation Loss per Fold')
plt.xlabel('Fold')
plt.ylabel('Loss')
plt.grid(True)

# Show plots
plt.tight_layout()
plt.show()
```

D InceptionV3 Appendix

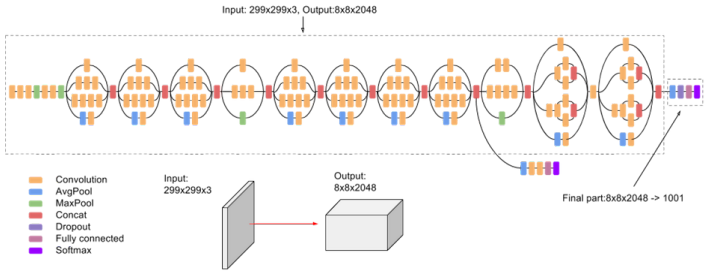


Figure 35: InceptionV3 Architecture

This is the architecture of the InceptionV3 model used in the experiments.

D.1 Jupyter Notebook

```
# Importing Libraries
import os

import keras

print("Keras = {}".format(keras.__version__))
import tensorflow as tf

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # or any {'0', '1', '2'}
import matplotlib.pyplot as plt
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
import seaborn as sns
import pandas as pd


# Print gpus
gpus = tf.config.experimental.list_physical_devices('GPU')
print("Num GPUs Available: ", len(gpus))

model_dir = './models/'
model_file = model_dir + 'inception.keras'

from google.colab import drive
drive.mount('/content/drive')

# Unzip Dataset
```

```
! unzip -o drive/MyDrive/dataset_19.zip

# Data Directories
dir = "dataset_19/"

# Data Distribution
data_distribution_count = pd.Series(
    {curr_index: len(os.listdir(os.path.join(dir, curr_index))) for curr_index in os.listdir
    (dir)})

data_distribution_count

fig, axis = plt.subplots(figsize=(13, 5))
axis.grid(True, alpha=0.1)
axis.set_title("Data Distribution Percentage (%)", fontsize=14)
sns.barplot(x=['\n'.join(curr_index.strip().split('_')).title() for curr_index in
    data_distribution_count.index],
            y=100 * data_distribution_count / data_distribution_count.sum(), ax=axis)
axis.set_xlabel("Tumor Class", fontsize=12)
axis.set_ylabel("% Total Observations", fontsize=12)
axis.tick_params(which='major', labelsize=12)
axis.text(2.5, 37, f'Total Observations: {data_distribution_count.sum()}', fontdict=dict(
    size=12))
sns.despine()
```

```
# Preprocess Data
from tqdm import tqdm
import cv2
import imutils

def crop_img(img):

  # Find extreme points on the image and crop the rectangular out

  gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
  gray = cv2.GaussianBlur(gray, (3, 3), 0)

  # threshold the image, then perform a series of erosions +
  # dilations to remove any small regions of noise
  thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
  thresh = cv2.erode(thresh, None, iterations=2)
  thresh = cv2.dilate(thresh, None, iterations=2)

  # find contours in thresholded image, then grab the largest one
  cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
  cnts = imutils.grab_contours(cnts)
  c = max(cnts, key=cv2.contourArea)

  # find the extreme points
  extLeft = tuple(c[c[:, :, 0].argmin()][0])
  extRight = tuple(c[c[:, :, 0].argmax()][0])
  extTop = tuple(c[c[:, :, 1].argmin()][0])
  extBot = tuple(c[c[:, :, 1].argmax()][0])
  ADD_PIXELS = 0
  new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight
    [0]+ADD_PIXELS].copy()

  return new_img

def preprocess_images(directory):
    for dir in os.listdir(directory):
        path = os.path.join(directory, dir)
        for img_name in os.listdir(path):
            img_path = os.path.join(path, img_name)
            img = cv2.imread(img_path)
            cropped_img = crop_img(img)
            processed_img = cv2.cvtColor(cropped_img, cv2.COLOR_RGB2GRAY)
            processed_img = cv2.bilateralFilter(processed_img, 2, 50, 50)
```

```
            processed_img = cv2.applyColorMap(processed_img, cv2.COLORMAP_BONE)
            processed_img = cv2.resize(processed_img, (224, 224))
            cv2.imwrite(img_path, processed_img)


# Preprocess the images before generating data
preprocess_images(dir)

# Display 9 image using matplotlib
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    for curr_index in os.listdir(dir):
        path = os.path.join(dir, curr_index)
        for img_name in os.listdir(path):
            img_path = os.path.join(path, img_name)
            image = plt.imread(img_path)
            plt.imshow(image)
            plt.title(curr_index)
            break
        break
```

```
# Splitting the Data

classes = os.listdir(dir)

batch_size = 10

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    horizontal_flip=True,
    rotation_range=20,
    width_shift_range=0.05,
    height_shift_range=0.05,
    validation_split=0.2)

validation_datagen = ImageDataGenerator(rescale=1. / 255,
                                        validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    dir,
    target_size=(224, 224),
    batch_size=batch_size,
    seed=42,
    subset='training'
)

test_generator = validation_datagen.flow_from_directory(
    dir,
    target_size=(224, 224),
    batch_size=batch_size,
    seed=42,
    shuffle = False,
    subset='validation')


print(test_generator.class_indices)
```

```
# Model training

## Installing Additional packages
!pip install tensorflow-addons optuna


## Model Creation
from keras.applications import InceptionV3
from tensorflow_addons.optimizers import RectifiedAdam
```

```
from keras.models import Model
from keras.layers import GlobalAveragePooling2D, Dense, Dropout
from keras.regularizers import l2
import tensorflow as tf

# Load the InceptionV3 model, include the top layers
base_model = InceptionV3(weights='imagenet', include_top=False,input_shape=(224,224,3))

# Exclude the bottom three layers (remove last 3 layers)
base_model_layers = base_model.layers[:-3]

# Create a new model with the remaining layers
x = base_model_layers[-1].output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.055)(x)

# Final dense layer with 4 neurons for classification
predictions = Dense(4, activation='softmax', kernel_regularizer=l2(0.1), dtype='float64')(x)

# Construct the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model with RectifiedAdam optimizer
optimizer = RectifiedAdam(learning_rate=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall(),
    'categorical_accuracy'])


# Summary of the model
model.summary()
tf.keras.utils.plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=
    True)

# Model training
from keras.callbacks import *

reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=10, factor=0.3, min_lr=1e-6)
checkpoint = ModelCheckpoint(model_file, monitor='val_loss', verbose=1, save_best_only=True)
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=20, verbose=1, mode
    ='auto')

# Fit the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=100,
    validation_data=test_generator,
    validation_steps=test_generator.samples // test_generator.batch_size,
    callbacks=[checkpoint, early_stopping, reduce_lr]
)

# Test the Model
# Learning curve
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

# Loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation','Loss', 'Value Loss'], loc='upper left')
plt.show()
```

```

model = keras.models.load_model(model_file)
model.evaluate(test_generator)
```

```
# Evaluation
import seaborn as sns
from sklearn.metrics import confusion_matrix, accuracy_score

# {'glioma': 0, 'meningioma': 1, 'notumor': 2, 'pituitary': 3}
classes = ['glioma', 'meningioma', 'notumor', 'pituitary']

def calculate_metrics(y_true, y_pred):
    # Confusion matrix
    cm = confusion_matrix(y_true, y_pred)

    # Plot the Confusion Matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=classes, yticklabels=
    classes)
    plt.xlabel('Predicted Values')
    plt.ylabel('True Values')
    plt.show()

    # Normalize the confusion matrix
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    # Plot the normalized confusion matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm_normalized, annot=True, fmt='.2f', cmap='Blues', xticklabels=classes,
    yticklabels=classes)
    plt.xlabel('Predicted Values')
    plt.ylabel('True Values')
    plt.title('Normalized Confusion Matrix')
    plt.show()

    # Calculate metrics for each class and average them
    dsc = np.mean([2.0 * cm[i, i] / (np.sum(cm[i, :]) + np.sum(cm[:, i])) for i in range(cm.
    shape[0])])
    sensitivity = np.mean([cm[i, i] / np.sum(cm[i, :]) for i in range(cm.shape[0])])
    specificity = np.mean([np.sum(np.delete(np.delete(cm, j, 0), j, 1)) / np.sum(np.delete(
    cm, j, 0)) for j in range(cm.shape[0])])

    # Accuracy
    accuracy = accuracy_score(y_true, y_pred)

    return dsc, sensitivity, specificity, accuracy

# Predict the output
predictions_prob = model.predict(test_generator)
predictions = np.argmax(predictions_prob, axis=1)

dsc, sensitivity, specificity, accuracy = calculate_metrics(test_generator.classes,
    predictions)
print(f"DSC: {dsc}, Sensitivity: {sensitivity}, Specificity: {specificity}, Accuracy: {
    accuracy}")

from sklearn.metrics import roc_curve, auc, classification_report
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt
import numpy as np

# Binarize the output
lb = LabelBinarizer()
y_test = lb.fit_transform(test_generator.classes)
y_pred = lb.transform(predictions)
```

```
# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(len(classes)):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], predictions_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure()
for i, class_name in enumerate(classes):
    plt.plot(fpr[i], tpr[i],
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(class_name, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()

# Print classification report
print(classification_report(y_test, y_pred, target_names=classes))
```

```
# K-Folds Validation

import os
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from tensorflow_addons.optimizers import RectifiedAdam
from keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
from keras.layers import GlobalAveragePooling2D, Dense, Dropout
from keras.regularizers import l2
from keras.models import Model

# Set parameters
dir = 'dataset_19'  # Update this to your dataset directory
batch_size = 10
input_shape = (224, 224, 3)
num_classes = 4
initial_learning_rate = 1e-4
k = 5

# Prepare data generators
datagen = ImageDataGenerator(
    rescale=1. / 255,
    horizontal_flip=True,
    rotation_range=20,
    width_shift_range=0.05,
    height_shift_range=0.05)

# List all images and labels
all_images = []
all_labels = []

for class_index, class_name in enumerate(os.listdir(dir)):
    class_dir = os.path.join(dir, class_name)
    for image_name in os.listdir(class_dir):
        all_images.append(os.path.join(class_dir, image_name))
        all_labels.append(str(class_index))  # Convert class index to string

```

```
# Convert lists to numpy arrays
all_images = np.array(all_images)
all_labels = np.array(all_labels)

# Define K-Fold cross-validation
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize lists to store metrics
fold_accuracies = []
fold_losses = []

# Iterate over each fold
for fold, (train_index, val_index) in enumerate(kf.split(all_images)):
    print(f'Fold {fold + 1}/{k}')

    train_images, val_images = all_images[train_index], all_images[val_index]
    train_labels, val_labels = all_labels[train_index], all_labels[val_index]

    # Create DataFrames for training and validation data
    train_df = pd.DataFrame({'filename': train_images, 'class': train_labels})
    val_df = pd.DataFrame({'filename': val_images, 'class': val_labels})

    # Create training and validation data generators
    train_generator = datagen.flow_from_dataframe(
        dataframe=train_df,
        x_col='filename',
        y_col='class',
        target_size=(224, 224),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=True
    )

    validation_generator = datagen.flow_from_dataframe(
        dataframe=val_df,
        x_col='filename',
        y_col='class',
        target_size=(224, 224),
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

    # Load the InceptionV3 model, include the top layers
    base_model = InceptionV3(weights='imagenet', include_top=False,input_shape=(224,224,3))

    # Exclude the bottom three layers (remove last 3 layers)
    base_model_layers = base_model.layers[:-3]

    # Create a new model with the remaining layers
    x = base_model_layers[-1].output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.055)(x)

    # Final dense layer with 4 neurons for classification
    predictions = Dense(4, activation='softmax', kernel_regularizer=l2(0.1), dtype='float64'
    )(x)

    # Construct the final model
    model = Model(inputs=base_model.input, outputs=predictions)

    # Compile the model with RectifiedAdam optimizer
    optimizer = RectifiedAdam(learning_rate=0.0001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
    model.compile(optimizer=optimizer,
                  loss='categorical_crossentropy',
                  metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall
    (), 'categorical_accuracy'])
```

```
    # Callbacks
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=10, factor=0.3, min_lr=1e-6, verbose=1)
    early_stopping = EarlyStopping(monitor='val_loss', patience=20, verbose=1, mode='auto')

    # Train the model
    history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // train_generator.batch_size,
        epochs=90,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples // validation_generator.batch_size,
        callbacks=[early_stopping, reduce_lr]
    )

    # Store the best validation accuracy and loss for the current fold
    best_val_accuracy = max(history.history['val_accuracy'])
    best_val_loss = min(history.history['val_loss'])
    fold_accuracies.append(best_val_accuracy)
    fold_losses.append(best_val_loss)

# Calculate mean and standard deviation of accuracies and losses
mean_accuracy = np.mean(fold_accuracies)
std_accuracy = np.std(fold_accuracies)
mean_loss = np.mean(fold_losses)
std_loss = np.std(fold_losses)

print(f'Validation Accuracy: {mean_accuracy:.4f}    {std_accuracy:.4f}')
print(f'Validation Loss: {mean_loss:.4f}    {std_loss:.4f}')

import matplotlib.pyplot as plt

# Plot validation accuracies
plt.figure(figsize=(12, 6))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(range(1, k + 1), fold_accuracies, marker='o', linestyle='--', color='b')
plt.title('Validation Accuracy per Fold')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.yticks(np.arange(0, 1.1, 0.05))
plt.grid(True)

# Loss
plt.subplot(1, 2, 2)
plt.plot(range(1, k + 1), fold_losses, marker='o', linestyle='--', color='r')
plt.title('Validation Loss per Fold')
plt.xlabel('Fold')
plt.ylabel('Loss')
plt.grid(True)

# Show plots
plt.tight_layout()
plt.show()
```

E ViT Appendix

Data Preparation

```
    import os
```

```
import keras

print("Keras = {}".format(keras.__version__))
import tensorflow as tf

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'  # or any {'0', '1', '2'}
import matplotlib.pyplot as plt
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
import seaborn as sns
import pandas as pd

# Print gpus
gpus = tf.config.experimental.list_physical_devices('GPU')
print("Num GPUs Available: ", len(gpus))

model_file = 'vit_brain_mri.keras'
```

Mounting Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Unzipping Dataset 19

```
! unzip -o /content/drive/MyDrive/dataset_19.zip

! pwd
```

Load the Data

```
# Data Directories
dir = "dataset_19/"
```

Check Data Distribution

```
data_distribution_count = pd.Series(
    {curr_index: len(os.listdir(os.path.join(dir, curr_index))) for curr_index in os.
listdir(dir)})

data_distribution_count
```

Plotting the Data Distribution

```
fig, axis = plt.subplots(figsize=(13, 5))
axis.grid(True, alpha=0.1)
axis.set_title("Data Distribution Percentage (%)", fontsize=14)
sns.barplot(x=['\n'.join(curr_index.strip().split('_')).title() for curr_index in
data_distribution_count.index],
            y=100 * data_distribution_count / data_distribution_count.sum(), ax=axis)
axis.set_xlabel("Tumor Class", fontsize=12)
axis.set_ylabel("% Total Observations", fontsize=12)
axis.tick_params(which='major', labelsize=12)
axis.text(2.5, 37, f'Total Observations: {data_distribution_count.sum()}', fontdict=dict
(size=12))
sns.despine()
```

Preprocess the Data To check if we can grab the image properly

```
import cv2

def display_images_from_subfolders(directory):
    subfolders = ["glioma", "meningioma", "notumor", "pituitary"]

    for subfolder in subfolders:
        subfolder_path = os.path.join(directory, subfolder)
        for img_name in os.listdir(subfolder_path):
            if img_name.lower().endswith(".jpg"):  # Ensure we are reading .jpg files
                img_path = os.path.join(subfolder_path, img_name)
```

```
                    print(f"Reading image from path: {img_path}")
                    img = cv2.imread(img_path)

                    # Check if the image was successfully loaded
                    if img is not None:
                        # Convert the image from BGR to RGB format
                        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

                        # Display the original image
                        plt.figure()
                        plt.imshow(img_rgb)
                        plt.title(f"Original Image from {subfolder}")
                        plt.axis('off')
                        plt.show()
                        break  # Display one image per subfolder
                    else:
                        print(f"Failed to load image: {img_path}")

# Display images from subfolders
display_images_from_subfolders(dir)
```

Converting Images into Patches
To check for 1 image to see whether it patches correctly back to original image.

```
import numpy as np
from tqdm import tqdm
import imutils
import matplotlib.pyplot as plt

def crop_img(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    gray = cv2.GaussianBlur(gray, (3, 3), 0)
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    if len(cnts) == 0:
        return img  # If no contours are found, return the original image
    c = max(cnts, key=cv2.contourArea)
    extLeft = tuple(c[c[:, :, 0].argmin()][0])
    extRight = tuple(c[c[:, :, 0].argmax()][0])
    extTop = tuple(c[c[:, :, 1].argmin()][0])
    extBot = tuple(c[c[:, :, 1].argmax()][0])
    ADD_PIXELS = 0
    new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:
extRight[0]+ADD_PIXELS].copy()
    return new_img

def divide_into_patches(img, patch_size):
    h, w, _ = img.shape
    patches = []
    for i in range(0, h, patch_size):
        for j in range(0, w, patch_size):
            patch = img[i:i+patch_size, j:j+patch_size]
            if patch.shape[0] == patch_size and patch.shape[1] == patch_size:
                patches.append(patch)
    return np.array(patches)

def save_patches(patches, subfolder, img_name):
    patch_dir = os.path.join("patches", subfolder)
    os.makedirs(patch_dir, exist_ok=True)
    for i, patch in enumerate(patches):
        patch_filename = f"{os.path.splitext(img_name)[0]}_patch_{i}.jpg"
        patch_path = os.path.join(patch_dir, patch_filename)
        cv2.imwrite(patch_path, patch)

def preprocess_images(directory, patch_size):
```

```
        subfolders = ["glioma", "meningioma", "notumor", "pituitary"]
        for subfolder in subfolders:
            subfolder_path = os.path.join(directory, subfolder)
            for img_name in os.listdir(subfolder_path):
                if img_name.lower().endswith(".jpg"):  # Ensure we are reading .jpg files
                    img_path = os.path.join(subfolder_path, img_name)
                    print(f"Reading image from path: {img_path}")
                    img = cv2.imread(img_path)
                    if img is None:
                        print(f"Failed to load image: {img_path}")
                        continue

                    # # Display original image
                    # img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    # plt.figure()
                    # plt.imshow(img_rgb)
                    # plt.title(f"Original Image from {subfolder}")
                    # plt.axis('off')
                    # plt.show()

                    cropped_img = crop_img(img)

                    # # Display cropped image
                    # plt.figure()
                    # plt.imshow(cv2.cvtColor(cropped_img, cv2.COLOR_BGR2RGB))
                    # plt.title("Cropped Image")
                    # plt.axis('off')
                    # plt.show()

                    processed_img = cv2.cvtColor(cropped_img, cv2.COLOR_RGB2GRAY)
                    processed_img = cv2.bilateralFilter(processed_img, 2, 50, 50)
                    processed_img = cv2.applyColorMap(processed_img, cv2.COLORMAP_BONE)
                    processed_img = cv2.resize(processed_img, (224, 224))

                    # Display processed image
                    plt.figure()
                    plt.imshow(cv2.cvtColor(processed_img, cv2.COLOR_BGR2RGB))
                    plt.title("Processed Image")
                    plt.axis('off')
                    plt.show()

                    # Extract patches
                    patches = divide_into_patches(processed_img, patch_size)

                    print(f"Original processed image dimensions: {processed_img.shape}")
                    print(f"Number of patches: {len(patches)}")

                    # Save patches
                    save_patches(patches, subfolder, img_name)
                    print(f"Saved {len(patches)} patches for image {img_name} in {subfolder}
    ")

                    # Display patches
                    plt.figure(figsize=(10, 10))
                    num_patches = len(patches)
                    grid_size = int(np.ceil(np.sqrt(num_patches)))
                    for i, patch in enumerate(patches):
                        plt.subplot(grid_size, grid_size, i + 1)
                        plt.imshow(cv2.cvtColor(patch, cv2.COLOR_BGR2RGB))
                        plt.axis('off')
                    plt.suptitle(f"Patches of {img_name} in {subfolder}")
                    plt.show()

                    # Reconstruct and display the image
                    reconstructed_img = reconstruct_image(patches, processed_img.shape,
    patch_size)
                    plt.figure()
```

```
                    plt.imshow(cv2.cvtColor(reconstructed_img, cv2.COLOR_BGR2RGB))
                    plt.title("Reconstructed Image")
                    plt.axis('off')
                    plt.show()

                    return  # Display patches for the first image only

# Parameters
patch_size = 16  # Since original image is 224 x 224 pixels = total 196 patches
                 # Caclulated from (224/16) * (224/16)

# Preprocess the images before generating data
preprocess_images(dir, patch_size)
```

Image Preprocessing: Cropping, Patching, and Saving MRI Scans by Tumor Class

```
import numpy as np
import matplotlib.pyplot as plt
import imutils

def crop_img(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    gray = cv2.GaussianBlur(gray, (3, 3), 0)
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    if len(cnts) == 0:
        return img  # If no contours are found, return the original image
    c = max(cnts, key=cv2.contourArea)
    extLeft = tuple(c[c[:, :, 0].argmin()][0])
    extRight = tuple(c[c[:, :, 0].argmax()][0])
    extTop = tuple(c[c[:, :, 1].argmin()][0])
    extBot = tuple(c[c[:, :, 1].argmax()][0])
    ADD_PIXELS = 0
    new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:
extRight[0]+ADD_PIXELS].copy()
    return new_img

def divide_into_patches(img, patch_size):
    h, w, _ = img.shape
    patches = []
    for i in range(0, h, patch_size):
        for j in range(0, w, patch_size):
            patch = img[i:i+patch_size, j:j+patch_size]
            if patch.shape[0] == patch_size and patch.shape[1] == patch_size:
                patches.append(patch)
    return np.array(patches)

def save_patches(patches, subfolder, img_name):
    patch_dir = os.path.join("patches", subfolder)
    os.makedirs(patch_dir, exist_ok=True)
    for i, patch in enumerate(patches):
        patch_filename = f"{os.path.splitext(img_name)[0]}_patch_{i}.jpg"
        patch_path = os.path.join(patch_dir, patch_filename)
        cv2.imwrite(patch_path, patch)

def preprocess_images(directory, patch_size):
    subfolders = ["glioma", "meningioma", "notumor", "pituitary"]
    for subfolder in subfolders:
        subfolder_path = os.path.join(directory, subfolder)
        for img_name in os.listdir(subfolder_path):
            if img_name.lower().endswith(".jpg"):  # Ensure we are reading .jpg files
                img_path = os.path.join(subfolder_path, img_name)
                print(f"Reading image from path: {img_path}")
                img = cv2.imread(img_path)
                if img is None:
```

```
                        print(f"Failed to load image: {img_path}")
                        continue

                    cropped_img = crop_img(img)
                    processed_img = cv2.cvtColor(cropped_img, cv2.COLOR_RGB2GRAY)
                    processed_img = cv2.bilateralFilter(processed_img, 2, 50, 50)
                    processed_img = cv2.applyColorMap(processed_img, cv2.COLORMAP_BONE)
                    processed_img = cv2.resize(processed_img, (224, 224))

                    # Extract patches
                    patches = divide_into_patches(processed_img, patch_size)

                    print(f"Original processed image dimensions: {processed_img.shape}")
                    print(f"Number of patches: {len(patches)}")

                    # Save patches
                    save_patches(patches, subfolder, img_name)
                    print(f"Saved {len(patches)} patches for image {img_name} in {subfolder}
")

# Parameters
patch_size = 64  # Modified to patch size of 64 as it was too computationally heavy

# Preprocess the images before generating data
preprocess_images(dir, patch_size)
```

Verify Patch Distribution

```
# Directory where patches are stored
patch_dir = "patches/"
subfolders = ["glioma", "meningioma", "notumor", "pituitary"]

# Count the number of patches in each class
patch_counts = {subfolder: len(os.listdir(os.path.join(patch_dir, subfolder))) for
subfolder in subfolders}
print("Patch distribution:", patch_counts)
```

Splitting the Data
Using original images directly instead of the patched images.

```
classes = os.listdir(dir)
batch_size = 16

train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=20,
    # width_shift_range=0.05,
    # height_shift_range=0.05,
    horizontal_flip=True,
    validation_split=0.2)

validation_datagen = ImageDataGenerator(rescale=1. / 255,
                                        validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    dir,
    target_size=(224, 224),
    batch_size=batch_size,
    seed=42,
    subset='training'
)

test_generator = validation_datagen.flow_from_directory(
    dir,
    target_size=(224, 224),
    batch_size=batch_size,
    seed=42,
    subset='validation',
    shuffle = False)
```

```

print(test_generator.class_indices)

# Get the number of samples in the training and testing data
num_train_samples = train_generator.samples
num_test_samples = test_generator.samples

# Get the distribution of classes in the training and testing data
train_class_distribution = np.bincount(train_generator.classes)
test_class_distribution = np.bincount(test_generator.classes)

# Create a figure with 3 subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plot the train/test split
axes[0].pie([num_train_samples, num_test_samples], labels=['Train', 'Test'], autopct='
%1.1f%%')
axes[0].set_title('Train/Test Split')

# Plot the distribution of classes in the training data
axes[1].pie(train_class_distribution, labels=classes, autopct='%1.1f%%')
axes[1].set_title('Class Distribution in Training Data')

# Plot the distribution of classes in the testing data
axes[2].pie(test_class_distribution, labels=classes, autopct='%1.1f%%')
axes[2].set_title('Class Distribution in Testing Data')

# Display the plots
plt.show()
```

Installing TensorFlow Addon and vit-keras Package

```
! pip install tensorflow-addons vit-keras
! pip install optuna
```

Training a Vision Transformer (ViT) model

```
import optuna
from tensorflow.keras.layers import Input, Dropout, Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy
from vit_keras import vit
from sklearn.utils.class_weight import compute_class_weight
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import optuna
from tensorflow.keras.regularizers import l2

def create_vit_model(input_shape, num_classes, learning_rate, dropout_rate=0.3):
    """
    Create a Vision Transformer (ViT) model with custom classification layers.

    Parameters:
    - input_shape: Tuple specifying the shape of input images (height, width, channels).
    - num_classes: Integer specifying the number of output classes.
    - learning_rate: Float specifying the learning rate for the optimizer.
    - dropout_rate: Float specifying the dropout rate for regularization (default is
0.3).

    Returns:
    - model: Compiled Keras model.
    """
    # Load the pre-trained ViT-B16 model
    vit_model = vit.vit_b16(
        image_size=input_shape[0],  # Image height and width
        activation='softmax',       # Activation function for the final layer
        pretrained=True,            # Use pre-trained weights
        include_top=False,          # Do not include the top classification layer
```

```
            pretrained_top=False,       # Do not include the pre-trained top layer
            classes=num_classes          # Number of output classes
        )

        inputs = Input(shape=input_shape)  # Define the input layer
        x = vit_model(inputs)              # Pass inputs through the ViT model

        # Apply global average pooling if the output has spatial dimensions
        if len(x.shape) == 4:  # Check if output includes spatial dimensions
            x = GlobalAveragePooling2D()(x)  # Apply global average pooling

        x = Dense(512, activation='relu')(x)  # Add a dense layer with 512 units and ReLU activation
        x = Dropout(dropout_rate)(x)           # Apply dropout for regularization
        outputs = Dense(num_classes, activation='softmax')(x)  # Add the output layer with softmax activation

        # Create the Keras model
        model = Model(inputs=inputs, outputs=outputs)
        # Compile the model with Adam optimizer and categorical crossentropy loss
        model.compile(optimizer=Adam(learning_rate),
                      loss=CategoricalCrossentropy(),
                      metrics=['accuracy'])
        return model

    # Define the parameters
    input_shape = (224, 224, 3)  # Shape of the input images
    num_classes = len(classes)   # Number of classes in the dataset
    learning_rate = 0.0001107117449413457  # Learning rate obtained from Optuna optimization
    dropout_rate = 0.32755569499826637     # Dropout rate obtained from Optuna optimization

    # Create the ViT model with the specified parameters
    model = create_vit_model(input_shape, num_classes, learning_rate, dropout_rate)

    # Print the model summary
    model.summary()

    # Setup callbacks for model training
    checkpoint = ModelCheckpoint(model_file, monitor='val_loss', save_best_only=True, verbose=1)
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=7, min_lr=1e-6)

    # Train the model using the training and validation data generators
    history = model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // train_generator.batch_size,
        epochs=50,  # Number of training epochs
        validation_data=test_generator,
        validation_steps=test_generator.samples // test_generator.batch_size,
        callbacks=[checkpoint, early_stopping, reduce_lr],  # Use the defined callbacks
    )
```

Plot the Model Architecture

```
    tf.keras.utils.plot_model(model, to_file='vit_model_architecture.png', show_shapes=True, show_layer_names=True)
```

Plotting the Learning Curve of the Model

```
    # Learning curve
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

# Loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('Model accuracy and loss')
```

```
plt.ylabel('Accuracy/Loss')
plt.xlabel('Epoch')
plt.legend(['Train Accuracy', 'Validation Accuracy', 'Train Loss', 'Validation Loss'], loc='
    upper left')
plt.show()
```

Plot Training VS Validation Accuracy

```
    # Plotting training and validation accuracy
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Model Visualisation (Evaluation)
Confusion Matrix

```
    import seaborn as sns
    from sklearn.metrics import confusion_matrix, accuracy_score

    # Define the class labels
    classes = ['glioma', 'meningioma', 'notumor', 'pituitary']

    def calculate_metrics(y_true, y_pred):
        """
        Calculate and display various evaluation metrics based on true and predicted labels.

        Parameters:
        - y_true: Array of true labels.
        - y_pred: Array of predicted labels.

        Returns:
        - dsc: Dice similarity coefficient (mean across classes).
        - sensitivity: Mean sensitivity across classes.
        - specificity: Mean specificity across classes.
        - accuracy: Overall accuracy.
        """
        # Compute the confusion matrix
        cm = confusion_matrix(y_true, y_pred)

        # Plot the confusion matrix
        plt.figure(figsize=(10, 8))
        sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=classes, yticklabels=
    classes)
        plt.xlabel('Predicted Values')
        plt.ylabel('True Values')
        plt.title('Confusion Matrix')
        plt.show()

        # Normalize the confusion matrix by row (true labels)
        cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

        # Plot the normalized confusion matrix
        plt.figure(figsize=(10, 8))
        sns.heatmap(cm_normalized, annot=True, fmt='.2f', cmap='Blues', xticklabels=classes,
     yticklabels=classes)
        plt.xlabel('Predicted Values')
        plt.ylabel('True Values')
        plt.title('Normalized Confusion Matrix')
        plt.show()

        # Calculate the Dice similarity coefficient (DSC) for each class and average them
        dsc = np.mean([2.0 * cm[i, i] / (np.sum(cm[i, :]) + np.sum(cm[:, i])) for i in range
    (cm.shape[0])])

```

```
        # Calculate the sensitivity (recall) for each class and average them
        sensitivity = np.mean([cm[i, i] / np.sum(cm[i, :]) for i in range(cm.shape[0])])

        # Calculate the specificity for each class and average them
        specificity = np.mean([np.sum(np.delete(np.delete(cm, j, 0), j, 1)) / np.sum(np.delete(cm, j, 0)) for j in range(cm.shape[0])])

        # Calculate the overall accuracy
        accuracy = accuracy_score(y_true, y_pred)

        return dsc, sensitivity, specificity, accuracy

    # Predict the class probabilities for the test dataset
    predictions_prob = model.predict(test_generator)

    # Convert probabilities to class labels
    predictions = np.argmax(predictions_prob, axis=1)

    # Calculate the evaluation metrics
    dsc, sensitivity, specificity, accuracy = calculate_metrics(test_generator.classes, predictions)

    # Print the evaluation metrics
    print(f"DSC: {dsc}, Sensitivity: {sensitivity}, Specificity: {specificity}, Accuracy: {accuracy}")
```

ROC Curve

```
    from sklearn.metrics import roc_curve, auc, classification_report
    from sklearn.preprocessing import LabelBinarizer
    import matplotlib.pyplot as plt
    import numpy as np

    # Initialize the LabelBinarizer
    lb = LabelBinarizer()

    # Binarize the true class labels for multi-class ROC AUC computation
    y_test = lb.fit_transform(test_generator.classes)

    # Binarize the predicted class labels
    y_pred = lb.transform(predictions)

    # Initialize dictionaries to store false positive rates, true positive rates, and AUC for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # Compute ROC curve and ROC area (AUC) for each class
    for i in range(len(classes)):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i], predictions_prob[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plot all ROC curves for each class
    plt.figure()
    for i, class_name in enumerate(classes):
        plt.plot(fpr[i], tpr[i],
                 label='ROC curve of class {0} (area = {1:0.2f})'
                 ''.format(class_name, roc_auc[i]))

    # Plot the diagonal line for random performance
    plt.plot([0, 1], [0, 1], 'k--')

    # Set plot limits
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])

    # Set plot labels and title
```

```
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) for Multi-Class Classification')

    # Add legend to the plot
    plt.legend(loc="lower right")

    # Display the plot
    plt.show()

    # Print the classification report for all classes
    print(classification_report(y_test, y_pred, target_names=classes))
```

Segmentation with patched images (Failed Attempt)

```
    # Define the image and patch sizes
    image_size = 224
    patch_size = 64

    # Create the ImageDataGenerator for normalizing the patches
    patch_datagen = ImageDataGenerator(rescale=1. / 255)

    # Directory containing the original images
    original_image_dir = "dataset_19"

    # Function to load and predict patches
    def load_and_predict_patches(patch_dir, model, classes):
        """
        Load image patches and predict their classes using the trained model.

        Parameters:
        - patch_dir: Directory containing image patches.
        - model: Trained model to predict patches.
        - classes: List of class names.

        Returns:
        - predictions: Array of model predictions for each patch.
        - filenames: List of filenames for the patches.
        """
        predictions = []
        filenames = []
        for subfolder in classes:
            subfolder_path = os.path.join(patch_dir, subfolder)
            for img_name in os.listdir(subfolder_path):
                if img_name.lower().endswith(".jpg"):
                    img_path = os.path.join(subfolder_path, img_name)
                    img = cv2.imread(img_path)
                    if img is None:
                        continue
                    # Resize patch to the model's input size
                    resized_img = cv2.resize(img, (224, 224))
                    resized_img = np.expand_dims(resized_img, axis=0) / 255.0
                    pred = model.predict(resized_img)
                    predictions.append(pred)
                    filenames.append(img_path)
        return np.array(predictions), filenames

    # Predict the patches
    predictions, filenames = load_and_predict_patches(patch_dir, model, classes)

    # Function to create segmentation masks from predictions
    def create_segmentation_masks(predictions, filenames, patch_size, image_size, threshold
    =0.5):
        """
        Create segmentation masks from model predictions.

        Parameters:
        - predictions: Array of model predictions for each patch.
```

```
        - filenames: List of filenames for the patches.
        - patch_size: Size of each patch.
        - image_size: Size of the original image.
        - threshold: Threshold for classifying a patch as an area of interest.

        Returns:
        - masks: List of segmentation masks for each original image.
        """
        num_patches = (image_size // patch_size) ** 2
        masks = []
        for i in range(0, len(predictions), num_patches):
            patch_preds = predictions[i:i + num_patches]
            mask = np.zeros((image_size, image_size))
            for j, pred in enumerate(patch_preds):
                y = (j // (image_size // patch_size)) * patch_size
                x = (j % (image_size // patch_size)) * patch_size
                if pred[0][np.argmax(pred)] > threshold:
                    mask[y:y + patch_size, x:x + patch_size] = 1  # Mark as area of interest
            masks.append(mask)
        return masks

    # Create segmentation masks
    segmentation_masks = create_segmentation_masks(predictions, filenames, patch_size,
    image_size)

    # Function to visualize the segmentation results
    def visualize_segmentation(filenames, masks, classes):
        """
        Visualize the segmentation results by overlaying masks on the original images.

        Parameters:
        - filenames: List of filenames for the patches.
        - masks: List of segmentation masks for each original image.
        - classes: List of class names.
        """
        plt.figure(figsize=(12, 6 * len(classes) * 2))
        displayed_classes_counts = {class_name: 0 for class_name in classes}
        num_images_per_class = 2
        image_index = 0

        for i, (img_path, mask) in enumerate(zip(filenames, masks)):
            # Extract class name from the patch file path
            actual_class_name = os.path.basename(os.path.dirname(img_path))
            if displayed_classes_counts[actual_class_name] >= num_images_per_class:
                continue

            # Construct path to the original image
            original_image_name = os.path.basename(img_path).split('_patch_')[0] + '.jpg'
            original_image_path = os.path.join(original_image_dir, actual_class_name,
    original_image_name)
            if not os.path.exists(original_image_path):
                print(f"File does not exist: {original_image_path}")
                continue

            original_image = cv2.imread(original_image_path)
            if original_image is None:
                print(f"Failed to load image: {original_image_path}")
                continue

            original_image_rgb = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)

            # Overlay the segmentation mask on the original image
            overlay = original_image_rgb.copy()
            mask_resized = cv2.resize(mask, (original_image_rgb.shape[1], original_image_rgb
    .shape[0]))  # Resize mask to match the original image size
            overlay[mask_resized > 0] = (0, 255, 0)  # Overlay mask in green color

```

```
            plt.subplot(len(classes) * num_images_per_class, 2, 2 * image_index + 1)
            plt.imshow(original_image_rgb)
            plt.title(f'Original: {actual_class_name}')
            plt.axis('off')

            plt.subplot(len(classes) * num_images_per_class, 2, 2 * image_index + 2)
            plt.imshow(overlay)
            plt.title(f'Segmentation')
            plt.axis('off')

            displayed_classes_counts[actual_class_name] += 1
            image_index += 1

            if all(count >= num_images_per_class for count in displayed_classes_counts.
    values()):
                break

        plt.tight_layout()
        plt.show()

    # Ensure that the model and other functions are defined above this code
    # Predict the patches
    predictions, filenames = load_and_predict_patches(patch_dir, model, classes)

    # Process predictions to create segmentation masks
    segmentation_masks = create_segmentation_masks(predictions, filenames, patch_size,
    image_size)

    # Visualize the segmentation results
    visualize_segmentation(filenames, segmentation_masks, classes)
```

K-Folds Validation

```
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow import keras
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping

# Set parameters
dir = 'dataset_19'
batch_size = 16
input_shape = (224, 224, 3)
num_classes = 4
initial_learning_rate = 1e-4
k = 5  # Number of folds for cross-validation

# Prepare data generators with data augmentation
datagen = ImageDataGenerator(
    rescale=1. / 255,       # Rescale pixel values to [0, 1]
    horizontal_flip=True,   # Randomly flip images horizontally
    rotation_range=20       # Randomly rotate images
)

# List all images and their corresponding labels
all_images = []
all_labels = []

# Iterate through each class folder to list images and assign labels
for class_index, class_name in enumerate(os.listdir(dir)):
    class_dir = os.path.join(dir, class_name)
    for image_name in os.listdir(class_dir):
        all_images.append(os.path.join(class_dir, image_name))
        all_labels.append(str(class_index))  # Convert class index to string

# Convert lists to numpy arrays
```

```
    all_images = np.array(all_images)
    all_labels = np.array(all_labels)

    # Define K-Fold cross-validation
    kf = KFold(n_splits=k, shuffle=True, random_state=42)

    # Initialize lists to store metrics for each fold
    fold_accuracies = []
    fold_losses = []

    # Iterate over each fold
    for fold, (train_index, val_index) in enumerate(kf.split(all_images)):
        print(f'Fold {fold + 1}/{k}')

        # Split data into training and validation sets
        train_images, val_images = all_images[train_index], all_images[val_index]
        train_labels, val_labels = all_labels[train_index], all_labels[val_index]

        # Create DataFrames for training and validation data
        train_df = pd.DataFrame({'filename': train_images, 'class': train_labels})
        val_df = pd.DataFrame({'filename': val_images, 'class': val_labels})

        # Create training and validation data generators
        train_generator = datagen.flow_from_dataframe(
            dataframe=train_df,
            x_col='filename',
            y_col='class',
            target_size=(224, 224),
            batch_size=batch_size,
            class_mode='categorical',
            shuffle=True
        )

        validation_generator = datagen.flow_from_dataframe(
            dataframe=val_df,
            x_col='filename',
            y_col='class',
            target_size=(224, 224),
            batch_size=batch_size,
            class_mode='categorical',
            shuffle=False
        )

        # Define and compile the ViT Model
        model = create_vit_model(input_shape, num_classes, initial_learning_rate,
    dropout_rate)
        optimizer = keras.optimizers.Adam(learning_rate=initial_learning_rate)
        model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['
    accuracy'])

        # Callbacks for reducing learning rate, early stopping, and model checkpointing
        reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=10, factor=0.3, min_lr=1e
    -6, verbose=1)
        # checkpoint = ModelCheckpoint(f'model_fold_{fold + 1}.h5', monitor='val_loss',
    verbose=1, save_best_only=True)
        early_stopping = EarlyStopping(monitor='val_loss', patience=15, verbose=1, mode='
    auto')

        # Train the model
        history = model.fit(
            train_generator,
            steps_per_epoch=train_generator.samples // train_generator.batch_size,
            epochs=20,
            validation_data=validation_generator,
            validation_steps=validation_generator.samples // validation_generator.batch_size
    ,
            callbacks=[early_stopping, reduce_lr]
```

```
        )

        # Store the best validation accuracy and loss for the current fold
        best_val_accuracy = max(history.history['val_accuracy'])
        best_val_loss = min(history.history['val_loss'])
        fold_accuracies.append(best_val_accuracy)
        fold_losses.append(best_val_loss)

    # Calculate mean and standard deviation of accuracies and losses across all folds
    mean_accuracy = np.mean(fold_accuracies)
    std_accuracy = np.std(fold_accuracies)
    mean_loss = np.mean(fold_losses)
    std_loss = np.std(fold_losses)

    # Print the final validation accuracy and loss
    print(f'Validation Accuracy: {mean_accuracy:.4f}    {std_accuracy:.4f}')
    print(f'Validation Loss: {mean_loss:.4f}    {std_loss:.4f}')
```

Plotting the K-Folds

```
import matplotlib.pyplot as plt

# Plot validation accuracies
plt.figure(figsize=(12, 6))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(range(1, k + 1), fold_accuracies, marker='o', linestyle='--', color='b')
plt.title('Validation Accuracy per Fold')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.yticks(np.arange(0, 1.1, 0.05))
plt.grid(True)

# Loss
plt.subplot(1, 2, 2)
plt.plot(range(1, k + 1), fold_losses, marker='o', linestyle='--', color='r')
plt.title('Validation Loss per Fold')
plt.xlabel('Fold')
plt.ylabel('Loss')
plt.grid(True)

# Show plots
plt.tight_layout()
plt.show()
```

References

[1] M. Nazar, M. M. Alam, E. Yafi, and M. M. Su'ud, "A systematic review of human–computer interaction and explainable artificial intelligence in healthcare with artificial intelligence techniques," *IEEE Access*, vol. 9, pp. 153 316–153 348, 2021, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3127881. [Online]. Available: https://ieeexplore.ieee.org/document/9614151 (visited on 06/08/2024).

[2] P. Doupe, J. Faghmous, and S. Basu, "Machine learning for health services researchers," *Value in Health*, vol. 22, no. 7, pp. 808–815, Jul. 1, 2019, Publisher: Elsevier, ISSN: 1098-3015, 1524-4733. DOI: 10.1016/j.jval.2019.02.012. [Online]. Available: https://www.valueinhealthjournal.com/article/S1098-3015(19)30146-9/fulltext?_returnURL=https%3A%2F%2Flinkinghub.elsevier.com%2Fretrieve%2Fpii%2FS1098301519301469%3Fshowall%3Dtrue (visited on 06/08/2024).

[3] D. Rolnick, P. L. Donti, L. H. Kaack, *et al.*, "Tackling climate change with machine learning," *ACM Computing Surveys*, vol. 55, no. 2, 42:1–42:96, Feb. 7, 2022, ISSN: 0360-0300. DOI: 10.1145/3485128. [Online]. Available: https://dl.acm.org/doi/10.1145/3485128 (visited on 06/08/2024).

[4] S. Lapointe, A. Perry, and N. A. Butowski, "Primary brain tumours in adults," *The Lancet*, vol. 392, no. 10145, pp. 432–446, Aug. 4, 2018, ISSN: 0140-6736. DOI: 10.1016/S0140-6736(18)30990-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140673618309905 (visited on 06/02/2024).

[5] J. B. Iorgulescu, M. Torre, M. Harary, *et al.*, "The misclassification of diffuse gliomas: Rates and outcomes," *Clinical Cancer Research*, vol. 25, no. 8, pp. 2656–2663, Apr. 15, 2019, ISSN: 1078-0432. DOI: 10.1158/1078-0432.CCR-18-3101. [Online]. Available: https://doi.org/10.1158/1078-0432.CCR-18-3101 (visited on 06/08/2024).

[6] L. Lenchik, L. Heacock, A. A. Weaver, *et al.*, "Automated segmentation of tissues using CT and MRI: A systematic review," *Academic Radiology*, vol. 26, no. 12, pp. 1695–1706, Dec. 1, 2019, Publisher: Elsevier, ISSN: 1076-6332, 1878-4046. DOI: 10.1016/j.acra.2019.07.006. [Online]. Available: https://www.academicradiology.org/article/S1076-6332(19)30353-8/abstract (visited on 06/08/2024).

[7] D. L. Krishna, N. V. Dedeepya Padmanabhuni, and G. JayaLakshmi, "Data augmentation based brain tumor detection using cnn and deep learning," in *2023 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)*, 2023, pp. 317–321. DOI: 10.1109/CISES58720.2023.10183465.

[8] J. S. Paul, A. Plassard, B. Landman, and D. Fabbri, "Deep learning for brain tumor classification," vol. 10137, 2017. DOI: 10.1117/12.2254195.

[9] S. Asif, W. Yi, Q. Ain, J. Hou, T. Yi, and J. Si, "Improving effectiveness of different deep transfer learning-based models for detecting brain tumors from mr images," *IEEE Access*, vol. 10, pp. 34 716–34 730, 2022. DOI: 10.1109/ACCESS.2022.3153306.

[10] F. Fahimi, S. Došen, K. Ang, N. Mrachacz-Kersting, and C. Guan, "Generative adversarial networks-based data augmentation for brain–computer interface," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 4039–4051, 2020. DOI: 10.1109/tnnls.2020.3016666.

[11] M. A. Mahmutoglu, C. J. Preetha, H. Meredig, *et al.*, "Deep learning–based identification of brain mri sequences using a model trained on large multicentric study cohorts," *Radiology: Artificial Intelligence*, vol. 6, no. 1, e230095, 2024, PMID: 38166331. DOI: 10.1148/ryai.230095. eprint: https://doi.org/10.1148/ryai.230095. [Online]. Available: https://doi.org/10.1148/ryai.230095.

[12] R. Imtiaz, M. W. Mirza, A. Siddiq, M. Farooq-i-Azam, I. R. Khan, and S. Rahardja, "Brain tumor segmentation from MR images using customized u-net for a smaller dataset," in *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, ISSN: 2766-4465, Oct. 2023, pp. 1–5. DOI: 10.1109/BioCAS58349.2023.10389092. [Online]. Available: https://ieeexplore.ieee.org/document/10389092 (visited on 05/28/2024).

[13] M. K. Abd-Ellah, A. I. Awad, A. A. M. Khalaf, and A. M. Ibraheem, "Automatic brain-tumor diagnosis using cascaded deep convolutional neural networks with symmetric u-net and asymmetric residual-blocks," *Scientific Reports*, vol. 14, no. 1, p. 9501, Apr. 25, 2024, ISSN: 2045-2322. DOI: 10.1038/s41598-024-59566-7. [Online]. Available: https://www.nature.com/articles/s41598-024-59566-7 (visited on 06/02/2024).

[14] H. Ding, J. Lu, J. Cai, Y. Zhang, and Y. Shang, "SLf-UNet: Improved UNet for brain MRI segmentation by combining spatial and low-frequency domain features," in *Advances in Computer Graphics*, B. Sheng, L. Bi, J. Kim, N. Magnenat-Thalmann, and D. Thalmann, Eds., Cham: Springer Nature Switzerland, 2024, pp. 415–426, ISBN: 978-3-031-50075-6. DOI: 10.1007/978-3-031-50075-6_32.

[15] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, Springer, 2018, pp. 3–11.

[16] B. Baheti, S. Innani, S. Gajre, and S. Talbar, "Eff-UNet: A novel architecture for semantic segmentation in unstructured environment," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 1473–1481, ISBN: 978-1-72819-360-1. DOI: 10.1109/CVPRW50498.2020.00187. [Online]. Available: https://ieeexplore.ieee.org/document/9150621/ (visited on 06/16/2024).

[17] Z. Yu, "Pneumonia detection with u-EfficientNet," in *2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC)*, Oct. 2021, pp. 591–594. DOI: 10.1109/DSC53577.2021.00093. [Online]. Available: https://ieeexplore.ieee.org/document/9750509 (visited on 06/16/2024).

[18] S.-Y. Lin and C.-L. Lin, "Brain tumor segmentation using u-net in conjunction with EfficientNet," *PeerJ Computer Science*, vol. 10, e1754, Jan. 2, 2024, ISSN: 2376-5992. DOI: 10.7717/peerj-cs.1754. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10773611/ (visited on 06/16/2024).

[19] M. M. Islam, P. Barua, M. Rahman, T. Ahammed, L. Akter, and J. Uddin, "Transfer learning architectures with fine-tuning for brain tumor classification using magnetic resonance imaging," *Healthcare Analytics*, vol. 4, p. 100 270, Dec. 2023. DOI: 10.1016/j.health.2023.100270.

[20] A. W. Thomas, K.-R. Müller, and W. Samek, "Deep transfer learning for whole-brain fmri analyses," in *OR 2.0 Context-Aware Operating Theaters and Machine Learning in Clinical Neuroimaging*, L. Zhou, D. Sarikaya, S. M. Kia, *et al.*, Eds., Cham: Springer International Publishing, 2019, pp. 59–67, ISBN: 978-3-030-32695-1.

[21] A. B. Mitta, A. H. Hegde, A. R. K. P., and G. S., "Brain tumor detection: An application based on transfer learning," in *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*, 2023, pp. 1424–1430. DOI: 10.1109/ICOEI56765.2023.10125766.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv (Cornell University)*, Jan. 2015. DOI: 10.48550/arxiv.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385.

[23] J. Cheng, W. Huang, S. Cao, *et al.*, "Enhanced performance of brain tumor classification via tumor region augmentation and partition," *PloS One*, vol. 10, no. 10, e0140381, Oct. 2015. DOI: 10.1371/journal.pone.0140381. [Online]. Available: https://doi.org/10.1371/journal.pone.0140381.

[24] O. O. Oladimeji and A. O. J. Ibitoye, "Brain tumor classification using resnet50-convolutional block attention module," *Applied Computing and Informatics*, Dec. 2023. DOI: 10.1108/aci-09-2023-0022. [Online]. Available: https://doi.org/10.1108/aci-09-2023-0022.

[25] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016. arXiv: 1610.02357. [Online]. Available: http://arxiv.org/abs/1610.02357.

[26] S. Li, H. Qu, X. Dong, B. Dang, H. Zang, and Y. Gong, *Leveraging deep learning and xception architecture for high-accuracy mri classification in alzheimer diagnosis*, 2024. arXiv: 2403.16212 [id='eess.IV' $full_name =' Image and Video Processing' is_active = True alt_name = None in_archive =' eess' is_general = False description =' Theory, algorithms, and architectures for the formation, capture, processing, communication, analysis, and display of images, vid mathematical, statistical, and perceptual image and video modeling and representation; linear and nonlinear filtering, de– blurring, enhancement, restoration, and reconstruction from degraded, low–resolution or tomographic data; lossless and lossy com

[27] G. Kaur, N. Sharma, D. Upadhyay, M. Singh, and R. Gupta, "Brain tumour disease detection in mri images using xception model," in *2024 3rd International Conference for Innovation in Technology (INOCON)*, 2024, pp. 1–5. DOI: 10.1109/INOCON60754.2024.10511504.

[28] D. Paranjpe, S. Vikram, S. Sonawane, and A. Deshpande, "Brain mri classification using transfer learning techniques," in *2023 International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*, 2023, pp. 507–512. DOI: 10.1109/ICSSAS57918.2023.10331713.

[29] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, Jan. 2017. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2017/html/Huang_Densely_Connected_Convolutional_CVPR_2017_paper.html.

[30] P. Rajpurkar, J. Irvin, K. Zhu, *et al.*, *Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning*, Dec. 2017. [Online]. Available: https://arxiv.org/abs/1711.05225.

[31] A. Esteva, B. Kuprel, R. A. Novoa, *et al.*, *Dermatologist-level classification of skin cancer with deep neural networks*, Jan. 2017. [Online]. Available: https://www.nature.com/articles/nature21056.

[32] C. Yang, Z. Yang, A. M. Khattak, *et al.*, "Structured pruning of convolutional neural networks via l1 regularization," *IEEE Access*, vol. 7, pp. 106 385–106 394, 2019. DOI: 10.1109/ACCESS.2019.2933032.

[33] J. D. Lanlan Liu, *Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution*, Apr. 2023. [Online]. Available: https://aaai.org/papers/11630-dynamic-deep-neural-networks-optimizing-accuracy-efficiency-trade-offs-by-selective-execution/.

[34] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[35] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [id='cs.CV' full$_name$=' ComputerVisionandPatternRecognition'is$_active$= True alt$_name$= None in$_archive$=' cs'is$_general$= False description=' Cover sim age processing, computer vision, pattern recognitio

[36] S. Tummala, S. Kadry, S. A. C. Bukhari, and H. T. Rauf, "Classification of brain tumor from magnetic resonance imaging using vision transformers ensembling," *Current Oncology*, vol. Toronto, Ont. Oct. 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9600395/.

[37] A. A. Asiri, A. Shaf, T. Ali, *et al.*, "Advancing brain tumor classification through fine-tuned vision transformers: A comparative study of pre-trained models," *Sensors (Basel, Switzerland)*, vol. 23, 2023. DOI: 10.3390/s23187913.

[38] A. Diker, "A performance comparison of pre-trained deep learning models to classify brain tumor," *IEEE EUROCON 2021 - 19th International Conference on Smart Technologies*, pp. 246–249, 2021. DOI: 10.1109/EUROCON52738.2021.9535636.

[39] F. A. Jibon, M. U. Khandaker, M. H. Miraz, *et al.*, "Cancerous and non-cancerous brain mri classification method based on convolutional neural network and log-polar transformation," *Healthcare*, vol. 10, no. 9, 2022, ISSN: 2227-9032. DOI: 10.3390/healthcare10091801. [Online]. Available: https://www.mdpi.com/2227-9032/10/9/1801.

[40] Z. Rasheed, Y.-K. Ma, I. Ullah, *et al.*, "Brain tumor classification from mri using image enhancement and convolutional neural network techniques," *Brain Sciences*, vol. 13, no. 9, 2023, ISSN: 2076-3425. DOI: 10.3390/brainsci13091320. [Online]. Available: https://www.mdpi.com/2076-3425/13/9/1320.

[41] B. B. Vimala, S. Srinivasan, S. K. Mathivanan, N. Mahalakshmi, P. Jayagopal, and G. T. Dalu, "Detection and classification of brain tumor using hybrid deep learning models," *Scientific Reports*, vol. 13, no. 1, Dec. 2023. DOI: 10.1038/s41598-023-50505-6. [Online]. Available: https://www.nature.com/articles/s41598-023-50505-6#citeas.

[42] S. Krishnapriya and Y. Karuna, "Pre-trained deep learning models for brain mri image classification," *Frontiers in Human Neuroscience*, vol. 17, 2023, ISSN: 1662-5161. DOI: 10.3389/fnhum.2023.1150120. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnhum.2023.1150120.

[43] A. Kujur, Z. Raza, A. A. Khan, and C. Wechtaisong, "Data complexity based evaluation of the model dependence of brain mri images for classification of brain tumor and alzheimer's disease," *IEEE Access*, vol. 10, pp. 112 117–112 133, 2022. DOI: 10.1109/ACCESS.2022.3216393.

[44] J. Nalepa, M. Marcinkiewicz, and M. Kawulok, "Data augmentation for brain-tumor segmentation: A review," *Frontiers in Computational Neuroscience*, vol. 13, Dec. 11, 2019, ISSN: 1662-5188. DOI: 10.3389/fncom.2019.00083. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncom.2019.00083 (visited on 06/02/2024).

[45] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, May 18, 2015. DOI: 10.48550/arXiv.1505.04597. arXiv: 1505.04597[cs]. [Online]. Available: http://arxiv.org/abs/1505.04597 (visited on 06/08/2024).

[46] P. Iakubovskii, *Segmentation models*, https://github.com/qubvel/segmentation_models, 2019.

[47] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, *UNet++: A nested u-net architecture for medical image segmentation*, Jul. 18, 2018. DOI: 10.48550/arXiv.1807.10165. arXiv: 1807.10165[cs,eess,stat]. [Online]. Available: http://arxiv.org/abs/1807.10165 (visited on 06/08/2024).

[48] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: Redesigning skip connections to exploit multiscale features in image segmentation," *IEEE Transactions on Medical Imaging*, 2019.

[49] Z. Zhou, "Towards annotation-efficient deep learning for computer-aided diagnosis," Ph.D. dissertation, Arizona State University, 2021.

[50] W. Hastomo, A. Satyo, E. Sestri, *et al.*, "Classification of brain image tumor using EfficientNet b1-b2 deep learning," *Semesta Teknika*, vol. 27, pp. 46–54, May 2, 2024. DOI: 10.18196/st.v27i1.19691.

[51] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, Dec. 11, 2015. DOI: 10.48550/arXiv.1512.00567. arXiv: 1512.00567[cs]. [Online]. Available: http://arxiv.org/abs/1512.00567 (visited on 06/06/2024).

[52] L. Liu, H. Jiang, P. He, *et al.*, *On the variance of the adaptive learning rate and beyond*, version: 1, Aug. 8, 2019. DOI: 10.48550/arXiv.1908.03265. arXiv: 1908.03265[cs,stat]. [Online]. Available: http://arxiv.org/abs/1908.03265 (visited on 06/06/2024).

[53] M. Z. Khaliki and M. S. Başarslan, "Brain tumor detection from images and comparison with transfer learning methods and 3-layer CNN," *Scientific Reports*, vol. 14, no. 1, p. 2664, Feb. 1, 2024, Publisher: Nature Publishing Group, ISSN: 2045-2322. DOI: 10.1038/s41598-024-52823-9. [Online]. Available: https://www.nature.com/articles/s41598-024-52823-9 (visited on 06/05/2024).

[54] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[55] K. Morani, E. K. Ayana, and D. Unay, "Covid-19 detection using modified xception transfer learning approach from computed tomography images," *International Journal of Advances in Intelligent Informatics*, vol. 9, no. 3, pp. 524–536, 2023, ISSN: 2548-3161. DOI: 10.26555/ijain.v9i3.1432. [Online]. Available: https://ijain.org/index.php/IJAIN/article/view/1432.

[56] B. Bulut, V. Kalin, B. B. Güneş, and R. Khazhin, "Deep learning approach for detection of retinal abnormalities based on color fundus images," in *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2020, pp. 1–6. DOI: 10.1109/ASYU50717.2020.9259870.

[57] S. H. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, pp. 1–41, 2021. DOI: 10.1145/3505244.

[58] B. Wu, C. Xu, X. Dai, *et al.*, "Visual transformers: Token-based image representation and processing for computer vision," *ArXiv*, 2020.

[59] S. Tummala, S. Kadry, S. A. C. Bukhari, and H. T. Rauf, "Classification of brain tumor from magnetic resonance imaging using vision transformers ensembling," *Current Oncology*, vol. 29, no. 10, pp. 7498–7511, 2022, ISSN: 1718-7729. DOI: 10.3390/curroncol29100590. [Online]. Available: https://www.mdpi.com/1718-7729/29/10/590.

[60] C. Matsoukas, J. F. Haslum, M. P. Soderberg, and K. Smith, "Is it time to replace cnns with transformers for medical images?" *ArXiv*, vol. abs/2108.09038, 2021.

[61] E. Simon and A. Briassouli, "Vision transformers for brain tumor classification," pp. 123–130, 2022. DOI: 10.5220/0010834300003123.

[62] Y. Wang, R. Huang, S. Song, Z. Huang, and G. Huang, "Not all images are worth 16x16$Words$: Dynamic vision transformers with adaptive sequence length," *ArXiv*, vol. abs/2105.15075, 2021.

[63] S. Al-Hadhrami, M. Menai, S. Al-ahmadi, and A. Alnafessah, "An effective med-vqa method using a transformer with weights fusion of multiple fine-tuned models," *Applied Sciences*, 2023. DOI: 10.3390/app13179735.

[64] O. R. U. Brawijaya, O. Rochmawanti, U. Brawijaya, F. U. U. Brawijaya, F. Utaminingrum, and O. M. A. Metrics, *Chest x-ray image to classify lung diseases in different resolution size using densenet-121 architectures: Proceedings of the 6th international conference on sustainable information engineering and technology*, Sep. 2021. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3479645.3479667.

[65] S. B. Thunuguntla, S. Murugaanandam, and R. Pitchai, *Densenet121-dnn-based hybrid approach*, Jan. 2023. [Online]. Available: https://inass.org/wp-content/uploads/2023/01/2023063013-2.pdf.

[66] G. Tanner, *Radam - rectified adam*, Aug. 2021. [Online]. Available: https://ml-explained.com/blog/radam-explained.

[67] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, Apr. 10, 2015. arXiv: 1409.1556[cs]. [Online]. Available: http://arxiv.org/abs/1409.1556 (visited on 06/12/2024).