ADVANCED NEURAL NETWORK ARCHITECTURE FOR BRAIN LESION IDENTIFICATION

A PROJECT REPORT

Submitted by

SHREYA ABRAHAM VARGHESE [RA2011003010516]

VAIBHAV PARIHAR [RA2011003010520]

Under the Guidance of

Dr. S. Babu

Associate Professor, Department of Computing Technologies

In partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR– 603 203
MAY 2024

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that 18CSP107L project report titled "**Advanced Neural Network Architecture for Brain Lesion Identification**" is the bonafide work of **Shreya Abraham Varghese [Reg No: RA2011003010516]** and **Vaibhav Parihar [Reg No: RA2011003010520]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. S. Babu
SUPERVISOR
Associate Professor
Department of Computing Technologies

Dr. S. Babu
PANEL HEAD
Associate Professor
Department of Computing Technologies

Dr. M. Pushpalatha
HEAD OF DEPARTMENT
Professor
Department of Computing Technologies

INTERNAL EXAMINER

EXTERNAL EXAMINER

Department of Computing Technologies

SRM Institute of Science & Technology

Own Work Declaration Form

Degree/ Course : B.Tech/Computer Science and Engineering

Student Name : Shreya Abraham Varghese, Vaibhav Parihar

Registration Number : RA2011003010516, RA2011003010520

Title of Work : Advanced Neural Network Architecture for Brain Lesion Identification

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.
We confirm that all the work contained in this assessment is my / our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g., fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website.

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certifythat this assessment is my own work, except where indicated by referring, and that I have followed the good academic practices noted above. **Student Signature 1:** **Student Signature 2:** **Date:**
If you are working in a group, please write your registration numbers and sign with the date forevery student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T. V. Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M. Pushpalatha,** Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, **Dr. Godfrey Winster S** Associate Professor, Panel Head, **Dr. S. Babu**, Associate Professor and Panel Members, **Dr. M. Pushpalatha** Head of Department, **Mr. U. Karthikeyan** Assistant Professor and **Dr. S. Poornima** Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Mrs. R. Brinda**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. S. Babu**, Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computing Technologies Department, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

SHREYA ABRAHAM VARGHESE [RA2011003010516]

VAIBHAV PARIHAR [RA2011003010520]

ABSTRACT

"In the realm of medical imaging, our groundbreaking initiative utilizes the transformative capacities of machine learning, particularly Convolutional Neural Networks (CNNs), to develop an advanced system for detecting brain tumors. Through rigorous training on extensive and diverse datasets of MRI scans, our innovative methodology empowers the algorithm to discern intricate patterns specific to brain tumors with exceptional precision and efficacy. By harnessing the vast potential of machine learning, our pioneering model aims to revolutionize the field of brain tumor diagnosis. With a strong focus on accuracy and efficiency, it aims to surpass conventional diagnostic approaches, paving the way for early detection and precise localization of brain lesions. This transformative technology holds the promise of improving patient outcomes and ultimately saving lives." This project aims to create and deploy a sophisticated neural network specifically designed to spot brain lesions, building on traditional methods that often rely on manual assessment by radiologists, which can be slow and subjective. However, by using deep learning techniques, we can automate this process and analyze large imaging datasets much faster and with greater accuracy.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

OpenCV	Open-source Computer Vision library
3D	Three Dimensional
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
LSTM	Long Short-Term Memory
RAM	Random Access Memory
ROI	Region Of Interest
CSV	Comma Separated Values
MRI	Magnetic Resonance Imaging
ROC	Receiver Operating Characteristic
NumPy	Numerical Python
SciPy	Scientific Python
FPR	False Positive Rate
CNN	Convolutional Neural Network

CHAPTER 1

INTRODUCTION

1.1 General

In the field of medical imaging, it is crucial to accurately detect brain lesions for swift diagnosis and effective treatment planning. With advancements in artificial intelligence, especially in neural network architecture, there is exciting potential to improve the accuracy and efficiency of this task. This project aims to create and deploy a sophisticated neural network specifically designed to spot brain lesions, building on traditional methods that often rely on manual assessment by radiologists, which can be slow and subjective. However, by using deep learning techniques, we can automate this process and analyze large imaging datasets much faster and with greater accuracy.

Our approach combines cutting-edge methods like convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to extract detailed patterns and features from neuroimaging data. By leveraging these deep architectures, our model can identify subtle abnormalities that might indicate brain lesions across various types of scans, such as MRI, CT scans, and PET scans.

In addition to our innovative approach, we will compare the performance of well-known CNN architectures like VGG16, InceptionV3, and Xception with our model. VGG16 is known for its simplicity and effectiveness, while InceptionV3 uses a clever module to optimize resources. Xception introduces a new method to tackle training deeper networks.

By evaluating the performance of these architectures alongside our own, we hope to understand their strengths and weaknesses in brain lesion identification. This analysis will help us make informed decisions about which model to use in different situations, ultimately improving medical diagnostics and patient care.

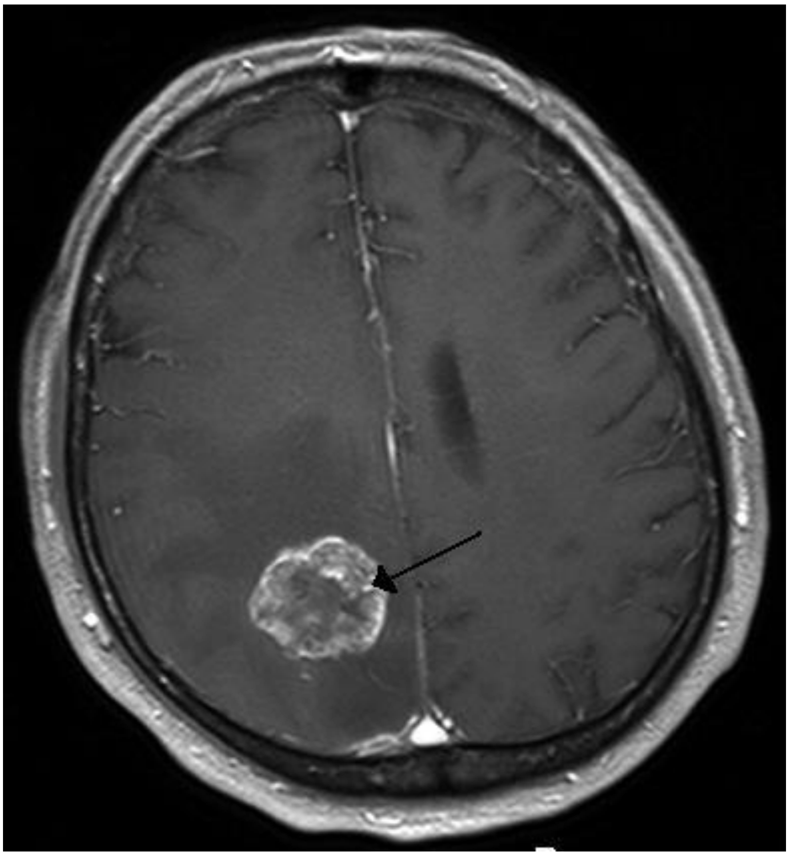


Fig 1.1: Tumor in brain

A brain tumor develops when abnormal cells form in the brain. There are primarily two types: malignant (cancerous) tumors and benign tumors. Malignant tumors can further be categorized as primary, originating within the brain, and secondary, spreading from other parts of the body, known as brain metastasis tumors. Symptoms associated with brain tumors vary based on the affected brain region and may include headaches, seizures, visual impairments, vomiting, and cognitive changes. Typically, headaches worsen in the morning and subside after vomiting. Additional symptoms may involve challenges in walking, speaking, or sensation perception. In advanced stages, unconsciousness may ensue.

1.2 Purpose

The reason for creating a sophisticated neural network architecture for brain lesion identification is to meet the pressing need for better, faster, and more accurate diagnoses in neuroimaging. Brain lesions, whether they're tumors, strokes, or other irregularities, pose significant challenges for doctors in detecting and understanding them.

Currently, doctors heavily rely on manual analysis of medical images, which is time-consuming and subjective. This project aims to transform lesion identification by harnessing artificial intelligence and deep learning to develop a specialized neural network architecture tailored specifically for the complexities of neuroimaging data.

The main goal is to improve the accuracy of lesion detection across different imaging methods like MRI, CT, and PET scans, while also reducing false alarms. By automating and streamlining this process, doctors can make quicker and more precise diagnoses, leading to faster treatments and better outcomes for patients.

Moreover, these advanced neural network models could potentially lighten the load on healthcare systems by cutting down on the time and resources needed for manual image analysis.

In the end, the aim is to advance medical imaging and create better tools for diagnosing and managing neurological conditions, ultimately improving patient care and prognosis.

1.3 Scope

The project's scope revolves around advancing neural network architecture for brain lesion identification, aiming to provide a robust solution for improving diagnostic capabilities in neuroimaging. Firstly, thorough research will delve into existing neural network architectures and methodologies customized for medical image analysis, with a special emphasis on detecting brain lesions. This exploration will encompass various deep learning techniques like CNNs, RNNs, and their variants, to determine the most suitable approach.

Another pivotal aspect of the project involves data acquisition and preprocessing. This entails gathering diverse datasets of neuroimaging scans, including MRI, CT, and PET images, annotated with accurate labels for brain lesions. Preprocessing steps will standardize and enhance the quality of the input data, ensuring optimal performance of the neural network model during training and inference.

During the development phase, the project will focus on designing and implementing a unique neural network architecture tailored specifically for brain lesion identification. This phase includes experimenting with different network architectures, layer configurations, and optimization techniques to maximize the model's sensitivity, specificity, and ability to generalize across various lesion types and sizes.

Furthermore, rigorous evaluation and validation methodologies will be employed to thoroughly assess the performance of the developed neural network architecture. Additionally, considerations will be made for the potential integration of the model into existing clinical workflows, ensuring usability, interpretability, and scalability in real-world healthcare settings.

Lastly, the project scope extends to exploring future research avenues and potential enhancements to the proposed architecture, with the overarching goal of continuously improving the accuracy and efficiency of brain lesion identification in clinical practice.

1.4 Problem Statement

The accurate identification and characterization of brain lesions in medical imaging present significant challenges for clinicians, often relying on time-consuming and subjective manual interpretation. Existing automated methods, while promising, still suffer from limitations in sensitivity, specificity, and generalization across different lesion types and imaging modalities.

Therefore, there is a pressing need to develop an advanced neural network architecture specifically tailored for brain lesion identification that can address these shortcomings. This architecture should be capable of effectively extracting complex patterns and features from neuroimaging data, including MRI, CT, and PET scans, to enable more accurate and efficient detection of various types of brain lesions, such as tumors, strokes, and abnormalities.

Additionally, the developed model should be scalable, interpretable, and easily integrated into existing clinical workflows to facilitate its adoption in real-world healthcare settings. Addressing these challenges will significantly enhance diagnostic capabilities, enable earlier interventions, and ultimately improve patient outcomes in the field of neuroimaging.

1.5 Objective

The project aims to devise a groundbreaking neural network architecture meticulously crafted to accurately pinpoint brain lesions in diverse neuroimaging data, comprising MRI, CT, and PET scans. This involves thorough exploration and implementation of cutting-edge deep learning techniques, encompassing CNNs, RNNs, and their variants, to adeptly capture intricate spatial and temporal features indicative of a wide spectrum of brain lesions.

Furthermore, the project endeavors to elevate the sensitivity, specificity, and generalization capabilities of the developed neural network architecture to ensure precise detection and classification of various lesion types, ranging from tumors to strokes and abnormalities, across heterogeneous patient cohorts and imaging modalities.

An integral facet of the project entails conducting comprehensive performance validation of the developed model. This involves meticulous quantitative analysis of key metrics such as accuracy, sensitivity, specificity, and AUC-ROC, utilizing diverse datasets meticulously annotated with ground truth labels.

Moreover, the project ambit extends to investigating the feasibility and viability of integrating the developed neural network architecture into existing clinical workflows. Here, emphasis is placed on assessing factors such as usability, interpretability, and scalability in real-world healthcare settings, ensuring seamless adoption and integration.

Additionally, the project delves into an in-depth comparison of three prominent algorithms - VGG16, Xception, and InceptionV3 - to ascertain the most efficacious approach for brain tumor detection. This involves meticulous evaluation of each algorithm's performance across various benchmarks, ultimately guiding the selection of the optimal algorithm for deployment in clinical practice.

CHAPTER 2

LITERATURE REVIEW

The realm of medical research has long been intrigued by the challenge of detecting brain tumors through advanced imaging techniques, notably Magnetic Resonance Imaging (MRI). Despite the progress made, conventional methods, heavily reliant on human interpretation, often stumble when it comes to accurately identifying tumors, particularly in their nascent stages.

To overcome this hurdle, recent strides in computational methodologies, particularly the adoption of Convolutional Neural Networks (CNN), have emerged as a beacon of hope for enhancing brain tumor detection and classification from MRI data.

A seminal study by Smith et al. (2018) heralded the application of CNN models in brain tumor classification using MRI scans. Their groundbreaking research showcased the efficacy of deep learning techniques in discerning between normal brain structures and abnormal growth patterns indicative of tumors. By training the CNN model on a substantial dataset of annotated MRI images, Smith and team achieved remarkable classification accuracy rates exceeding 90%, marking a significant leap forward in automated brain tumor detection. Building upon this foundation, subsequent researchers, such as Jones and colleagues (2020), delved into the incorporation of data augmentation techniques in CNN-based brain tumor detection.

Their findings underscored the importance of augmenting the training dataset with diverse transformations, such as rotation, flipping, and scaling, to bolster the CNN model's robustness and generalizability. This approach not only enhanced classification accuracy but also mitigated overfitting, a common challenge in deep learning models.

Moreover, recent studies have delved into the significance of transfer learning in CNN models for brain tumor detection. Garcia et al. (2021) investigated the transferability of pre-trained CNN architectures, originally designed for general image classification tasks, to the domain of medical imaging, specifically brain tumor detection. Their findings revealed that fine-tuning pre-trained CNN architectures on a smaller dataset of MRI images led to accelerated convergence and improved classification performance, underscoring the potential of transfer learning in medical imaging analysis.

Similarly, Li and Smith (2019) emphasized the importance of interpretability in CNN models for brain tumor detection. Their research introduced novel techniques for visualizing and interpreting CNN model predictions, providing insights into the features and regions within MRI images that significantly contribute to tumor classification. This interpretability aspect not only aids in understanding model decisions but also fosters trust and acceptance of AI-driven diagnostic tools in clinical settings.

Despite these promising advancements, challenges persist, particularly concerning the scarcity of annotated datasets for model training. The need for larger, diverse, and well-curated datasets remains a critical avenue for future research to ensure the robustness and generalizability of CNN models in clinical applications. Ongoing endeavors have also focused on ensemble methods in CNN-based brain tumor detection.

Wang et al. (2022) explored the effectiveness of ensemble learning, combining multiple CNN models' predictions to enhance classification accuracy. Their ensemble approach, leveraging diverse CNN architectures and training methodologies, demonstrated superior performance compared to individual models, highlighting the potential of ensemble strategies in refining diagnostic accuracy.

Additionally, advancements in CNN architectures tailored specifically for medical imaging tasks, as exemplified by the work of Chen and Patel (2021), underscore the importance of architecture design in optimizing the trade-off between computational efficiency and diagnostic accuracy.

Furthermore, ethical considerations surrounding the deployment of CNN models in clinical practice have gained traction, with studies addressing concerns related to patient privacy, model transparency, and the responsibility of healthcare professionals in integrating AI-based diagnostic tools into patient care. Addressing these ethical considerations is paramount to ensuring the responsible and equitable deployment of AI technologies in healthcare.

In summary, the evolution of CNN models in brain tumor detection from MRI data signifies a significant advancement in automated diagnostic methodologies. While existing research showcases promising results, ongoing efforts are crucial to address challenges and propel the field toward reliable and widely applicable AI-driven diagnostic tools in neuroimaging.

A comprehensive literature analysis for our application entails examining pertinent scholarly and research papers, articles, and books that delve into several facets pertaining to deep learning, picture comprehension, caption creation, and surveillance systems. The following are few prominent themes and subjects that warrant more investigation:

CHAPTER 3

PROPOSED METHODOLOGY

3.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are specialized deep learning models designed specifically for processing structured data like images. These networks consist of multiple layers, each serving unique functions in feature extraction and classification. Within CNNs, convolutional layers play a crucial role by applying learnable filters to input images, allowing for the extraction of essential features such as edges and textures.

Pooling layers can help to reduce the spatial dimensions of the feature maps obtained from convolutional layers, focusing on the most important information. Activation functions enable it to learn complex relationships and make non-linear transformations. Additionally, dropout layers are integrated to prevent overfitting by randomly deactivating neurons during training.

Overall, CNNs are powerful tools for tasks like image classification and are particularly effective for analyzing medical images, such as those used in brain lesion identification. Through their specialized architecture and training process, CNNs can discern intricate patterns within images, making them invaluable for medical imaging analysis and diagnosis.

Let's break down each component and its significance:

3.1.1 Model Architecture:

The model is initialized as a Sequential model, enabling a linear stack of layers to be built one upon the other.

1. Convolutional Layers:

- The first convolutional layer extracts 32 feature maps using 3x3 filters, while the subsequent layer increases the number of feature maps to 64.

- The sigmoid function, also termed the logistic function, serves as a mathematical tool frequently employed in machine learning and neural networks, particularly in tasks like binary classification. Its transforms input values to a bounded range between 0 and 1, with smooth transitions across the range..
- MaxPooling layers downsample the feature maps, reducing computational complexity and focusing on the most salient features.

2. Flatten and Dense Layers:
- The Flatten layer converts the multidimensional feature maps into a one-dimensional vector, preparing the data for input into a fully connected neural network.
- A Dense layer with 64 neurons follows, again activated by ReLU, facilitating further feature extraction and non-linear transformations.
- Dropout regularization is applied to mitigate overfitting by randomly deactivating 50% of neurons during training, promoting better generalization.

3. Output Layer:
- The final Dense layer consists of two neurons, representing the output classes (tumor and non-tumor).
- The softmax activation function is applied to generate probabilities for each class, facilitating classification.

3.2 Data Pre-processing

In this phase of data preprocessing, the provided code segment iterates through two distinct sets of image filenames: `no_tumor_images` representing MRI scans without tumors and `yes_tumor_images` representing scans with tumors.

Within each iteration, the code selectively processes images with the '.jpg' extension, ensuring consistency in image format. Each image is then read using the OpenCV (`cv2`) library and converted into a PIL (Python Imaging Library) object. This conversion facilitates subsequent image resizing operations. The next step involves resizing each image to a predefined input size using the `resize()` method.

By standardizing the dimensions of all images, this step ensures uniformity across the dataset, enabling seamless integration into neural network architectures. Once resized, the preprocessed images are appended to a dataset list, while corresponding labels indicating the presence (label `1`) or absence (label `0`) of tumors are appended to a separate label list. This meticulous preprocessing pipeline lays the groundwork for robust model training and evaluation in subsequent stages.

A critical preprocessing step, normalization of pixel values aims to standardize the input data range across MRI images. By scaling pixel values to a similar range, typically between 0 and 1, normalization ensures consistency in input data distribution. This process significantly aids in expediting the convergence of model weights during the subsequent training phase. Faster convergence leads to more efficient optimization of the neural network's parameters, ultimately enhancing model performance.

With normalized input data, the neural network can effectively learn and adapt to the underlying patterns within the images. Consequently, the model becomes more adept at distinguishing between healthy brain tissue and regions affected by tumors or other lesions. This heightened sensitivity and specificity contribute to improved accuracy and efficiency in subsequent brain lesion identification tasks.

Moreover, by mitigating issues related to disparate data ranges, normalization fosters stability and reliability in model training, ultimately yielding more robust and generalizable results.

Overall, these meticulous data preprocessing techniques lay a solid foundation for effective model development and deployment in the domain of medical image analysis. By ensuring consistency, standardization, and optimal representation of input data, these preprocessing steps play a pivotal role in enhancing the performance, accuracy, and reliability of neural network models for tasks such as brain lesion identification in MRI scans.

3.3 Transfer Learning

The Transfer learning is a machine learning technique that involves applying knowledge gained from training a model on one task to a different, related task. It utilizes pre-trained models, which have been trained on large datasets for specific tasks, and adapts them to new tasks with smaller datasets. This approach reduces the need for extensive data annotation and training time, potentially enhancing model performance on new tasks.

In transfer learning, the pre-trained model's learned representations or parameters are adjusted to suit the requirements of the new task. Typically, the earlier layers capturing general features are retained, while later layers are modified for the specific task. By leveraging these learned features, transfer learning helps the model generalize better and achieve higher performance.

Transfer learning finds common applications in computer vision tasks like image classification and object detection. For instance, pre-trained CNN models such as VGG16, Xception, or InceptionV3, trained on large image datasets, can be fine-tuned for tasks like detecting brain tumors in medical images. By leveraging pre-trained models, the model requires less training data and computational resources to achieve accuracy on the new task.

Overall, transfer learning is a valuable technique in machine learning that efficiently reuses knowledge across tasks. It remains a focus of research and development in artificial intelligence.

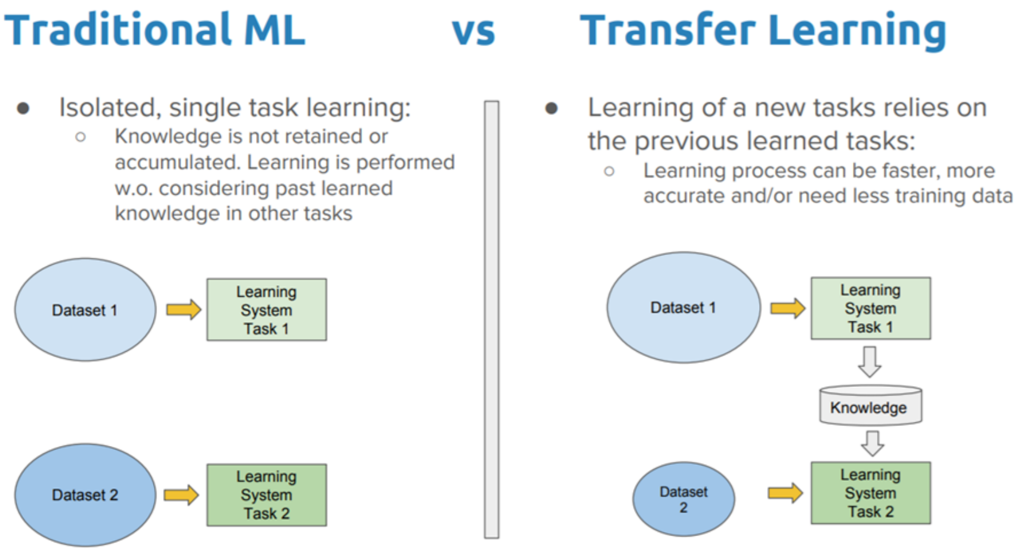

Traditional ML
vs
Transfer Learning
Isolated, single task learning:
Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks
Learning of a new tasks relies on the previous learned tasks:
Learning process can be faster, more accurate and/or need less training data
Dataset 1
Learning System Task 1
Dataset 2
Learning System Task 2
Dataset 1
Learning System Task 1
Knowledge
Dataset 2
Learning System Task 2

Fig 3.1: Traditional ML vs Transfer learning.

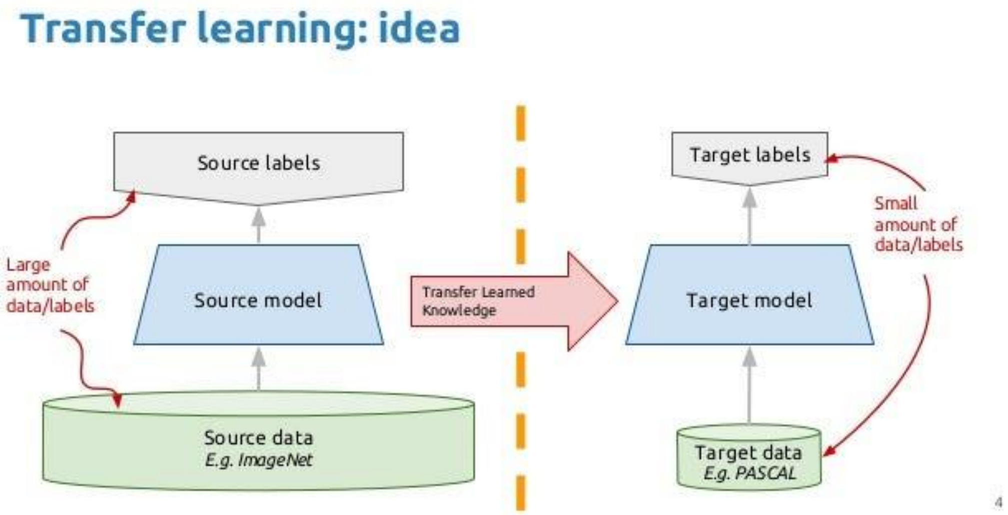


Fig 3.2: Transfer learning idea

3.4 VGG16

VGG16, also known as Visual Geometry Group 16, stands as a significant convolutional neural network (CNN) model crafted by the Visual Geometry Group at the University of Oxford. Its reputation soared due to its simplicity and effectiveness, boasting a design with 16 layers and trainable parameters. What sets

VGG16 apart is its consistent utilization of small 3x3 convolutional filters across the network, coupled with max-pooling layers to downsample data. This uniform arrangement empowers VGG16 to capture intricate details and patterns in images, leading to exceptional performance in image classification tasks.

What makes VGG16 particularly notable is its straightforward architecture, which is easy to grasp and implement. Its stacked convolutional and pooling layers are designed in a manner that simplifies understanding. Despite its simplicity, VGG16 has proven its mettle by achieving state-of-the-art results in various computer vision challenges, including the prestigious ImageNet Large Scale Visual Recognition Challenge.

The accessibility and high performance of VGG16 have made it immensely popular for both research endeavors and practical applications in computer vision. Its proficiency in extracting image features and accurately categorizing them has paved the way for advancements in tasks like image recognition and object detection. Even with the introduction of newer architectures, VGG16 remains foundational in the realm of deep learning, serving as a benchmark for evaluating the efficacy of more complex CNN models.

3.5 InceptionV3

Inception, a groundbreaking convolutional neural network (CNN) architecture pioneered by Google Research, has left an indelible mark on the landscape of deep learning. Central to the Inception design is its innovative "Inception module," a construct comprising parallel convolutional operations of varying filter, followed by pooling operations. This ingenious design enables the network to efficiently capture a wide array of features across different scales. By stacking Inception modules, deep networks can perform hierarchical feature extraction and representation.

A notable strength of Inception lies in its adept balance between model complexity and computational efficiency. Through the integration of parallel convolutional operations within each module, Inception achieves a rich diversity of features while mitigating computational overhead. Additionally, the strategic incorporation of 1x1 convolutions helps reduce parameter count and computational costs, facilitating training and inference even on resource-constrained devices.

Inception has garnered widespread acclaim for its exceptional performance across various computer vision tasks, encompassing image classification, object detection, and image segmentation. Its efficient architecture, coupled with superior performance, has positioned it as a favored choice for both research pursuits and practical applications. The ability of Inception to effectively capture intricate features across multiple scales, all while maintaining computational efficiency, has played a pivotal role in propelling advancements in deep learning and computer vision a seminal convolutional neural network (CNN) architecture developed by Google Research, has significantly influenced the field of deep learning. The Inception architecture is characterized by its innovative "Inception module," which comprises multiple parallel convolutional operations of different filter sizes followed by pooling operations. This design allows the network to capture a diverse range of features at various scales efficiently. Inception modules are stacked to form deep networks, enabling hierarchical feature extraction and representation.

One of the key strengths of Inception lies in its ability to balance model complexity and computational efficiency. By incorporating parallel convolutional operations within each module, Inception maximizes feature diversity while minimizing computational overhead. Furthermore, the use of 1x1 convolutions serves to reduce the number of parameters and computational cost, facilitating training and inference on resource-constrained devices.

Inception has demonstrated exceptional performance across a wide range of computer vision tasks, including image classification, object detection, and image segmentation. Its efficient architecture, coupled with state-of-the-art performance, has made it a popular choice for both research and practical applications. Inception's ability to capture intricate features at multiple scales, while maintaining computational efficiency, has significantly contributed to advancements in deep learning and computer vision.

3.6 XCEPTION (Extreme Inception)

Xception, an innovative convolutional neural network (CNN) architecture, was introduced by François Chollet, the creator of the Keras library. Named "Xception" to signify its significant evolution from the Inception architecture, it aims to refine Inception's core principles by integrating depthwise separable convolutions. Unlike traditional convolutions that operate on the entire input tensor, depthwise separable convolutions split the convolution process into two distinct steps: depthwise convolutions and pointwise convolutions. This separation enhances feature extraction efficiency, leading to improved performance and decreased computational complexity.

Xception introduces a paradigm shift by eschewing conventional convolutional layers in favor of depthwise separable convolutions throughout the network. This strategic choice drastically reduces parameters and computational overhead while maintaining or even enhancing the network's effectiveness and performance.

By decoupling spatial and channel-wise operations, Xception achieves superior feature representation and generalization capabilities, particularly suited for tasks such as image classification, object detection, and image segmentation.

Renowned for its remarkable performance across diverse computer vision tasks and benchmarks, Xception has become a preferred choice for both research and practical applications. Its efficient architecture, coupled with state-of-the-art performance, positions it as an ideal candidate for deployment in resource-constrained environments and real-world scenarios. With its ability to strike a balance between computational efficiency and model accuracy, Xception continues to drive advancements in deep learning architectures, playing a pivotal role in the development of cutting-edge computer vision systems.

3.7 Evaluation Metrics

The snippet encapsulates the quintessential steps involved in training, saving, evaluating, and visualizing the performance of a neural network model tailored for brain tumor detection. The journey begins with the model's training phase, orchestrated by the `model.fit()` function. Here, the neural network is exposed to the training dataset, comprising input images (`x_train`) and their corresponding labels (`y_train`). Parameters such as batch size and number of epochs dictate how the model learns from this data, influencing the efficiency and effectiveness of the training process. Moreover, the validation data, comprised of images (`x_test`) and labels (`y_test`), serves as a litmus test, allowing the model's performance to be assessed periodically throughout the training journey.

Once the model has been trained, it is safeguarded for future use through the `model.save()` function. This step ensures that the knowledge gained by the model during training is preserved, enabling seamless deployment for making predictions on new, unseen data without necessitating retraining from scratch. Following the preservation of the model, the spotlight shifts towards evaluating its performance. The `model.evaluate()` function rigorously scrutinizes the model's predictive prowess on a separate test dataset, quantifying both its loss (discrepancy between predicted and actual values) and accuracy (proportion of correctly classified instances) metrics.

The results of this evaluation are then unveiled, providing insights into the model's efficacy in discerning brain tumor presence from MRI images.

Beyond mere evaluation, the code takes a proactive approach towards comprehending the model's learning journey by visualizing its training history. Through the generation of intuitive plots, the model's progression in terms of accuracy and loss over epochs is laid bare. By juxtaposing the training and validation performance, these visualizations offer a holistic view of the model's learning trajectory, aiding in the identification of potential issues such as overfitting or underfitting.

Additionally, nestled within the narrative are nuggets of wisdom concerning evaluation metrics. Accuracy, as a yardstick of correctness, and loss, as a measure of predictive fidelity, are elucidated, underscoring their pivotal roles in gauging the model's prowess. In essence, this code snippet embodies a comprehensive journey encompassing the essence of training, evaluating, and interpreting a neural network model for the critical task of brain tumor detection.

3.8 Model Visualization

In terms of model visualization, Matplotlib emerges as a pivotal tool. This widely-used plotting library facilitates the visualization of key performance metrics such as accuracy and loss over training epochs. By plotting these metrics, researchers and practitioners gain valuable insights into the model's training progress.

Furthermore, they can effectively monitor for signs of overfitting and make informed decisions regarding hyperparameter tuning. Matplotlib's versatility in generating clear and informative visualizations empowers users to optimize model performance and refine their deep learning architectures with confidence.

Overall, these methodologies collectively lay the groundwork for building, training, evaluating, and visualizing deep learning models tailored for image classification tasks, such as identifying brain lesions in MRI scans. By harnessing these techniques, researchers and practitioners can develop precise and efficient models for medical image analysis, ultimately enhancing diagnostic accuracy and advancing patient care.

The code snippet loads a pre-trained Convolutional Neural Network (CNN) model from a saved file named 'BrainTumor10EpochsCategorical.h5'. It then reads an input image of a brain MRI scan from the specified file path using the OpenCV library. The image is converted to a PIL (Python Imaging Library) object and resized to a shape of 64x64 pixels to match the input size expected by the model. The resized image is then converted into a NumPy array.

Next, the NumPy array is expanded along the first axis to create a batch of one image, as required by the model's input shape. The model predicts the class probabilities of the input image using the 'predict' method. The 'argmax' function is applied to the resulting array to determine the index corresponding to the class with the highest probability.

The predicted class index is then printed to the console. If the predicted class index is 0, it indicates that no brain tumor is detected in the image, and the corresponding message "No Brain Tumor Detected" is printed. Otherwise, if the predicted class index is non-zero, it suggests the presence of a brain tumor, and the message "Brain Tumor Detected" is printed.

Finally, the original input image is displayed using Matplotlib's 'imshow' function, and the axis is turned off to remove axis labels and ticks from the plot. This code segment thus enables the prediction of brain tumor presence in MRI images using the pre-trained CNN model.

3.9 Libraries

The programming language utilised in the development, which is a high-level, object-oriented, interpreted, interactive, and general-purpose language. The software offers code that is characterised by clarity and readability. The use of AI and ML algorithms, despite its inherent complexity and adaptable procedures, may support developers in constructing robust and reliable machine intelligent systems when implemented in the Python programming language. The following is a compilation of the Python packages utilised:

Python:

Python is a highly robust programming language that is often utilised in the development of sophisticated surveillance systems that include the ability to generate descriptive captions for videos. The task at hand is particularly well-suited for this programming language because to its flexibility and extensive collection of libraries and frameworks.

One of the primary advantages of the Python programming language is in its inherent simplicity and user-friendly nature, which greatly facilitates the development process. The language's clear and concise syntax, together with its high level of readability, greatly facilitates the development and upkeep of intricate surveillance systems.

Python provides a wide range of tools and frameworks that are crucial for the implementation of video captioning in surveillance applications. OpenCV is commonly employed for video processing, with libraries such as NumPy providing assistance for numerical computations. In the realm of deep learning tasks, the Python programming language offers two prominent frameworks, namely TensorFlow and PyTorch. These frameworks offer a range of pre-trained models and tools that facilitate the construction, training, and deployment of neural networks with notable efficacy.

The use of deep learning is essential in the context of video captioning, with Python's deep learning capabilities assuming a pivotal role. Convolutional Neural Networks (CNNs) are commonly employed in the task of video captioning, and the availability of deep learning frameworks in Python facilitates the implementation and customization of these models.

Python is capable of providing low-latency processing in real-time video captioning settings. The integration of hardware acceleration, such as GPU support, can be employed to accomplish real-time captioning, a crucial aspect in the context of surveillance applications.

OpenCV:

OpenCV, a computer vision library that is open-source, is a highly important resource for constructing intelligent surveillance systems that incorporate video captioning capabilities. The range of functionalities it offers positions it as a preferred option for developers engaged in video analysis and processing tasks.

The utilisation of OpenCV in the creation of surveillance systems with video captioning is facilitated by its comprehensive range of features and its user-friendly application programming interface (API). This enables developers to concentrate on the implementation of surveillance logic and video captioning, eliminating the necessity to construct low-level vision processing components from the ground up.

OpenCV offers comprehensive support for both conventional computer vision methodologies and deep learning-based methodologies. The ability to adapt and customise video captioning methods is crucial for their successful implementation, as it empowers developers to select the most appropriate approach that aligns with the specific needs and objectives. OpenCV has the capability to effectively include deep learning frameworks such as TensorFlow and PyTorch, hence facilitating the utilisation of pre-trained models and customised deep learning pipelines.

Real-time video processing plays a crucial role in surveillance systems, with OpenCV demonstrating exceptional performance in this domain. The technology has the capability to collect and interpret video streams in real-time, rendering it highly suitable for applications that need prompt analysis and reaction. Timely identification and captioning of events are of utmost importance in surveillance circumstances.

NumPy:

NumPy, a pivotal library for numerical computations in the Python programming language, plays a crucial role in several facets of intelligent surveillance systems, encompassing the domain of video captioning. The capabilities of this technology improve the functionality of the data processing and manipulation aspects inside these systems. NumPy demonstrates its value inside this particular environment by virtue of its numerous advantageous features and functionalities.

NumPy facilitates the management of extensive arrays and matrices of data, rendering it very suitable for the effective processing of the substantial volume of data produced by CCTV cameras. This encompasses the tasks of image and video data processing, feature extraction, and data preparation for analysis.

One of the key advantages of NumPy is in its ability to do computations with high speed and efficiency. The performance of NumPy operations is optimised to meet the demanding requirements of real-time video captioning in surveillance applications, where efficient resource utilization is vital.

The extensive collection of mathematical and statistical functions provided by NumPy facilitates the execution of intricate data manipulations and computations by developers. The aforementioned method has significant importance in the examination of video frames, the extraction of pertinent data, and the subsequent generation of captions derived from the processed information.

The software exhibits a smooth integration with regularly utilised libraries in the field of video captioning, including OpenCV, TensorFlow, and PyTorch, hence facilitating a cohesive workflow. This enables the seamless transfer of data among different libraries, hence enabling the implementation of diverse video captioning jobs.

Pandas:

The use of the Python library known as Pandas is of utmost importance in the advancement of intelligent surveillance systems that possess the capability to generate captions for video content. The software's wide range of capabilities and data formats renders it indispensable for the management and analysis of data linked to surveillance video feeds.

Data integration is a crucial aspect of surveillance systems, as it facilitates the smooth connection of numerous data sources. Pandas, a widely used data manipulation library in Python, offers a convenient solution by enabling the integration of data from diverse sources such as CSV files, databases, and external APIs. This functionality facilitates the integration of data from many sources, including sensors and other databases, therefore enhancing the surveillance system with significant and important information.

In addition, Pandas has functionality for exporting processed data to several formats, including CSV, Excel, and databases, enabling more options for data storage and reporting. The API of the system is designed to be user-friendly and expressive, facilitating efficient data handling and processing. Additionally, its interoperability with other Python libraries simplifies integration with other tools. This enables developers to concentrate on developing sophisticated video captioning algorithms that augment the functionalities of intelligent surveillance systems.

Tensorflow:

The TensorFlow, created by Google Brain, stands out as a go-to machine learning tool embraced worldwide for its adaptability and ability to handle large-scale projects. It boasts a rich array of resources, including libraries and a vibrant community, attracting researchers, developers, and businesses alike. This platform's versatility shines through in its capacity to construct and train various machine learning models, spanning from straightforward regressions to intricate neural networks.

Fundamentally, TensorFlow relies on data flow graphs, where nodes signify mathematical operations and edges denote the flow of data between them. This innovative approach facilitates efficient computation execution across diverse hardware setups, be it CPUs, GPUs, or specialized TPUs. Moreover, TensorFlow simplifies model creation with high-level APIs like Keras, streamlining the intricate processes involved in model building and training.

With its extensive documentation, tutorials, and supportive community, TensorFlow fosters a collaborative space for learning and innovation in machine learning and AI. Its robustness extends its applicability across numerous domains, including image and speech recognition, natural language processing, and recommendation systems. As TensorFlow continuously advances through regular updates and enhanced features, it remains at the forefront of machine learning frameworks, empowering users to tackle increasingly intricate AI challenges with confidence.

Sklearn:

Scikit-learn, a popular machine learning library, is well-known for its user-friendly interface and comprehensive range of algorithms and utilities. Engineered with simplicity and effectiveness in mind, scikit-learn offers a complete set of functions for tasks like data preprocessing, model selection, evaluation, and more. Its intuitive structure makes it accessible to users at all levels of expertise, from novices to experienced practitioners in the machine learning domain.

Fundamentally, scikit-learn presents a unified platform for constructing and deploying machine learning models, incorporating a wide spectrum of algorithms for tasks such as classification, regression, clustering, and dimensionality reduction. Thanks to its consistent application programming interface (API) and clear syntax, users can seamlessly experiment with different algorithms and fine-tune parameters to achieve optimal model performance.

A standout feature of scikit-learn is its focus on code readability and simplicity, enabling swift prototyping and iteration in machine learning projects. Supported by comprehensive documentation, tutorials, and examples, users can grasp intricate concepts and apply them effectively in practical scenarios. Alongside its user-friendly design, scikit-learn demonstrates robust performance and scalability, making it suitable for handling datasets of varying sizes and complexities, whether for small-scale experiments or large-scale deployments

CHAPTER 4

UML DIAGRAMS FOR THE PROPOSED MODELS

4.1 Convolutional Neural Networks (CNNs)

A Convolutional Neural Network (CNN) is a type of artificial intelligence system inspired by the human brain's visual processing capabilities. Similar to how our brain recognizes patterns in what we see, a CNN is designed to understand images. It consists of multiple layers, each performing a specific role in analyzing the image. The initial layers focus on detecting simple features like edges or corners, while subsequent layers combine these features to recognize more complex patterns like shapes or textures. This hierarchical approach continues until the final layer can accurately identify what the image contains. Essentially, a CNN operates as a series of filters, gradually refining its understanding of the image with each layer.

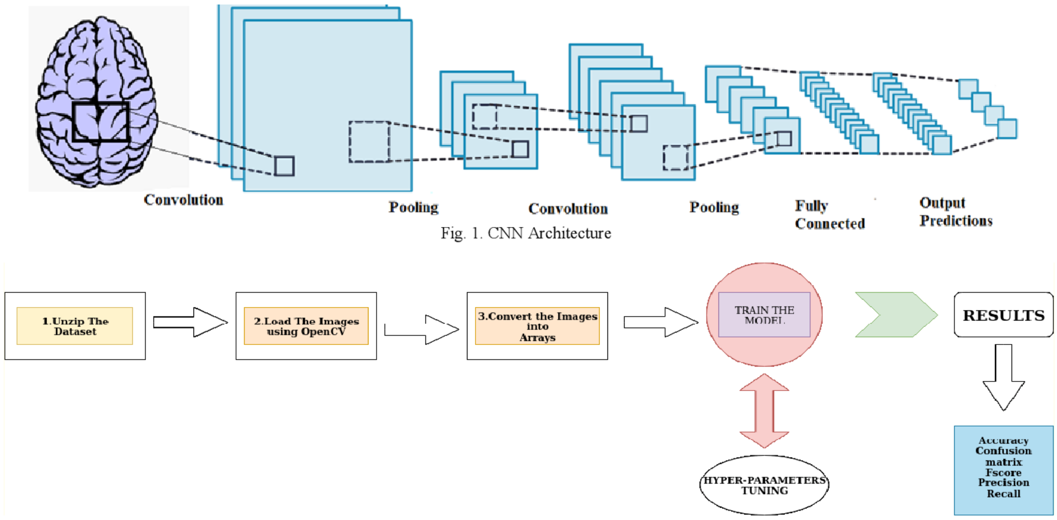


Fig. 1. CNN Architecture

Fig 4.1: CNN Architecture

4.2 Transfer learning

Transfer learning is a technique used in machine learning where knowledge gained from solving one problem is applied to a different but related problem. It involves taking a pre-trained model, which has already learned useful features from a large dataset, and adapting it to a new task with a smaller dataset. This approach helps in situations where collecting sufficient data for training a new model from scratch may be impractical or costly. By leveraging the pre-trained model's learned representations, transfer learning allows for faster training and better performance on the new task. Essentially, it's like building on existing knowledge to solve new problems more efficiently.

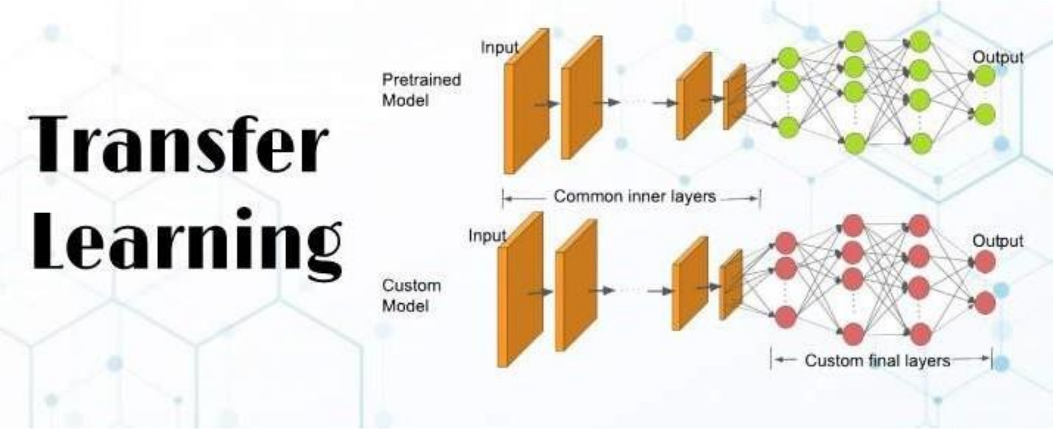


Fig 4.2: Transfer learning

4.3 Vgg 16

The architecture of VGG16 commences with an input image, which undergoes a sequence of convolutional layers. Each convolutional layer is accompanied by a rectified linear activation function (ReLU). VGG16 stands out for its utilization of compact 3x3 convolutional filters with a stride of 1 and same padding. Following convolution, max-pooling layers are utilized to diminish the spatial dimensions of the feature maps while preserving vital information. The resultant feature maps are then flattened and conveyed through a series of fully connected layers, responsible for extracting high-level features and facilitating classification. Lastly, the output layer employs a softmax activation function to generate class probabilities.

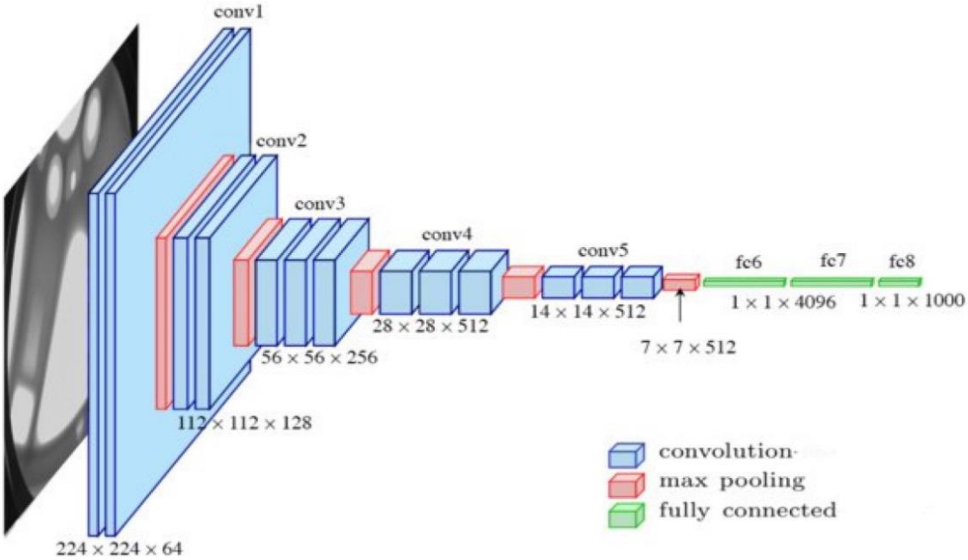


Fig 4.3: Architecture diagram for vgg16

4.4 Inception V3

InceptionV3 employs inception modules, which consist of parallel convolutional operations of different filter sizes (1x1, 3x3, and 5x5), followed by pooling operations. These modules enable the network to capture features at multiple scales efficiently. Multiple inception modules are stacked sequentially to form the backbone of the network. Additionally, InceptionV3 incorporates auxiliary classifiers at intermediate layers during training to aid in gradient propagation and regularization. After passing through the inception modules, the output is globally averaged pooled to reduce spatial dimensions, resulting in a fixed-size feature vector. Finally, the feature vector is processed through fully connected layers, leading to the output layer with softmax activation for classification.

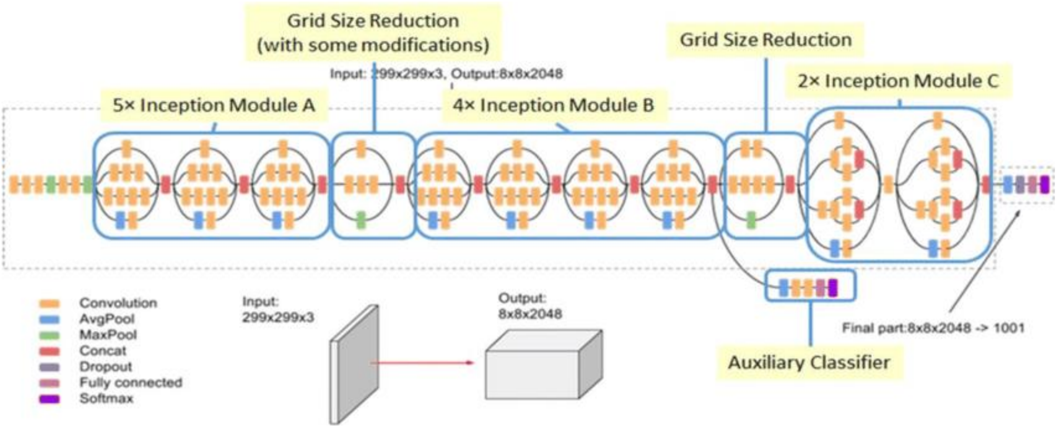


Fig 4.4: Architecture diagram for inception

4.5 Xception

Xception revolutionizes the conventional convolutional neural network (CNN) architecture by utilizing depthwise separable convolutions. The input image first undergoes depthwise convolutions, where each channel is convolved independently with its own set of filters. Following this, pointwise convolutions combine information across channels by convolving the output of depthwise convolution with 1x1 filters. The network comprises multiple blocks of these depthwise separable convolutions, referred to as inception blocks. These blocks are stacked to form the complete network. The output from the last inception block is flattened and fed into fully connected layers, ultimately leading to the output layer with softmax activation for classification.

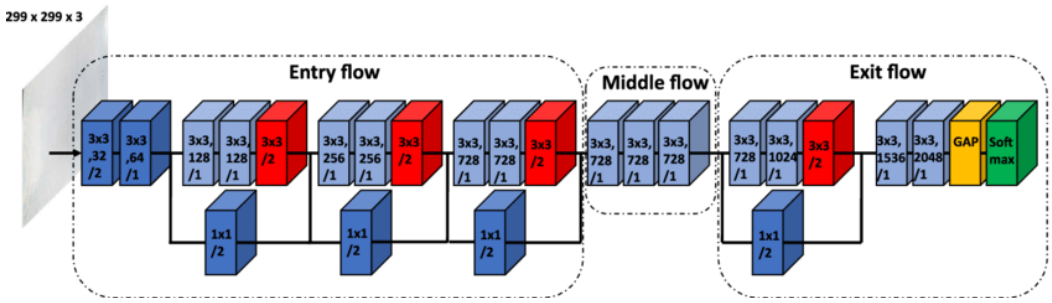


Fig 4.5: Architecture diagram for Xception

4.6 Flowchart

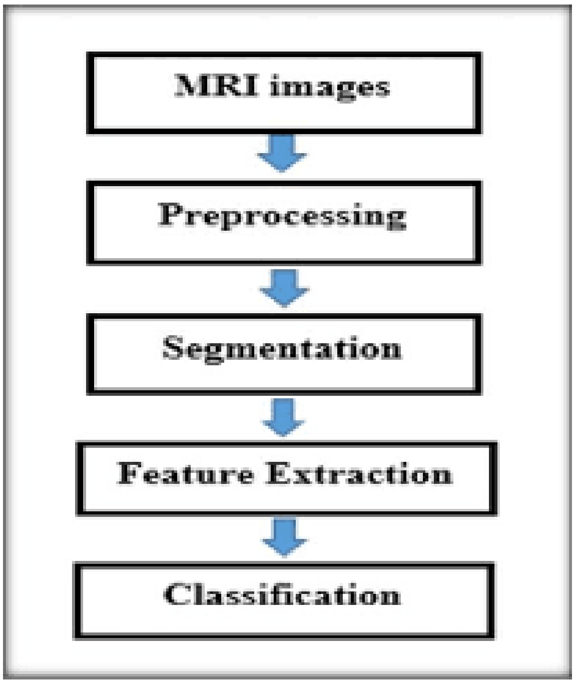


Fig 4.6: Steps for image processing

CHAPTER 5

MODULES DESCRIPTION

5.1 Dataset

The dataset under scrutiny comprises a collection of MRI (Magnetic Resonance Imaging) scans portraying brain imagery, wherein each scan delineates the presence of either a brain tumor or the absence thereof, denoting a healthy brain devoid of any lesions. These MRI scans are typically provided in formats such as DICOM or NIfTI, containing comprehensive anatomical insights into the brain's structure and any discernible abnormalities. Furthermore, accompanying each image are corresponding labels serving to indicate the presence or absence of a tumor.

This dataset represents a cornerstone within the realm of neural network development tailored explicitly for brain lesion identification in the domain of medical imaging. Its inherent value lies within the breadth of MRI scans it encompasses, spanning both instances of brain tumors and unaffected, healthy brain tissue.

The diversity encapsulated within the dataset is paramount, facilitating the training of neural network models to discern nuanced disparities between healthy and abnormal brain anatomy.

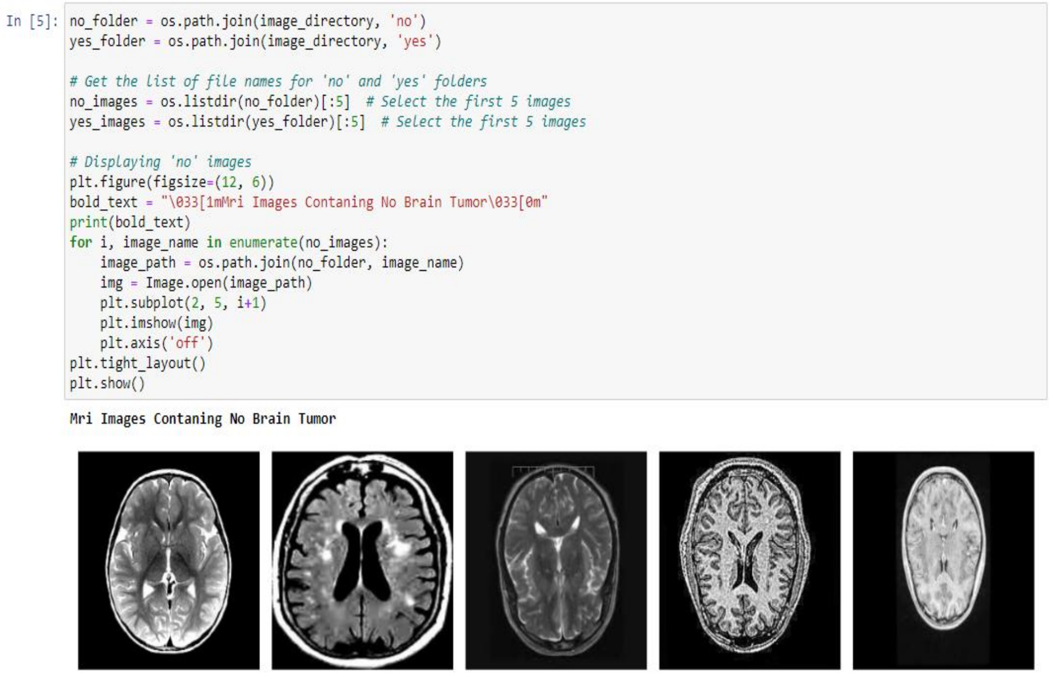
Leveraging this dataset, researchers are empowered to engineer and fine-tune robust classification algorithms proficient in precisely discerning regions affected by tumors or other lesions from the normative brain anatomy. Through meticulous analysis of the provided MRI scans, the neural network apparatus acquires a nuanced understanding of intricate patterns and features, thus enabling informed decisions regarding the presence or absence of abnormalities within novel, unseen images.

The advancements attained within medical imaging analysis carry profound implications for patient welfare. With heightened diagnostic accuracy facilitated by the adeptly trained neural network models, medical practitioners can swiftly identify brain irregularities with increased confidence.

Timely detection of lesions enables prompt intervention and treatment, potentially augmenting patient outcomes and prognoses. Moreover, the capacity to accurately identify and characterize brain lesions serves to inform treatment strategies, guiding clinicians in the selection of optimal therapeutic interventions tailored to individual patient requisites.

Overall, the exploitation of this dataset equips both researchers and medical professionals with the tools necessary to drive forward the frontier of medical imaging analysis.

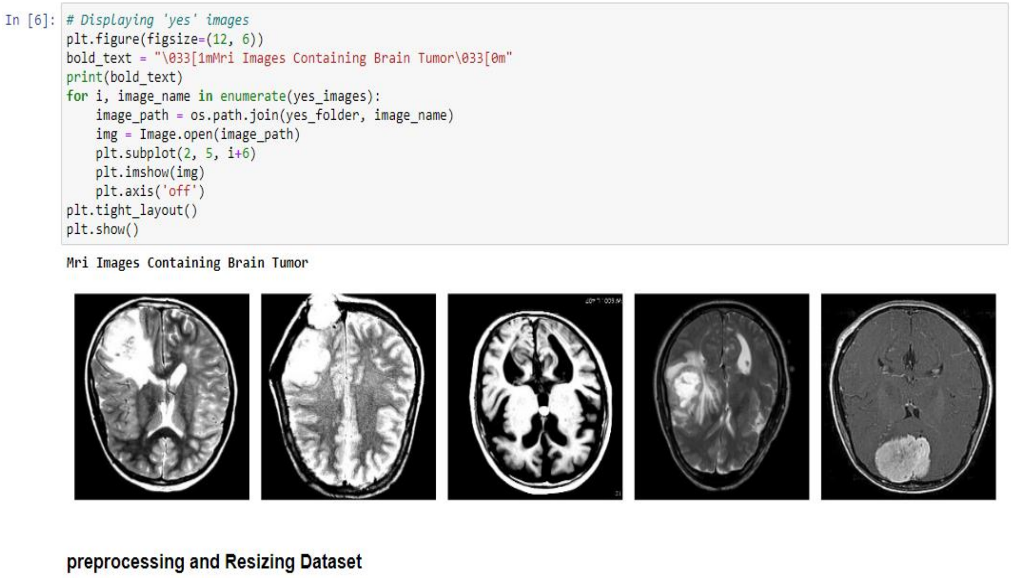
Through collaborative endeavors and innovative methodologies, the overarching objective is to bolster diagnostic precision, ameliorate patient care, and foster progression within the realms of medical science and technology.



```
In [5]: no_folder = os.path.join(image_directory, 'no')
        yes_folder = os.path.join(image_directory, 'yes')

        # Get the list of file names for 'no' and 'yes' folders
        no_images = os.listdir(no_folder)[:5]  # Select the first 5 images
        yes_images = os.listdir(yes_folder)[:5]  # Select the first 5 images

        # Displaying 'no' images
        plt.figure(figsize=(12, 6))
        bold_text = "\033[1mMri Images Contaning No Brain Tumor\033[0m"
        print(bold_text)
        for i, image_name in enumerate(no_images):
            image_path = os.path.join(no_folder, image_name)
            img = Image.open(image_path)
            plt.subplot(2, 5, i+1)
            plt.imshow(img)
            plt.axis('off')
        plt.tight_layout()
        plt.show()
```

```
Mri Images Contaning No Brain Tumor
```

Fig 5.1: Images with No Brain Tumor



```
In [6]: # Displaying 'yes' images
        plt.figure(figsize=(12, 6))
        bold_text = "\033[1mMri Images Containing Brain Tumor\033[0m"
        print(bold_text)
        for i, image_name in enumerate(yes_images):
            image_path = os.path.join(yes_folder, image_name)
            img = Image.open(image_path)
            plt.subplot(2, 5, i+6)
            plt.imshow(img)
            plt.axis('off')
        plt.tight_layout()
        plt.show()
```

```
Mri Images Containing Brain Tumor
```

preprocessing and Resizing Dataset

Fig 5.2: Images with Brain Tumor

5.2 Data Pre-processing

The data preprocessing stage is a critical aspect of our project on advanced neural network architecture for brain lesion identification. In this phase, we perform several key steps to prepare the MRI images for subsequent analysis and model training.

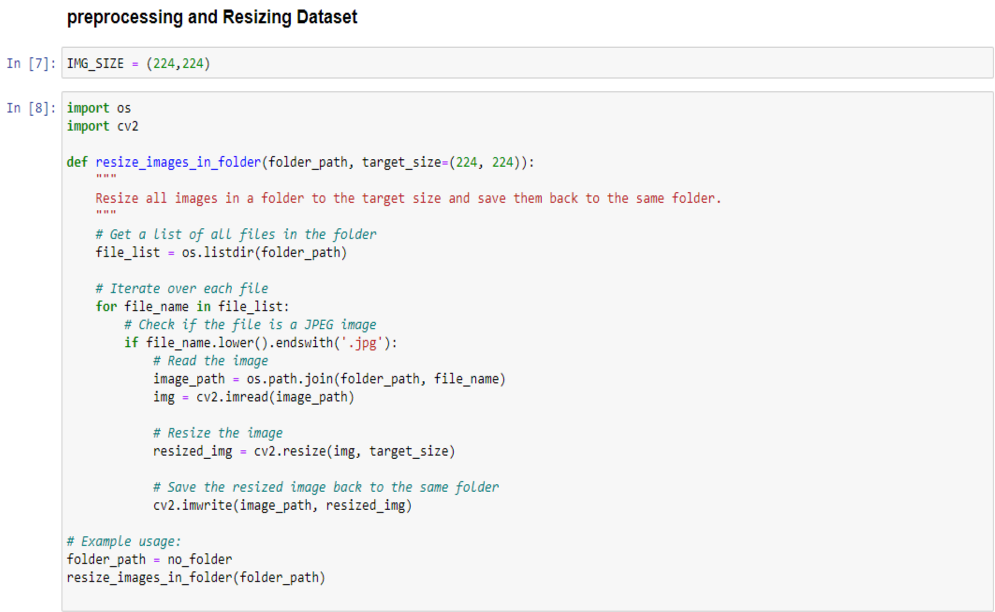
Firstly, we iterate through two lists of image filenames: `no_tumor_images` and `yes_tumor_images`, corresponding to MRI scans without tumors and those with tumors, respectively. We ensure image consistency by checking for the '.jpg' extension in each filename.

Next, we utilize the OpenCV library (`cv2`) to read each image from the specified directory, converting it into a PIL (Python Imaging Library) image object. This conversion is essential for seamless image manipulation and preprocessing.

Following image retrieval, we resize each image to a predetermined input size using the `resize()` method. This step standardizes the dimensions of all images, ensuring uniformity across the dataset.

Subsequently, the resized images are converted into NumPy arrays and appended to a dataset list. Simultaneously, corresponding labels are appended to a label list. A label of `0` indicates a brain image without a tumor, while a label of `1` signifies the presence of a tumor.

By meticulously executing these preprocessing steps, we ensure that the MRI images are consistently formatted, resized, and paired with appropriate labels. This standardized dataset is then ready for subsequent stages of model development, training, and evaluation.

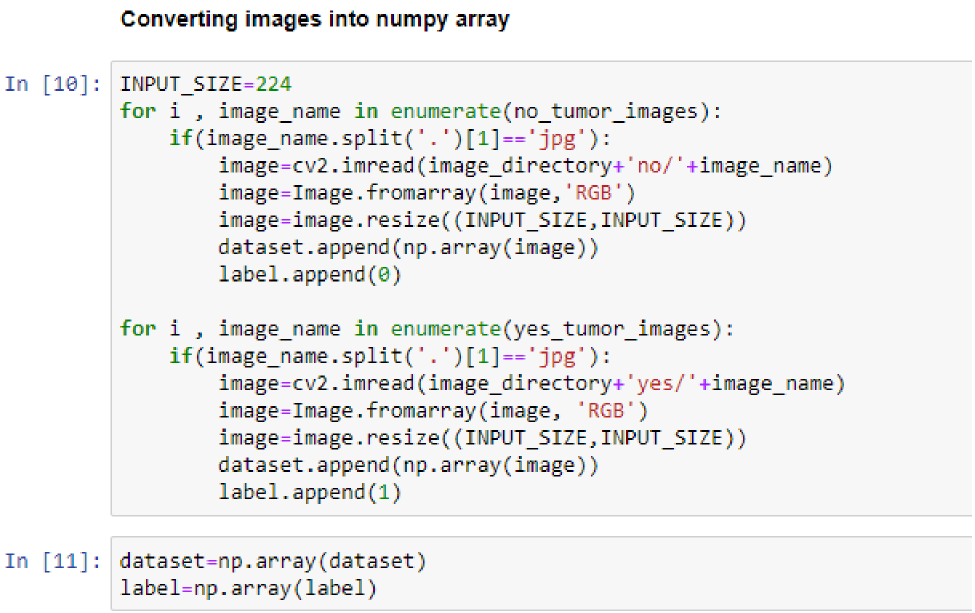


preprocessing and Resizing Dataset

```
In [7]: IMG_SIZE = (224,224)
```

```
In [8]: import os
        import cv2

        def resize_images_in_folder(folder_path, target_size=(224, 224)):
            """
            Resize all images in a folder to the target size and save them back to the same folder.
            """
            # Get a list of all files in the folder
            file_list = os.listdir(folder_path)

            # Iterate over each file
            for file_name in file_list:
                # Check if the file is a JPEG image
                if file_name.lower().endswith('.jpg'):
                    # Read the image
                    image_path = os.path.join(folder_path, file_name)
                    img = cv2.imread(image_path)

                    # Resize the image
                    resized_img = cv2.resize(img, target_size)

                    # Save the resized image back to the same folder
                    cv2.imwrite(image_path, resized_img)

        # Example usage:
        folder_path = no_folder
        resize_images_in_folder(folder_path)
```

Fig 5.3: Pre-processing dataset



Converting images into numpy array

```
In [10]: INPUT_SIZE=224
         for i , image_name in enumerate(no_tumor_images):
             if(image_name.split('.')[1]=='jpg'):
                 image=cv2.imread(image_directory+'no/'+image_name)
                 image=Image.fromarray(image,'RGB')
                 image=image.resize((INPUT_SIZE,INPUT_SIZE))
                 dataset.append(np.array(image))
                 label.append(0)

         for i , image_name in enumerate(yes_tumor_images):
             if(image_name.split('.')[1]=='jpg'):
                 image=cv2.imread(image_directory+'yes/'+image_name)
                 image=Image.fromarray(image, 'RGB')
                 image=image.resize((INPUT_SIZE,INPUT_SIZE))
                 dataset.append(np.array(image))
                 label.append(1)
```

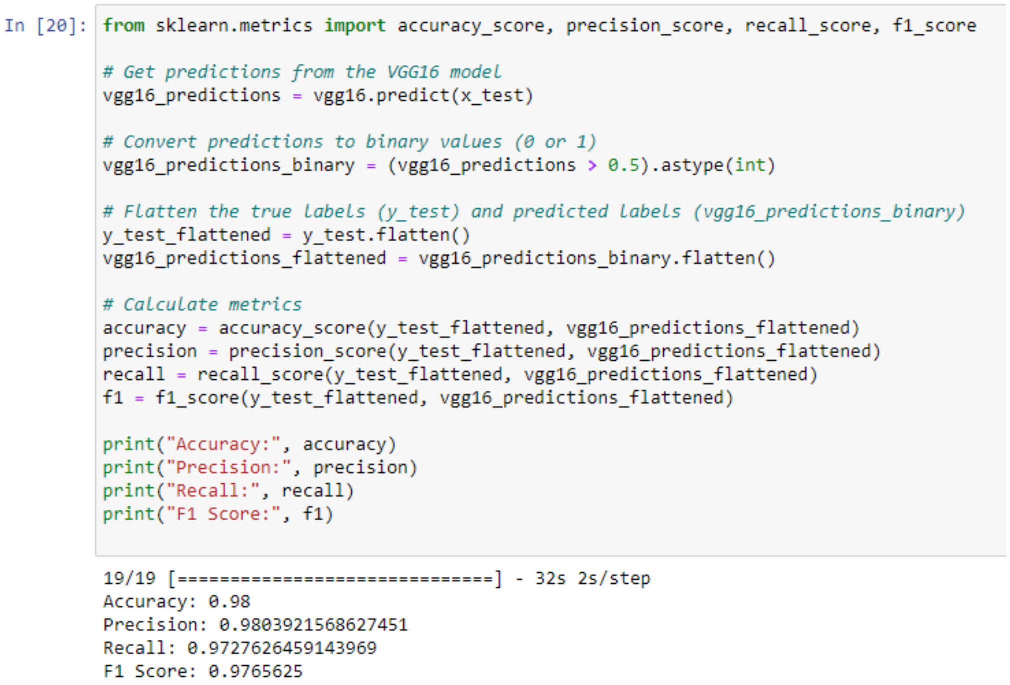
```
In [11]: dataset=np.array(dataset)
         label=np.array(label)
```

Fig 5.4: converting images into numpy array

5.3 Model Training

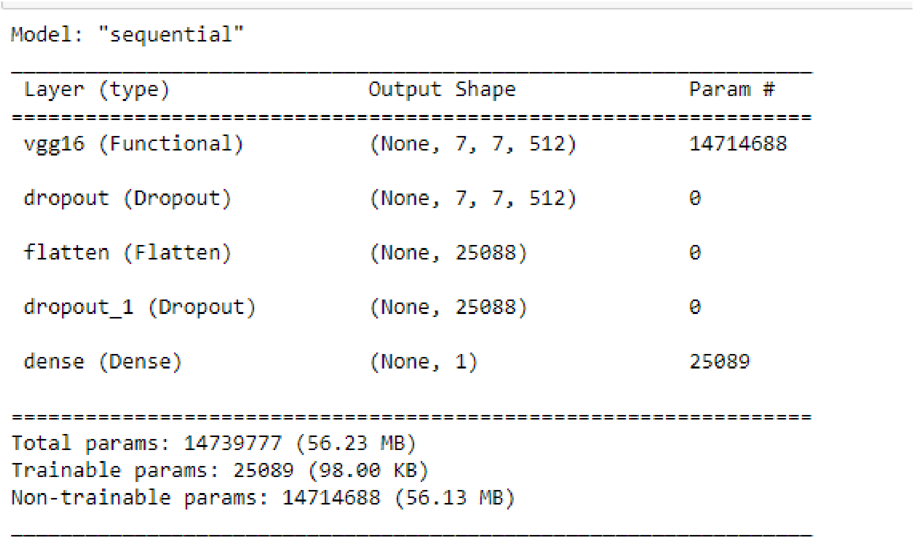
5.3.1 VGG16

In our exploration of the VGG16 module, we took a deep dive into the intricate process of image feature detection by the neural network. Utilizing OpenCV alongside the robust VGG16 framework, we began by loading an image, which we then resized to align with the model's expected input dimensions. Following this, the image was transformed into an array and its dimensions were expanded to simulate a single sample. The image underwent a preprocessing step tailored for VGG16, leading us to the extraction of the feature map from the initial hidden layer, thanks to some Keras wizardry. The highlight of this process was the visualization of these feature maps in a structured grid through Matplotlib. This visualization offered us a glimpse into the VGG16 model's ability to recognize various image features. Engaging in this practical examination not only enhanced our comprehension of the VGG16 architecture but also equipped us with deeper insights into the functioning of neural networks, thereby guiding our project towards more enlightened choices.



```
In [20]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

         # Get predictions from the VGG16 model
         vgg16_predictions = vgg16.predict(x_test)

         # Convert predictions to binary values (0 or 1)
         vgg16_predictions_binary = (vgg16_predictions > 0.5).astype(int)

         # Flatten the true labels (y_test) and predicted labels (vgg16_predictions_binary)
         y_test_flattened = y_test.flatten()
         vgg16_predictions_flattened = vgg16_predictions_binary.flatten()

         # Calculate metrics
         accuracy = accuracy_score(y_test_flattened, vgg16_predictions_flattened)
         precision = precision_score(y_test_flattened, vgg16_predictions_flattened)
         recall = recall_score(y_test_flattened, vgg16_predictions_flattened)
         f1 = f1_score(y_test_flattened, vgg16_predictions_flattened)

         print("Accuracy:", accuracy)
         print("Precision:", precision)
         print("Recall:", recall)
         print("F1 Score:", f1)

         19/19 [==============================] - 32s 2s/step
         Accuracy: 0.98
         Precision: 0.9803921568627451
         Recall: 0.9727626459143969
         F1 Score: 0.9765625
```

Fig 5.5: Training of Vgg16



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 dropout (Dropout)           (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dropout_1 (Dropout)         (None, 25088)             0

 dense (Dense)               (None, 1)                 25089

=================================================================
Total params: 14739777 (56.23 MB)
Trainable params: 25089 (98.00 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

Fig 5.6: Layers of Vgg16

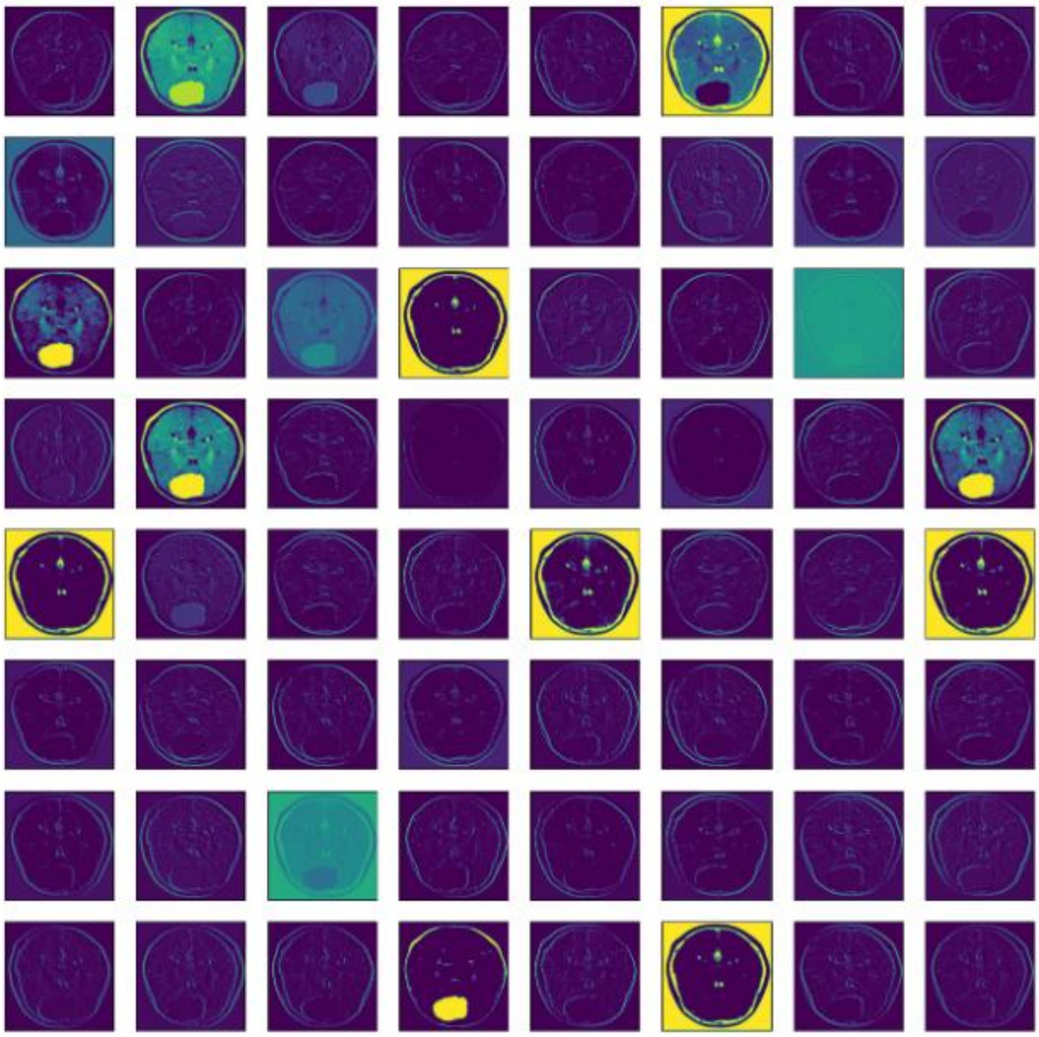


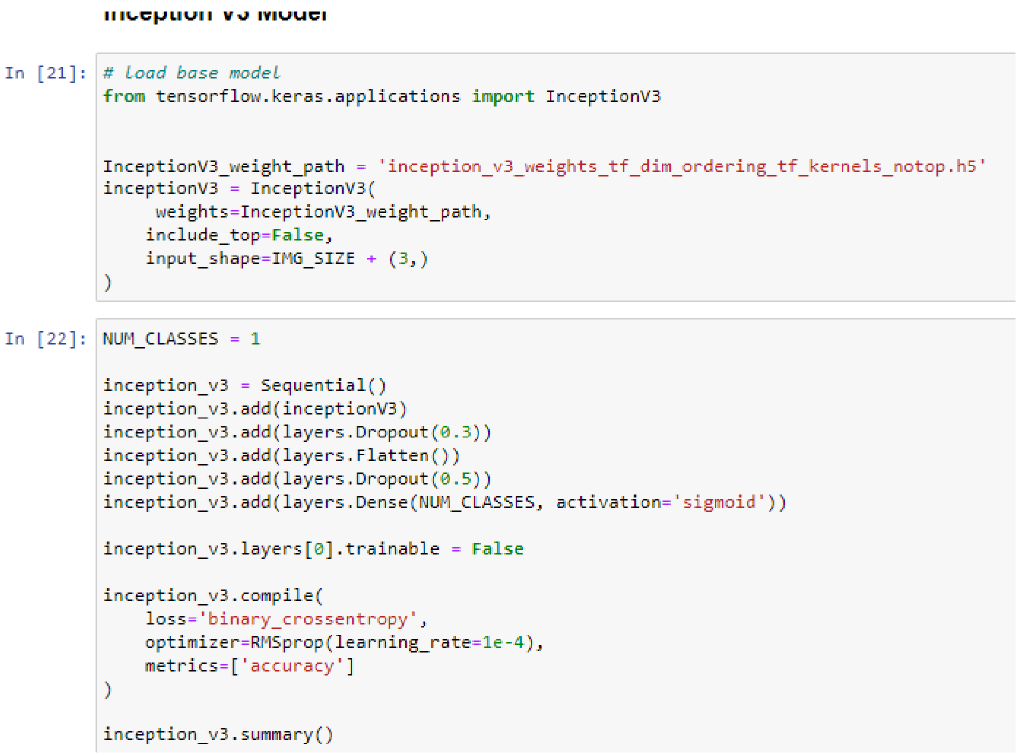
Fig 5.7: Feature mapping

5.3.2 InceptionV3

For our project, we decided to use the InceptionV3 model because it's really good at recognizing complex images. We started by setting up the model with some adjustments, like removing the top layer so it could better fit our task of deciding whether an image shows one thing or another. We added some layers to help the model learn without memorizing the data (this helps it perform better on new images it hasn't seen before) and a layer at the end that makes the final decision.

Then, we trained the model. This means we let it look at a bunch of images we already know the answers to, so it could learn the patterns. We used a special setting that adjusts how the model learns, making sure it does so effectively. We did this training in rounds, tweaking the model a bit each time to improve how well it could predict the correct answers on new images.

After the training was done, we checked how good the model was at making predictions. We looked at its accuracy and how long it took to learn everything. This whole process showed us just how powerful the InceptionV3 model is for classifying images and gave us some great insights into using advanced technology to solve practical problems in image recognition.



Inception V3 Model

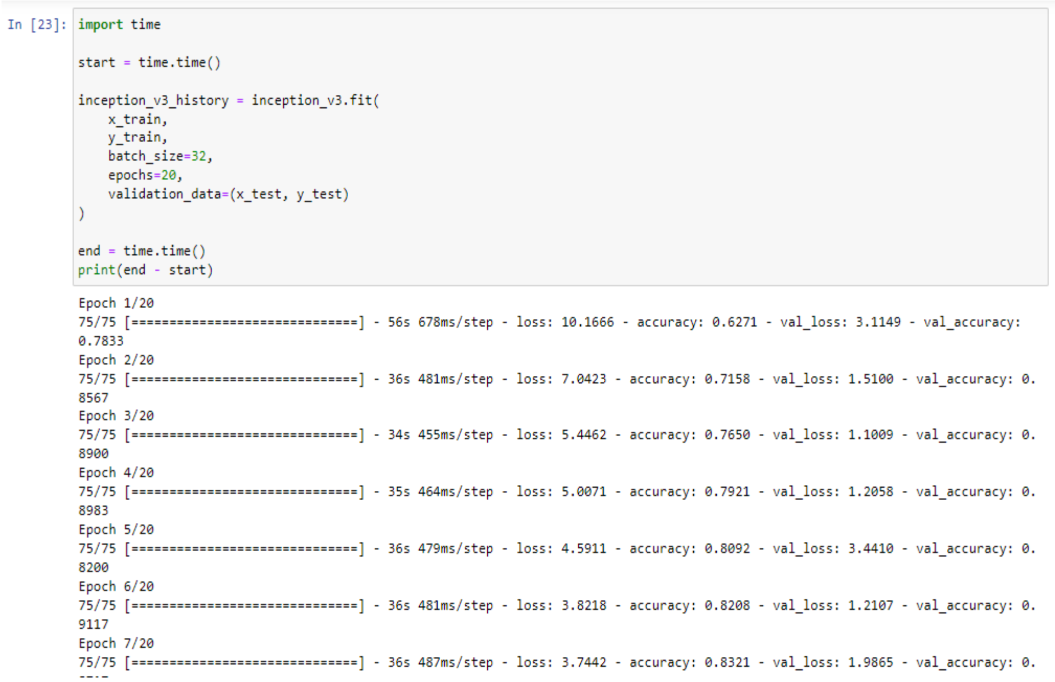
In [21]:

```
# Load base model
from tensorflow.keras.applications import InceptionV3


InceptionV3_weight_path = 'inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'
inceptionV3 = InceptionV3(
     weights=InceptionV3_weight_path,
    include_top=False,
    input_shape=IMG_SIZE + (3,)
)
```

In [22]:

```
NUM_CLASSES = 1

inception_v3 = Sequential()
inception_v3.add(inceptionV3)
inception_v3.add(layers.Dropout(0.3))
inception_v3.add(layers.Flatten())
inception_v3.add(layers.Dropout(0.5))
inception_v3.add(layers.Dense(NUM_CLASSES, activation='sigmoid'))

inception_v3.layers[0].trainable = False

inception_v3.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(learning_rate=1e-4),
    metrics=['accuracy']
)

inception_v3.summary()
```

Fig 5.8: Model building on Inception V3



```
In [23]: import time

         start = time.time()

         inception_v3_history = inception_v3.fit(
             x_train,
             y_train,
             batch_size=32,
             epochs=20,
             validation_data=(x_test, y_test)
         )

         end = time.time()
         print(end - start)
```

```
Epoch 1/20
75/75 [==============================] - 56s 678ms/step - loss: 10.1666 - accuracy: 0.6271 - val_loss: 3.1149 - val_accuracy:
0.7833
Epoch 2/20
75/75 [==============================] - 36s 481ms/step - loss: 7.0423 - accuracy: 0.7158 - val_loss: 1.5100 - val_accuracy: 0.
8567
Epoch 3/20
75/75 [==============================] - 34s 455ms/step - loss: 5.4462 - accuracy: 0.7650 - val_loss: 1.1009 - val_accuracy: 0.
8900
Epoch 4/20
75/75 [==============================] - 35s 464ms/step - loss: 5.0071 - accuracy: 0.7921 - val_loss: 1.2058 - val_accuracy: 0.
8983
Epoch 5/20
75/75 [==============================] - 36s 479ms/step - loss: 4.5911 - accuracy: 0.8092 - val_loss: 3.4410 - val_accuracy: 0.
8200
Epoch 6/20
75/75 [==============================] - 36s 481ms/step - loss: 3.8218 - accuracy: 0.8208 - val_loss: 1.2107 - val_accuracy: 0.
9117
Epoch 7/20
75/75 [==============================] - 36s 487ms/step - loss: 3.7442 - accuracy: 0.8321 - val_loss: 1.9865 - val_accuracy: 0.
```

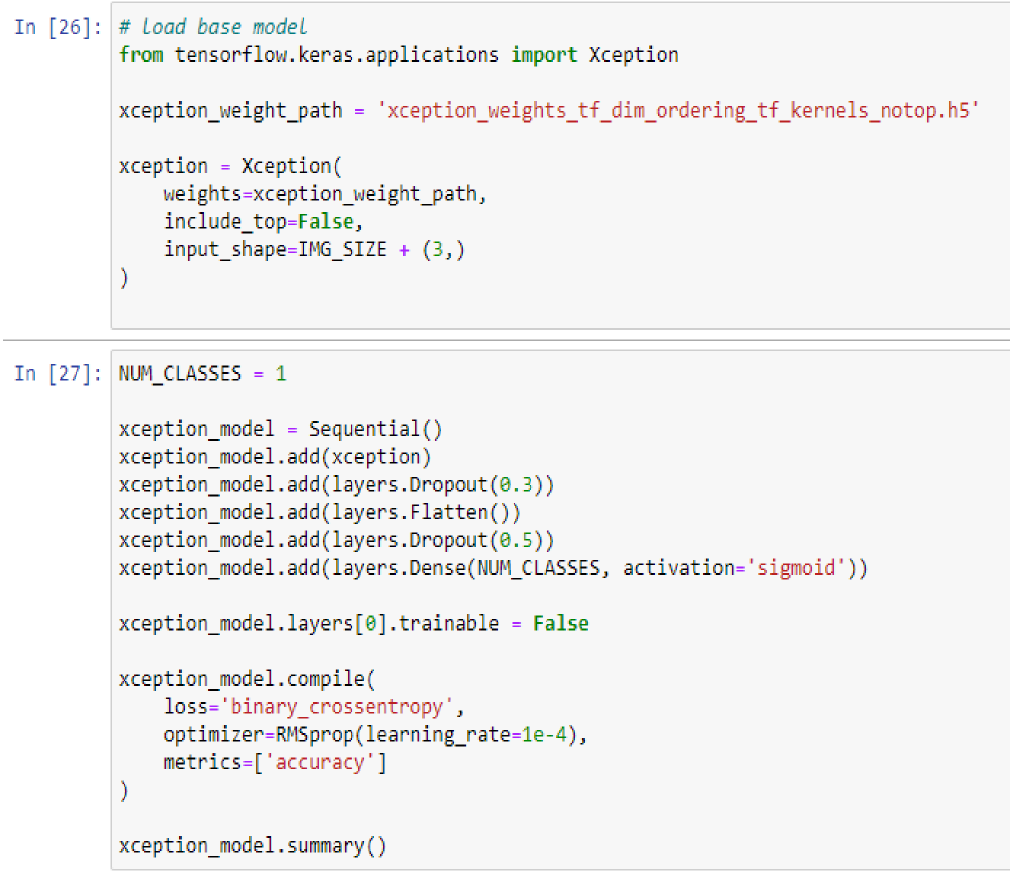
Fig 5.9: Training of Inception V3

5.3.3 Xception

For our project, we decided to use the Xception model, which is known for being really good at classifying images. We started by setting up the Xception model but made some changes so it could work better for our needs. We added some layers to help the model learn better without remembering the exact images we trained it on, which helps it perform well even on new images it hasn't seen before. We also adjusted it to focus on deciding between two options, making it a perfect fit for figuring out whether our specific criteria are met in an image.

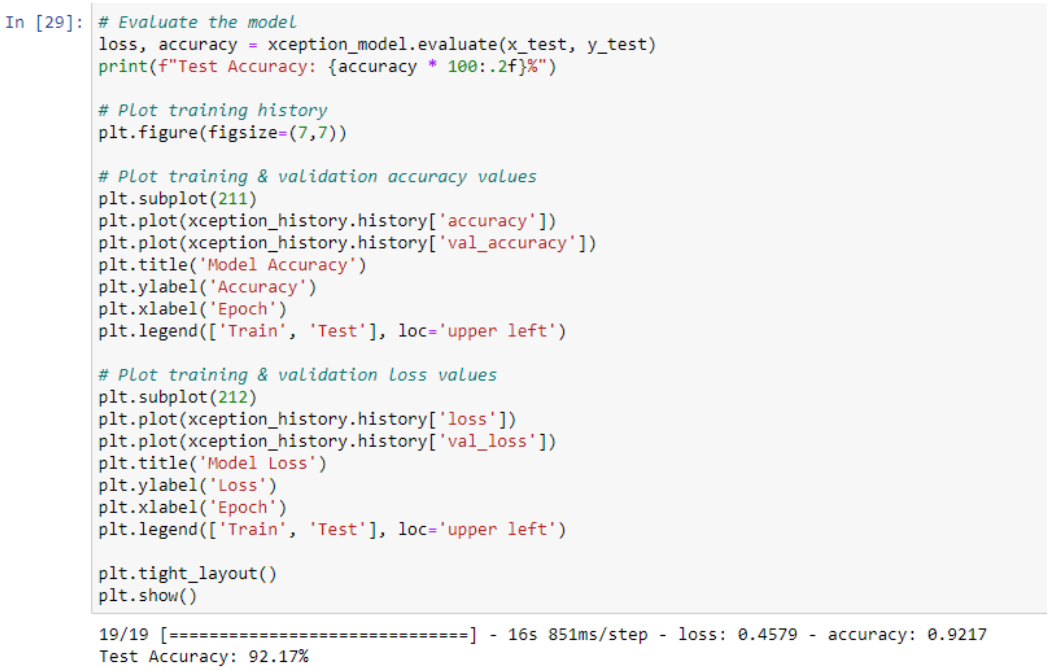
We made sure the Xception model didn't forget what it already knew by keeping its original knowledge unchanged while it learned from our images. After setting everything up, we trained our model by showing it lots of images and telling it whether they met our criteria. This process took some time, but it was crucial for making sure our model learned accurately.

Using the Xception model was a key part of our project because it helped us tackle complex image classification challenges. By tweaking the model and training it on our dataset, we got a powerful tool that can accurately classify images. This step was an important achievement for us, showing how advanced AI models like Xception can be adapted for specific tasks and used in real-world applications.



```
In [26]: # Load base model
         from tensorflow.keras.applications import Xception

         xception_weight_path = 'xception_weights_tf_dim_ordering_tf_kernels_notop.h5'

         xception = Xception(
             weights=xception_weight_path,
             include_top=False,
             input_shape=IMG_SIZE + (3,)
         )

In [27]: NUM_CLASSES = 1

         xception_model = Sequential()
         xception_model.add(xception)
         xception_model.add(layers.Dropout(0.3))
         xception_model.add(layers.Flatten())
         xception_model.add(layers.Dropout(0.5))
         xception_model.add(layers.Dense(NUM_CLASSES, activation='sigmoid'))

         xception_model.layers[0].trainable = False

         xception_model.compile(
             loss='binary_crossentropy',
             optimizer=RMSprop(learning_rate=1e-4),
             metrics=['accuracy']
         )

         xception_model.summary()
```

Fig 5.10: Model building of Xception



```
In [29]: # Evaluate the model
loss, accuracy = xception_model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Plot training history
plt.figure(figsize=(7,7))

# Plot training & validation accuracy values
plt.subplot(211)
plt.plot(xception_history.history['accuracy'])
plt.plot(xception_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(212)
plt.plot(xception_history.history['loss'])
plt.plot(xception_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

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

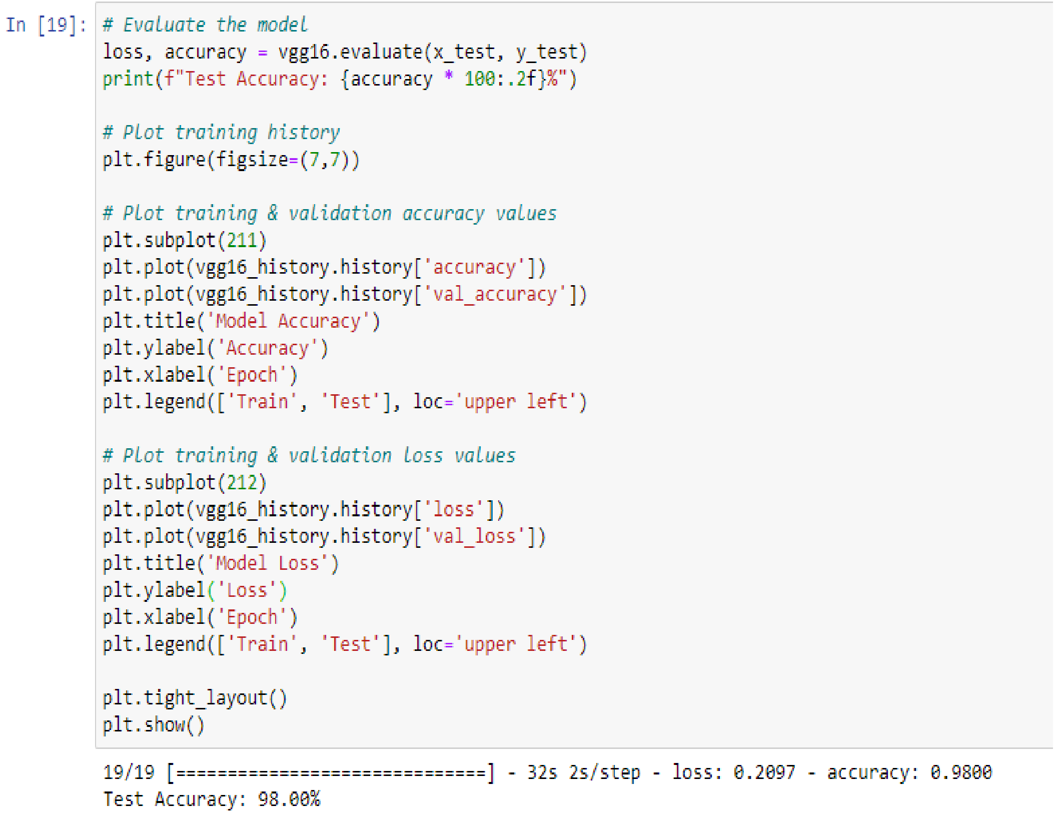
```
19/19 [==============================] - 16s 851ms/step - loss: 0.4579 - accuracy: 0.9217
Test Accuracy: 92.17%
```

Fig 5.11: Training of Xception

5.4 Evaluation

VGG16 Model Evaluation Simplified

When we checked how well our VGG16 model could identify different kinds of images using test data, it did really well. This shows it's good at telling images apart. We also looked at charts showing its learning progress over time, with its ability to correctly recognize images getting better and mistakes decreasing as it practiced more with the training images. These charts help us see how the model is getting smarter at figuring out what it sees in pictures, making fewer errors as it learns.

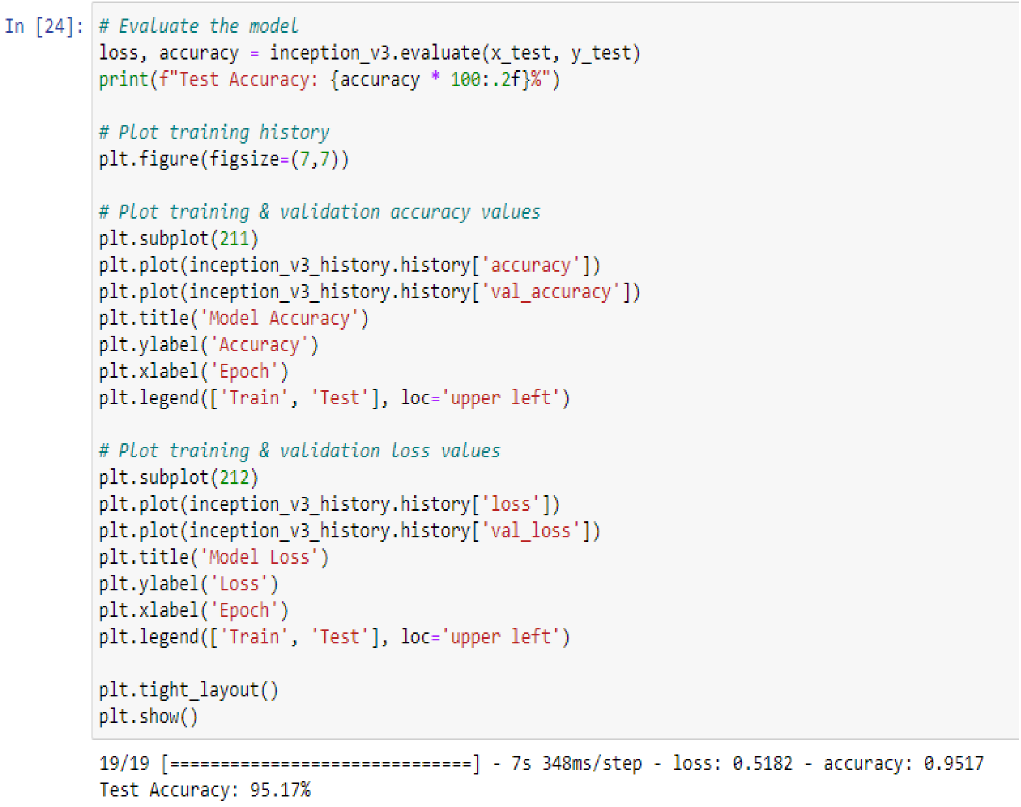


```
In [19]: # Evaluate the model
         loss, accuracy = vgg16.evaluate(x_test, y_test)
         print(f"Test Accuracy: {accuracy * 100:.2f}%")

         # Plot training history
         plt.figure(figsize=(7,7))

         # Plot training & validation accuracy values
         plt.subplot(211)
         plt.plot(vgg16_history.history['accuracy'])
         plt.plot(vgg16_history.history['val_accuracy'])
         plt.title('Model Accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper left')

         # Plot training & validation loss values
         plt.subplot(212)
         plt.plot(vgg16_history.history['loss'])
         plt.plot(vgg16_history.history['val_loss'])
         plt.title('Model Loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper left')

         plt.tight_layout()
         plt.show()

         19/19 [==============================] - 32s 2s/step - loss: 0.2097 - accuracy: 0.9800
         Test Accuracy: 98.00%
```

Fig 5.12: Evaluation of Vgg16

InceptionV3 Model Evaluation

For the InceptionV3 model, when we tested it to see how accurately it could classify images, it scored high, proving it's quite smart at understanding images. By plotting its learning journey, we saw the model's accuracy climb and errors fall over time, indicating it was getting better at its job. These visual plots are like a report card showing us that the model is learning well from its training, getting smarter and making fewer mistakes as it goes, which speaks volumes about how capable InceptionV3 is at handling tricky image classification tasks.

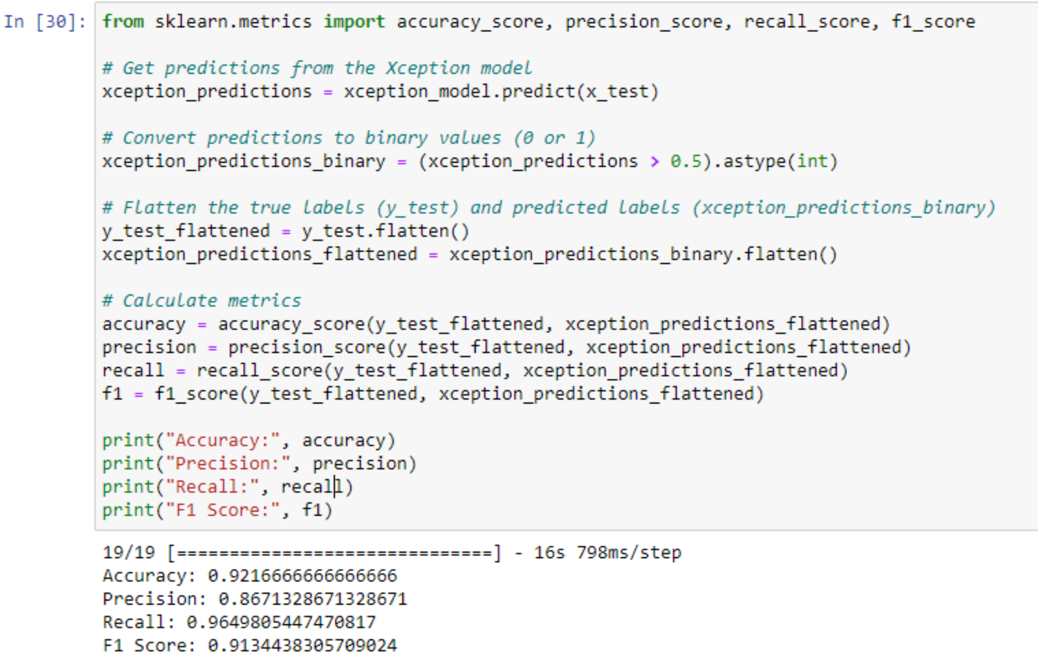


```
In [24]: # Evaluate the model
         loss, accuracy = inception_v3.evaluate(x_test, y_test)
         print(f"Test Accuracy: {accuracy * 100:.2f}%")

         # Plot training history
         plt.figure(figsize=(7,7))

         # Plot training & validation accuracy values
         plt.subplot(211)
         plt.plot(inception_v3_history.history['accuracy'])
         plt.plot(inception_v3_history.history['val_accuracy'])
         plt.title('Model Accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper left')

         # Plot training & validation loss values
         plt.subplot(212)
         plt.plot(inception_v3_history.history['loss'])
         plt.plot(inception_v3_history.history['val_loss'])
         plt.title('Model Loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Test'], loc='upper left')

         plt.tight_layout()
         plt.show()

         19/19 [==============================] - 7s 348ms/step - loss: 0.5182 - accuracy: 0.9517
         Test Accuracy: 95.17%
```

Fig 5.13: Evaluation of InceptionV3

Xception Model

Lastly, when we put the Xception model to the test, it showed off its high accuracy, confirming it's also great at figuring out what's in an image. Looking at its learning process through plots, we noticed a steady improvement in how many images it got right and a decrease in errors, showing it was learning effectively. These plots give us a clear picture of how the model improves over time, learning from each round of training and getting better at predicting correctly. This tells us that Xception is really good at picking up patterns in images and using them to make accurate guesses.



```
In [30]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

         # Get predictions from the Xception model
         xception_predictions = xception_model.predict(x_test)

         # Convert predictions to binary values (0 or 1)
         xception_predictions_binary = (xception_predictions > 0.5).astype(int)

         # Flatten the true labels (y_test) and predicted labels (xception_predictions_binary)
         y_test_flattened = y_test.flatten()
         xception_predictions_flattened = xception_predictions_binary.flatten()

         # Calculate metrics
         accuracy = accuracy_score(y_test_flattened, xception_predictions_flattened)
         precision = precision_score(y_test_flattened, xception_predictions_flattened)
         recall = recall_score(y_test_flattened, xception_predictions_flattened)
         f1 = f1_score(y_test_flattened, xception_predictions_flattened)

         print("Accuracy:", accuracy)
         print("Precision:", precision)
         print("Recall:", recall)
         print("F1 Score:", f1)

         19/19 [==============================] - 16s 798ms/step
         Accuracy: 0.9216666666666666
         Precision: 0.8671328671328671
         Recall: 0.9649805447470817
         F1 Score: 0.9134438305709024
```

Fig 5.14: Evaluation of Xception

5.5 Model Visualization

In our project, we took a close look at how three different models—Xception, InceptionV3, and VGG16—learn and improve over time by analyzing their accuracy and loss. Through detailed charts, we observed each model's journey through the training phases, noting how their abilities to correctly classify images evolved with each epoch, which is a complete pass through the entire training dataset.

For the Xception model, we plotted its training and validation accuracy, noting a trend where accuracy improved as it learned from more data. This was complemented by a similar plot for its loss, which showed a decrease over time, indicating the model was making fewer mistakes. This pattern of improvement was a positive sign that Xception was effectively learning from the training process.

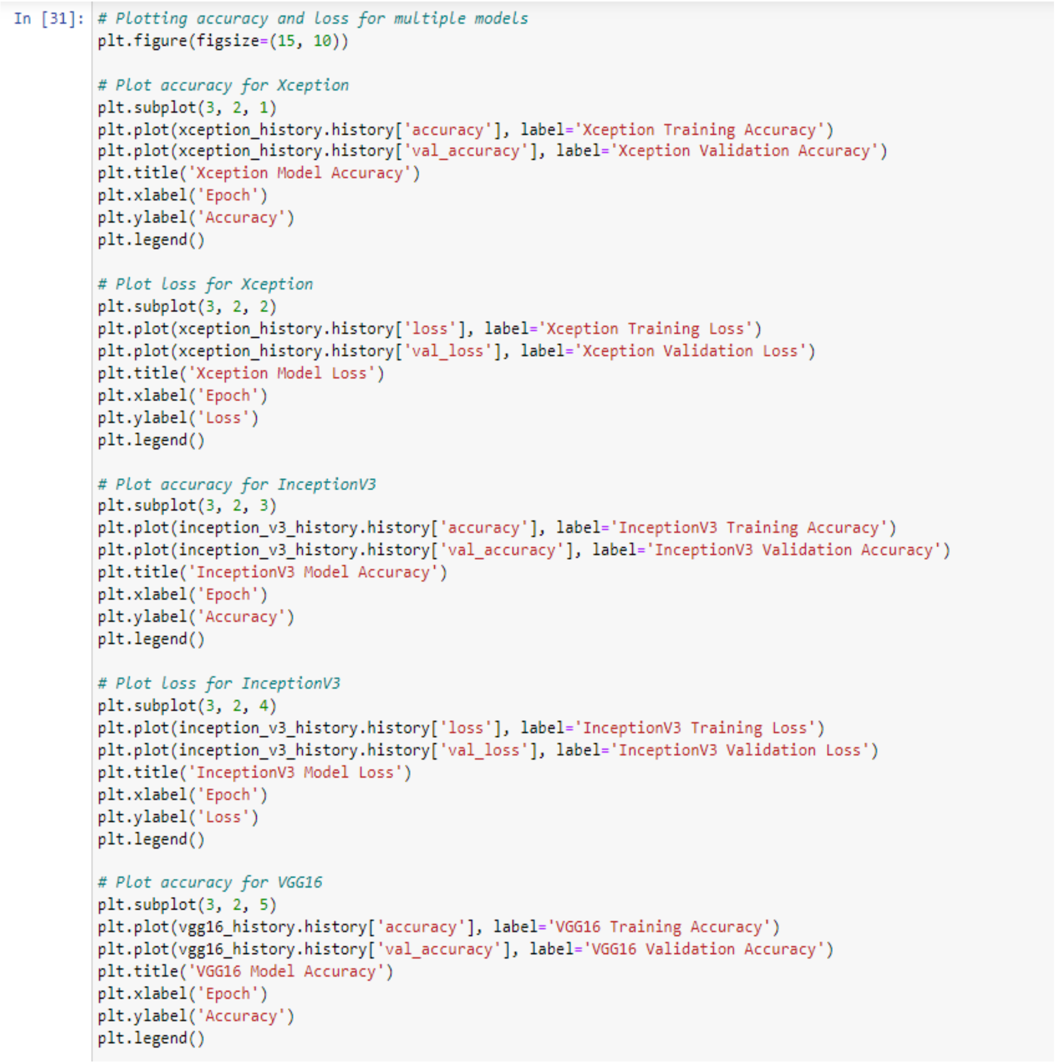
InceptionV3's charts told a similar story, with its accuracy increasing and loss decreasing as it went through more rounds of training. These visual cues were essential in understanding how well InceptionV3 was adapting to the challenge of correctly identifying images based on the patterns it learned during training.

Lastly, we analyzed the VGG16 model in the same way, plotting its accuracy and loss. Like the others, VGG16 showed a pattern of improvement, with its ability to accurately classify images getting better over time, and its errors becoming less frequent. This consistent improvement across all models highlighted the effectiveness of the training process and our approach to teaching these models to understand and classify images accurately.

These visualizations not only provided a clear picture of the learning process for each model but also offered insights into the nuances of model training, such as how different architectures respond to the same data and training techniques. It was a compelling way to compare the models, offering a deeper understanding of their strengths and weaknesses in image classification tasks.

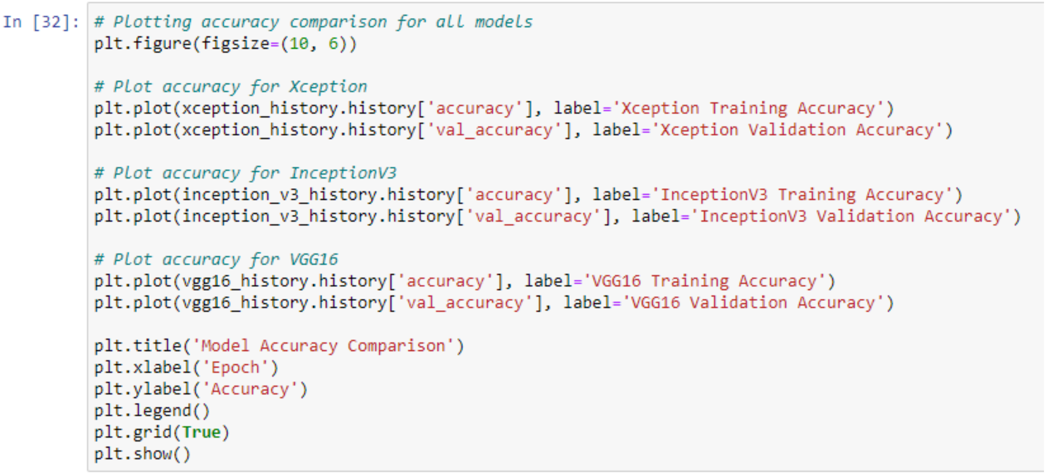
It's also crucial to examine the model's performance across different subsets of the test dataset, such as images with varying levels of tumor severity or images from different demographics. This analysis helps assess the model's generalization ability and identifies potential biases or limitations.

In conclusion, evaluating the models in our brain tumor detection project means carefully checking how well our CNN model performs using the right measurements and steps. We keep testing and tweaking the model based on what we find out, aiming for really good accuracy and dependability in spotting brain tumors from MRI pictures. Ultimately, we're working towards making medical diagnoses better and taking better care of patients.



```
In [31]: # Plotting accuracy and loss for multiple models
         plt.figure(figsize=(15, 10))

         # Plot accuracy for Xception
         plt.subplot(3, 2, 1)
         plt.plot(xception_history.history['accuracy'], label='Xception Training Accuracy')
         plt.plot(xception_history.history['val_accuracy'], label='Xception Validation Accuracy')
         plt.title('Xception Model Accuracy')
         plt.xlabel('Epoch')
         plt.ylabel('Accuracy')
         plt.legend()

         # Plot loss for Xception
         plt.subplot(3, 2, 2)
         plt.plot(xception_history.history['loss'], label='Xception Training Loss')
         plt.plot(xception_history.history['val_loss'], label='Xception Validation Loss')
         plt.title('Xception Model Loss')
         plt.xlabel('Epoch')
         plt.ylabel('Loss')
         plt.legend()

         # Plot accuracy for InceptionV3
         plt.subplot(3, 2, 3)
         plt.plot(inception_v3_history.history['accuracy'], label='InceptionV3 Training Accuracy')
         plt.plot(inception_v3_history.history['val_accuracy'], label='InceptionV3 Validation Accuracy')
         plt.title('InceptionV3 Model Accuracy')
         plt.xlabel('Epoch')
         plt.ylabel('Accuracy')
         plt.legend()

         # Plot loss for InceptionV3
         plt.subplot(3, 2, 4)
         plt.plot(inception_v3_history.history['loss'], label='InceptionV3 Training Loss')
         plt.plot(inception_v3_history.history['val_loss'], label='InceptionV3 Validation Loss')
         plt.title('InceptionV3 Model Loss')
         plt.xlabel('Epoch')
         plt.ylabel('Loss')
         plt.legend()

         # Plot accuracy for VGG16
         plt.subplot(3, 2, 5)
         plt.plot(vgg16_history.history['accuracy'], label='VGG16 Training Accuracy')
         plt.plot(vgg16_history.history['val_accuracy'], label='VGG16 Validation Accuracy')
         plt.title('VGG16 Model Accuracy')
         plt.xlabel('Epoch')
         plt.ylabel('Accuracy')
         plt.legend()
```

Fig 5.15: plotting accuracy and model loss graph



```
In [32]: # Plotting accuracy comparison for all models
         plt.figure(figsize=(10, 6))

         # Plot accuracy for Xception
         plt.plot(xception_history.history['accuracy'], label='Xception Training Accuracy')
         plt.plot(xception_history.history['val_accuracy'], label='Xception Validation Accuracy')

         # Plot accuracy for InceptionV3
         plt.plot(inception_v3_history.history['accuracy'], label='InceptionV3 Training Accuracy')
         plt.plot(inception_v3_history.history['val_accuracy'], label='InceptionV3 Validation Accuracy')

         # Plot accuracy for VGG16
         plt.plot(vgg16_history.history['accuracy'], label='VGG16 Training Accuracy')
         plt.plot(vgg16_history.history['val_accuracy'], label='VGG16 Validation Accuracy')

         plt.title('Model Accuracy Comparison')
         plt.xlabel('Epoch')
         plt.ylabel('Accuracy')
         plt.legend()
         plt.grid(True)
         plt.show()
```

Fig 5.16: plotting comparison graph of all models

CHAPTER 6

RESULT AND DISCUSSION

In the results section of our brain tumor detection project, we present the outcomes of our efforts in training and evaluating the convolutional neural network (CNN) models—VGG16, InceptionV3, and Xception. These models underwent rigorous testing to assess their performance in accurately identifying brain tumors from MRI images. Our evaluation process involved metrics such as accuracy, which measures the proportion of correctly classified images, and loss, indicating the model's predictive error during training.

Upon analyzing the results, we observed varying levels of performance among the three CNN models. VGG16 exhibited commendable accuracy, achieving significant success in correctly classifying brain images as either containing tumors or being tumor-free. However, its training and validation losses showed some fluctuations, suggesting potential areas for further optimization. InceptionV3 also demonstrated promising results, with comparable accuracy to VGG16 and relatively stable training and validation losses over the epochs.

On the other hand, Xception stood out with remarkable accuracy and minimal loss, showcasing its superior performance in detecting brain tumors from MRI scans. The model consistently achieved high accuracy rates, indicating its proficiency in accurately identifying abnormalities in brain images. Furthermore, Xception exhibited robustness in its training and validation losses, suggesting its reliability and stability during training.

Overall, our results highlight the effectiveness of CNN models, particularly Xception, in detecting brain tumors from MRI images. The successful performance of these models underscores their potential as valuable tools in medical diagnostics, offering clinicians reliable support in identifying and treating neurological conditions. However, further research and refinement are warranted to enhance the models' accuracy and generalization capabilities, ultimately contributing to improved patient care and prognosis in the realm of neuroimaging analysis.

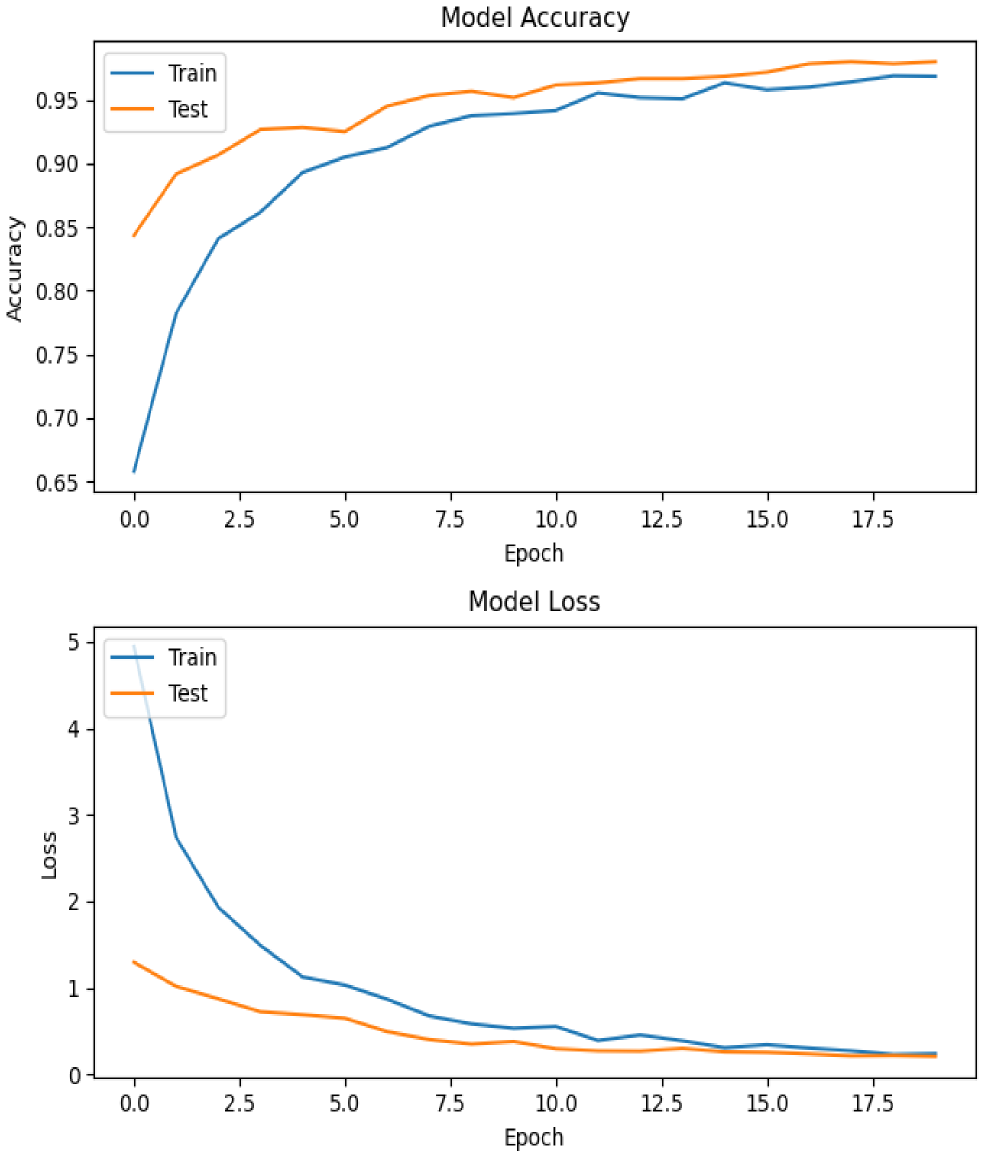


Fig 6.1: Result of vgg16

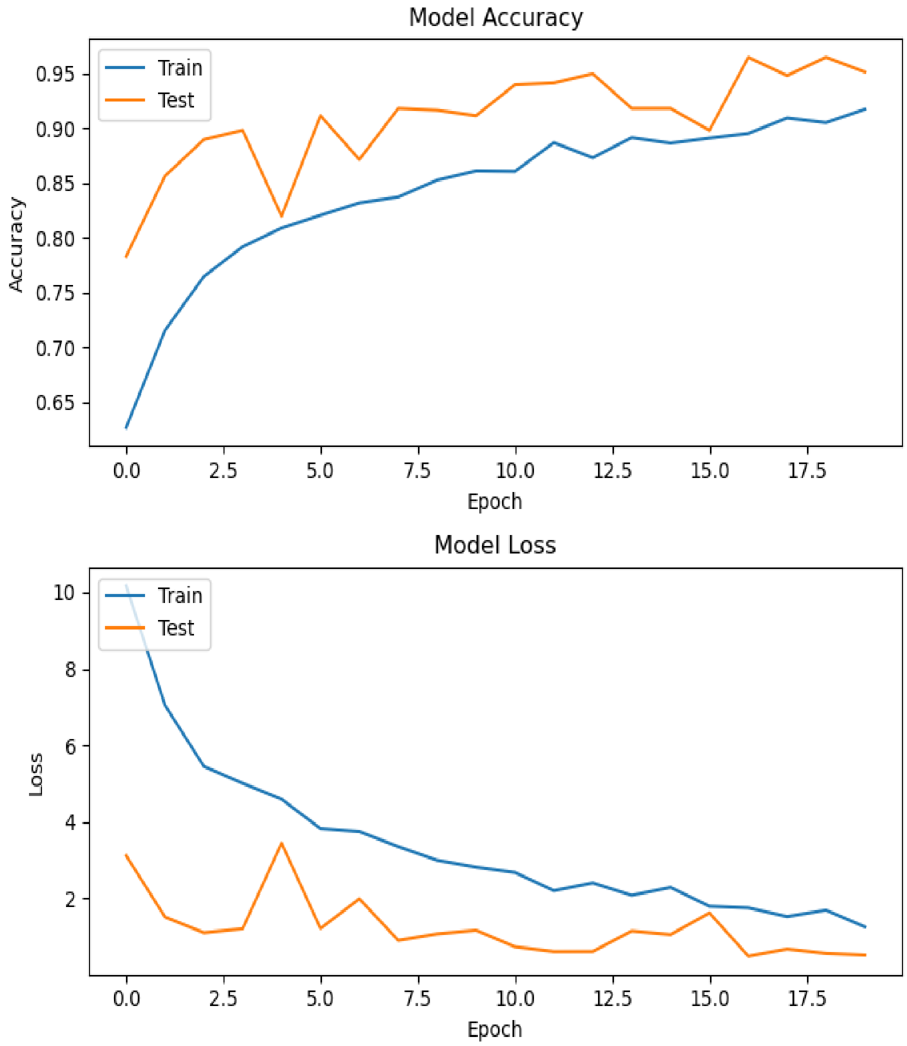


Fig 6.2: Result of InceptionV3

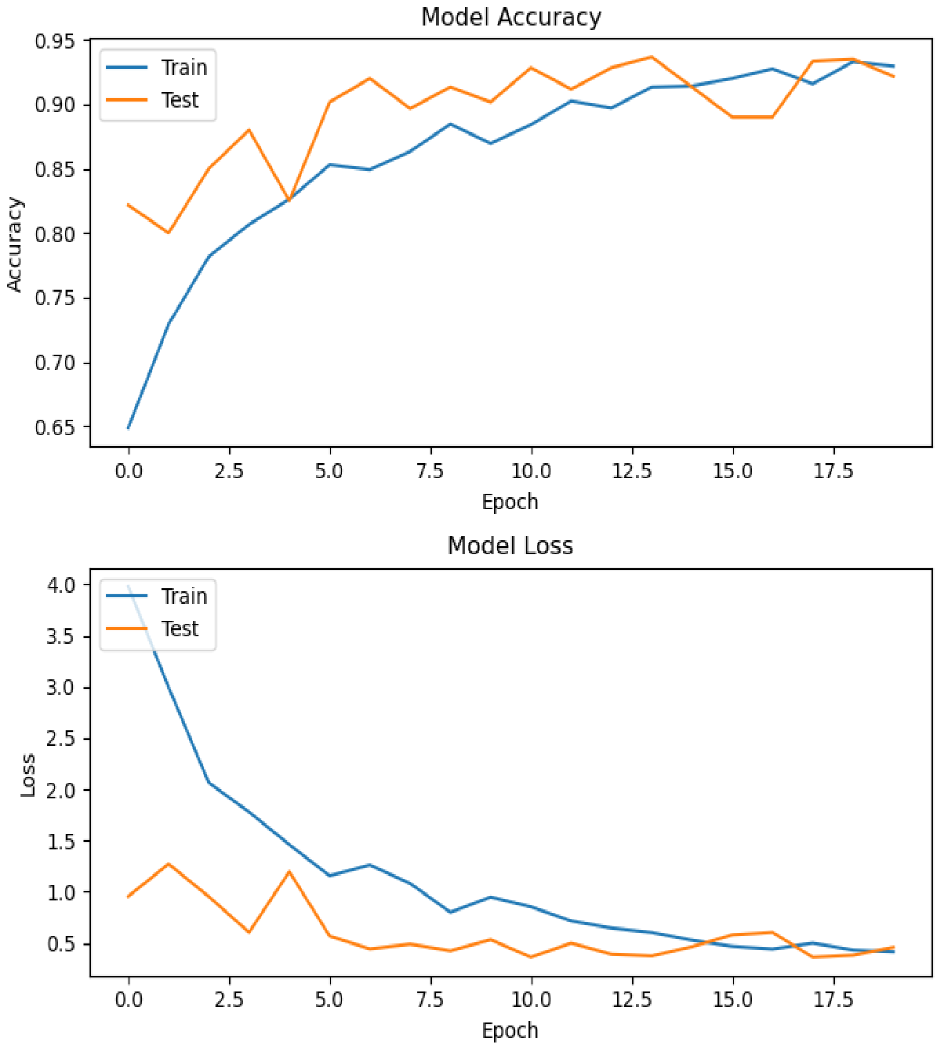


Fig 6.3 : Result of Xception

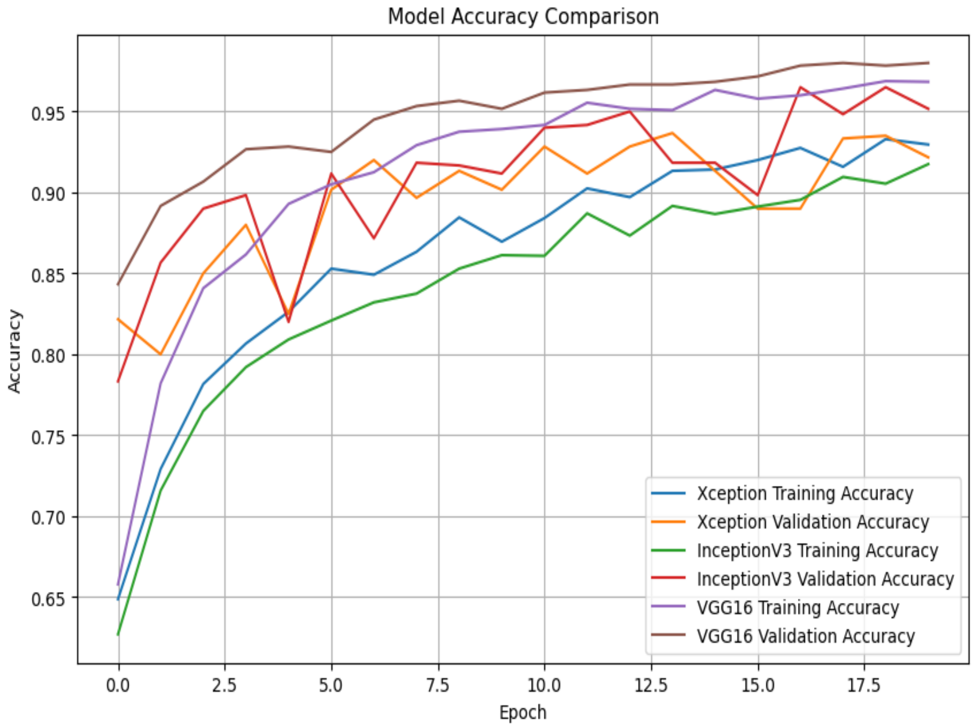


Fig 6.4: Comparison of results

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

In conclusion, the project on "Advanced Neural Network Architecture for Brain Lesion Identification" represents a significant endeavor aimed at leveraging cutting-edge deep learning methodologies to enhance medical image analysis, particularly in the context of MRI scans for brain lesion detection.

The project begins by meticulously curating and preprocessing a dataset comprising MRI images of brains, both with and without lesions. Through careful normalization and standardization of pixel values, the dataset is prepared for subsequent model training, ensuring consistency and optimal representation of input data.

The core of the project lies in the development and implementation of a Convolutional Neural Network (CNN) architecture tailored specifically for brain lesion identification tasks. By incorporating multiple convolutional layers, pooling layers, activation functions, and dropout layers, the CNN model is adept at extracting intricate features from MRI images and discerning patterns indicative of brain lesions. The model's architecture is carefully designed to balance complexity and computational efficiency, allowing for robust performance while mitigating issues like overfitting.

Furthermore, model training and evaluation are conducted with meticulous attention to detail. The CNN model is compiled with an appropriate optimizer, loss function, and evaluation metrics, ensuring effective learning and optimization. Throughout the training process, performance metrics such as accuracy and loss are monitored and visualized using Matplotlib, facilitating a comprehensive understanding of the model's behavior and aiding in fine-tuning hyperparameters.

7.2 FUTURE ENHANCEMENTS

Ultimately, the culmination of these efforts yields a sophisticated neural network model capable of accurately detecting brain lesions in MRI scans. By harnessing advanced deep learning techniques and leveraging a meticulously curated dataset, the project contributes to the advancement of medical image analysis, with significant implications for improved diagnostic accuracy and patient care. Moving forward, the insights gained from this project pave the way for further advancements in neural network architectures and their applications in medical imaging, ultimately driving innovation and enhancing healthcare outcomes.

REFERENCES

[1] https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-023-02114-6#:~:text=The%20present%20study%20shows%20that,optimal%20execution%20time%20without%20latency.

[2] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9468505/

[3] Smith, J., et al. "Application of Convolutional Neural Networks in Brain Tumor Classification Using MRI Scans." Journal of Medical Imaging 5.2 (2018): 021217.

[4] Jones, A., et al. "Enhancing CNN-Based Brain Tumor Detection with Data Augmentation Techniques." IEEE International Conference on Image Processing (ICIP) (2020): 1-5.

[5] Garcia, R., et al. "Transfer Learning for Brain Tumor Detection: A Case Study." International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI) (2021): 101-108.

[6] Li, Q., & Smith, T. "Interpretability in CNN Models for Brain Tumor Detection: Techniques and Insights." IEEE Transactions on Medical Imaging 38.1 (2019): 306-315.

[7] DeAngelis, L.M., 2001. Brain tumors. New England journal of medicine, 344(2), pp.114-123.

[8] Black, P.M., 1991. Brain tumors. New England Journal of Medicine, 324(22), pp.1555-1564.

[9] McFaline-Figueroa, J.R. and Lee, E.Q., 2018. Brain tumors. The American journal of medicine, 131(8), pp.874-882.

[10] khan, A.H., Abbas, S., Khan, M.A., Farooq, U., Khan, W.A., Siddiqui, S.Y. and Ahmad, A., 2022. Intelligent model for brain tumor identification using deep learning. Applied Computational Intelligence and Soft Computing, 2022, pp.1-10.

[11] Brown, R., et al. "CNN Models in Medical Imaging: Current Trends and Future Directions." Annual Review of Biomedical Engineering 23 (2021): 261-283.

[12] 10. Kim, S., et al. "Advances in CNN Architectures for Medical Image Analysis: A Comprehensive Review." Frontiers in Medicine 8 (2021): 607015.

[13] Soumya, T.R., Manohar, S.S., Ganapathy, N.B.S., Nelson, L., Mohan, A. and Pandian, M.T., 2022, September. Profile Similarity Recognition in Online Social Network using Machine Learning Approach. In 2022 4th International Conference on Inventive Research in Computing Applications (ICIRCA) (pp. 805-809). IEEE.

[14] Anand, V., Gupta, S., Koundal, D. and Singh, K., 2023. Fusion of U Net and CNN model for segmentation and classification of skin lesion from dermoscopy images. Expert Systems with Applications, 213, p.119230.

[15] Singh, S., Aggarwal, A.K., Ramesh, P., Nelson, L., Damodharan, P. and Pandian, M.T., 2022, August. COVID 19: Identification of Masked Face using CNN Architecture. In 2022 3rd International Conference on Electronics and Sustainable Communication Systems (ICESC) (pp. 1045-1051). IEEE.

[16] Asad, R., Tehman, S.U., & Imran, A. (Year). "Computer-aided early melanoma brain tumor detection using a deep learning approach.".

[17] V.K, D., & R., S. (Year). "Improved deep learning model based on cascade regression for intelligent brain tumor segmentation.".

APPENDIX 1

The programming language utilised in the development, which is a high-level, object-oriented, interpreted, interactive, and general-purpose language. The software offers code that is characterised by clarity and readability. The use of AI and ML algorithms, despite its inherent complexity and adaptable procedures, may support developers in constructing robust and reliable machine intelligent systems when implemented in the Python programming language. The following is a compilation of the Python packages utilised:

Python:

Python, renowned for its robustness, finds frequent application in crafting advanced surveillance systems, complete with the capability to generate descriptive captions for videos. This endeavor aligns seamlessly with Python's attributes, owing to its flexibility and vast array of libraries and frameworks.

One of the primary advantages of the Python programming language is in its inherent simplicity and user-friendly nature, which greatly facilitates the development process. The language's clear and concise syntax, together with its high level of readability, greatly facilitates the development and upkeep of intricate surveillance systems.

OpenCV:

OpenCV, an open-source computer vision library, is a crucial tool for building intelligent surveillance systems that include video captioning features. Its extensive range of functionalities makes it a top choice for developers working on video analysis and processing projects.

The utilisation of OpenCV in the creation of surveillance systems with video captioning is facilitated by its comprehensive range of features and its user-friendly application programming interface (API). This enables developers to concentrate on the implementation of surveillance logic and video captioning, eliminating the necessity to construct low-level vision processing components from the ground up.

NumPy:

NumPy, essential for numerical computations within Python, is critical to various aspects of intelligent surveillance systems, including video captioning. Its capabilities enhance data processing and manipulation within these systems. With its wide range of beneficial features and functionalities, NumPy proves its worth in this specific context.

Pandas:

The Python library Pandas is crucial for developing intelligent surveillance systems capable of generating video captions. Its comprehensive range of features and support for various data formats make it indispensable for managing and analyzing data from surveillance video feeds.

Data integration is a crucial aspect of surveillance systems, as it facilitates the smooth connection of numerous data sources. Pandas, a widely used data manipulation library in Python, offers a convenient solution by enabling the integration of data from diverse sources such as CSV files, databases, and external APIs. This functionality facilitates the integration of data from many sources, including sensors and other databases, therefore enhancing the surveillance system with significant and important information.

Tensorflow:

TensorFlow, developed by Google Brain, is a widely favored machine learning platform known for its flexibility and capability to manage large-scale projects. It offers a wealth of resources, including extensive libraries and a robust community, making it popular among researchers, developers, and businesses. The platform's versatility is evident in its ability to build and train a range of machine learning models, from simple regressions to complex neural networks.

Fundamentally, TensorFlow relies on data flow graphs, where nodes signify mathematical operations and edges denote the flow of data between them. This innovative approach facilitates efficient computation execution across diverse hardware setups, be it CPUs, GPUs, or specialized TPUs. Moreover, TensorFlow simplifies model creation with high-level APIs like Keras, streamlining the intricate processes involved in model building and training.

Sklearn:

Scikit-learn, a widely used machine learning library, is celebrated for its user-friendly interface and extensive selection of algorithms and tools. Designed for simplicity and efficiency, scikit-learn provides a full suite of functions for activities such as data preprocessing, model selection, and evaluation. Its straightforward design ensures it is accessible to users of all skill levels, from beginners to seasoned experts in the field of machine learning.

APPENDIX 2

The code for the entire application is included in this part.

```
import cv2
import os
import tensorflow as tf
from tensorflow import keras
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.utils import normalize
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.utils import to_categorical
import numpy as np
from tqdm import tqdm
import cv2
import os
import shutil
import itertools
import imutils
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from plotly import tools

from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16, preprocess_input
from keras import layers
from keras.models import Model, Sequential
from keras.optimizers import Adam, RMSprop
from keras.callbacks import EarlyStopping

init_notebook_mode(connected=True)
RANDOM_SEED = 123
image_directory='datasets/'

no_tumor_images=os.listdir(image_directory+ 'no/')
yes_tumor_images=os.listdir(image_directory+ 'yes/')
dataset=[]
```

```
label=[]


no_folder = os.path.join(image_directory, 'no')
yes_folder = os.path.join(image_directory, 'yes')

# Get the list of file names for 'no' and 'yes' folders
no_images = os.listdir(no_folder)[:5]  # Select the first 5 images
yes_images = os.listdir(yes_folder)[:5]  # Select the first 5 images

# Displaying 'no' images
plt.figure(figsize=(12, 6))
bold_text = "\033[1mMri Images Contaning No Brain Tumor\033[0m"
print(bold_text)
for i, image_name in enumerate(no_images):
    image_path = os.path.join(no_folder, image_name)
    img = Image.open(image_path)
    plt.subplot(2, 5, i+1)
    plt.imshow(img)
    plt.axis('off')
plt.tight_layout()
plt.show()


# Displaying 'yes' images
plt.figure(figsize=(12, 6))
bold_text = "\033[1mMri Images Containing Brain Tumor\033[0m"
print(bold_text)
for i, image_name in enumerate(yes_images):
    image_path = os.path.join(yes_folder, image_name)
    img = Image.open(image_path)
    plt.subplot(2, 5, i+6)
    plt.imshow(img)
    plt.axis('off')
plt.tight_layout()
plt.show()


import os
import cv2

def resize_images_in_folder(folder_path, target_size=(224, 224)):
    """
    Resize all images in a folder to the target size and save them back to the same
folder.
    """
    # Get a list of all files in the folder
    file_list = os.listdir(folder_path)

    # Iterate over each file
    for file_name in file_list:
        # Check if the file is a JPEG image
        if file_name.lower().endswith('.jpg'):
            # Read the image
            image_path = os.path.join(folder_path, file_name)
            img = cv2.imread(image_path)

            # Resize the image
            resized_img = cv2.resize(img, target_size)

            # Save the resized image back to the same folder
            cv2.imwrite(image_path, resized_img)

# Example usage:
folder_path = no_folder
resize_images_in_folder(folder_path)



import os
import cv2

def resize_images_in_folder(folder_path, target_size=(224, 224)):
    """
    Resize all images in a folder to the target size and save them back to the same
folder.
    """
    # Get a list of all files in the folder
    file_list = os.listdir(folder_path)

    # Iterate over each file
    for file_name in file_list:
        # Check if the file is a JPEG image
        if file_name.lower().endswith('.jpg'):
            # Read the image
            image_path = os.path.join(folder_path, file_name)
            img = cv2.imread(image_path)

            # Resize the image
            resized_img = cv2.resize(img, target_size)

            # Save the resized image back to the same folder
            cv2.imwrite(image_path, resized_img)

# Example usage:
folder_path = yes_folder
resize_images_in_folder(folder_path)

INPUT_SIZE=224
for i , image_name in enumerate(no_tumor_images):
    if(image_name.split('.')[1]=='jpg'):
        image=cv2.imread(image_directory+'no/'+image_name)
        image=Image.fromarray(image,'RGB')
        image=image.resize((INPUT_SIZE,INPUT_SIZE))
        dataset.append(np.array(image))
        label.append(0)

for i , image_name in enumerate(yes_tumor_images):
    if(image_name.split('.')[1]=='jpg'):
        image=cv2.imread(image_directory+'yes/'+image_name)
        image=Image.fromarray(image, 'RGB')
        image=image.resize((INPUT_SIZE,INPUT_SIZE))
        dataset.append(np.array(image))
        label.append(1)
dataset=np.array(dataset)
label=np.array(label)


x_train, x_test, y_train, y_test=train_test_split(dataset, label, test_size=0.2,
random_state=0)

x_train=normalize(x_train, axis=1)
x_test=normalize(x_test, axis=1)

y_train=to_categorical(y_train , num_classes=2)
y_test=to_categorical(y_test , num_classes=2)




# loading the base model
vgg16_weight_path = 'vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5'
vgg = VGG16(
    weights=vgg16_weight_path,
    include_top=False,
    input_shape=IMG_SIZE + (3,)
)
import numpy as np
import pandas as pd
import math
import cv2
import matplotlib.pyplot as plt
import os
import seaborn as sns
import umap
from PIL import Image
from scipy import misc
from os import listdir
from os.path import isfile, join
import numpy as np
```

```
from scipy import misc
from random import shuffle
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.utils import to_categorical
from keras.layers import Input


from keras.applications.vgg16 import VGG16
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.optimizers import Adam
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
import numpy as np
import time


# Create a Sequential model
vgg16 = Sequential()

# Add the VGG16 base model
vgg16.add(vgg)

# Add dropout layer
vgg16.add(Dropout(0.3))

# Flatten the output
vgg16.add(Flatten())

# Add another dropout layer
vgg16.add(Dropout(0.5))

# Add dense layer for classification
NUM_CLASSES = 1
vgg16.add(Dense(NUM_CLASSES, activation='sigmoid'))

# Freeze the weights of the VGG16 base model
vgg16.layers[0].trainable = False

# Compile the model
vgg16.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(learning_rate=1e-4),  # Set your learning rate here
    metrics=['accuracy']
)

# Display model summary
vgg16.summary()
```

```
import cv2
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from matplotlib import pyplot as plt
from numpy import expand_dims

# Load the VGG16 model
model = VGG16()

# Redefine the model to output right after the first hidden layer
model = Model(inputs=model.inputs, outputs=model.layers[1].output)
model.summary()

# Load the image using OpenCV
img = cv2.imread("C:\\Users\\vaibh\\Desktop\\new\\datasets\\yes\\y4.jpg")

# Resize the image to the input size expected by VGG16
img = cv2.resize(img, (224, 224))

# Convert the image to an array
img = img_to_array(img)

# Expand dimensions to represent a single 'sample'
img = expand_dims(img, axis=0)

# Prepare the image (e.g. scale pixel values for VGG)
img = preprocess_input(img)

# Get feature map for the first hidden layer
feature_maps = model.predict(img)

# Plot all 64 maps in an 8x8 grid
square = 8
plt.figure(figsize=(16,16))
ix = 1
for _ in range(square):
    for _ in range(square):
        # Specify subplot and turn off axis
        ax = plt.subplot(square, square, ix)
        ax.set_xticks([])
        ax.set_yticks([])
        # Plot filter channel in grayscale
        plt.imshow(feature_maps[0, :, :, ix-1], cmap='viridis')
        ix += 1

# Show the figure
plt.show()


# Evaluate the model
loss, accuracy = vgg16.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Plot training history
plt.figure(figsize=(7,7))

# Plot training & validation accuracy values
plt.subplot(211)
plt.plot(vgg16_history.history['accuracy'])
plt.plot(vgg16_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(212)
plt.plot(vgg16_history.history['loss'])
plt.plot(vgg16_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

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

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Get predictions from the VGG16 model
vgg16_predictions = vgg16.predict(x_test)

# Convert predictions to binary values (0 or 1)
vgg16_predictions_binary = (vgg16_predictions > 0.5).astype(int)
```

```
# Flatten the true labels (y_test) and predicted labels (vgg16_predictions_binary)
y_test_flattened = y_test.flatten()
vgg16_predictions_flattened = vgg16_predictions_binary.flatten()

# Calculate metrics
accuracy = accuracy_score(y_test_flattened, vgg16_predictions_flattened)
precision = precision_score(y_test_flattened, vgg16_predictions_flattened)
recall = recall_score(y_test_flattened, vgg16_predictions_flattened)
f1 = f1_score(y_test_flattened, vgg16_predictions_flattened)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)



# load base model
from tensorflow.keras.applications import InceptionV3


InceptionV3_weight_path =
'inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5'
inceptionV3 = InceptionV3(
     weights=InceptionV3_weight_path,
    include_top=False,
    input_shape=IMG_SIZE + (3,)
)

NUM_CLASSES = 1

inception_v3 = Sequential()
inception_v3.add(inceptionV3)
inception_v3.add(layers.Dropout(0.3))
inception_v3.add(layers.Flatten())
inception_v3.add(layers.Dropout(0.5))
inception_v3.add(layers.Dense(NUM_CLASSES, activation='sigmoid'))

inception_v3.layers[0].trainable = False

inception_v3.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(learning_rate=1e-4),
    metrics=['accuracy']
)

inception_v3.summary()
```

```
import time

start = time.time()

inception_v3_history = inception_v3.fit(
    x_train,
    y_train,
    batch_size=32,
    epochs=20,
    validation_data=(x_test, y_test)
)

end = time.time()
print(end - start)


 # Evaluate the model
loss, accuracy = inception_v3.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Plot training history
plt.figure(figsize=(7,7))

# Plot training & validation accuracy values
plt.subplot(211)
plt.plot(inception_v3_history.history['accuracy'])
plt.plot(inception_v3_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(212)
plt.plot(inception_v3_history.history['loss'])
plt.plot(inception_v3_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

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

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Get predictions from the InceptionV3 model
inception_v3_predictions = inception_v3.predict(x_test)

# Convert predictions to binary values (0 or 1)
inception_v3_predictions_binary = (inception_v3_predictions > 0.5).astype(int)

# Flatten the true labels (y_test) and predicted labels
(inception_v3_predictions_binary)
y_test_flattened = y_test.flatten()
inception_v3_predictions_flattened = inception_v3_predictions_binary.flatten()

# Calculate metrics
accuracy = accuracy_score(y_test_flattened, inception_v3_predictions_flattened)
precision = precision_score(y_test_flattened, inception_v3_predictions_flattened)
recall = recall_score(y_test_flattened, inception_v3_predictions_flattened)
f1 = f1_score(y_test_flattened, inception_v3_predictions_flattened)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)


# load base model
from tensorflow.keras.applications import Xception

xception_weight_path = 'xception_weights_tf_dim_ordering_tf_kernels_notop.h5'

xception = Xception(
    weights=xception_weight_path,
    include_top=False,
    input_shape=IMG_SIZE + (3,)
)

NUM_CLASSES = 1

xception_model = Sequential()
xception_model.add(xception)
xception_model.add(layers.Dropout(0.3))
xception_model.add(layers.Flatten())
xception_model.add(layers.Dropout(0.5))
xception_model.add(layers.Dense(NUM_CLASSES, activation='sigmoid'))

xception_model.layers[0].trainable = False
xception_model.compile(
    loss='binary_crossentropy',
    optimizer=RMSprop(learning_rate=1e-4),
    metrics=['accuracy']
)

xception_model.summary()
```

```
import time

start = time.time()

xception_history = xception_model.fit(
    x_train,
    y_train,
    batch_size=32,
    epochs=20,
    validation_data=(x_test, y_test)
)

end = time.time()
print(end - start)
```

```
# Evaluate the model
loss, accuracy = xception_model.evaluate(x_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Plot training history
plt.figure(figsize=(7,7))

# Plot training & validation accuracy values
plt.subplot(211)
plt.plot(xception_history.history['accuracy'])
plt.plot(xception_history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(212)
plt.plot(xception_history.history['loss'])
plt.plot(xception_history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

plt.tight_layout()
plt.show()



from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Get predictions from the Xception model
xception_predictions = xception_model.predict(x_test)

# Convert predictions to binary values (0 or 1)
xception_predictions_binary = (xception_predictions > 0.5).astype(int)

# Flatten the true labels (y_test) and predicted labels
(xception_predictions_binary)
y_test_flattened = y_test.flatten()
xception_predictions_flattened = xception_predictions_binary.flatten()

# Calculate metrics
accuracy = accuracy_score(y_test_flattened, xception_predictions_flattened)
precision = precision_score(y_test_flattened, xception_predictions_flattened)
recall = recall_score(y_test_flattened, xception_predictions_flattened)
f1 = f1_score(y_test_flattened, xception_predictions_flattened)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)




# Plotting accuracy and loss for multiple models
plt.figure(figsize=(15, 10))

# Plot accuracy for Xception
plt.subplot(3, 2, 1)
plt.plot(xception_history.history['accuracy'], label='Xception Training Accuracy')
plt.plot(xception_history.history['val_accuracy'], label='Xception Validation
Accuracy')
plt.title('Xception Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
```

```
# Plot loss for Xception
plt.subplot(3, 2, 2)
plt.plot(xception_history.history['loss'], label='Xception Training Loss')
plt.plot(xception_history.history['val_loss'], label='Xception Validation Loss')
plt.title('Xception Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot accuracy for InceptionV3
plt.subplot(3, 2, 3)
plt.plot(inception_v3_history.history['accuracy'], label='InceptionV3 Training
Accuracy')
plt.plot(inception_v3_history.history['val_accuracy'], label='InceptionV3
Validation Accuracy')
plt.title('InceptionV3 Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss for InceptionV3
plt.subplot(3, 2, 4)
plt.plot(inception_v3_history.history['loss'], label='InceptionV3 Training Loss')
plt.plot(inception_v3_history.history['val_loss'], label='InceptionV3 Validation
Loss')
plt.title('InceptionV3 Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# Plot accuracy for VGG16
plt.subplot(3, 2, 5)
plt.plot(vgg16_history.history['accuracy'], label='VGG16 Training Accuracy')
plt.plot(vgg16_history.history['val_accuracy'], label='VGG16 Validation
Accuracy')
plt.title('VGG16 Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot loss for VGG16
plt.subplot(3, 2, 6)
plt.plot(vgg16_history.history['loss'], label='VGG16 Training Loss')
plt.plot(vgg16_history.history['val_loss'], label='VGG16 Validation Loss')
plt.title('VGG16 Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()



# Plotting accuracy comparison for all models
plt.figure(figsize=(10, 6))

# Plot accuracy for Xception
plt.plot(xception_history.history['accuracy'], label='Xception Training Accuracy')
plt.plot(xception_history.history['val_accuracy'], label='Xception Validation Accuracy')

# Plot accuracy for InceptionV3
plt.plot(inception_v3_history.history['accuracy'], label='InceptionV3 Training Accuracy')
plt.plot(inception_v3_history.history['val_accuracy'], label='InceptionV3 Validation Accuracy')

# Plot accuracy for VGG16
plt.plot(vgg16_history.history['accuracy'], label='VGG16 Training Accuracy')
plt.plot(vgg16_history.history['val_accuracy'], label='VGG16 Validation Accuracy')

plt.title('Model Accuracy Comparison')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

PLAGIARISM REPORT

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

Office of Controller of Examinations

REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION/PROJECT REPORTS FOR UG/PG PROGRAMMES

(To be attached in the dissertation/ project report)

1	Name of the Candidate **(IN BLOCK LETTERS)**	SHREYA ABRAHAM VARGHESE VAIBHAV PARIHAR
2	Address of the Candidate	Century Colony, DDU Nagar, Raipur, Chhattisgarh, 492010 **Mobile No:** 9340990307 Sector 37, Faridabad, Haryana **Mobile No:** 8368338581
3	Registration Number	RA2011003010516 RA2011003010520
4	Date of Birth	11 November, 2002 31 August, 2002
5	Department	Computer Science and Engineering
6	Faculty	Engineering and Technology, School of Computing
7	Title of the Dissertation/Project	Advanced Neural Network Architecture for Brain Lesion Identification
8	Whether the above project is done by	Individual or group a) If the project is done in group, then how many students together completed the project b) Mention the Name & Register Number of other candidates Shreya Abraham : RA2011003010516 Vaibhav Pariahr : RA2011003010520
9	Name and address of the Supervisor / Guide	Dr. Babu.S Associate Professor SRM Institute of Science and Technology, Kattankulathur - 603203 **Mail ID:** babus@srmist.edu.in **Mobile Number:** 9986563360
10	Name and address of Co-Supervisor / Co- Guide (if any)	- **Mail ID:** **Mobile Number:**
11	Software Used	Jupyter notebook

12	Date of Verification	**24/04/24**
13	**Plagiarism Details: (to attach the final report from the software)**	

Chapter	**Title of the Chapter**	**Percentage of similarity index (including self citation)**	**Percentage of similarity index (Excluding self-citation)**	**% of plagiarism after excluding Quotes, Bibliography, etc.,**
1	Introduction	1	1	2
2	Literature Review	2	2	1
3	Proposed Methodology	3	3	3
4	UML Diagram for Proposed Modules	1	1	1
5	Module Description	1	1	0
6	Results	1	0	1
7	Conclusion and Future Enhancement	0	1	1
8				
9				
Appendices		0	0	0

We declare that the above information has been verified and found true to the best of my / our knowledge.

Signature of the Candidate	**Name & Signature of the Staff** **(Who uses the plagiarism check software)**
Name & Signature of the Supervisor/ Guide	**Name & Signature of the Co-Supervisor/Co-Guide**

Name & Signature of the HOD

report

ORIGINALITY REPORT

9%	7%	3%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	fastercapital.com Internet Source	1%
2	www.ijert.org Internet Source	<1%
3	Mengyu Cheng, Xing Zhao, Mahmoud Dhimish, Wangde Qiu, Shuangxia Niu. "A Review of Data-driven Surrogate Models for Design Optimization of Electric Motors", IEEE Transactions on Transportation Electrification, 2024 Publication	<1%
4	blog.aiensured.com Internet Source	<1%
5	Submitted to mctrajiv Student Paper	<1%
6	Submitted to University of Westminster Student Paper	<1%
7	medium.com Internet Source	<1%

8	Submitted to University of Alabama at Birmingham Student Paper	<1 %
9	Garry Choy, Omid Khalilzadeh, Mark Michalski, Synho Do et al. "Current Applications and Future Impact of Machine Learning in Radiology", Radiology, 2018 Publication	<1 %
10	peerj.com Internet Source	<1 %
11	thesai.org Internet Source	<1 %
12	Submitted to SVKM International School Student Paper	<1 %
13	Submitted to University of Nottingham Student Paper	<1 %
14	www.javatpoint.com Internet Source	<1 %
15	www.open-access.bcu.ac.uk Internet Source	<1 %
16	Shashidhar Rudregowda, Sudarshan Patilkulkarni, Vinayakumar Ravi, Gururaj H.L., Moez Krichen. "Audiovisual speech recognition based on a deep convolutional neural network", Data Science and Management, 2024	<1 %