University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project
7PAM2002-0509-2023
Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Advanced Deep Learning for Automated Brain Tumor Detection and Classification via MRI Images

Student Name and SRN:

Sohan Kumar Narayanaswamy | STUDENT ID: 22085239

Supervisor: Alyssa Drake

Date Submitted: 29 August 2024

Word Count: 7498

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Sohan Kumar Narayanaswamy

Student Name signature: Sohan.N

Student SRN number: 22085239

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Acknowledgement

I like to convey my sincere appreciation to those individuals who have provided their assistance in assisting me in the successful completion of this project. Initially, I express my gratitude to the Almighty for bestowing upon me the courage and ability required to succeed in this journey.

I am profoundly grateful of my supervisor, Alyssa Drake, for her essential mentorship and assistance that facilitated the effective completion of this project. The prompt feedback and suggestions she provided played a crucial role in enhancing the project and elevating it to a higher standard. I express my gratitude to Carolyn Devereux, the module leader, for her unwavering support and encouragement during this journey.

I would like to express my sincere gratitude to my family for their invaluable mental and emotional support, which played a crucial role in enabling me to successfully accomplish this project. Their unbreakable faith in me provided me with the fortitude to endure and surmount obstacles.

Lastly, I would like to express my gratitude to all of my peers who generously provided their assistance and encouragement throughout this research. Their invaluable efforts enabled me to successfully accomplish this research within the stipulated timeframe and to the utmost of my capabilities.

Abstract

Brain tumor detection in the healthcare sector has become a crucial and challenging endeavor due to the various shapes, locations, and imaging intensity of the tumors. An automated system is crucial in assisting physicians and radiologists in the identification and categorization of brain tumors. This research explores various machine learning and deep learning algorithms often employed for image classification. I have conducted a comparative analysis of various models employed for tumor classification using both machine learning and deep learning techniques. This research is based on MRI scans of Glioma tumor, Pituitary tumor, Meningioma tumor and No tumor. I have examined various methodologies and the accuracy of different classification models using these MRI images. I employed various pre-trained deep learning models to train images of brain tumors. The pre-trained models have demonstrated exceptional performance while using less power and processing time. CNN has achieved the highest accuracy of 99.16% compared to other models and traditional machine learning algorithms. The experimental findings of this model have demonstrated its superiority and efficiency in tumor identification and classification, surpassing other recent studies.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Brain tumors are abnormal and uncontrolled proliferations of cells. Some originate in the brain itself, in which case they are termed primary. Others spread to this location from somewhere else in the body through metastasis, and are termed secondary. Primary brain tumors do not spread to other body sites, and can be malignant or benign. Secondary brain tumors are always malignant. Gliomas are the most common type of brain tumor in adults, accounting for 78% of malignant brain tumors. Meningiomas can occur in various locations and are also a serious concern. Although pituitary cancer is rare, it can cause significant problems by producing excess hormones (Jiang, S., Gu, Y., & Kumar, E. (2023)). Early diagnosis and treatment are essential for reducing the mortality rate associated with brain tumors. The World Health Organization (WHO) states that a correct diagnosis of a brain tumor entails its discovery, localization, and classification based on its degree, kind, and severity. This research comprises finding the tumor, grading it according to type, and classifying it according to grade in the diagnosis of brain tumors using magnetic resonance imaging (MRI). This approach has experimented with using several models rather than a single model for classification task in order to categorize brain MRI data. MRI is used for brain imaging as it is less harmful than CT scans.

The protons and neutrons within an atom's nucleus possess angular momentum, referred to as spin. The spins will nullify when the number of subatomic particles in a nucleus is an even number. Nuclei possessing an odd number of particles will have a resulting spin. This serves as the foundation for magnetic resonance imaging. An MRI scanner utilizes strong magnets to align and stimulate hydrogen nuclei (single proton) in human tissue, generating a detectable signal that is spatially recorded, ultimately producing images of the body. The MRI equipment emits a radio frequency (RF) pulse that selectively attaches to hydrogen atoms. The technology transmits the pulse to the precise region of the body that requires examination. As a result of the RF pulse, protons in that region assimilate the requisite energy to induce a change in their rotational orientation. The term "resonance" in the context of MRI refers to the phenomenon where some atomic nuclei in the body respond to a magnetic field by emitting detectable signals. The RF pulse induces the protons to rotate at the Larmor frequency, in a predetermined direction. The frequency is determined by the specific tissue being scanned and the intensity of the primary magnetic field. MRI employs three electromagnetic fields: a powerful static magnetic field that aligns the hydrogen nuclei, a less intense time-varying field for spatial encoding, and a weak radio frequency field to manipulate the hydrogen nuclei and generate measurable signals. These signals are then captured by a radio frequency antenna. (Joseph et al. (2014)).

1.1 Challenges of the Proposed Approach

Complexity of Brain Tumor Detection: Detecting brain tumors from medical images like MRI scans can be a complex task due to the variability in tumor shapes, sizes, and locations within the brain. This complexity requires advanced algorithms and models to accurately identify and classify tumors.

Need for Advanced Machine Learning: With the increasing demand for advanced machine learning techniques, there is a requirement to leverage sophisticated models like CNNs to

handle the complexities involved in brain tumor detection. These models need to be trained effectively to improve the accuracy of tumor identification.

I**mbalance in Image Data**: Imbalance between normal MRI brain tumor images can lead to potential misdiagnosis if not properly managed.

Image Quality Issues: Degradation and noise in MRI images during acquisition, making them unsuitable for direct analysis. Subtle variations and similarities between original and affected brain regions.

Overfitting: Deep learning models, particularly those with many parameters, are prone to overfitting, especially when trained on small datasets. This can lead to models that perform well on training data but poorly on unseen data.

Computational Resources: Training deep learning models, especially with large datasets and complex architectures like VGG-16, requires substantial computational power and resources, which may not be accessible to all research institutions.

1.2 Motivation for the Proposed Approach

The impetus for this research arises from the necessity to enhance the efficacy of brain tumor detection in medical imaging. Early and precise detection of these tumors is essential for optimizing patient outcomes and customizing effective treatment strategies. The objective of this research project is to utilize advanced machine learning and deep learning techniques to develop a dependable system for the detection and categorization of tumors. The central focus of this research project involves conducting a comprehensive comparative analysis to compare the performance of CNN, VGG-16, KNN, and Random Forest. By evaluating each method on the same dataset, we can gain insights into the strengths and weaknesses of each algorithm in terms of accuracy, speed, and computational efficiency. This comparison will yield valuable insights into the most effective and practical approach for real-world medical applications, enabling us to identify the optimal algorithm for rapid and precise brain tumor detection. Through a comparative analysis of these methods, my aim is to determine the most precise and efficient model for this proposed approach.

1.3 Research Aim and Objectives

The aim of this research project is to create a dependable system for identifying and classifying brain tumors. The study aims to achieve the following objective:

- Create and apply machine learning models, such as CNN, VGG-16, KNN, and Random Forest, designed specifically for identifying brain tumors.
- Optimize the performance of each model by fine-tuning them to accurately detect and classify brain tumors, ensuring their suitability for medical imaging tasks.
- Assess the accuracy, speed, and computational efficacy of each model by employing an extensive dataset of MRI scans, offering a thorough evaluation of their capabilities.
- Perform a comprehensive comparative analysis of the four models, evaluating their performance metrics to determine their individual strengths and weaknesses.
- Suggest the optimal algorithm for fast and accurate brain tumor detection, enabling its implementation in real-life medical environments to improve diagnostic precision and efficiency.

1.4 Ethical Consideration:

As it is a public domain Licensed dataset it is free to use no copyright control. But as this is a medical data, identifiers such as Name and medical records are removed to protect privacy.

Chapter 2

Literature Review

Machine learning and deep learning techniques have revolutionized medical imaging, particularly in the classification and detection of diseases like brain tumors. This literature review investigates the applications and efficacy of CNNs, VGG-16, KNN, and Random Forest algorithms in medical imaging, with a particular emphasis on brain tumor detection and classification.

(Rathi et al., 2014) investigated the classification of MRI brain tumor images by selecting and extracting features using Principal Component Analysis and Linear Discriminant Analysis. The dataset, which included 140 tumor-containing MR images from the Internet Brain Segmentation Repository, was preprocessed and normalized to grayscale. LDA outperformed other methods like KNN and AdaBoost, achieving a classification accuracy of 98.87%. However, issues such as feature redundancy, computational complexity, class variability, and the risk of overfitting were identified. To improve the classification system, the study recommends using advanced feature selection techniques, effective dimensionality reduction methods, continuous classifier training, and combining multiple features (shape, intensity, texture) for comprehensive tumour characterization. These enhancements are intended to improve the robustness and efficiency of MRI brain tumor classification systems.

(Gurusamy, R et al., 2017) studied MRI brain tumor classification using real-time MRI images, using preprocessing to remove noise, feature extraction with wavelet transforms, and classification with a Support Vector Machine (SVM) classifier. The dataset included 140 MRI brain images, and the SVM classifier outperformed other methods like Neural Networks (97%) and K-Nearest Neighbor (96%) with 98% accuracy in both positive and negative predictive values. The identified challenges included noise in MRI images, feature extraction complexity, classification uncertainties, and computational demands. To improve performance, the study recommends advanced denoising techniques, effective feature selection methods, strong classification algorithms, and hybrid approaches that combine multiple filtering methods.

(Machhale, K et al., 2015) present a hybrid classifier for classifying MRI brain images as normal or abnormal using Support Vector Machine (SVM) and K-Nearest Neighbor. The model was tested on 50 MRI images using preprocessing techniques like filtering and skull masking, as well as feature extraction methods like grayscale, texture, and symmetry. The SVM-KNN hybrid classifier outperformed the SVM alone, which achieved a maximum accuracy of 96% using a quadratic kernel. This hybrid approach combines the strengths of SVM and KNN to improve classification accuracy, demonstrating its potential for accurate brain cancer detection in medical imaging.

(Rajini, N.H et al., 2015) paper describes an approach for automated diagnosis of brain MRI images using two classification techniques such as KNN and feed-forward back propagation artificial neural network. The study extracts features using discrete wavelet transformation and reduces them using PCA. The dataset contains MRI images of normal and pathological brains, with classification accuracies of 99% for KNN and 90% for FP-ANN. The identified challenges include the need for more accurate and robust feature extraction techniques. Future improvements could focus on incorporating more advanced machine learning techniques as well as larger, more diverse datasets in order to improve classification model generalizability and accuracy.

(Amiri,S et al., 2017) present a novel method for automatic brain tumor segmentation in MRI images. The method combines a Random Forest and a SVM in two steps. First, the RF is trained to match MRI intensities to tumor labels. The SVM then refines the predictions. When tested on the BRATS dataset of 20 patients, the method performed significantly better than individual RF and SVM approaches, with a mean Dice score of 0.72 versus 0.63 for RF and 0.59 for SVM alone. This method effectively captures the complex features of brain tumors, thereby improving segmentation accuracy and reliability.

(Belaid et al., 2020) investigate brain tumour classification with a pre-trained VGG-16 CNN and gray-level co-occurrence matrix (GLCM) features. Using a dataset of 3064 T1-weighted contrast-enhanced MRI images, including glioma, meningioma, and pituitary tumors, the model achieved 96.5% accuracy by combining original images with GLCM energy maps. This method outperformed other models, demonstrating the value of using GLCM features to improve classification accuracy. Future research plans to incorporate segmentation steps to improve tumor grading.

(Irmak, E et al., 2021) investigates the use of CNNs to classify brain tumors from MRI images, with impressive results 99.33% accuracy for detecting tumors, 92.66% for identifying specific types (normal, glioma, meningioma, pituitary, and metastatic), and 98.14% for grading gliomas (Grades II, III, and IV). The study demonstrates the potential of CNNs in medical imaging by utilizing large datasets such as RIDER, REMBRANDT, and TCGA-LGG, as well as fine-tuning model parameters with grid search. However, it also emphasizes issues such as the need for large, well-labeled datasets and the computational demands of deep learning models. Future efforts will seek to improve model reliability and efficiency in clinical applications.

(Díaz-Pernas, F.J et al., 2021) created an automatic brain tumor segmentation and classification model using a multiscale CNN. This method, modeled after the Human Visual System, processes MRI images at three spatial scales to classify and segment meningiomas, gliomas, and pituitary tumors without the need for preprocessing to remove non-brain tissue. Using 3064 T1-CE MRI slices from 233 patients, the model had a tumor classification accuracy of 97.3%, outperforming previous methods on the dataset. The multiscale CNN performed admirably, with average Dice, Sensitivity, and Pttas scores of 0.828, 0.940, and 0.967, respectively. The study emphasizes the value of multiscale processing for improving feature extraction, classification, and segmentation accuracy.

(Younis et al., 2022) investigated brain tumor detection using deep learning techniques, specifically the VGG-16 CNN architecture and ensemble learning approaches. The study used a dataset of 253 MRI brain images, 155 of which contained tumors classified as meningioma, glioma, or pituitary tumor. The VGG-16 model achieved an impressive 98.15% accuracy, which the ensemble model improved to 98.41%. The identified challenges include dealing with tumor variability and the computational complexity of deep learning models. Optimization of training processes and the incorporation of advanced image pre-processing techniques are among the recommended improvements. Overall, the study emphasizes the potential of VGG-16 and ensemble models to achieve high accuracy for brain tumor detection, as well as the need for further refinement in deep learning methodologies for medical imaging.

(Jiang, S et al., 2023) analysed 250 images from each of the following categories such as normal tissue, glioma, meningioma, and pituitary tumors. True Skill Statistics (TSS) were used to measure performance, and k-NN had the highest TSS value of 0.2154, while Decision Tree

had the lowest. Challenges identified include dataset imbalance, difficulty recognizing low-level features, and lengthy training procedures. To improve accuracy, the study recommends adjusting model parameters, balancing datasets, improving feature extraction methods, and reducing training time. Overall, k-NN performed best, but more work is needed for robust and efficient MRI brain tumor classification.

(Imam.R et al., 2023) investigated the use of deep learning, specifically transfer learning combined with CNN, to identify brain tumors in MRI images. Using a dataset of 4200 MRI images, the study investigates data imbalance using models such as VGG16, EfficientNet, ResNet, DenseNet, MobileNet, GoogleNet, and XceptionNet. When combined with data augmentation techniques, the VGG16-CNN model achieved a 96% accuracy rate. Several approaches to dealing with data imbalance were investigated, including focal loss, cross-entropy loss, and oversampling techniques like SMOTE and ADASYN. The VGG16-CNN model had the most detailed and accurate features, making it the best at classifying brain tumors. The study recommends further optimisation of models and hyperparameters to improve robustness and efficiency for real-world applications.

(Gayathri et al., 2023) discovered that the VGG-16 CNN architecture can accurately detect brain tumors through deep learning. The VGG-16 model was fine-tuned and trained on a dataset of 1655 brain MRI images with tumors and 1598 without tumors, resulting in an initial accuracy of 91% that improved to 94% after hyperparameter optimization. The model's performance metrics, including sensitivity (93.97%), specificity (92.54%), precision (93.28%), recall (93.27%), and F1 score (93.27%), demonstrated its efficacy. Compared to other techniques such as EasyDL, GoogLeNet, GrayNet, and CNNs, VGG-16 demonstrated comparable accuracy. However, the study emphasizes the need for larger and more diverse datasets, as well as the exploration of other deep learning architectures and the improvement of model interpretability, in order to improve clinical relevance and medical diagnostic application.

(Sutradhar, P et al., 2021) compare the performance of different machine learning and deep learning algorithms for detecting brain tumors on MRI images. The study contrasts traditional machine learning algorithms like SVM, KNN, Random Forest, and Decision Tree with deep learning models like CNN and transfer learning models like EfficientNet-B3, ResNet-150V2, Inception-ResNetV2, and VGG16. The study used a dataset of 3264 MRI images and discovered that EfficientNet-B3 had the highest accuracy of 98.16%, outperforming other models in both accuracy and computational efficiency. The findings highlight the superior performance of deep learning models, particularly CNNs and transfer learning approaches, in accurately classifying brain tumors, and call for further investigation and optimization of these models for improved diagnostic applications.

(Muis, A et al., 2024) looks into improving brain tumor image classification accuracy with CNN-based approaches, specifically the AlexNet and GoogleNet architectures. Using a dataset of 7023 MRI brain images, the study achieved 98% classification accuracy with AlexNet and 96% with GoogleNet. The study covered preprocessing steps like resizing and grayscale conversion, as well as advanced techniques for managing computational load and improving model performance. Despite the high accuracy rates, the study recommends further optimization and integration of additional techniques to improve diagnostic applications in medical imaging.

Reference	Technique	Dataset	Accuracy
(Rathi et al., 2014)	SVM	MRI brain images	98.87%
(Gurusamy, R et al., 2017)	Neural Network, KNN, SVM	MRI brain images	Neural Network (97%), KNN (96%) and SVM (98%)
(Imam.R et al., 2023)	CNN, VGG16, EfficientNetB0, EfficientNetB3, ResNet50	MRI brain images	CNN (96%), VGG-16 (98.5%)
(Younis, A. et al., 2022)	VGG-16 Ensembling Learning	MRI brain images	98.14%
(Gayathri, P. et al., 2023)	VGG-16 Ensembling Learning	MRI brain images	94%
(Belaid. et al., 2020)	VGG-16	MRI brain images	96.5%
(Irmak, E et al., 2021)	CNN	Multiple datasets: RIDER, REMBRANDT, TCGA-LGG, and Cheng's brain tumor dataset	92.66%
(Muis, A. et al., 2024)	Cnn using AlexNet and GoogLeNet	MRI brain images	AlexNet (98%), GoogleNet (96%)
(Díaz-Pernas, F.J et al., 2021)	CNN	T1-weighted contrast-enhanced MRI images	97.3%
(Machhale, K. et al., 2015)	Hybrid Classifier (SVM-KNN)	MRI brain images	98%
(Rajini, N.H. et al., 2011)	KNN, ANN	MRI brain images	KNN (99%), ANN (90%)
(Amiri, S. et al., 2016)	Deep Random Forest-based Learning Transfer to SVM	Brain Tumor Image Segmentation Challenge (BRATS) dataset	72%
(Jiang, S, et al., 2023)	k-NN, Decision Tree, SVM, Logistic Regression, SGD	MRI brain tumor image dataset from Figshare	-
(Sutradhar, P. et al., 2021)	EfficientNet-B3, Random Forest	MRI brain images	98.16%

Table 2.1: Summary of existing Models

2.1. Research gap:

Although there have been notable improvements in using machine learning and deep learning models to classify brain tumors, there is still a crucial research gap in achieving the best possible performance on diverse and large datasets. This gap specifically pertains to finding the right balance between model accuracy and computational efficiency. Prior research has shown that models such as CNN, VGG-16, KNN, and Random Forest have the ability to detect and classify brain tumors. However, these models often face challenges such as overfitting

caused by limited dataset sizes, the requirement for extensive preprocessing, and the difficulty of capturing low-level features in datasets with imbalanced distribution. Furthermore, although certain techniques such as hybrid models and ensemble learning exhibit potential, their effectiveness is still constrained by their reliance on particular data attributes and computational limitations. This research project seeks to address this deficiency by implementing and comparing CNN, CNN-VGG16, KNN, and Random Forest models on a broader and more extensive dataset. It will also employ data augmentation techniques to reduce overfitting and optimize hyperparameters to improve model generalization and efficiency. The project aims to fill these gaps in order to enhance the development of more resilient and dependable brain tumor classification systems that can be efficiently implemented in clinical environments.

Chapter 3

Methodology

Brain tumor classification from MRI images has become an essential task in medical image processing in recent years. The application of modern machine learning techniques, such as CNN and KNN, in combination with ensemble approaches like Random Forests, has greatly improved the accuracy and efficiency of tumor identification. This study investigates four different methods for classifying brain tumors. The models I'm using, CNN, CNN utilizing the VGG16 architecture, KNN algorithm, and a Random Forest classifier. Every technique goes through a thorough process of data preprocessing, which includes scaling, normalization, and label encoding, in order to guarantee the best possible performance of the model. The overall architecture is shown below.

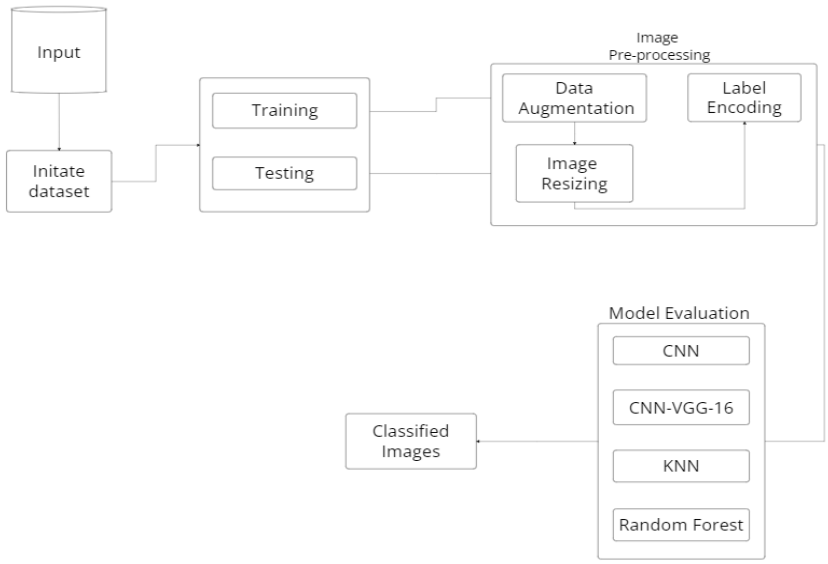


Figure 3.1: Overall System Architecture

3.1 Data Collection

The dataset utilized in this experiment comprises MRI images depicting various categories of brain tumors, namely glioma, meningioma, and pituitary tumors. The images were split into training and testing sets, with each set being saved in distinct directories within Google Drive. This process assisted in methodically organizing the data for model training and evaluation. The dataset is partitioned into training and testing sets, with images split into separate directories for each type of tumor. A total of 7023 images were counted in each class for both the training and testing datasets. The program iterates through the directories to collect the image locations and labels. The count of tumors in training and testing dataset is shown is figure 3.2 and sample images of tumors are shown in figure 3.3.

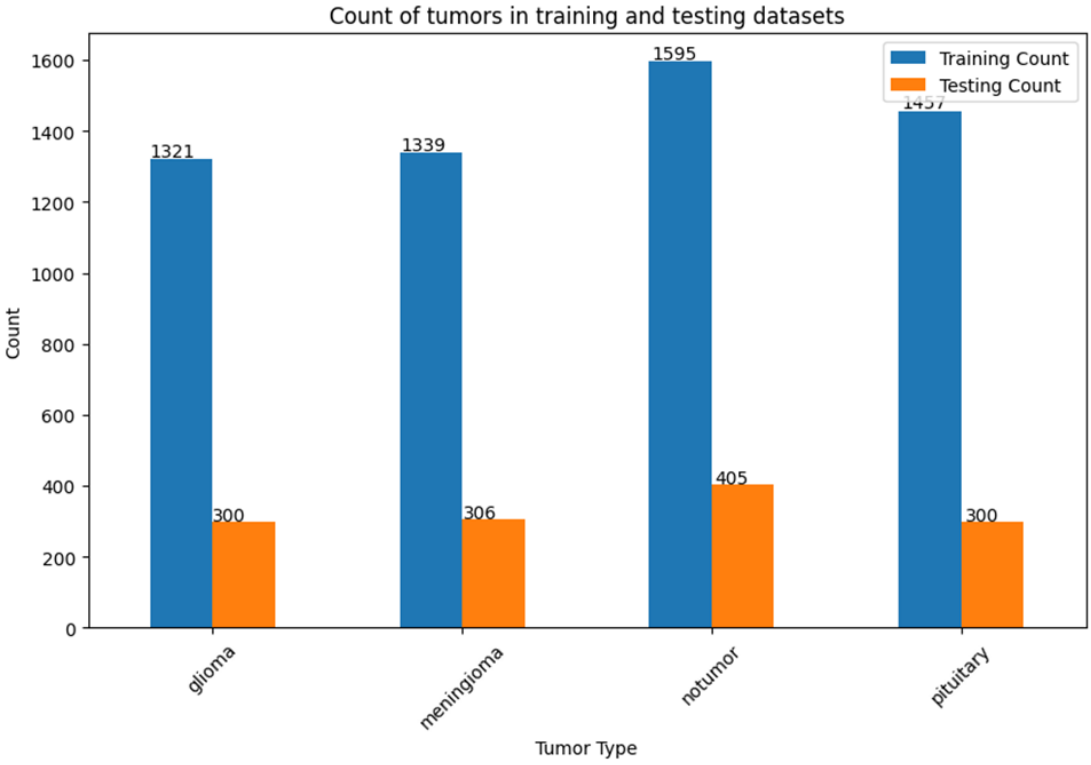


Figure 3.2 : Count of testing and training dataset

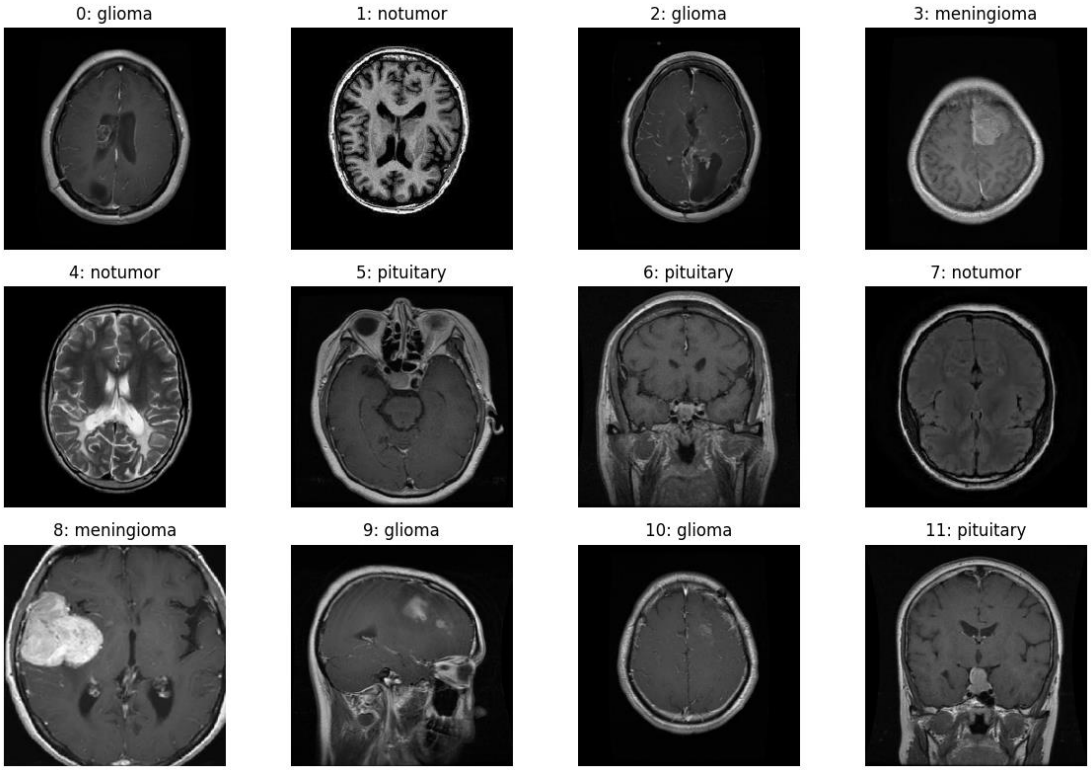


Figure 3.3: Sample images of tumors

3.2 Data preprocessing

Data preprocessing for deep learning, includes several essential stages to ready the MRI images for training the CNN model. Initially, the images underwent a process of resizing to a standardized dimension of 250x250 pixels and were subsequently transformed into grayscale to ensure uniformity. In order to increase the variety of the training dataset and minimize overfitting, several data augmentation techniques were employed, including random flipping, rotation, contrast adjustment, zooming, and translation as shown in figure 3.4. The modified images enhanced the dataset's strength for training. In addition, the pixel values of the images were standardized to a range from 0 to 1, and the tumor labels were transformed into numerical indices to facilitate processing by the neural network. Ultimately, the preprocessed images and labels were systematically arranged into TensorFlow datasets. These datasets were subsequently divided into batches and prefetched to enhance the efficiency of the training process.

The preprocessing for machine learning, includes multiple stages to ready them for the KNN classification and Random Forest. The images were resized to a resolution of 250x250 pixels in order to achieve consistency. Subsequently, the pixel values across the dataset were standardized by normalizing them to a range of [0, 1]. The images were transformed into 1D array in order to ensure compatibility with both models, which necessitates vectorized input data. In addition, the labels that represent the type of tumor were transformed into numerical indices, which makes them appropriate for machine learning algorithms.

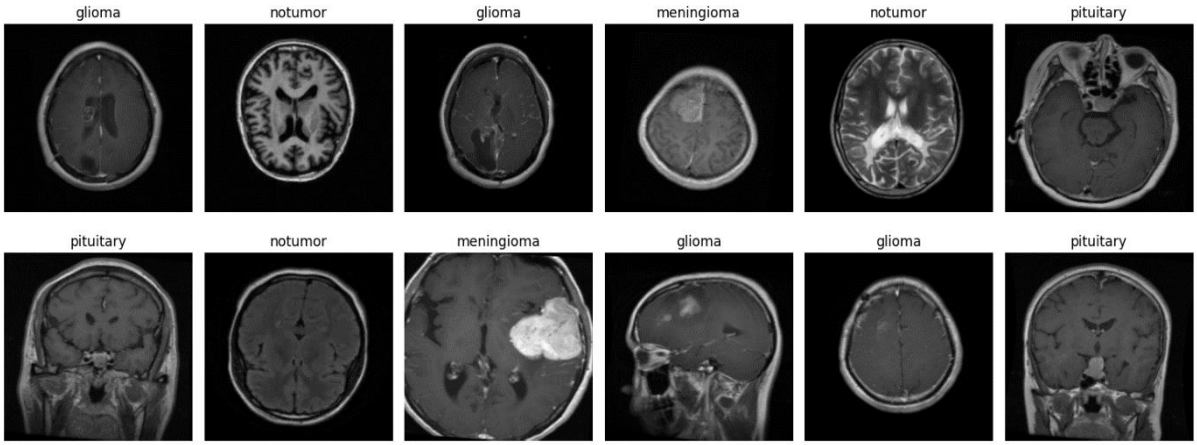


Figure 3.4: Augmented data images

3.3 Model Architecture

CNN:

The CNN model developed for brain tumor classification comprises several layers specifically intended to extract and analyze characteristics from MRI images. The architecture commences with an input layer that receives 250x250 RGB images. Subsequently, a few convolutional layers are employed, with each layer employing ReLU activation and different kernel sizes to extract complex spatial characteristics from the images. Max-pooling layers are inserted between convolutional layers to decrease the size of the feature maps, thus lowering their dimensionality while preserving crucial information. Subsequently, fully linked layers are utilized to combine these features and generate the ultimate classification probabilities. Dropout layers are included in the architecture to reduce overfitting by randomly discarding neurons during the training phase. The last layer consists of an output layer that utilizes softmax activation to categorize the images into distinct tumor categories. Figure 3.5 illustrates how a CNN model processes an input image via different layers, and then uses fully connected layers to make the final prediction.

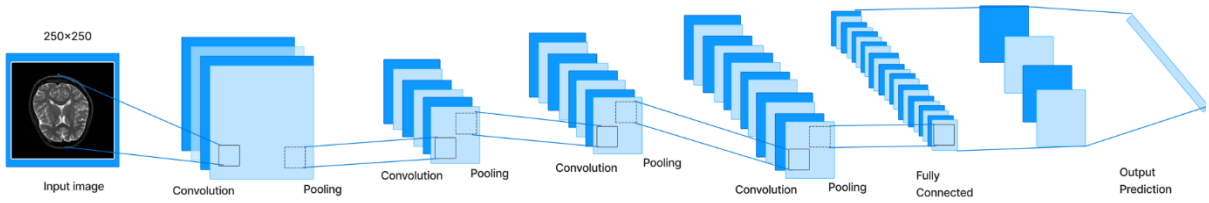


Figure 3.5: An example showing how a CNN model processes an input image via different layers, and then uses fully connected layers to make the final prediction.

The training and optimization procedure for the CNN model encompassed various essential steps and hyperparameters. The model underwent training using the Adam optimizer, employing a learning rate of 0.001. Additionally, the beta_1 and beta_2 values were assigned as 0.85 and 0.9925, respectively. beta_1 and beta_2 is hyperparameter of Adam's where they control the exponential decay rate for first moment and second moment estimation. The training process consisted of 50 epochs, with each epoch comprising a batch size of 32. Several callbacks were utilized to optimize the training process, including reducing the learning rate when the validation loss reached a plateau, saving the best model based on validation accuracy using a model checkpoint, and implementing a custom learning rate scheduler to adjust the learning rate at specific accuracy thresholds.

The training method employed the preprocessed training dataset, whereas the testing dataset was used for validation. During the training process, I have monitored carefully the accuracy and loss metrics to assess the performance of the model and ensure that it was learning optimally. The adoption of this all-encompassing strategy for model structure and training guaranteed the creation of a strong and precise CNN model for the classification of brain tumors.

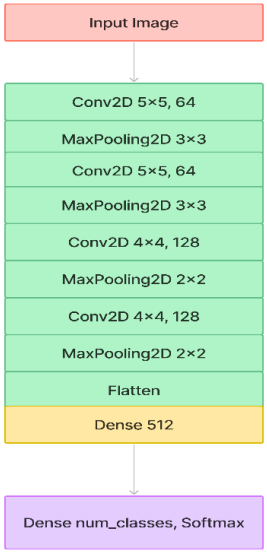


Figure 3.6: Architecture of CNN Model

CNN-VGG-16

For this investigation, I employed a CNN that is built on the widely recognized VGG16 framework, noted for its exceptional performance in tasks involving image identification. The VGG16 model consists of multiple layers that are specifically built to gradually extract and enhance information from input images, hence aiding the model in effectively comprehending and classifying them. The block diagram of CNN-VGG-16 model is shown below.

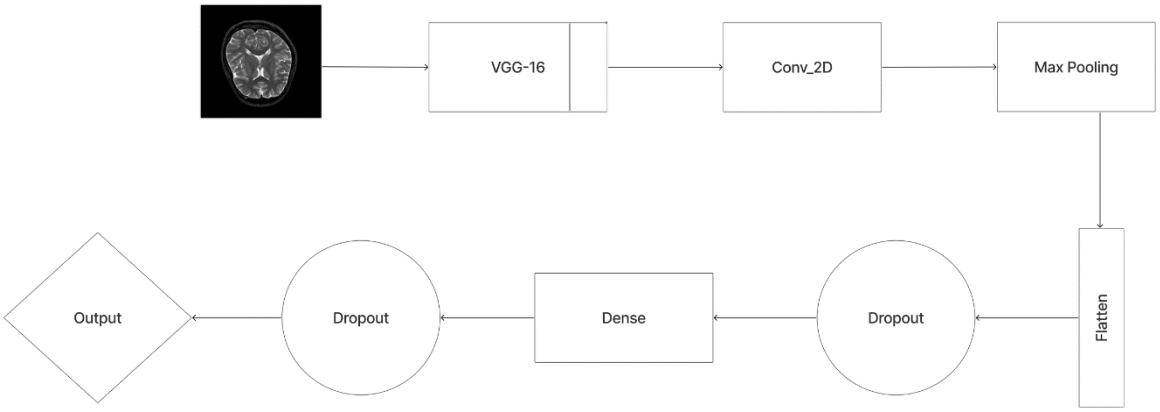


Figure 3.7: Block diagram of CNN-VGG-16 model

The initial layer of the VGG16 model is the input layer, which receives images that have been scaled to dimensions of 250x250 pixels. The images are in RGB format, which guarantees that the model receives all the required color information for precise classification. The subsequent model incorporates several convolutional layers, each employing ReLU (Rectified Linear Unit) activation and 3x3 kernel sizes. The function of these layers is to analyze the images using small filters, enabling the model to detect and capture precise spatial characteristics such as edges, textures, and patterns. Utilizing the Rectified Linear Unit (ReLU) activation function facilitates the incorporation of non-linear elements, hence allowing the model to acquire more intricate representations of the input.

Max-pooling layers are inserted between the convolutional layers. These layers decrease the spatial dimensions of the feature maps by choosing the maximum value within tiny portions of the feature maps. Downsampling is a technique that reduces the computational load and prevents overfitting by summarizing the most significant features. Following the convolutional and pooling layers, the model includes fully connected layers. The thick layers in the neural network combine the high-level features obtained from the preceding layers to create the ultimate classification decisions. The presence of these layers guarantees that the model is capable of establishing a connection between the characteristics and distinct types of tumors.

In order to mitigate the issue of overfitting, dropout layers are incorporated into the architecture. During the training process, certain layers of the model randomly eliminate some neurons. This technique encourages the model to acquire more resilient characteristics that can be applied well to unfamiliar data. The last layer of the VGG16 model is the output layer, which employs a softmax activation function. This layer assigns probabilities to each tumor type, enabling the model to accurately classify the images into distinct groups.

In order to customize the pre-trained VGG16 model for the specific objective of brain tumor classification, I utilized a technique known as fine-tuning. At first, I immobilized the initial layers of the model, maintaining their pre-trained weights without any modifications. Subsequently, I reactivated the remaining layers, enabling them to undergo retraining using the updated dataset. This strategy utilizes the existing knowledge incorporated in the pre-trained model while modifying it to enhance performance on the brain tumor images.

The VGG16-based CNN, which has been meticulously constructed and optimized, successfully categorizes various brain tumors from MRI images. It combines the advantages of pre-trained models with the necessary precision for medical image analysis.

KNN:

The KNN technique was used for data classification. This technique employs a method of classification that involves identifying the ‘k’ closest training samples in the feature space to categorize the input data. The KNN model was instantiated with a value of 5 for the number of neighbors (k=5), and it employed the Euclidean distance metric to compute the similarity between the training and testing data points. The model underwent training using the preprocessed training dataset, where the labels denoted the specific type of tumor.

Once the KNN model was trained, its performance was assessed using the testing dataset. The accuracy of the model was calculated by comparing its predictions with the true labels of the test images. A confusion matrix analysis was conducted to offer comprehensive insights into the classification outcomes, emphasizing the quantities of true positives, true negatives, false positives, and false negatives. The block diagram of KNN Model is shown below.

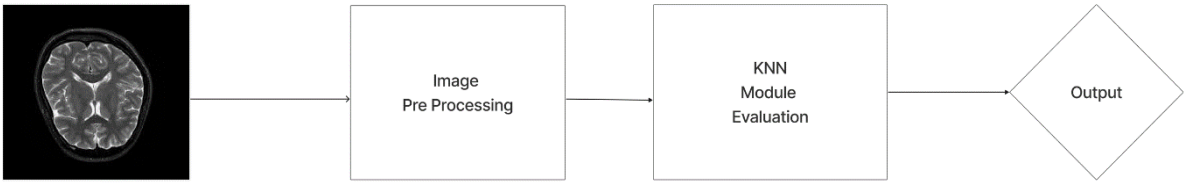


Figure 3.8: Block diagram of KNN Model

Random Forest

The Random Forest (RF) model, an ensemble learning method, was used for the classification task. The Random Forest model was instantiated with 100 trees (n_estimators=100) to guarantee resilience and precision. The model was trained using the training dataset, which included preprocessed images and their accompanying labels. The process entailed constructing numerous decision trees and combining their predictions to appropriately identify the input images. The block diagram of Random Forest Model is shown below.

Figure 3.9: Block diagram of Random Forest Model

Following the completion of training, the model's performance was assessed using the testing dataset. The Random Forest model, which had been trained, made predictions on the tumor types of the test images. These predictions were then compared to the actual labels in order to determine the accuracy.

3.4 Performance Evaluation of the models:

In order to ensure a comprehensive and standard evaluation of the performance of each model, I applied a standardized methodology that involves the computation of precision, recall, and F1-score for each tumor class. This evaluation methodology enables me to precisely evaluate the efficacy and dependability of the models, which will be elaborated upon in subsequent sections.

Precision is the quotient obtained by dividing the number of accurately anticipated positive observations by the total number of predicted positives. It quantifies the accuracy of predicting positive situations as being truly positive. The mathematical formula is provided as:

$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)}$$

Recall is the proportion of accurately predicted positive observations in relation to the total number of observations in the actual class. It quantifies the model's capacity to accurately detect all relevant examples.

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negitive(FN)}$$

The F1-score is calculated as the reciprocal of the arithmetic mean of the reciprocals of precision and recall. It offers a unified measurement that takes into account both the accuracy and completeness of the model.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

A confusion matrix was generated to visually represent the classification results of the model and detect any instances of misclassification. In addition, I showed sample predictions to demonstrate the model's proficiency in accurately categorizing various brain tumor kinds. The thorough assessment revealed the model's efficacy and dependability in categorizing MRI scans of brain tumors.

Chapter 4

Result and Discussion

4.1 Comprehensive Review of Research Outcomes and Performance:

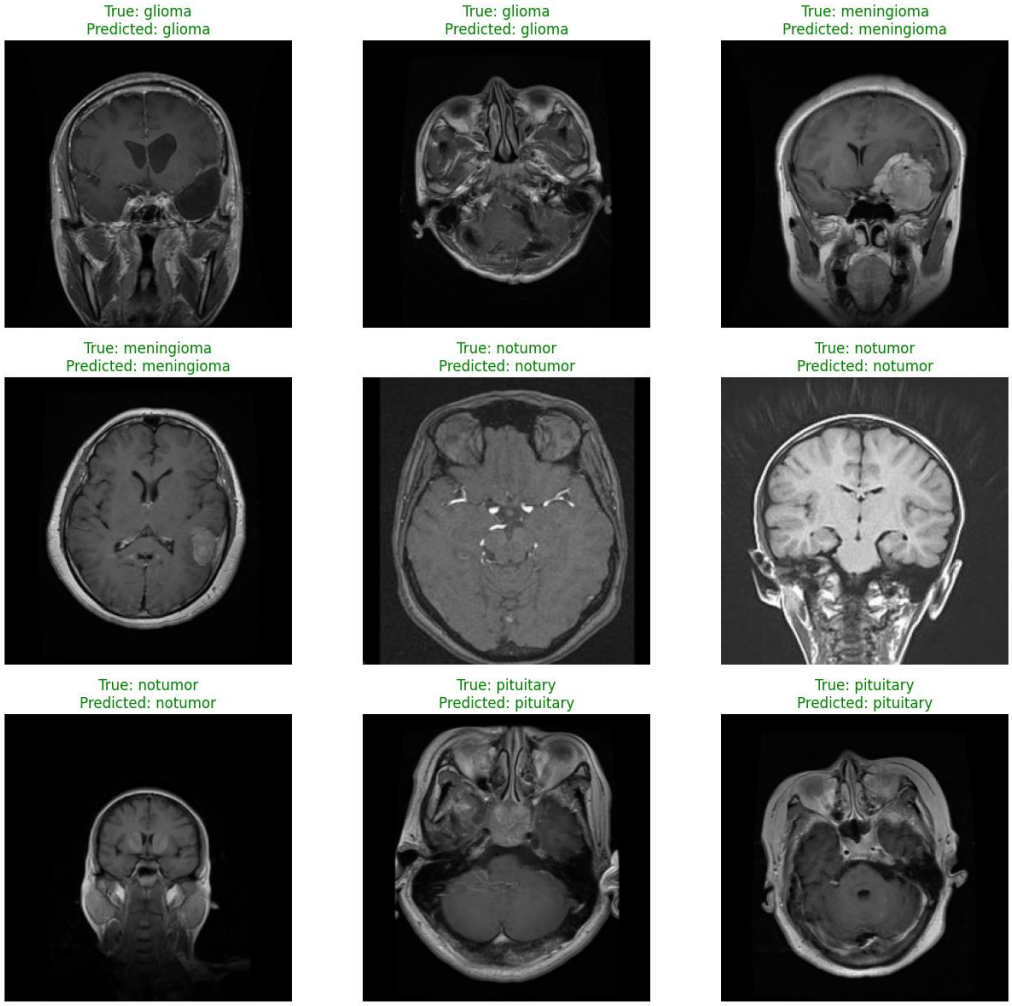


Figure 4.1: Output prediction of CNN, CNN-VGG-16 Model

Figure 4.1 shows the output of brain tumor classification using a CNN model and a CNN-VGG-16 model. Both models correctly classify nine MRI samples, which include glioma, meningioma, pituitary tumors, and cases with no tumor. The CNN model outperforms the CNN-VGG-16 model with an accuracy of 99.16% compared to 97.64%, demonstrating its superior performance in this task.

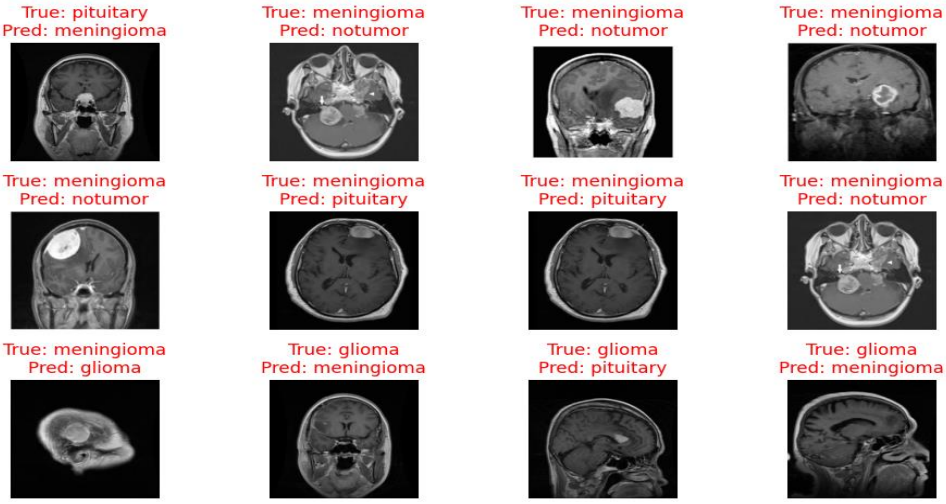


Figure 4.2: Misclassifications of CNN Model

Figure 4.2 shows instances of misclassification by the CNN model during brain tumor classification. Only 12 of the 1311 testing samples were misclassified. The image highlights these cases, demonstrating both the true labels and the incorrect predictions, including confusions between pituitary, meningioma, glioma, and no tumor categories. Despite the overall high accuracy, these examples highlight the difficulties in distinguishing between similar tumor types in certain instances.

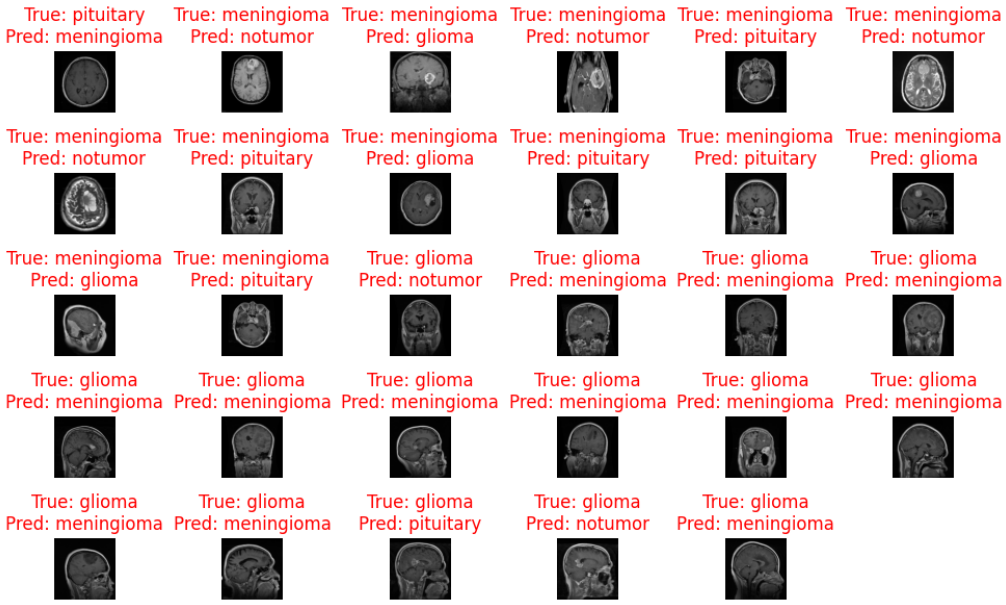


Figure 4.3: Misclassifications of CNN-VGG-16 Model

Figure 4.3 depicts 29 instances where the CNN-VGG-16 model incorrectly classified samples out of 1311 testing cases. These errors specifically involve the model's inability to accurately differentiate between pituitary, meningioma, glioma, and the no tumor categories. This highlights the difficulties faced by the model in accurately differentiating between these types of tumors in specific cases.

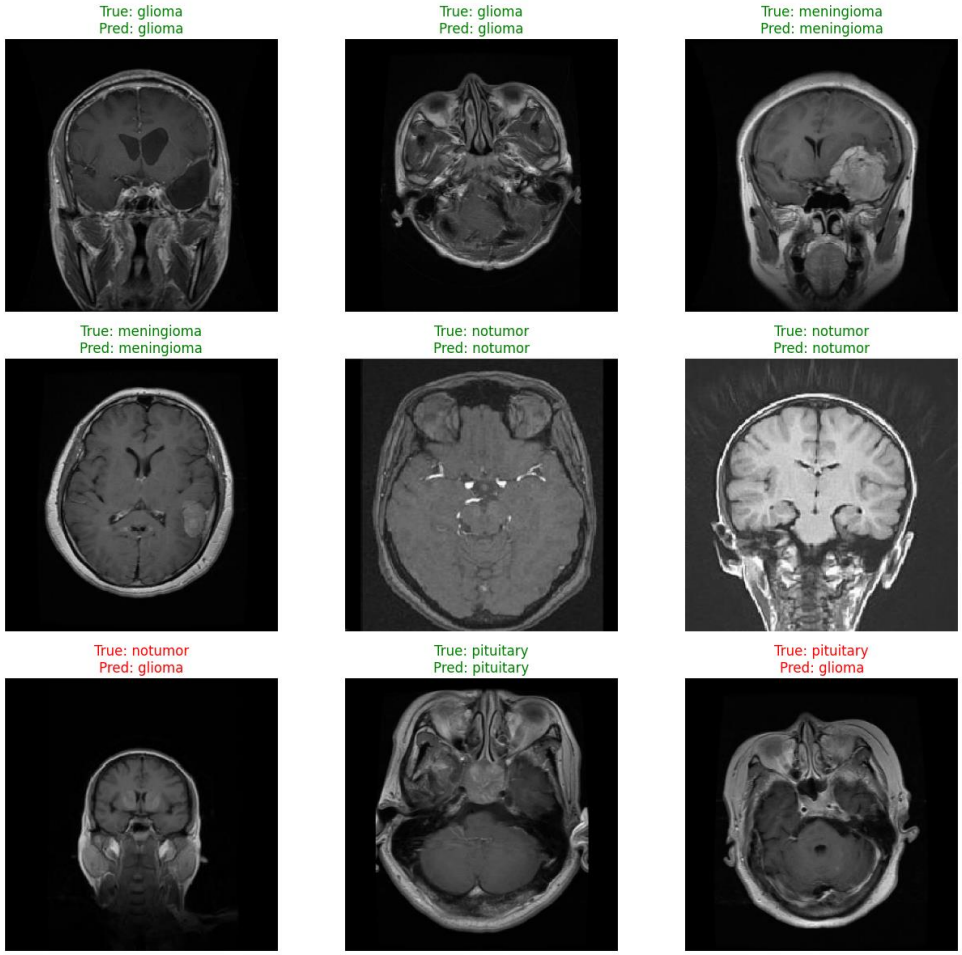


Figure 4.4: Output prediction of KNN Model

Figure 4.4 demonstrates the output predictions of the KNN model. The model achieved an accuracy of 84.74% by correctly classifying 7 out of 9 selected test samples, with 2 samples being misclassified. The true and predicted labels are displayed, emphasizing both accurate and inaccurate predictions.

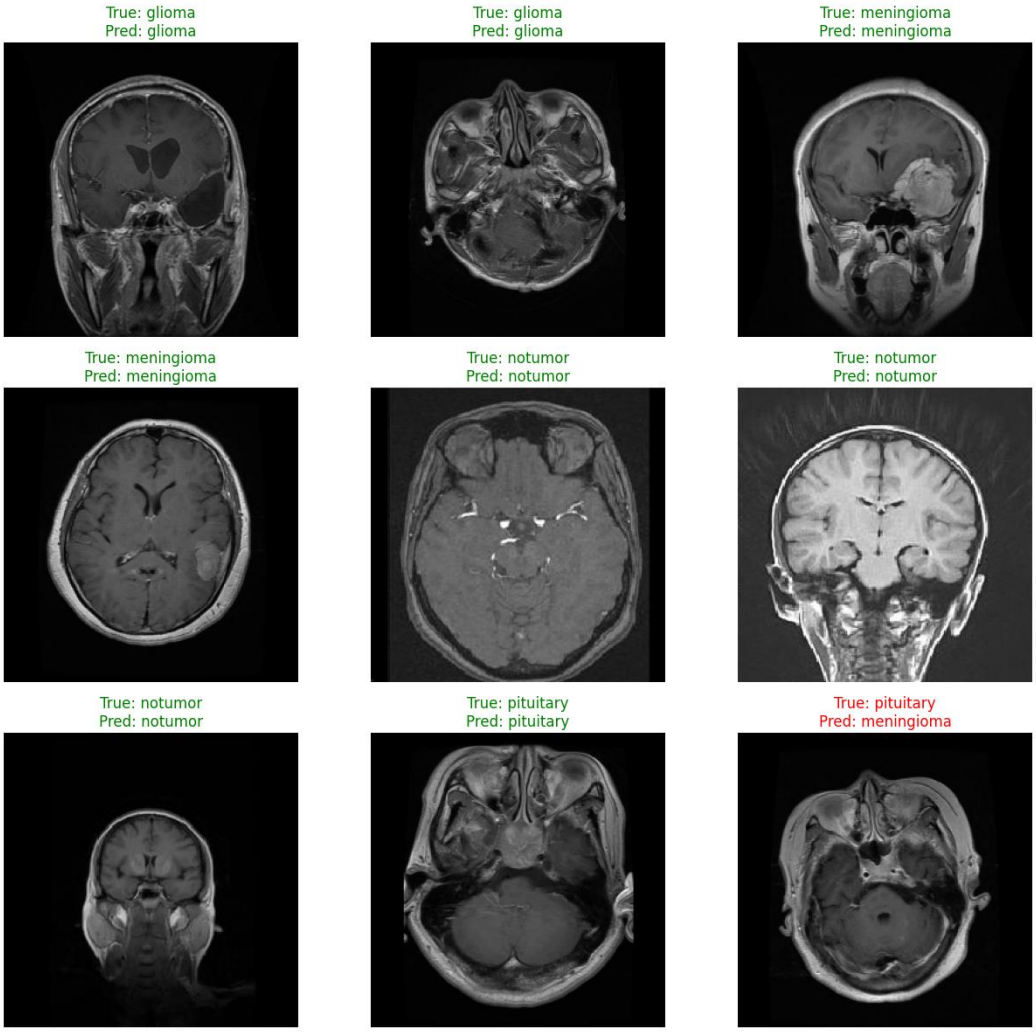


Figure 4.5: Output prediction of Random Forest Model

The output predictions of the Random Forest model for the classification of brain tumors are shown in Figure 4.5. The model achieved a 93.36% accuracy by correctly classifying 8 out of 9 selected test samples, with only 1 sample being misclassified.

4.2 Evaluation of my Research Outcomes

I describe and evaluate the efficacy of different machine learning models for classifying brain tumors using MRI images. The evaluated models consist of CNN, a CNN utilizing the VGG-16 architecture, KNN, and Random Forest. The efficacy of these models is evaluated using accuracy measures and other performance indicators, such as precision, recall, and F1-score. In addition, I analyzed the learning curves, confusion matrices, and individual examples of accurate and inaccurate classifications to gain a thorough understanding of the strengths and limitations of each model. This discussion also examines potential concerns such as overfitting and the influence of dataset size on model performance, providing insights into the optimization and practical use of these models in real-world medical diagnoses.

Model	Training Accuracy	Testing Accuracy
CNN	99.89%	99.16%
CNN-VGG-16	98.44%	97.63%
KNN	92.87%	85.35%
Random Forest	100%	93.82%

Table 4.1: Tabulated the accuracy of the model used.

The table 4.1, displays the training and testing accuracies of four distinct models utilized for brain tumor classification. The CNN model demonstrates strong generalization with a testing accuracy of 99.16%, while the CNN-VGG-16 model closely trails behind with an accuracy of 97.63%. Although the Random Forest model attains a flawless accuracy on the training set, its testing accuracy decreases to 93.82%, suggesting the possibility of overfitting. The KNN model exhibits the lowest testing accuracy, measuring at 85.35%. This result indicates difficulties in extrapolating from the training data.

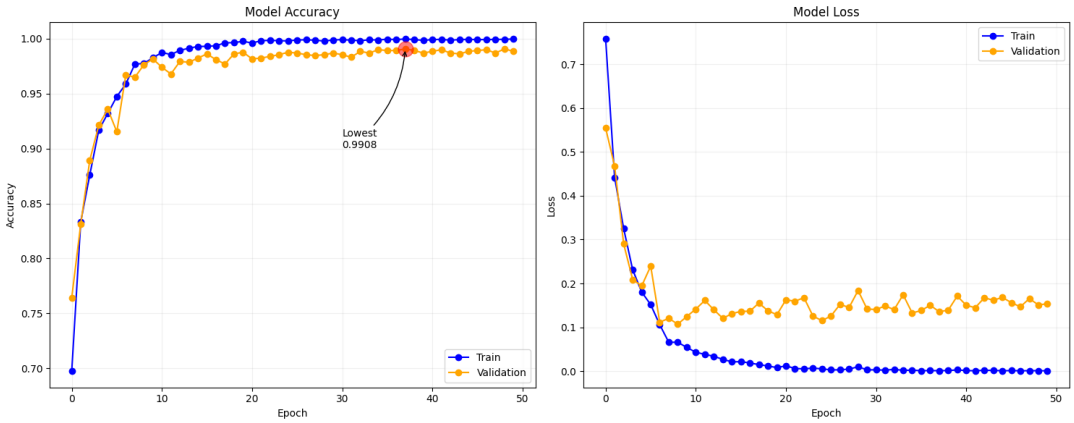
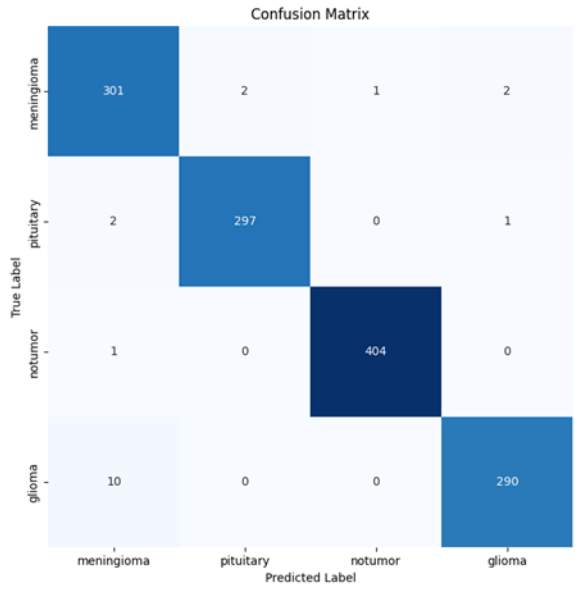


Figure 4.6: Learning curve of CNN model

The CNN model exhibits strong performance as shown above, with an accuracy of nearly 100% on the training set and a high level of accuracy on the validation set about 99%. The loss curves indicate that the model is successfully reducing error, while the small variations in validation loss signal a possible issue of overfitting. Nevertheless, the degree of overfitting appears to be minimal, given that the validation accuracy consistently remains high. In overall, the model seems to be well-trained and has the ability to apply its knowledge to new data, with only a few small areas where it could be improved. The confusion matrix and classification report obtained from evaluating the CNN model are visually represented in Figure 4.7. The analysis of the confusion matrix exposes the effectiveness of the CNN model in categorizing the images, offering comprehensive observations regarding the model's performance in each class.



Classification Report:

	precision	recall	f1-score	support
meningioma	0.99	0.98	0.98	306
pituitary	0.99	1.00	1.00	300
notumor	0.99	1.00	1.00	405
glioma	1.00	0.99	0.99	300
accuracy			0.99	1311
macro avg	0.99	0.99	0.99	1311
weighted avg	0.99	0.99	0.99	1311

Figure 4.7: Confusion Matrix and Classification Report for CNN model

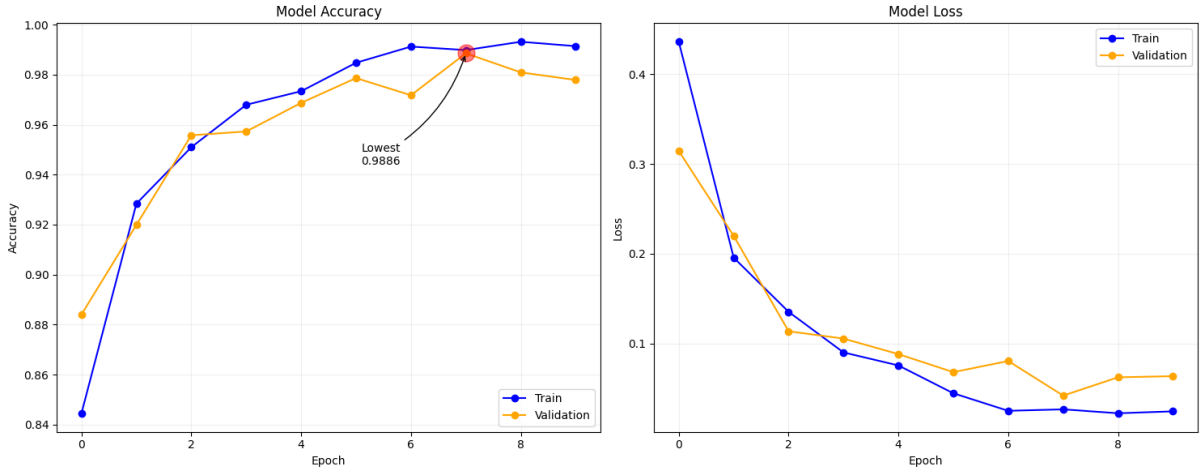
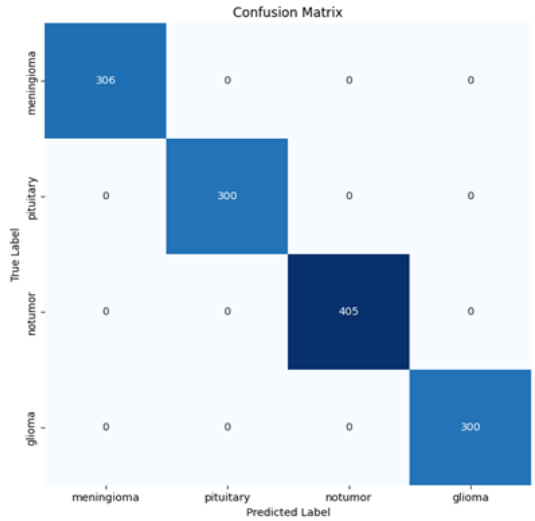


Figure 4.8: Learning curve of VGG-16 based CNN model

The VGG-16 based CNN model demonstrates solid learning skills as shown in figure 4.8, as seen by the quick improvement in both training accuracy and validation accuracy in the early epochs. However, after around 4-6 epochs, indications of overfitting begin to show up, as demonstrated by the modest deviation between the accuracy of the training and validation sets, as well as the fluctuating loss in the validation set. Although these variations, the model consistently achieves a high level of accuracy on the validation set, suggesting strong generalization capabilities with potential for further enhancement in addressing overfitting. Potential approaches to tackle this issue could involve methods like as regularization, increased utilization of dropout, or implementing early stopping. The confusion matrix and classification report obtained from the evaluation of the CNN-based VGG-16 model are shown in Figure 4.9. The provided visuals display the classification outcomes for each class, illustrating the proficiency of the CNN-based VGG-16 model in accurately categorizing images. This is achieved through a thorough analysis of the confusion matrix and the accompanying report.



Classification Report:

	precision	recall	f1-score	support
meningioma	0.94	0.97	0.95	306
pituitary	0.98	1.00	0.99	300
notumor	1.00	0.99	1.00	405
glioma	0.98	0.94	0.96	300
accuracy			0.98	1311
macro avg	0.97	0.98	0.97	1311
weighted avg	0.98	0.98	0.98	1311

Figure 4.9: Confusion Matrix and Classification Report of CNN-VGG-16

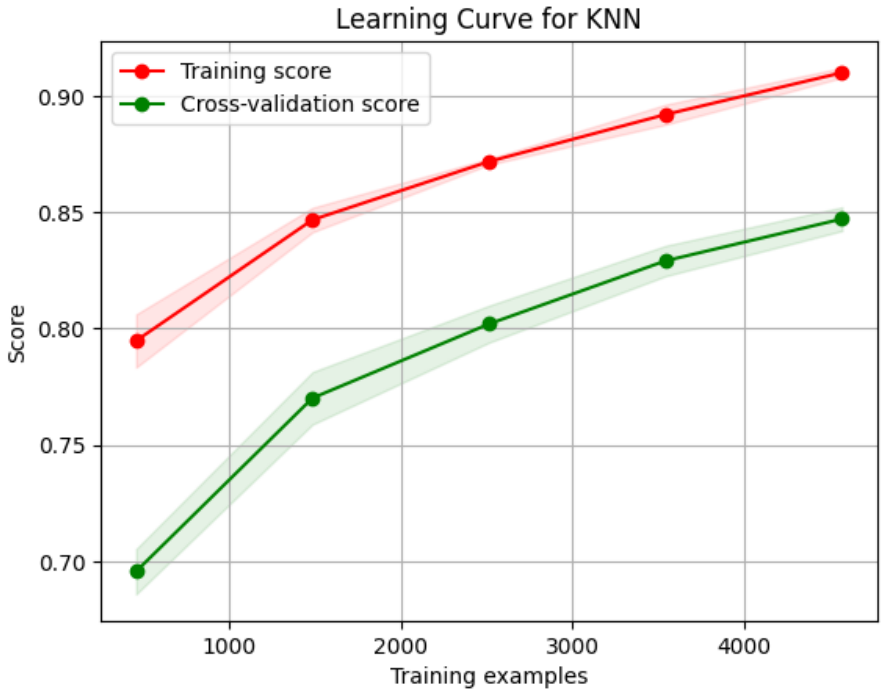
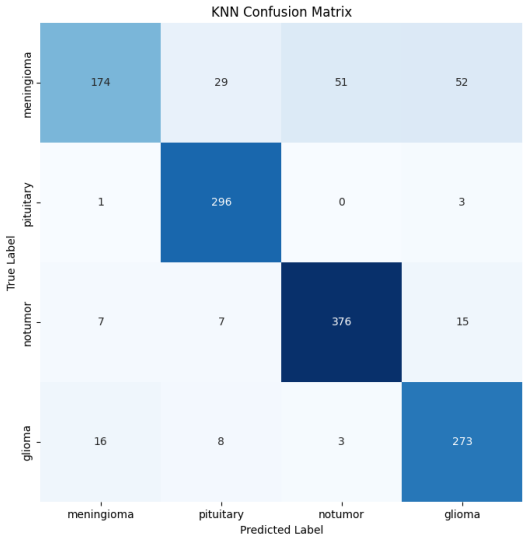


Figure 4.10: Learning curve of KNN model

The learning curve for the KNN model demonstrated in Figure 4.10, that the model exhibits good behaviour overall, as both the training and cross-validation performance improve with the utilization of more data. The overfitting initially noticed diminishes as the size of the training set increases, which is a favourable outcome. The model's performance becomes more consistent and better when more data is utilized, suggesting that the model's ability to apply its knowledge to new data is improved with a larger training set. This implies that augmenting the training data's volume is advantageous for this model, resulting in more dependable and precise forecasts on unseen data. The confusion matrix and classification report from the evaluation of the KNN model are shown in Figure 4.11. This integrated visual representation displays the

classification outcomes for each class, showing the effectiveness of the KNN model in image classification.

.



Classification Report:

	precision	recall	f1-score	support
meningioma	0.88	0.57	0.69	306
pituitary	0.87	0.99	0.93	300
notumor	0.87	0.93	0.90	405
glioma	0.80	0.91	0.85	300
accuracy			0.85	1311
macro avg	0.85	0.85	0.84	1311
weighted avg	0.86	0.85	0.85	1311

Figure 4.11: Confusion Matrix and Classification Report of KNN

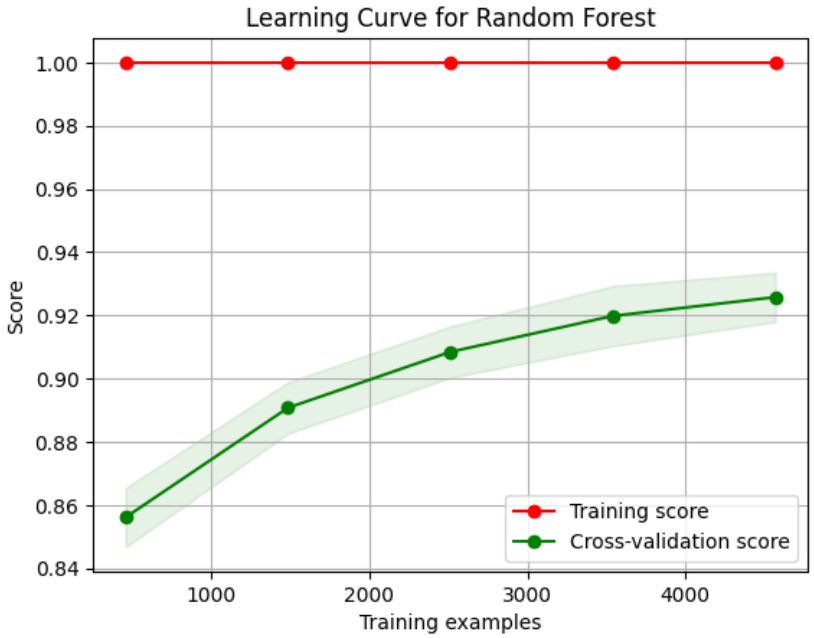
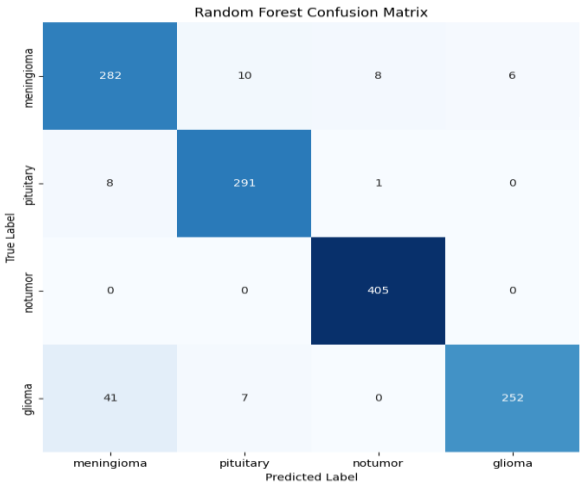


Figure 4.12: Learning curve of Random Forest model

The learning curve shown above, for the Random Forest model exhibits typical indications of overfitting. The model attains excellent accuracy on the training data. However, this proficiency does not correspond to equally robust performance on the cross-validation data. The cross-validation score has a positive correlation with the size of the training set, suggesting that the model gains advantages from an augmented amount of data. Nevertheless, the consistent difference between the training and cross-validation scores indicates that the model

is overfitting, resulting in suboptimal generalization to unseen data. To prevent overfitting, we can employ regularization techniques, minimize the number of trees or depth in the Random Forest, or augment the training data to enhance the model's generalization ability. The confusion matrix and classification report resulting from the evaluation of the Random Forest model are shown in Figure 4.13. This visual representation illustrates the classification outcomes for each class, showcasing the efficacy of the Random Forest model in image classification.



Classification Report:

	precision	recall	f1-score	support
glioma	0.97	0.83	0.89	300
meningioma	0.85	0.92	0.88	306
notumor	0.98	1.00	0.99	405
pituitary	0.95	0.98	0.96	300
accuracy			0.94	1311
macro avg	0.94	0.93	0.93	1311
weighted avg	0.94	0.94	0.94	1311

Figure 4.13: Confusion Matrix and Classification Report of Random Forest model

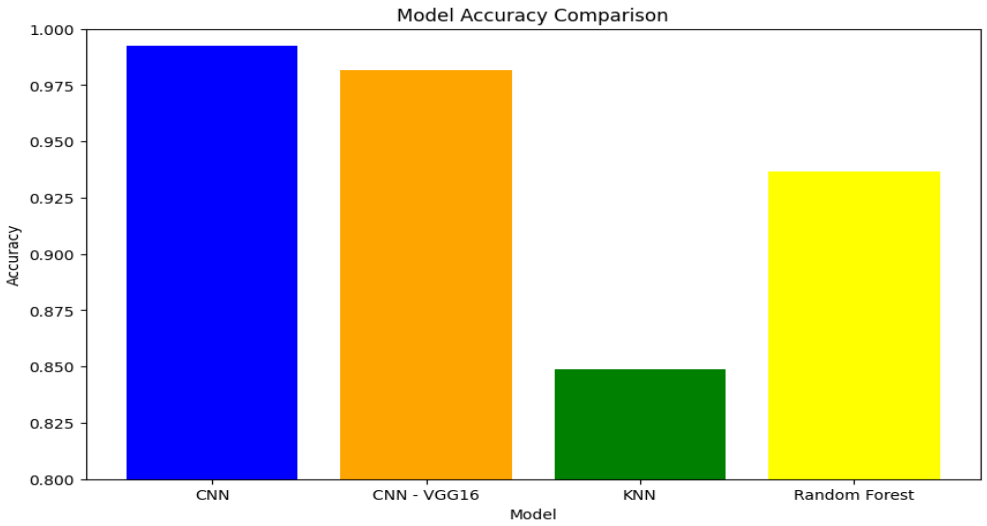


Figure 4.14: Comparison between model’s accuracy

In figure 4.14, it clearly shows that CNN model has high level of performance. From the research, CNN model has achieved 99.16% and CNN-VGG-16 achieved 97.63%. Whereas KNN and Random Forest achieved 85.35% and 93.82%.

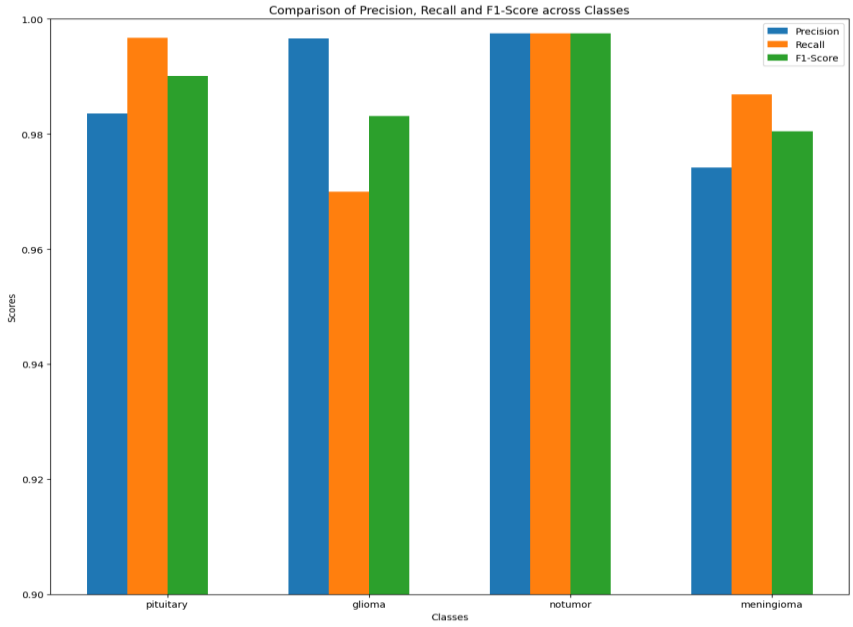


Figure 4.15: Comparison of Precision, Recall and F1-Score across Classes

Figure 4.15 depicts the comparison of Precision, Recall, F1-Score, and relative Support across four distinct classes: glioma, meningioma, notumor, and pituitary. Each metric constantly displays high scores approaching 1.0, demonstrating the model's strong performance. The relative support bars accurately indicate the proportionate distribution of classes in the dataset.

4.2 Comparison with the Existing Models

The table below presents a comparative analysis of the models formulated in my research with regard to the existing models gathered from previous investigation. The comparison is predicated on the level of precision attained by each model throughout several research endeavours.

Model	Authors	Accuracy
CNN	My report	99.16%
VGG-16 based CNN	My report	98.85%
KNN	My report	84.74%
Random Forest	My report	93.36%
CNN	(Irmak, E et al., 2021)	92.66%
CNN	(Díaz-Pernas, F.J et al., 2021)	97.3%
Cnn using AlexNet and GoogLeNet	(Muis, A. et al., 2024)	98%
VGG-16 Ensembling Learning	(Younis, A. et al., 2022)	98.14
VGG-16 Ensembling Learning	(Gayathri, P. et al., 2023)	94%
VGG-16	(Nait Belaid, O. et al., 2020)	96.5%
KNN	(Wasule, V. et al., 2017)	86% (clinical database) and 72.5% (Brats database)
(KNN-SVM)	(Machhale, K. et al., 2015)	98%
KNN	(Rajini, N.H. et al., 2011)	99%
Random Forest	(Sutradhar, P. et al., 2021)	77.87%
Random Forest	(Amiri, S. et al., 2016)	72%

Table 4.2: Comparing with Existing model.

This section provides a comparison between the models created in my research and the models found in other studies. The table presents the accuracy of numerous machine learning models, such as CNN, VGG-16 based CNN, KNN, and Random Forest, as utilized in various research studies. The CNN model in my research achieved an accuracy of 99.16%, while the VGG-16 based CNN achieved an accuracy of 98.85%. Both models outperformed several prior studies, including those conducted by (Irmak et al., 2021) and Díaz -Pernas, F.J et al. (2021), who reported accuracies of 92.66% and 97.3% respectively. The KNN model in my research attained an accuracy of 84.74%, which is commendable but marginally inferior to previous studies, such as the 99% reported by Rajini, N.H. et al. (2011). The Random Forest model in my study achieved a 93.36% accuracy, outperforming the models of (Sutradhar, P. et al., 2021) and (Amiri, S. et al., 2016), which had accuracies of 77.87% and 72%, respectively. In summary, the models I constructed in my research consistently exhibit high performance, frequently surpassing the accuracy of similar models found in existing literature.

4.3 Discussion and Challenges:

The main hurdle in my study was the lack of compressive and diverse dataset, resulting in substantial overfitting in the machine learning models I used, such as CNN, VGG16, KNN, and Random Forest. Deep learning algorithms like as CNNs and VGG16 are particularly susceptible to overfitting when trained on an insufficient amount of data, whilst KNN and Random Forest models have difficulties in capturing the whole spectrum of properties required for accurate classification. This emphasized the need of having enough data to efficiently train these models.

In order to ease these difficulties, I used several tactics. The technique of cross-validation was used to more accurately measure the performance of the model and mitigate the problem of overfitting. Hyperparameter optimization was performed to enhance the performance of the model while working within the limitations of the data that was provided. Data augmentation was used on the CNN and VGG16 models to increase the dataset size and enhance generalization. However, the models' capacity to apply knowledge to new data was limited due to the limited dataset, emphasizing the urgent need for large and diverse datasets in order to create efficient machine learning models.

In addition, finding the most suitable value for k in the KNN model posed additional difficulty. Initial grid searches indicated that a value of k=1 resulted in significant overfitting. In order to tackle this issue, I carried out a methodical trial-and-error procedure utilizing grid search to identify an optimal k value that avoided overfitting while preserving model efficiency. The approach was laborious and emphasized the intricacies of hyperparameter modification, especially in situations with limited data.

Chapter 5

Conclusion

This study focused on utilizing machine learning and deep learning methods to detect and classify brain tumors by analyzing MRI data. The study encompassed the creation and assessment of multiple models, such as Convolutional Neural Networks (CNN), a CNN based on the VGG-16 architecture, K-Nearest Neighbors (KNN), and Random Forest. By conducting thorough experimentation and comparison, it was noted that deep learning models, specifically the CNN and VGG-16 architectures, exhibited higher accuracy and better generalization skills in contrast of traditional machine learning models. The CNN model attained an accuracy of 99.16%, while the VGG-16 derived model closely trailed with 98.85%. On the other hand, the KNN and Random Forest models, while successful, exhibited diminished accuracy, emphasizing the difficulties associated with employing less sophisticated models for complex image classification tasks. In addition to its achievements, the research faced difficulties such as overfitting, especially in deep learning models, and the constraints provided by the dataset's size and diversity. These findings emphasize the significance of selecting the appropriate model, fine-tuning the hyperparameters, and enhancing the data in order to optimize the performance of machine learning models in medical imaging applications.

Research Questions:

Question 1: How do various machine learning and deep learning algorithms including CNN, CNN-VGG-16, KNN, and Random Forest improve the accuracy and efficiency of image classification of MRI brain tumor?

Answer: Utilizing machine learning and deep learning techniques such as CNN, CNN-VGG-16, KNN, and Random Forest greatly improves the accuracy and efficiency of classifying MRI brain tumor images. These algorithms leverage their distinct abilities to interpret and learn from image data. CNNs and CNN-based models, like VGG-16, are highly proficient at extracting features from detailed MRI image data. This is attributed to their deep architectures, which enable them to automatically recognize and understand complex patterns within the images.

However, KNN and Random Forest make contributions through their own methodologies. The K-nearest neighbors (KNN) approach, which is both simple and efficient, achieves a classification accuracy of 85.35% by comparing new data points with existing data points. The ensemble learning capability of Random Forest enhances classification accuracy to 93.82% by combining the predictions of numerous decision trees, thereby reducing the possibility of overfitting and enhancing generalization.

These algorithms work well together, as CNNs and VGG-16 excel at extracting complex features, while KNN and Random Forest enhance the system's reliability with their effective classification methods.

Question 2: Is Data augmentation important in deep learning and machine learning to improve their performance?

Answer: Yes, Data augmentation is crucial in deep learning and machine learning, particularly for tasks that entail picture data, such as the diagnosis of brain tumors. The process involves

creating new training samples by applying various transformations, such as rotations, flips, and zooms, to the original data. This methodology allows models such as CNNs and VGG-16 to enhance their ability to make generalizations by training on a more diverse dataset. As a result, it reduces the problem of overfitting and enhances their performance on new and unexplored data. Data augmentation can significantly improve the model's capacity to see patterns and generate precise predictions when there is a lack of data.

Future works:

This research has the potential to be expanded from multiple ideas such as:

- Gathering more extensive and varied datasets to enhance model performance and generalization on unseen data.
- Investigating cutting-edge methods of data augmentation, like the generation of synthetic data using Generative Adversarial Networks (GANs).
- Utilizing transfer learning with architectures such as ResNet, EfficientNet, or Vision Transformers that goes beyond VGG-16.
- Enhancing classification accuracy by combining CNNs with additional machine learning methods for decision-level fusion.
- Enhancing generalization and lowering variance by combining several models through ensemble learning techniques.

Reference

1. Agarwal, M., Rani, G., Kumar, A., Kumar, P., Manikandan, R. and Gandomi, A.H., 2024. Deep learning for enhanced brain Tumor Detection and classification. *Results in Engineering*, *22*, p.102117.
2. Ait Amou, M., Xia, K., Kamhi, S. and Mouhafid, M., 2022. A Novel MRI Diagnosis Method for Brain Tumor Classification Based on CNN and Bayesian Optimization. Healthcare 2022, 10, 494.
3. Amiri, S., Rekik, I. and Mahjoub, M.A., 2016, March. Deep random forest-based learning transfer to SVM for brain tumor segmentation. In *2016 2nd International Conference on advanced technologies for signal and image processing (ATSIP)* (pp. 297-302). IEEE.
4. Belaid, O.N. and Loudini, M., 2020. Classification of brain tumor by combination of pre-trained vgg16 cnn. *Journal of Information Technology Management*, *12*(2), pp.13-25.
5. Chithra, P.L. and Dheepa, G., 2020. An Efficient Cascaded CNN Architecture for Brain Tumor Detection in MRI Images. *Int J Innov Technol Explor Eng*, *9*(3), pp.1663-1668.
6. Chithra, P.L. and Dheepa, G., 2020. An Efficient Cascaded CNN Architecture for Brain Tumor Detection in MRI Images. *Int J Innov Technol Explor Eng*, *9*(3), pp.1663-1668.
7. Díaz-Pernas, F.J., Martínez-Zarzuela, M., Antón-Rodríguez, M. and González-Ortega, D., 2021, February. A deep learning approach for brain tumor classification and segmentation using a multiscale convolutional neural network. In *Healthcare* (Vol. 9, No. 2, p. 153). MDPI.
8. Gayathri, P., Dhavileswarapu, A., Ibrahim, S., Paul, R. and Gupta, R., 2023. Exploring the potential of vgg-16 architecture for accurate brain tumor detection using deep learning. *Journal of Computers, Mechanical and Management*, *2*(2), pp.13-22.
9. Gurusamy, R. and Subramaniam, V., 2017. A machine learning approach for MRI brain tumor classification. *Computers, Materials and Continua*, *53*(2), pp.91-109.
10. Imam, R. and Alam, M.T., 2023, August. Optimizing Brain Tumor Classification: A Comprehensive Study on Transfer Learning and Imbalance Handling in Deep Learning Models. In *International Workshop on Epistemic Uncertainty in Artificial Intelligence* (pp. 74-88). Cham: Springer Nature Switzerland.
11. Irmak, E., 2021. Multi-classification of brain tumor MRI images using deep convolutional neural network with fully optimized framework. *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*, *45*(3), pp.1015-1036.
12. Jiang, S., Gu, Y. and Kumar, E., 2023. Magnetic resonance imaging (mri) brain tumor image classification based on five machine learning algorithms. *Cloud Computing and Data Science*, pp.122-133.
13. Joseph, R.P., Singh, C.S. and Manikandan, M., 2014. An Efficient Method of Modified Centroid K Means Clustering Algorithm for Medical Images.C.S. and Manikandan, M., 2014. Brain tumor MRI image segmentation and detection in image processing. *International Journal of Research in Engineering and Technology*, *3*(1), pp.1-5.
14. Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., Van Der Laak, J.A., Van Ginneken, B. and Sánchez, C.I., 2017. A survey on deep learning in medical image analysis. *Medical image analysis*, *42*, pp.60-88.
15. Machhale, K., Nandpuru, H.B., Kapur, V. and Kosta, L., 2015, May. MRI brain cancer classification using hybrid classifier (SVM-KNN). In *2015 International Conference on Industrial Instrumentation and Control (ICIC)* (pp. 60-65). IEEE.

16. Muis, A., Sunardi, S. and Yudhana, A., 2024. Cnn-based approach for enhancing brain tumor image classification accuracy. *International Journal of Engineering*, *37*(5), pp.984-996.
17. Rajini, N.H. and Bhavani, R., 2011, June. Classification of MRI brain images using k-nearest neighbor and artificial neural network. In *2011 International conference on recent trends in information technology (ICRTIT)* (pp. 563-568). IEEE.
18. Soltaninejad, M., Zhang, L., Lambrou, T., Yang, G., Allinson, N. and Ye, X., 2018. MRI brain tumor segmentation and patient survival prediction using random forests and fully convolutional networks. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: Third International Workshop, BrainLes 2017, Held in Conjunction with MICCAI 2017, Quebec City, QC, Canada, September 14, 2017, Revised Selected Papers 3* (pp. 204-215). Springer International Publishing.
19. Wasule, V. and Sonar, P., 2017, May. Classification of brain MRI using SVM and KNN classifier. In *2017 third international conference on sensing, signal processing and security (ICSSS)* (pp. 218-223). IEEE.
20. Younis, A., Qiang, L., Nyatega, C.O., Adamu, M.J. and Kawuwa, H.B., 2022. Brain tumor analysis using deep learning and VGG-16 ensembling learning approaches. *Applied Sciences*, *12*(14), p.7282.

Appendix

Literature Review

Joseph et al. (2014) studied MRI brain tumor image segmentation and detection using K-means clustering followed by morphological filtering. The dataset contains a variety of brain tumors, and the proposed method was tested on pre-processed MRI images to remove noise and convert to grayscale. The challenges include image degradation during acquisition, the labor-intensive nature of manual segmentation, minute differences between healthy and tumorous tissues, and the computational complexity of current algorithms. To improve accuracy and efficiency, the study suggests improving preprocessing techniques, employing advanced and less computationally intensive clustering methods, developing fully automated segmentation processes, and optimizing algorithms for real-time performance.

Background Study

In modern image recognition, convolutional neural networks (CNNs) are very important because they handle complex data well by using shared weights and convolutional filters, which make the computer's job easier while improving feature detection. VGG-16 is one of these architectures that stands out for its organized simplicity and efficiency. With the use of tiny 3x3 filters, VGG-16 uses sixteen layers—thirteen convolutional layers and three fully connected layers—to capture intricate patterns in images. This architecture is especially useful for medical imaging applications, like brain tumor detection, as it performs exceptionally well in recognizing and categorizing tumors from MRI scans. Furthermore, traditional machine learning techniques such as Random Forest and k-Nearest Neighbors (k-NN), which offer accurate classification and regression solutions, work well with CNNs to make diagnostic procedures more accurate and dependable in many fields.

CNN

Convolutional neural networks (CNNs) use neurons that have weights and biases. These neurons can do a variety of tasks, such as applying non-linearities, adding biases, and performing dot products. While standard neural networks assume that the input data are images, CNNs reduce the number of parameters by using shared weights and convolutional filters. These filters detect features that appear repeatedly in various regions of the image. Since pooling layers are widely used to reduce dimensionality and improve feature recognition, CNNs are highly helpful for tasks involving images. Gu et al. (2015) claim that this methodical approach saves computational resources while improving performance in image and video identification tasks.

VGG-16

The Visual Geometry Group at the University of Oxford created the VGG-16 architecture, which is a well-liked convolutional neural network (CNN) for image identification applications. Thirteen convolutional layers, three fully connected layers, and five max-pooling layers make up its sixteen weight layers. This can automatically learn and recognize patterns within images. VGG-16, with its deep layers and fine-tuned filters, is particularly good at capturing detailed spatial features, making it excellent for distinguishing between different types of brain tumors. The VGG-16 retains a manageable number of settings while capturing complex patterns in input images using modest 3x3 filters. Highly regarded for its ease of use

and effectiveness, this design is perfect for a range of image classification and identification applications, including medical imaging. VGG-16's performance and applicability can be increased by fine-tuning it on smaller datasets, such as those used to identify brain tumors. The model's hierarchical structure allows it to learn strong feature representations, which improves its ability to detect brain tumors from MRI scans. Research has demonstrated that VGG-16 can distinguish between different types of tumors, including meningiomas, gliomas, and pituitary tumors. This ability can significantly support medical diagnosis (Belaid & Loudini, 2020; Gayathri et al., 2023; Younis et al., 2022).

KNN

Fix and Hodges created the k-Nearest Neighbors (k-NN) technique in 1951 as a non-parametric approach for regression and classification applications. It is predicated on the idea that similar objects are found in close proximity to one another. By using a distance metric, typically Euclidean distance, to determine the k nearest neighbors, the k-NN algorithm classifies a data point based on how its neighbors are classified, with a majority vote among them. Due to its straightforward yet effective methodology, k-NN is already widely used in machine learning and pattern recognition applications (Machhale, K et al., 2015; Wasule & Sonar, 2017).

Random Forest

Leo Breiman created Random Forest, an ensemble learning technique, in 2001. It produces a huge number of decision trees during training and outputs the class mode for classification tasks. In order to decrease overfitting and increase forecast accuracy, this method makes use of several decision trees. To ensure tree diversity and boost robustness, each tree in the forest is trained on a random subset of the data using replacement (bootstrap sampling). At each node, a random set of characteristics is selected for splitting. The method's inherent ability to estimate missing data and maintain accuracy even when a sizable portion of the data is missing, together with its ability to handle high-dimensional data, are its strongest points. It is used in a lot of different areas, including medical imaging. Using Random Forest to separate and label brain tumors from MRI scans has been shown to greatly improve the accuracy and efficiency of diagnostics (Amiri et al., 2016; Wasule & Sonar, 2017; Breiman, 2001).

Script for Brain tumor classification

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

# Importing libraries
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from PIL import Image

import random
import tensorflow as tf
from tensorflow import keras
```

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.layers import RandomFlip, RandomRotation, RandomContrast,
RandomZoom, RandomTranslation
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import VGG16

from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import learning_curve
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import joblib
# Defining testing and training path
directory_path = '/content/drive/MyDrive/Final project'
testing_path = directory_path + '/Testing'
training_path = directory_path + '/Training'
def load_images(dir_path):
    """
    Function to load images from directory
    """
    image_count_dict = {}
    image_paths = []
    image_labels = []
    for class_image_dir in os.listdir(dir_path):
        class_image_path = os.path.join(dir_path, class_image_dir)
        if os.path.isdir(class_image_path):
            image_count_dict[class_image_dir] = len(os.listdir(class_image_path))
            for image_name in os.listdir(class_image_path):
                image_paths.append(os.path.join(class_image_path, image_name))
                image_labels.append(class_image_dir)
    return image_count_dict, image_paths, image_labels

training_count, training_image_paths, training_labels = load_images(training_path)
testing_count, testing_image_paths, testing_labels = load_images(testing_path)

train_data_df = pd.DataFrame(list(training_count.items()), columns=['Tumor Type', 'Training
Count'])
test_data_df = pd.DataFrame(list(testing_count.items()), columns=['Tumor Type', 'Testing
Count'])

merged_data_df = pd.merge(train_data_df, test_data_df, on='Tumor Type')
merged_data_df.set_index('Tumor Type', inplace=True)

ax = merged_data_df.plot(kind='bar', figsize=(10, 6))
plt.title('Tumor count in the training and testing datasets')
plt.xlabel('Tumor Type')
```

```
plt.ylabel('Count')
plt.xticks(rotation=45)
for p in ax.patches:
    ax.annotate(str(int(p.get_height())), (p.get_x() * 1.005, p.get_height() * 1.005))

plt.show()

unique_name_list = list(set(training_labels))
print('unique names: ', unique_name_list)
class_namings = {label: idx for idx, label in enumerate(unique_name_list)}
print('class naming: ', class_namings)

training_labels_indx = [class_namings[label] for label in training_labels]
testing_labels_indx = [class_namings[label] for label in testing_labels]
def display_images(path, label_idx, class_namings, index_list=range(10), img_size=250,
figsize=(12, 8)):
    """
    Function to display dataset images
    """
    img_count = len(index_list)
    row = (img_count + 3) // 4
    index_to_class = {v: k for k, v in class_namings.items()}
    _, plot_axes = plt.subplots(nrows=row, ncols=4, figsize=figsize)
    plot_axes = plot_axes.flatten()

    for i, indx in enumerate(index_list):
        if i >= img_count or indx >= len(path):
            plot_axes[i].axis('off')
            continue
        img = load_img(path[indx], target_size=(img_size, img_size))
        plot_axes[i].imshow(img)
        class_name = index_to_class[label_idx[indx]]
        plot_axes[i].set_title(f'{indx}: {class_name}')
        plot_axes[i].axis('off')

    for j in range(img_count, len(plot_axes)):
        plot_axes[j].axis('off')

    plt.tight_layout()
    plt.show()

combined_data = list(zip(training_image_paths, training_labels_indx))
random.shuffle(combined_data)
training_image_paths, training_labels_indx = zip(*combined_data)
display_images(training_image_paths, training_labels_indx, class_namings,
index_list=range(12))
data_augmentations = Sequential([
    RandomFlip("horizontal"),
    RandomRotation(0.02, fill_mode='constant'),
    RandomContrast(0.1),
```

```
    RandomZoom(height_factor=0.01, width_factor=0.05),
    RandomTranslation(height_factor=0.0015, width_factor=0.0015, fill_mode='constant'),
])
img_dim = (250, 250)
batch_size = 32

def get_image(image_path, label):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, img_dim)
    return img, label

def preprocess_training(image, label):
    img = data_augmentations(image) / 255.0
    return img, label

def preprocess_testing(img, label):
    return img / 255.0, label

training_image_paths = np.array(training_image_paths)
training_labels_indx = np.array(training_labels_indx)
testing_image_paths = np.array(testing_image_paths)
testing_labels_indx = np.array(testing_labels_indx)

train_data = tf.data.Dataset.from_tensor_slices((training_image_paths, training_labels_indx))
train_data = train_data.map(lambda x, y: get_image(x, y),
num_parallel_calls=tf.data.AUTOTUNE)
train_data = train_data.map(preprocess_training, num_parallel_calls=tf.data.AUTOTUNE)

test_data = tf.data.Dataset.from_tensor_slices((testing_image_paths, testing_labels_indx))
test_data = test_data.map(lambda x, y: get_image(x, y),
num_parallel_calls=tf.data.AUTOTUNE)
test_data = test_data.map(preprocess_testing, num_parallel_calls=tf.data.AUTOTUNE)

train_data_preprocessed = train_data.batch(32).prefetch(buffer_size=tf.data.AUTOTUNE)
test_data_preprocessed = test_data.batch(32).prefetch(buffer_size=tf.data.AUTOTUNE)
def display_augmented_images(dataset, shape, class_naming, figsize=(15, 6)):
    """
    Function to display augmented images
    """
    plt.figure(figsize=figsize)
    idx_to_class = {v: k for k, v in class_naming.items()}
    for img, label in dataset.take(1):
        for i in range(shape[0] * shape[1]):
            ax = plt.subplot(shape[0], shape[1], i + 1)
            plt.imshow(img[i].numpy().squeeze())
            plt.title(idx_to_class[label.numpy()[i]])
            plt.axis("off")

    plt.tight_layout()
```

```
    plt.show()

display_augmented_images(train_data_preprocessed, shape=(2, 6),
class_naming=class_namings)
num_class = len(class_namings.keys())
img_shape = (img_dim[0], img_dim[1], 3)
epoch = 50

inv_class_namings = {v: k for k, v in class_namings.items()}
print(f'Inverse class mappings: {inv_class_namings}')

print(f'Number of Classes: {num_class}')
print(f'Image shape: {img_shape}')
print(f'Epochs: {epoch}')
# print(f'Batch size: {batch_size}')

def label_encoder(image, label):
    return image, tf.one_hot(label, depth=num_class)

train_data_preprocessed = train_data_preprocessed.map(label_encoder,
num_parallel_calls=tf.data.AUTOTUNE)
test_data_preprocessed = test_data_preprocessed.map(label_encoder,
num_parallel_calls=tf.data.AUTOTUNE)
def cnn_model_creation(input_shape, num_class):
    """Function to create cnn model"""

    model = Sequential([
        Input(shape=input_shape),
        Conv2D(64, (5, 5), activation="relu"),
        MaxPooling2D(pool_size=(3, 3)),
        Conv2D(128, (4, 4), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(128, (4, 4), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(512, activation="relu"),
        Dense(num_class, activation="softmax")
    ])
    return model
input_shape = (img_dim[0], img_dim[1], 3)
cnn_model = cnn_model_creation(input_shape, num_class)
optimizer = Adam(learning_rate=0.001, beta_1=0.85, beta_2=0.9925)
cnn_model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])
class AdjustLROnMultipleAccuracies(tf.keras.callbacks.Callback):
    """
    Callback to adjust learning rate based on multiple accuracies.
    """
    def __init__(self, threshold, factor, monitor='val_accuracy', verbose=1):
        super(AdjustLROnMultipleAccuracies, self).__init__()
```

```
        self.threshold = threshold
        self.factor = factor
        self.monitor = monitor
        self.verbose = verbose
        self.threshold_reached = [False] * len(threshold)

    def after_epoch_completion(self, epoch, logs=None):
        epoch_accuracy = logs.get(self.monitor)
        for i, threshold in enumerate(self.threshold):
            if epoch_accuracy >= threshold and not self.threshold_reached[i]:
                optimizer = self.model.optimizer
                old_lr = optimizer.learning_rate.numpy()
                new_lr = old_lr * self.factor
                optimizer.learning_rate.assign(new_lr)
                self.thresholds_reached[i] = True
                if self.verbose > 0:
                    print(f"\nEpoch {epoch+1}: {self.monitor} reached {threshold}. Reducing
learning rate from {old_lr} to {new_lr}.")
thresholds = [0.96, 0.99, 0.9935]
model_file = 'cnn_model.keras'
lr_scheduler = AdjustLROnMultipleAccuracies(threshold=thresholds, factor=0.75,
monitor='val_accuracy', verbose=False)
model_lr_reduction = ReduceLROnPlateau(monitor='val_loss', factor=0.8, min_lr=1e-4,
patience=4, verbose=False)
model_checkpoint = ModelCheckpoint(model_file, monitor='val_accuracy', mode='max',
save_best_only=True, verbose=False)
# Train the model
cnn_history = cnn_model.fit(
    train_data_preprocessed,
    epochs=epoch,
    validation_data=test_data_preprocessed,
    callbacks=[model_lr_reduction, model_checkpoint, lr_scheduler],
    verbose=True
)
# Load the best model
cnn_model = load_model('cnn_model.keras')

# Evaluate the model on the training set
training_loss, training_accuracy = cnn_model.evaluate(train_data_preprocessed)
print(f"Training accuracy: {training_accuracy*100:.4f}%")

# Evaluate the model on the test set
testing_loss, cnn_testing_accuracy = cnn_model.evaluate(test_data_preprocessed)
print(f"Testing accuracy: {cnn_testing_accuracy*100:.4f}%")
_, plot_axes = plt.subplots(ncols=2, figsize=(15, 6))

# Displaying training and validation accuracy over epochs
plot_axes[0].plot(cnn_history.history['accuracy'], marker='o', linestyle='-', color='blue')
plot_axes[0].plot(cnn_history.history['val_accuracy'], marker='o', linestyle='-',
color='orange')
```

```
plot_axes[0].set_title('Model Accuracy')
plot_axes[0].set_xlabel('Epoch')
plot_axes[0].set_ylabel('Accuracy')
plot_axes[0].legend(['Train', 'Validation'], loc='lower right')
plot_axes[0].grid(alpha=0.2)

# Displaying training and validation loss over epochs
plot_axes[1].plot(cnn_history.history['loss'], marker='o', linestyle='-', color='blue')
plot_axes[1].plot(cnn_history.history['val_loss'], marker='o', linestyle='-', color='orange')
plot_axes[1].set_title('Model Loss')
plot_axes[1].set_xlabel('Epoch')
plot_axes[1].set_ylabel('Loss')
plot_axes[1].legend(['Train', 'Validation'], loc='upper right')
plot_axes[1].grid(alpha=0.2)

# Highlight lowest validation accuracy
min_val_acc_epoch = np.argmax(cnn_history.history['val_accuracy'])
min_val_acc = np.max(cnn_history.history['val_accuracy'])
plot_axes[0].plot(min_val_acc_epoch, min_val_acc, 'ro', markersize=15, alpha=0.5)
plot_axes[0].annotate(f'Lowest\n{min_val_acc:.4f}', xy=(min_val_acc_epoch, min_val_acc),
            xytext=(min_val_acc_epoch - 100, min_val_acc - 100), textcoords='offset points',
            arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=.2'))

plt.tight_layout()
plt.show()
# Using test data to make predictions that are true
true_label_list = []
predicted_label_list = []

# Iterate  over the dataset to gather accurate labels and predictions.
for img, lbs in test_data_preprocessed.unbatch():
    # Store true labels (Convert one-hot to index)
    true_label = np.argmax(lbs.numpy())
    true_label_list.append(true_label)

    # Get model prediction (predicts batch dimension).
    prediction = cnn_model.predict(tf.expand_dims(img, 0), verbose=False)
    predicted_label = np.argmax(prediction)
    predicted_label_list.append(predicted_label)
def display_confusion_matrix(true_labels, predicted_labels, class_naming, metrics=False,
cmap='Blues'):
    """
    Function to determine the confusion matrix
    """

    con_mat = confusion_matrix(true_labels, predicted_labels)
    plt.figure(figsize=(8, 8))
    sns.heatmap(con_mat, annot=True, fmt="d", cmap=cmap, cbar=False)
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted Label")
```

```
    plt.ylabel("True Label")

    # Index mapping to class names in class_naming
    plt.xticks(ticks=np.arange(num_class) + 0.5, labels=class_naming.keys(), ha='center')
    plt.yticks(ticks=np.arange(num_class) + 0.5, labels=class_naming.keys(), va='center')
    plt.show()

    if metrics:
        # Compute Precision, Recall, and F1-Score for each class & overall accuracy
        precision = np.diag(con_mat) / np.sum(con_mat, axis=0)
        recall = np.diag(con_mat) / np.sum(con_mat, axis=1)
        f1_scores = 2 * precision * recall / (precision + recall)
        accuracy = np.sum(np.diag(con_mat)) / np.sum(con_mat)

        print("Class-wise metrics:")
        for i in range(len(class_naming)):
            class_name = list(class_naming.keys())[i]
            print(f"Class: {class_name}")
            print(f"Precision: {precision[i]:.4f}")
            print(f"Recall: {recall[i]:.4f}")
            print(f"F1-Score: {f1_scores[i]:.4f}\n")

        print(f"Overall Accuracy: {accuracy:.4f}")
# Display confusion matrix
display_confusion_matrix(true_label_list,
                         predicted_label_list,
                         class_namings,
                         metrics=True)
y_pred_prob = cnn_model.predict(test_data_preprocessed)
y_pred = np.argmax(y_pred_prob, axis=1)

# Step 2: Get the true labels from the test dataset and convert to class indices if needed
y_true = np.concatenate([y for x, y in test_data_preprocessed], axis=0)
if len(y_true.shape) > 1 and y_true.shape[1] > 1:
    y_true = np.argmax(y_true, axis=1)

# Step 3: Generate and print the classification report
classif_report = classification_report(y_true, y_pred,
target_names=inv_class_namings.values())

print("Classification Report:\n")
print(classif_report)
def display_sample_predictions(model, folder_path, indx_to_class, figsize=(10, 10)):
    """
    Displaying the sample images, true labels, and predicted labels.
    """

    plt.figure(figsize=figsize)
    sub_folders = [f for f in os.listdir(folder_path) if os.path.isdir(os.path.join(folder_path, f))]
    img_files = []
```

```
    true_labels = []
    for folder in sub_folders:
        subfolder_path = os.path.join(folder_path, folder)
        for img_file in os.listdir(subfolder_path):
            if img_file.endswith(('.png', '.jpg', '.jpeg')):
                img_files.append(os.path.join(subfolder_path, img_file))
                true_labels.append(folder)

    sample_count = len(img_files)
    if sample_count == 0:
        print("No images were located in the designated directory.")
        return

    num_cols = int(np.sqrt(sample_count))
    rows = (sample_count // num_cols) + (sample_count % num_cols > 0)

    for i, img_path in enumerate(img_files):
        sample_img = Image.open(img_path).resize((250, 250))  # Resize to 250x250
        img_array = np.array(sample_img) / 255.0  # Normalize the image
        batch_img = np.expand_dims(img_array, axis=0)

        prediction = model.predict(batch_img, verbose=False)
        predicted_label = np.argmax(prediction, axis=1)[0]
        class_predcition = indx_to_class[predicted_label]

        true_class = true_labels[i]

        plt.subplot(rows, num_cols, i + 1)
        plt.imshow(img)
        plt.title(f"True: {true_class}\nPredicted: {class_predcition}", color='green')
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Display samples with predictions
prediction_dir = '/content/drive/MyDrive/Final project/ImagePrediction'
display_sample_predictions(cnn_model, prediction_dir, inv_class_namings, figsize=(13, 12))
def display_misclassified_samples(model, dataset, indx_to_class, figsize=(10, 10)):
    """
    Display misclassified samples from test data
    """

    misclassified_image_list = []
    misclassified_label_list = []
    misclassified_prediction_list = []

    # Iterate over dataset to gather misclassified images
    for img, true_label in dataset.unbatch():
        batch_img = tf.expand_dims(img, 0)
```

```
            model_prediction = model.predict(batch_img, verbose=False)
            predicted_label = np.argmax(model_prediction, axis=1)[0]
            true_class_idx = np.argmax(true_label.numpy())

            if true_class_idx != predicted_label:
                misclassified_image_list.append(img.numpy().squeeze())
                misclassified_label_list.append(indx_to_class[true_class_idx])
                misclassified_prediction_list.append(indx_to_class[predicted_label])

    # Compute number of rows and columns for subplot
    misclassified_count = len(misclassified_image_list)
    cols = int(np.sqrt(misclassified_count)) + 1
    rows = misclassified_count // cols + (misclassified_count % cols > 0)

    # Displaying misclassified images
    misclassified_zip = zip(misclassified_image_list, misclassified_label_list,
misclassified_prediction_list)
    plt.figure(figsize=figsize)
    for i, (image, true_label, predicted_label) in enumerate(misclassified_zip):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(image, cmap='gray')
        plt.title(f"True: {true_label}\nPred: {predicted_label}", color='red')
        plt.axis('off')

    plt.tight_layout()
    plt.show()
# Display misclassified samples
display_misclassified_samples(model=cnn_model,
                              dataset=test_data_preprocessed,
                              indx_to_class=inv_class_namings,
                              figsize=(10, 6))
# Create CNN-VGG16 model
def vgg16_cnn_model(input_shape, num_class):
    """Create vgg16-cnn model"""

    base_model = VGG16(input_shape=input_shape, include_top=False, weights='imagenet')
    # Freeze all the layers in the base model
    for layer in base_model.layers:
        layer.trainable = False

    # Unfreeze the last few layers of the base model
    base_model.layers[-2].trainable = True
    base_model.layers[-3].trainable = True
    base_model.layers[-4].trainable = True

    inputs = Input(shape=input_shape)
    x = base_model(inputs, training=False)
    x = Flatten()(x)
    x = Dropout(0.3)(x)
    x = Dense(128, activation='relu')(x)
```

```
    x = Dropout(0.2)(x)
    outputs = Dense(num_class, activation='softmax')(x)

    model = tf.keras.Model(inputs, outputs)
    return model

input_shape = (img_dim[0], img_dim[1], 3)
epoch = 10
vgg16_model = vgg16_cnn_model(input_shape, num_class)
optimizer = Adam(learning_rate=0.0001)
vgg16_model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

thresholds = [0.96, 0.99, 0.9935]
vgg16_cnn_model_file = 'cnn_vgg16_model.keras'
lr_scheduler = AdjustLROnMultipleAccuracies(threshold=thresholds, factor=0.75,
monitor='val_accuracy', verbose=False)
model_lr_reduction = ReduceLROnPlateau(monitor='val_loss', factor=0.8, min_lr=1e-4,
patience=4, verbose=False)
model_checkpoint = ModelCheckpoint(vgg16_cnn_model_file, monitor='val_accuracy',
mode='max', save_best_only=True, verbose=False)
# Train the model
vgg16_cnn_history = vgg16_model.fit(
    train_data_preprocessed,
    epochs=epoch,
    validation_data=test_data_preprocessed,
    callbacks=[model_lr_reduction, model_checkpoint, lr_scheduler],
    verbose=True
)
# Load the vgg16-cnn model
vgg16_cnn = load_model(vgg16_cnn_model_file)

training_loss, training_accuracy = vgg16_cnn.evaluate(train_data_preprocessed)
print(f"Training accuracy: {training_accuracy*100:.4f}%")

testing_loss, cnn_vgg16_testing_accuracy = vgg16_cnn.evaluate(test_data_preprocessed)
print(f"Testing accuracy: {cnn_vgg16_testing_accuracy*100:.4f}%")
_, plot_axes = plt.subplots(ncols=2, figsize=(15, 6))

# Displaying training and validation accuracy over epochs
plot_axes[0].plot(vgg16_cnn_history.history['accuracy'], marker='o', linestyle='-',
color='blue')
plot_axes[0].plot(vgg16_cnn_history.history['val_accuracy'], marker='o', linestyle='-',
color='orange')
plot_axes[0].set_title('Model Accuracy')
plot_axes[0].set_xlabel('Epoch')
plot_axes[0].set_ylabel('Accuracy')
plot_axes[0].legend(['Train', 'Validation'], loc='lower right')
plot_axes[0].grid(alpha=0.2)
```

```
# Diplaying training and validation loss over epochs
plot_axes[1].plot(vgg16_cnn_history.history['loss'], marker='o', linestyle='-', color='blue')
plot_axes[1].plot(vgg16_cnn_history.history['val_loss'], marker='o', linestyle='-',
color='orange')
plot_axes[1].set_title('Model Loss')
plot_axes[1].set_xlabel('Epoch')
plot_axes[1].set_ylabel('Loss')
plot_axes[1].legend(['Train', 'Validation'], loc='upper right')
plot_axes[1].grid(alpha=0.2)

# Highlight lowest validation accuracy
min_val_acc_epoch = np.argmax(vgg16_cnn_history.history['val_accuracy'])
min_val_acc = np.max(vgg16_cnn_history.history['val_accuracy'])
plot_axes[0].plot(min_val_acc_epoch, min_val_acc, 'ro', markersize=15, alpha=0.5)
plot_axes[0].annotate(f'Lowest\n{min_val_acc:.4f}', xy=(min_val_acc_epoch, min_val_acc),
            xytext=(min_val_acc_epoch - 100, min_val_acc - 100), textcoords='offset points',
            arrowprops=dict(arrowstyle='->', connectionstyle='arc3,rad=.2'))

plt.tight_layout()
plt.show()
true_label_list = []
predicted_label_list = []

# Iterate over dataset to gather predictions and true labels
for img, lab in test_data_preprocessed.unbatch():
    true_label = np.argmax(lab.numpy())
    true_label_list.append(true_label)

    prediction = vgg16_cnn.predict(tf.expand_dims(img, 0), verbose=False)
    predicted_label = np.argmax(prediction)
    predicted_label_list.append(predicted_label)
# Display confusion matrix for VGG16 Model
display_confusion_matrix(true_label_list, true_label_list, class_namings, metrics=True)
y_pred_prob = vgg16_cnn.predict(test_data_preprocessed)
y_pred = np.argmax(y_pred_prob, axis=1)

# Get the true labels from the test dataset and convert to class indices
y_true = np.concatenate([y for x, y in test_data_preprocessed], axis=0)
if len(y_true.shape) > 1 and y_true.shape[1] > 1:
    y_true = np.argmax(y_true, axis=1)

classif_report = classification_report(y_true, y_pred,
target_names=inv_class_namings.values())

print("Classification Report:\n")
print(classif_report)
# Ploting sample predictions for VGG16 Model
prediction_folder = '/content/drive/MyDrive/Final project/ImagePrediction'
display_sample_predictions(vgg16_cnn, prediction_folder, inv_class_namings, figsize=(13,
12))
```

```
# Displaying samples with predictions
display_misclassified_samples(
    model=vgg16_cnn,
    dataset=test_data_preprocessed,
    indx_to_class=inv_class_namings,
    figsize=(10, 6)
    )
def preprocess_data_images(image_path, image_size=(250, 250)):
    """
    Function to preprocess the images for machine learning model
    """
    image = load_img(image_path, target_size=image_size)
    image_array = img_to_array(image).flatten() / 255.0  # Flatten the image and normalize
    return image_array
X_train = np.array([preprocess_data_images(img_path) for img_path in
training_image_paths])
y_train = np.array(training_labels_indx)

X_test = np.array([preprocess_data_images(img_path) for img_path in testing_image_paths])
y_test = np.array(testing_labels_indx)
# Create and train KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Save the model
joblib.dump(knn_model, 'knn_model.pkl')

# Predicting based on the training set to determine training accuracy
y_train_prediction = knn_model.predict(X_train)
training_accuracy = accuracy_score(y_train, y_train_prediction)
print(f"Training Accuracy: {training_accuracy*100:.2f}%")

# Predicting based on the test set in order to determine test accuracy.
y_pred_knn = knn_model.predict(X_test)
knn_test_accuracy = accuracy_score(y_test, y_pred_knn)
print(f"Test Accuracy: {knn_test_accuracy*100:.2f}%")
# Display confusion matrix for knn model
con_mat_knn = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(8, 8))
sns.heatmap(con_mat_knn, annot=True, fmt="d", cmap='Blues', cbar=False,
xticklabels=class_namings.keys(), yticklabels=class_namings.keys())
plt.title("KNN Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

classif_report = classification_report(y_test, y_pred_knn,
target_names=class_namings.keys())
print("Classification Report:\n")
print(classif_report)
```

```
def display_learning_curve(estimator, title, X, y, cv=None, n_jobs=None,
train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Function to plot the learning curve for the model
    """
    plt.figure()
    plt.title(title)
    plt.xlabel("Training examples")
    plt.ylabel("Score")

    trainiing_sizes, training_scores, test_scores = learning_curve(estimator, X, y, cv=cv,
n_jobs=n_jobs, train_sizes=train_sizes)

    training_scores_mean = np.mean(training_scores, axis=1)
    training_scores_std = np.std(training_scores, axis=1)
    testing_scores_mean = np.mean(test_scores, axis=1)
    testing_scores_std = np.std(test_scores, axis=1)

    plt.grid()

    plt.fill_between(train_sizes, training_scores_mean - training_scores_std,
training_scores_mean + training_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, testing_scores_mean - testing_scores_std,
testing_scores_mean + testing_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, training_scores_mean, 'o-', color="r", label="Training score")
    plt.plot(train_sizes, testing_scores_mean, 'o-', color="g", label="Cross-validation score")

    plt.legend(loc="best")
    return plt

# Plot learning curve for knn model
display_learning_curve(knn_model, "Learning Curve for KNN", X_train, y_train, cv=5)
plt.show()
def predict_tumor_type(image_array, model, class_naming):
    """
    Function to predict the tumor type for sample image
    """
    image_array = image_array.reshape(1, -1)
    prediction = model.predict(image_array)

    # Decode the prediction to determine the type of tumor.
    inv_class_naming = {v: k for k, v in class_naming.items()}
    tumor_type = inv_class_naming[prediction[0]]
    return tumor_type

def display_predictions(model, folder_path, class_naming, image_size=(250, 250),
figsize=(13, 12)):
    """
    Function to plot predictions for all images in the prediction folder
    """
```

```
    image_path_list = []
    true_label_list = []

    # Collect image paths and corresponding labels
    for class_name in os.listdir(folder_path):
        class_folder = os.path.join(folder_path, class_name)
        if os.path.isdir(class_folder):
            for img_file in os.listdir(class_folder):
                if img_file.lower().endswith(('.png', '.jpg', '.jpeg')):
                    image_path_list.append(os.path.join(class_folder, img_file))
                    true_label_list.append(class_name)

    sample_count = len(image_path_list)
    rows = int(np.sqrt(sample_count))
    num_cols = (sample_count // rows) + (sample_count % rows > 0)

    plt.figure(figsize=figsize)

    for i in range(sample_count):
        img_path = image_path_list[i]
        true_label = true_label_list[i]
        img_array = preprocess_data_images(img_path, image_size=image_size)
        predicted_label = predict_tumor_type(img_array, model, class_naming)

        plt.subplot(rows, num_cols, i + 1)
        plt.imshow(img_array.reshape(image_size[0], image_size[1], 3))
        title_color = 'green' if true_label == predicted_label else 'red'
        plt.title(f"True: {true_label}\nPred: {predicted_label}", color=title_color)
        plt.axis('off')

    plt.tight_layout()
    plt.show()
# Predict and display the sample images
prediction_folder = '/content/drive/MyDrive/Final project/ImagePrediction'
knn_model = joblib.load('knn_model.pkl')
display_predictions(knn_model, prediction_folder, class_namings)
# Initialize and train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

rf_model.fit(X_train, y_train)

# Save the model
joblib.dump(rf_model, 'random_forest_model.pkl')
rf_model_file = joblib.load('random_forest_model.pkl')
y_prediction_rf = rf_model.predict(X_test)

# Predict on the training set to compute training accuracy
y_train_pred_rf = rf_model.predict(X_train)
training_accuracy_rf = accuracy_score(y_train, y_train_pred_rf)
print(f"Training Accuracy: {training_accuracy_rf*100:.4f}%")
```

```
# Predict on the test set to compute testing accuracy
testing_accuracy_rf = accuracy_score(y_test, y_prediction_rf)
print(f"Test Accuracy: {testing_accuracy_rf*100:.4f}%")

print("Classification Report:")
print(classification_report(y_test, y_prediction_rf, target_names=class_namings.keys()))
# Display confusion matrix fro random forest
con_mat_rf = confusion_matrix(y_test, y_prediction_rf)

plt.figure(figsize=(8, 8))
sns.heatmap(con_mat_rf, annot=True, fmt="d", cmap='Blues', cbar=False,
xticklabels=class_namings.keys(), yticklabels=class_namings.keys())
plt.title("Random Forest Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
# Plotting learning curve for random forest model
display_learning_curve(rf_model, "Learning Curve for Random Forest", X_train, y_train,
cv=5)
plt.show()
# Predict and display the sample images for random forest model
prediction_folder = '/content/drive/MyDrive/Final project/ImagePrediction'
rf_model = joblib.load('random_forest_model.pkl')
display_predictions(rf_model, prediction_folder, class_namings)
accuracies = {
   'CNN': cnn_testing_accuracy,
   'CNN - VGG16': cnn_vgg16_testing_accuracy,
   'KNN': knn_test_accuracy,
   'Random Forest': testing_accuracy_rf
}

# Plot accuracies
plt.figure(figsize=(10, 6))
plt.bar(accuracies.keys(), accuracies.values(), color=['blue', 'orange', 'green', 'yellow'])
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.ylim(0.8,1)
plt.show()
```