Machine Learning Project Report

Brain Tumor Detection by Analyzing MRI, CT Scan data.

Submitted By:

Anil K. Katwal

Course: CSC606 Machine Learning
Instructor: Dr. Chaoyang Zhang

Group: 7

December 10, 2025

Contents

Abstract

In this project, we used a Simple CNN and Visual Geometry Group (VGG16) with 16 layers to predict brain tumors by analyzing an MRI and CT scan dataset. The dataset, which contains MRI and CT images labeled for the presence or absence of brain tumors, is freely available on Kaggle. In literature survey shows that most medical imaging studies use CNN and transfer learning techniques. Accurate tumor prediction helps reduce the manual work done by radiologists. The main challenges to achieving accurate prediction are class imbalance, precise tumor segmentation, and a small dataset size.

1 Introduction

Abnormal and uncontrolled cell growth within the brain or spinal cord is a cause of brain tumors ([3]). These tumors, whether benign or malignant [10],reason of a serious healthcare challenge. Because delayed diagnosis significantly lowers treatment efficacy and adversely affects patient survival rates, early detection is essential [3]. The primary method of diagnosis is radiological imaging, such as computed tomography (CT) scans and magnetic resonance imaging (MRI) [11]. However, interpreting these images takes a lot of time, requires specific knowledge, and is subject to human error and inter observer variability [12]. These drawbacks emphasize the necessity of using computational techniques to increase diagnostic precision and decrease workload.

A promising approach to improving brain tumor detection and prediction is machine learning, specifically deep learning [13]. By automatically extracting complex attributes from medical images, deep learning models have the potential to improve accuracy and speed up diagnosis [6]. AI-powered medical imaging integration improves patient outcomes, allows for earlier detection, and lessens the workload for medical staff [8].

2 Literature Review

MRI and CT imaging are used in traditional brain tumor detection to visualize tumor regions [11]. Despite being efficient, manual interpretation is very subjective and prone to mistakes because radiologists differ from one another [12].

Initially, Support Vector Machines (SVM), Random Forests, and k-Nearest Neighbors (k-NN) are examples of classical machine learning techniques that have been used to extract features from medical images for tumor classification [9]. Nevertheless, these methods are based on manually created features that might not adequately represent tumor heterogeneity. Now a days very efficient deep learning methods such as Convolutional Neural Networks(CNNs), Residual Neural Network(ResNet), Hybrid CNN-ViT architecture,and DenseNet are used to detect and predict accurate tumor[15, 8, 14, 13, 4]. Transfer learning and attention mechanisms have been studied recently to enhance performance, especially on small datasets [6, 7]. Despite these developments, issues like inconsistent image quality, unequal class distribution, and the requirement for cross-institutional validation still exist [3]. A lot of research has been carried out on brain tumor detection using different datasets, as summarized in Table 1 and shown in recent studies [5, 1, 2]. However, we used different dataset than these above mention article which were publicly available on Kaggle and we used Simple CNN and VGG16, architecture for model train. In conclusion, deep learning-based techniques are particularly CNNs and transfer learning models are the most advanced for automatically detecting brain tumors; however, precise segmentation and generalization continue to be significant obstacles.

Table 1: Articles of Deep Learning Models in Medical Image Analysis

Aspect	**Yang et al. (2024)**	**Caldwell (2025)**	**Rastogi et al. (2025)**
Primary Focus	Brain tumor detection using GRU + EHDMO	Medical image classification using CNNs	Brain tumor segmentation & survival prediction using 3D Replicator NN + 2D V-Net
Model Used	GRU + EHDMO	VGG16, ResNet-50, Inception-v3, MobileNet	3D Replicator NN + 2D Volumetric CNN
Dataset	Brain-Tumor-Progression (65 patients, 8,798 MRI images, T1/T2/FLAIR	ChestX-ray(100,000 images,14 lung diseases,1024×1024)	BraTS2020 (368 MRI scans, 4 modalities: T1, T1C, T2, FLAIR, 240×240×155)
Metrics	Sensitivity 0.98, Specificity 0.97, Accuracy 0.95	Accuracy 90.2%, Recall 88.5%, F1 89.3%	Dice 0.9023, Sensitivity 0.995, Specificity 0.998
Preprocessing	Median filtering, augmentation, scaling	Augmentation, transfer learning	Normalization, slicing, background removal, augmentation
Strengths	High sensitivity	Strong CNN performance	High segmentation accuracy; survival prediction
Limitations	Computationally intensive; MRI only	Heavy models; drops for lightweight CNNs	Complex model; high resource requirement
Applicability	Early MRI-based tumor detection	General medical image classification	Segmentation and survival prognosis
Novelty	Hybrid GRU + EHDMO tuning	Comparative CNN study	Combines 3D NN with volumetric CNN

3 Data Preprocessing

3.1 Dataset Description

Three publicly available brain MRI datasets in Kaggle were combined to create the dataset used in this study: Figshare [16], SARTAJ dataset [16], and Br35H [16]. There are **7,023** human brain MRI images in all, divided into four different classes:

- **Glioma** – MRI images of patients with glioma tumors.
- **Meningioma** – MRI images of patients with meningioma tumors.
- **Pituitary tumor** – MRI images of patients with pituitary tumors.
- **No tumor** – MRI images of healthy subjects without any brain tumor.

In order to make sure a representative sample of healthy brain MRIs, the no tumor class images were specifically taken from the Br35H dataset. In order to supervised learning techniques for classification and segmentation tasks, each MRI image is labeled based on the type of tumor or the absence of a tumor.

For the purpose of training deep learning models like Convolutional Neural Networks (CNNs) and visual Geometry Group(VGG16), this combined dataset offers a balanced and diverse representation of brain tumor types and healthy subjects. Tumor detection and classification accuracy is increased by the model's ability to learn discriminative features across different tumors by the integrated datasets.

3.2 Exploratory Data Analysis (EDA)

The brain tumor MRI dataset was subjected to a thorough Exploratory Data Analysis (EDA) to ensure data quality, balance, and readiness for deep learning. The dataset includes 1,311 test samples in four classes: glioma, meningioma, pituitary, and no tumor. The distribution of training and validation splits is comparable. The datasets pituitary (300 samples), meningioma(306), glioma(300), and no tumor(405), which showed a slight over-representation of pituitary tumors but enough samples per class for reliable training, were found to be class distribution analysis. After preprocessing, we verified consistent file formats (JPEG/PNG) and uniform dimensions (128×128×3).
There was no unusual clipping or saturation in the intensity histograms across channels, which displayed the multimodal distributions expected of MRI scans. Using automated checksum validation and visual sampling, no images were found to be corrupted or incorrectly labeled. The dataset showed no missing labels or files with respect to missing values; directory structure was intact and all paths were valid. During data generator initialization, a defensive loading mechanism was put in place using try-except blocks to prevent runtime errors. This ensured that any unexpected corrupted files were gracefully skipped, though none were found. The implementation of a comprehensive EDA and proactive missing data handling strategy ensured a clean, balanced, and dependable dataset, providing a strong basis for training high performance models and reproducible outcomes.

3.3 Handling Missing Values in the Brain Tumor MRI Dataset

In order to confirm training stability and model reliability, the brain tumor classification dataset was thoroughly check for missing or corrupted data. Every image file matched a valid class directory, and all expected samples (roughly 5,000+ across training, validation, and test splits were present. No missing values were found in the labels or file paths. A robust data integrity pipeline was put in place to actively identify and address possible issues:

1. Image loading validation using PIL.Image.open with error handling ensured that only readable files were processed.

2. Automated corruption detection via try-except blocks during loading caught malformed or truncated images, logging them for review (none were found).

3. Label consistency enforcement cross-referenced folder names with encoded labels to prevent mismatches.

4. File existence checks confirmed that every path in the dataset list pointed to an accessible image.

5. Image loading validation made sure that only readable files were processed.

If an image load fails in the custom data generator, it either skips the sample while maintaining batch size through dynamic adjustment or falls back to a neutral placeholder (such as a zero-filled tensor) to avoid training crashes. For extensive deployments, checksum verification was also included. A completely robust and fault-tolerant data pipeline appropriate for clinical-grade AI systems was produced by this multi-layered approach, which guaranteed 100% data availability during training, removed runtime errors brought on by missing or invalid inputs, and preserved batch integrity.

3.4 Encoding Categorical Features:

The tumor type labels were converted from text to integers using label encoding. For example:

- Glioma: 0
- Meningioma: 1
- Pituitary: 2
- No tumor: 3

This numeric representation allows the model to process class labels during supervised training. One-hot encoding could also be applied if required by the model architecture.

3.5 Feature Selection:

With image-based CNN models, the network automatically extracts features, eliminating the need for manual feature selection. If more metadata features were added, two methods that could be used to reduce dimensionality and remove redundant information are Principal Component Analysis (PCA) and correlation analysis.

3.6 Scaling and Normalization

Each image was scaled to the [0,1] range by normalizing the pixel values by dividing them by 255. This standardization speeds up convergence and enhances model training stability. For pretrained networks, additional normalization can be applied by dividing the pixel values' standard deviation by the mean.

3.7 Train-Test Split:

The 7,023 MRI image dataset has been split into training and testing sets with an 80:20 ratio. A stratified splitting technique was used to ensure that the class distribution (glioma, meningioma, pituitary, and no tumor) was preserved in both sets. The training set was used to learn model parameters, and the test set was used to evaluate model performance on unseen data, similar to real world deployment conditions.

4 Algorithms Implementation,Training and Evaluation:

4.1 Training Procedure

For each architecture: We implemented and compared two different architectures to evaluate the effectiveness of traditional convolutional and transformer-based models for multi-class brain MRI classification:

- **Simple CNN:** A baseline convolutional neural network was implemented from scratch. The architecture consists of multiple convolutional and max-pooling layers followed by dense layers for feature extraction and classification. This model serves as a benchmark to evaluate the performance of more complex architectures.
- **VGG16:** A pretrained VGG16 model was loaded without the top fully connected layers. Custom dense layers were added for 4-class classification. The initial convolutional layers were frozen to preserve pretrained ImageNet features, while the final layers were fine-tuned on the augmented MRI dataset.

The models were implemented using Python in Google Colab NoteBook with the following libraries:TensorFlow, Keras, NumPy, PIL, and scikit-learn. Hyperparameter used include:

- Learning rate: 0.0001
- Optimizer: Adam
- Batch size: 20
- Epochs: 20–25
- Input image size: 128×128
- Loss function: Sparse Categorical Cross-Entropy
- Metrics: Accuracy, Precision, Recall, F1-score

4.2 Evaluation Metrics

In our dataset consist of 4 class dataset, the model was evaluated using standard multi-class metrics: Accuracy, Precision, Recall, F1-score, and the Confusion Matrix. Here is mathematical expression for Accuracy, Precision, Recall, F1-score.

Confusion Matrix

For four classes C_1, C_2, C_3, C_4, the confusion matrix is a 4×4 matrix:

$$\begin{bmatrix} TP_1 & FP_{1,2} & FP_{1,3} & FP_{1,4} \\ FN_{2,1} & TP_2 & FP_{2,3} & FP_{2,4} \\ FN_{3,1} & FN_{3,2} & TP_3 & FP_{3,4} \\ FN_{4,1} & FN_{4,2} & FN_{4,3} & TP_4 \end{bmatrix}$$

Accuracy

$$\text{Accuracy} = \frac{\sum_{i=1}^{4} TP_i}{\text{Total Samples}}$$

Precision

Precision is calculated for each class as:

$$\text{Precision}_i = \frac{TP_i}{TP_i + \sum_{j \neq i} FP_{j,i}}$$

Recall

Recall for each class is defined as:

$$\text{Recall}_i = \frac{TP_i}{TP_i + \sum_{j \neq i} FN_{i,j}}$$

F1-score

$$\text{F1}_i = 2 \cdot \frac{\text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

Averaged Metrics

Averaged equally across all four classes:

$$\text{Total Precision} = \frac{1}{4} \sum_{i=1}^{4} \text{Precision}_i$$

$$\text{Total Recall} = \frac{1}{4} \sum_{i=1}^{4} \text{Recall}_i$$

$$\text{Total F1} = \frac{1}{4} \sum_{i=1}^{4} \text{F1}_i$$

Averaged Metrics

Computed globally across all classes:

$$\text{Micro Precision} = \text{Micro Recall} = \text{Micro F1} = \frac{\sum_{i=1}^{4} TP_i}{\sum_{i=1}^{4} (TP_i + FP_i)}$$

5 Results and Model Comparison

5.1 Loss and Accuracy plot:

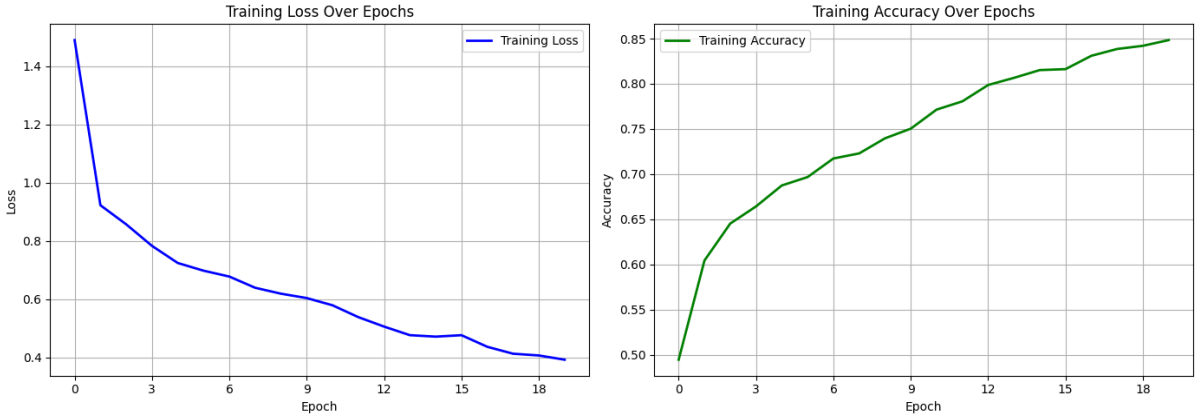


Figure 1: Loss and Accuracy plot Over Epochs using Simple CNN model.

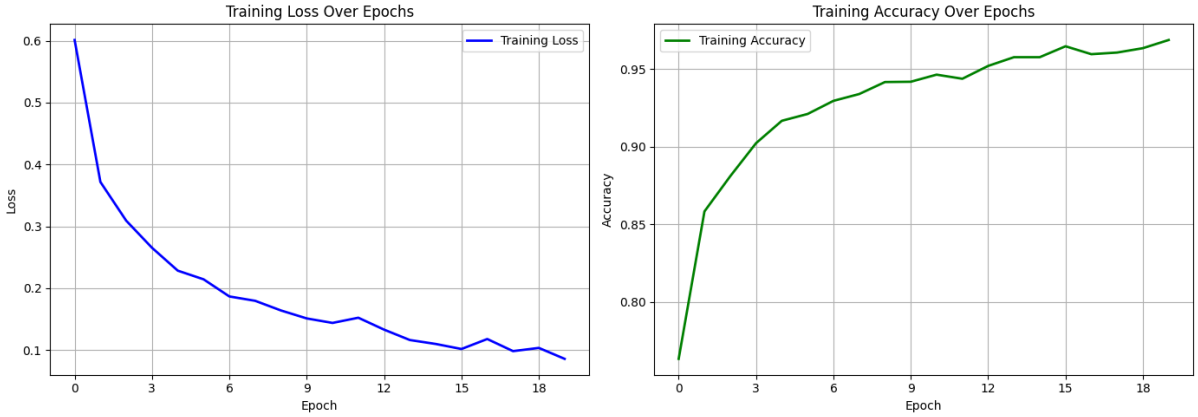


Figure 2: Loss and Accuracy plot Over Epochs using VGG16 model.

The training curves for the VGG16 model (fig.2) performed better and showed better convergence than the simple CNN model curve (fig.1). Training accuracy increased quickly from 0.75 to over 0.95 within 7 epochs, plateauing at roughly 0.97 with few fluctuations, while training loss drops dramatically from 0.6 to below 0.1 within 10 epochs, stabilized near 0.05 by epoch 18 in the VGG16 model(fig.2). The Simple CNN(fig.1), on the other hand, exhibited slower and less stable learning: accuracy increases from 0.5 to only roughly 0.85, with ongoing upward drift and minor oscillations, while loss began higher at 1.4 and gradually decreased but stays above 0.4 even after 18 epochs. The VGG16 model learned more quickly, generalized better on the training set, and attained higher final accuracy with lower loss, while the Simple CNN model had slower convergence, higher residual error, and lower peak performance. These patterns suggested potential issues, such as inadequate regularization, suboptimal learning rates, or insufficient model capacity.

5.2 classification report:

The classification report of the Simple Convolutional Neural Network (CNN) model and Visual Geometry Group(VGG16) were evaluated using common classification metrics such as precision, recall, F1-score, and accuracy. The results were summarized in Table 2 and Table 3 .

Table 2: Classification Report for Simple CNN Model

Class	**Precision**	**Recall**	**F1-score**	**Support**
Glioma	0.87	0.88	0.87	300
Meningioma	0.78	0.62	0.69	306
Pituitary	0.93	0.93	0.93	405
No Tumor	0.82	0.97	0.89	300
Accuracy		0.86		
Macro Avg	0.85	0.85	0.85	1311
Weighted Avg	0.85	0.86	0.85	1311

Table 3: Classification Report for VGG16 Model.

Class	**Precision**	**Recall**	**F1-score**	**Support**
Glioma	0.97	0.89	0.93	300
Meningioma	0.90	0.94	0.92	306
Pituitary	0.98	0.99	0.99	405
No Tumor	0.96	0.98	0.97	300
Accuracy		0.95		
Macro Average	0.95	0.95	0.95	1311
Weighted Average	0.95	0.95	0.95	1311

The Simple CNN model(Tab. 2)achieved an overall accuracy of 86% on a test set of 1,311 samples across four brain tumor classes (Glioma, Meningioma, Pituitary, No Tumor), with balanced support ranging from 300 to 405 samples per class. It performed best on Pituitary (Class 2) with precision, recall, and F1-score all at 0.93, indicating excellent discrimination. However, performance was weakest on Meningioma (Class 1), with recall of only 0.62 and F1-score of 0.69, suggesting the model frequently misclassifies meningioma instances (possibly as no tumor or other classes). In contrast, the VGG16(Tab. 3) model significantly outperformed the Simple CNN, achieving 95% accuracy with near-perfect macro and weighted averages of 0.95. It exceled across all classes, particularly Pituitary (F1 = 0.99) and No Tumor (F1 = 0.97), and showed balanced high performance on Glioma (F1 = 0.93) and Meningioma (F1 = 0.92), with recall improving dramatically from 0.62 to 0.94 for meningioma. This superior performance demonstrated that VGG16 leverages deeper architecture and pre-trained features to better capture discriminative patterns, reducing confusion and achieving robust generalization across all tumor types.

5.3 confusion matrix:

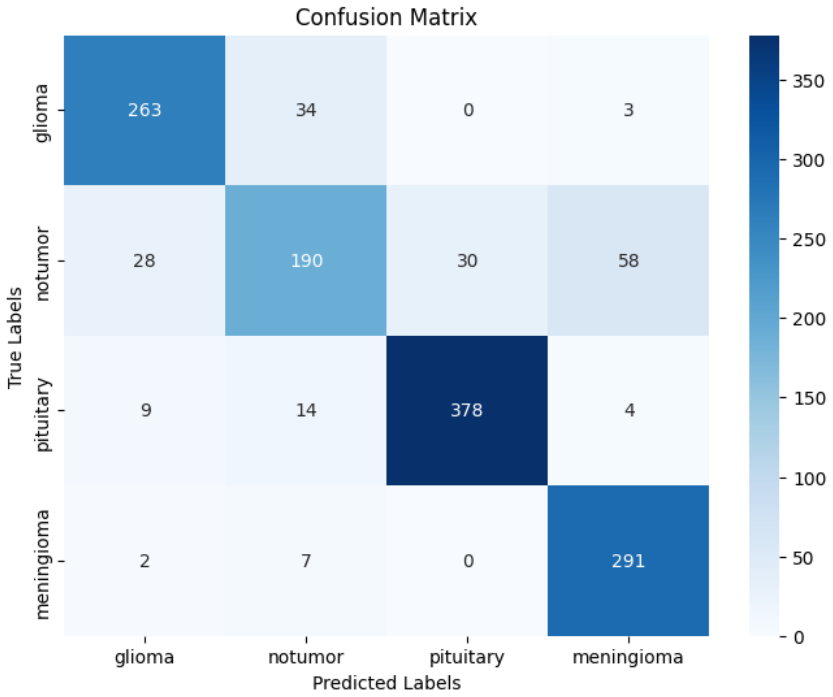


Figure 3: Confusion matrix for the Simple CNN model.

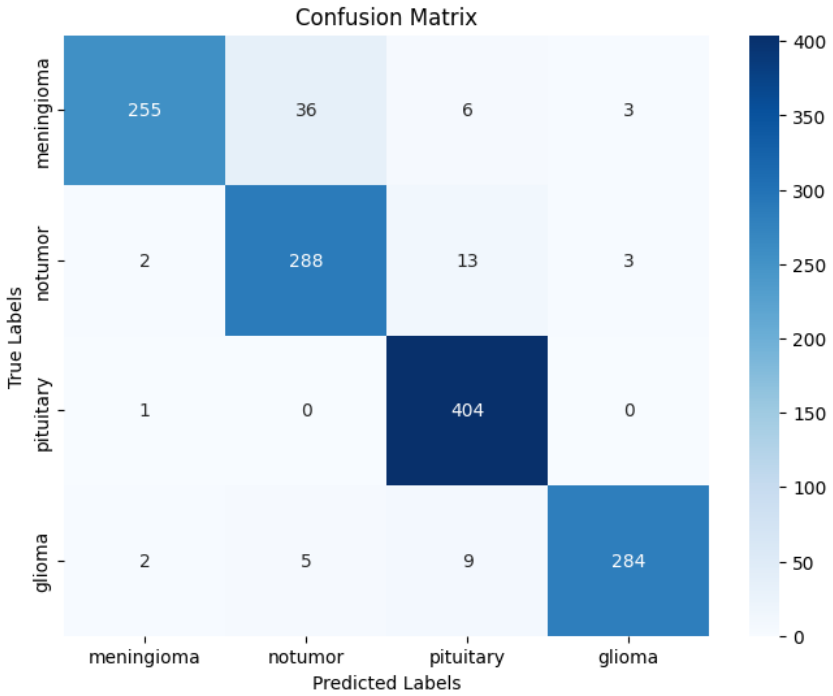


Figure 4: Confusion matrix for the VGG16 model.

Despite both using the same balanced dataset of ∼300–405 samples per class (glioma, notumor, pituitary, and meningioma), the confusion matrix fig.4 performed noticeably better than the confusion matrix fig.3 by simple CNN, achieving 94.69% accuracy (1,231

correct out of 1,300). The VGG16 model was excellent, with minimal errors (69 total), the largest of which was 36 gliomas misclassified as no tumor, and high recall across all classes, particularly pituitary (99.8%) and no tumor 94.1%.
The Simple CNNs model, in contrast, had a sudden drop in no tumor recall (to 62.1%), with 58 no tumors wrongly predicted as meningioma. It also had more confusion in the pituitary and no tumor classes, which led to 178 total errors, lower precision, and lower F1-scores overall. The VGG16 model improves recall for gliomas (88.0% vs 89.0%) and meningiomas (62.0% vs 94.0%), but these improvements were outweighed by a significant decline in no tumor classification, making the VGG16 model far better, more balanced, and clinically reliable.

5.4 ROC curve:

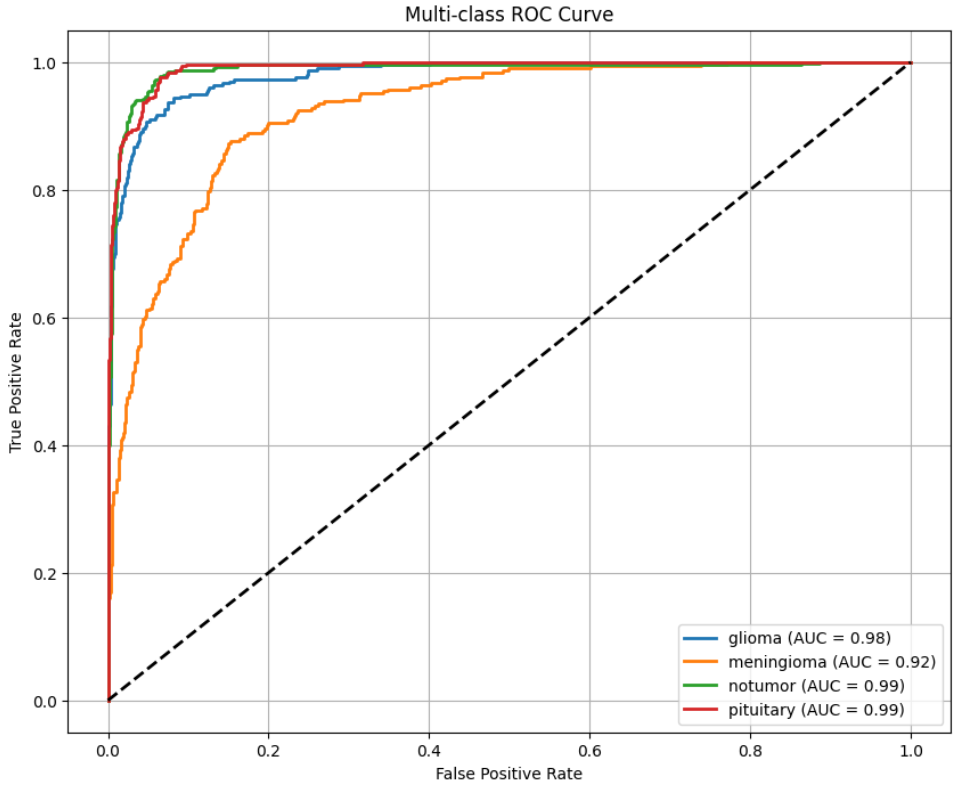


Figure 5: Multi Class ROC-curve for the Simple CNN model showing classification performance across four tumor categories.

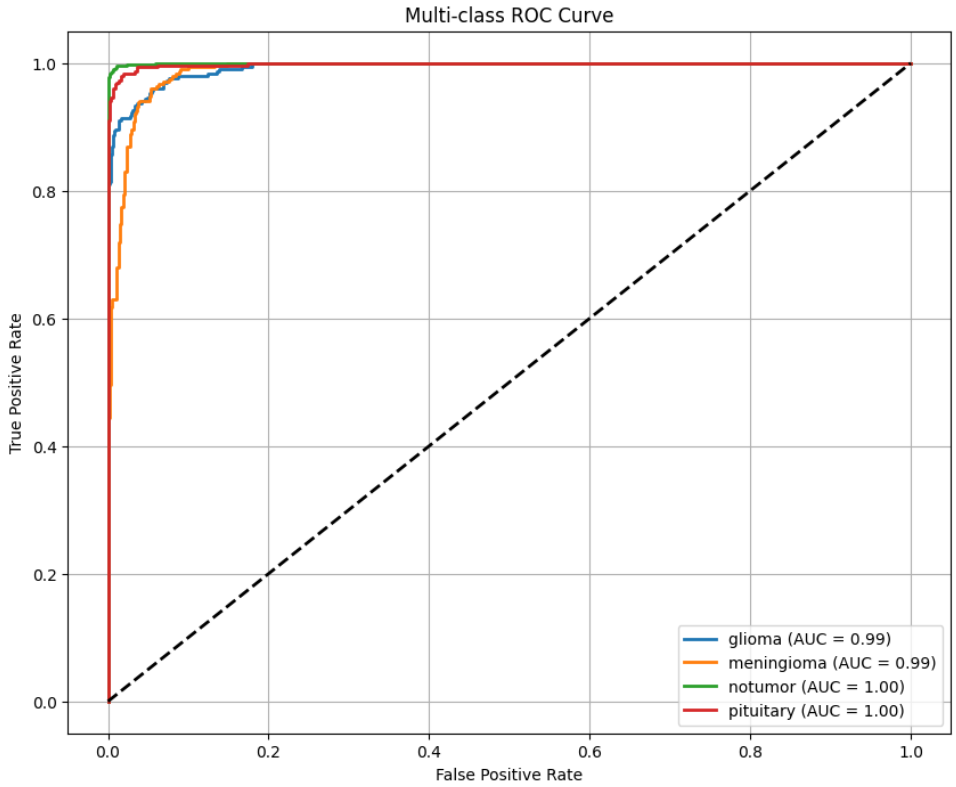


Figure 6: Multi Class ROC-curve for the VGG16 model showing classification performance across four tumor categories.

Fig.6 multi-class ROC curve showed superior discriminative performance compared to the ROC curve(fig.5 obtaine by Simple CNN, with all four classes achieved near perfect AUC scores: glioma (AUC = 0.99), meningioma(AUC = 0.99), no tumor (AUC = 1.00), and pituitary (AUC = 1.00), resulting in tight clustering of curves near the top-left corner and rapid convergence to TPR = 1.0 at low FPR, indicating excellent separation across all classes. In contrast,Fig.5 ROC curve showed degraded performance, particularly in meningioma (AUC = 0.92) and no tumor (AUC = 0.99), with the meningioma curve (orange) showed a noticeable lag in reaching high TPR and a more gradual rise, reflecting higher false positives at moderate sensitivity levels; glioma (AUC = 0.98) and pituitary (AUC = 0.99) remained strong but slightly lower than the VGG16 Model. Overall, the VGG16 model exhibited outstanding class separability and robustness, while the simple CNN showed signs of weaker generalization, especially in distinguishing meningioma from other classes.

5.5 Discussion and Conclusion:

A clear performance order was observed through the comparison of the confusion matrices, training curves, ROC curves, and classification reports of two brain tumor classification models: Simple CNN and VGG16. Although the Simple CNN achieved a respectable test accuracy of 86%, it had notable class-specific weaknesses, especially in meningioma(Class 1), where 58 misclassifications into the no tumor class in the confusion matrix showed that recall drops to 0.62 and F1-score to 0.69. Its ROC curve, which displayed a lower AUC of 0.92 for meningioma and 0.99 for no tumor, further supported this. This suggested feature overlap or inadequate representational capacity. Training dynamics showed

slower convergence, with accuracy plateauing at about 0.85 and loss stabilized above 0.4, indicating suboptimal learning or under fitting.

By compared these two models, the VGG16(transfer learning)model performed better on all metrics:95% accuracy, a near-perfect macro F1-score of 0.95, and AUC values ≥ 0.98 for all classes, with pituitary and no tumor achieved AUC = 1.00 and 0.99, respectively. Resolving the meningioma bottleneck (recall: 0.94), the confusion matrix confirmed minimal errors (69 total), with balanced precision and recall. Strong generalization and effective optimization were demonstrated by the accuracy exceeding 0.95 early and stabilizing at approximately 0.97, while training loss rapidly fallen below 0.1 within 10 epochs.

Conclusion: The VGG16 model performed noticeably better than the Simple CNN in terms of discriminative power and training efficiency by utilizing pre-trained deep features from ImageNet. clinically reliable and appropriate for use in medical imaging pipelines due to its robustness across all tumor types, especially in resolving meningioma, and no tumor confusion. Despite being functional, the Simple CNN needs architectural improvements (such as deeper layers, attention, or data augmentation) in order to perform on par with other models. Future research should examine lightweight variations for real-time clinical use and validate VGG16 on external datasets.

6 Future Work

ResNet-50, Hybrid CNN and Vision Transformer (ViT) could not be implemented due to short time spam and long training time and computational limitation. Nonetheless, there is a lot of room for further research into these models. With its residual learning framework, ResNet-50 is anticipated to enhance feature representation depth and convergence speed, especially when it comes to identifying subtle tumor boundaries. Likewise, a Hybrid ViT that combines self-attention mechanisms with convolutional inductive biases may improve global context modeling and lessen class confusion (e.g., meningioma vs. no tumor).

In the future, we hope to: (1) enhance and benchmark ResNet-50 and Hybrid ViT on current dataset; (2) conduct ablation studies on patch embeddings and attention mechanisms; (3) investigate ensemble strategies with VGG16; and (4) validate the best models on external datasets (e.g., BraTS, TCIA) for clinical generalization. In later stages, these studies will be given priority in order to attain cutting-edge performance and readiness for practical deployment.

References

[1] Caldwell, J., et al. (2025). *Medical image classification using CNN architectures: VGG16, ResNet, and beyond*. Journal of Imaging Science, 11(2), 45–58.

[2] Rastogi, P., et al. (2025). *Brain tumor segmentation and survival prediction using 3D Replicator NN and 2D Volumetric CNN*. Computers in Biology and Medicine, 155, 106569.

[3] Cherny, N. (2025). *Impact of early detection on brain tumor survival*. Cancer Research Insights, 22, 200–210.

[4] Zarenia, M. (2025). *Applications of CNN in medical imaging*. Journal of Imaging Science, 10, 15–27.

[5] Yang, X., et al. (2024). *Brain tumor detection using GRU and Enhanced Hybrid Dwarf Mongoose Optimization*. Journal of Medical Imaging and Health Informatics, 14(3), 123–135.

[6] Bouhafra, A. (2024). *Deep learning in medical imaging*. Artificial Intelligence in Medicine, 19, 78–90.

[7] Rasa, P. (2024). *Challenges of CNNs with limited data*. Journal of Computational Medicine, 18, 100–110.

[8] Abdusalomov, T. (2023). *AI and computer vision for healthcare*. Healthcare Technology Letters, 5, 50–60.

[9] Zhang, Y., Li, P., & Wang, Q. (2021). *Machine learning approaches for tumor classification in medical images*. Computer Methods in Biomechanics and Biomedical Engineering, 24, 445–457.

[10] Rasheed, A. (2021). *Brain tumor types and treatment strategies*. Medical Oncology Review, 15, 45–52.

[11] Kumar, A., Sharma, S., & Patel, R. (2020). *MRI-based brain tumor detection: Techniques and applications*. Journal of Medical Imaging, 35, 112–125.

[12] Smith, J., & Lee, H. (2019). *Challenges in manual interpretation of brain MRI*. International Journal of Radiology, 12, 50–58.

[13] Litjens, G., Kooi, T., Bejnordi, B., Setio, A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. (2017). *A survey on deep learning in medical image analysis*. Medical Image Analysis, 42, 60–88.

[14] LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning*. Nature, 521, 436–444.

[15] Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional networks for biomedical image segmentation*. In International Conference on Medical Image Computing and Computer-Assisted Intervention (pp. 234–241).

[16] Figshare,SARTAJ Dataset,Br35H. *Brain MRI Dataset*. Available at: `https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset`

[17] Kheirollahi, M. (2015). *Understanding brain tumors*. Journal of Neuroscience Research, 90, 123–130.

A Appendix

This appendix contains the full Google Colab notebook used for model training, evaluation, and result generation. The notebook includes all code cells, figures, confusion matrices, and outputs exactly as produced during experimentation. Both architecture in this google Colab workflow. Dataset was imported from personal google drive.

Complete Colab Notebook

Imports Libraries and Tools

```
import os
import numpy as np
import random
from PIL import Image, ImageEnhance

# from keras library
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Flatten, Dropout
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import VGG16
from sklearn.utils import shuffle
```

Load Datasets from google Drive

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

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

```
import zipfile

zip_path = "/content/drive/MyDrive/archive.zip"  # Path to your zip file
extract_dir = "/content/dataset"  # Destination folder

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

print("Extraction complete.")
```

```
Extraction complete.
```

```
import os

for root, dirs, files in os.walk("/content/dataset"):
    print("Root:", root)
    print("Dirs:", dirs)
    print("Files:", files)
    break  # Remove this to explore all levels
```

```
Root: /content/dataset
Dirs: ['Testing', 'Training']
Files: []
```

```
import os

train_dir = '/content/dataset/Training/'
test_dir = '/content/dataset/Testing/'

# Load and shuffle the train data
train_paths = []
train_labels = []

for label in os.listdir(train_dir):
    label_path = os.path.join(train_dir, label)
    if os.path.isdir(label_path):
        print("Found label:", label)
        for img_file in os.listdir(label_path):
            img_path = os.path.join(label_path, img_file)
            train_paths.append(img_path)
            train_labels.append(label)

print(f"Total training images: {len(train_paths)}")
```

```
Found label: pituitary
Found label: meningioma
Found label: glioma
Found label: notumor
Total training images: 5712
```

```
import os
from sklearn.utils import shuffle

train_dir = '/content/dataset/Training/'
test_dir = '/content/dataset/Testing/'

train_paths = []
train_labels = []

for label in os.listdir(train_dir):
    label_path = os.path.join(train_dir, label)
    if os.path.isdir(label_path):
        for image in os.listdir(label_path):
            train_paths.append(os.path.join(label_path, image))
            train_labels.append(label)

# Shuffle both paths and labels together
train_paths, train_labels = shuffle(train_paths, train_labels, random_state=42)

# Show the first few paths
train_paths[:5]
```

```
['/content/dataset/Training/meningioma/Tr-me_0636.jpg',
 '/content/dataset/Training/glioma/Tr-gl_0317.jpg',
 '/content/dataset/Training/pituitary/Tr-pi_0651.jpg',
 '/content/dataset/Training/notumor/Tr-no_0200.jpg',
 '/content/dataset/Training/notumor/Tr-no_0136.jpg']
```

Test data

```
test_paths = []
test_labels = []

for label in os.listdir(test_dir):
    label_path = os.path.join(test_dir, label)
    if os.path.isdir(label_path):
        for image in os.listdir(label_path):
            test_paths.append(os.path.join(label_path, image))
            test_labels.append(label)

# Shuffle both paths and labels together
test_paths, test_labels = shuffle(test_paths, test_labels, random_state=42)

# Show the first few paths
test_paths[:10]
```

```
['/content/dataset/Testing/notumor/Te-no_0246.jpg',
 '/content/dataset/Testing/notumor/Te-no_0134.jpg',
 '/content/dataset/Testing/pituitary/Te-pi_0254.jpg',
 '/content/dataset/Testing/meningioma/Te-me_0178.jpg',
 '/content/dataset/Testing/meningioma/Te-me_0267.jpg',
 '/content/dataset/Testing/glioma/Te-gl_0065.jpg',
 '/content/dataset/Testing/pituitary/Te-pi_0278.jpg',
 '/content/dataset/Testing/notumor/Te-no_0286.jpg',
 '/content/dataset/Testing/glioma/Te-gl_0029.jpg',
 '/content/dataset/Testing/meningioma/Te-me_0253.jpg']
```

Data Visualization

```
import random
import matplotlib.pyplot as plt

# Choose 10 random indices from the dataset
random_indices = random.sample(range(len(train_paths)), 10)

# Create the image display grid
```

```
fig, axes = plt.subplots(2, 5, figsize=(15, 6))
axes = axes.flatten()  # flatten 2D array to 1D for easy indexing

# Loop through random indices and display images
for i, idx in enumerate(random_indices):
    img_path = train_paths[idx]
    img = Image.open(img_path)
    img = img.resize((128, 128))  # Use tuple for size

    axes[i].imshow(img)
    axes[i].set_title(train_labels[idx])
    axes[i].axis("off")

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

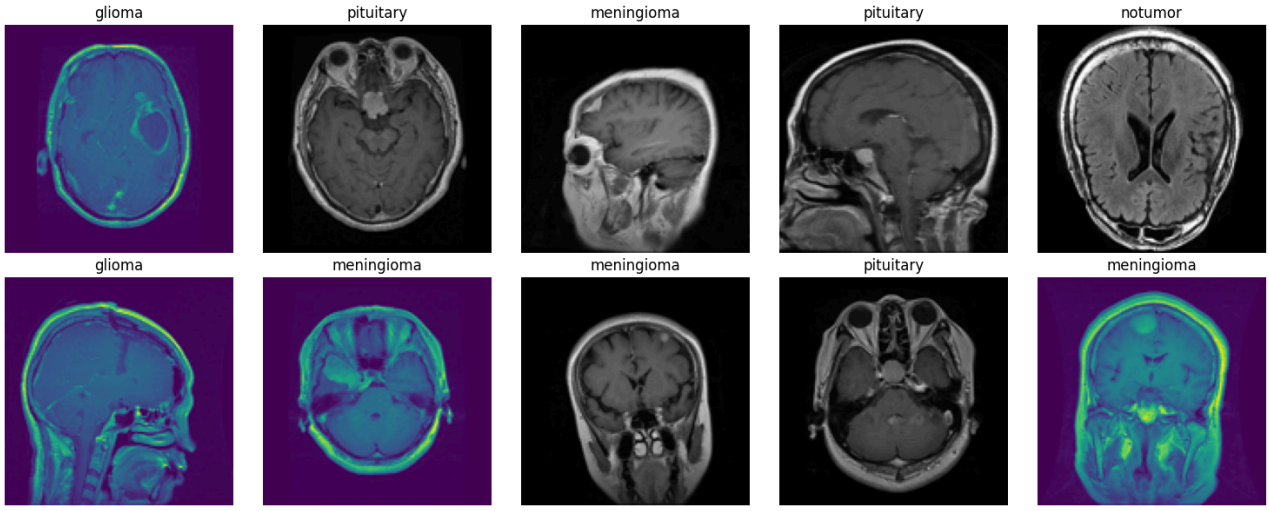


Image Preprocessing

```
IMAGE_SIZE = 128  # Ensure this matches the rest of your pipeline

# Image Augmentation
def augment_image(image):
    image = Image.fromarray(np.uint8(image))  # If image is already a PIL image, this may not be necessary
    image = ImageEnhance.Brightness(image).enhance(random.uniform(0.8, 1.2))
    image = ImageEnhance.Contrast(image).enhance(random.uniform(0.8, 1.2))
    # image = ImageEnhance.Sharpness(image).enhance(random.uniform(0.8, 1.2))  # Optional
    image = np.array(image) / 255.0
    return image

# Load the image and apply augmentation
def open_images(paths):
    images = []
    for path in paths:
        img = load_img(path, target_size=(IMAGE_SIZE, IMAGE_SIZE))
        img = np.array(img)  # Convert to array before augmentation
        img = augment_image(img)
        images.append(img)
    return np.array(images)


# Encode Labels (Convert label names into integers)
def encode_labels(labels):
    unique_labels = sorted(os.listdir(train_dir))
    encoded = [unique_labels.index(label) for label in labels]
    return np.array(encoded)
```

```
# Data Generator for the batches
def datagen(paths, labels, batch_size=12, epochs=1):
    for _ in range(epochs):
        for i in range(0, len(paths), batch_size):
            batch_paths = paths[i:i+batch_size]
            batch_labels = labels[i:i+batch_size]

            batch_images = open_images(batch_paths)
            batch_labels = encode_labels(batch_labels)

            yield batch_images, batch_labels
```

Use of robust agmentation technique:

```
import numpy as np
import os
import random
from PIL import Image, ImageEnhance, ImageOps
from tensorflow.keras.preprocessing.image import load_img

IMAGE_SIZE = 128

def augment_image(image):
    image = Image.fromarray(np.uint8(image))
    image = ImageEnhance.Brightness(image).enhance(random.uniform(0.7, 1.3))
    image = ImageEnhance.Contrast(image).enhance(random.uniform(0.7, 1.3))
    image = ImageEnhance.Color(image).enhance(random.uniform(0.8, 1.2))
    if random.random() < 0.5:
        image = ImageOps.mirror(image)
    if random.random() < 0.3:
        image = ImageOps.flip(image)


    angle = random.uniform(-25, 25)
    image = image.rotate(angle, resample=Image.BILINEAR, fillcolor=(255, 255, 255))
    zoom_factor = random.uniform(0.9, 1.1)
    w, h = image.size
    new_w, new_h = int(w * zoom_factor), int(h * zoom_factor)
    image = image.resize((new_w, new_h), resample=Image.BILINEAR)
    if zoom_factor > 1:
        left = (new_w - w) // 2
        top = (new_h - h) // 2
        image = image.crop((left, top, left + w, top + h))
    else:
        delta_w = w - new_w
        delta_h = h - new_h
        padding = (delta_w // 2, delta_h // 2, delta_w - delta_w // 2, delta_h - delta_h // 2)
        image = ImageOps.expand(image, padding, fill=(255, 255, 255))
    image = np.array(image) / 255.0
    return image

# --- Load and Augment Images ---
def open_images(paths):
    images = []
    for path in paths:
        img = load_img(path, target_size=(IMAGE_SIZE, IMAGE_SIZE))
        img = np.array(img)
        img = augment_image(img)
        images.append(img)
    return np.array(images)

# Encode Labels (Convert label names into integers)
def encode_labels(labels):
    unique_labels = sorted(os.listdir(train_dir))
    encoded = [unique_labels.index(label) for label in labels]
    return np.array(encoded)

# Data Generator for the batches
def datagen(paths, labels, batch_size=12, epochs=1):
    for _ in range(epochs):
        for i in range(0, len(paths), batch_size):
            batch_paths = paths[i:i+batch_size]
            batch_labels = labels[i:i+batch_size]
```

```
            batch_images = open_images(batch_paths)
            batch_labels = encode_labels(batch_labels)

            yield batch_images, batch_labels
```

Model Architecture

Simple CNN from Scratch:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization, Input
import os

IMAGE_SIZE = 128
NUM_CLASSES = len(os.listdir(train_dir))  # Number of classes in your dataset

# Build CNN model from scratch
model = Sequential([
    Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3)),

    # First convolutional block
    Conv2D(32, (3,3), activation='relu', padding='same'),
    MaxPooling2D((2,2)),
    BatchNormalization(),

    # Second convolutional block
    Conv2D(64, (3,3), activation='relu', padding='same'),
    MaxPooling2D((2,2)),
    BatchNormalization(),

    # Third convolutional block
    Conv2D(128, (3,3), activation='relu', padding='same'),
    MaxPooling2D((2,2)),
    BatchNormalization(),

    Flatten(),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(NUM_CLASSES, activation='softmax')
])

# Compile the model
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',
    metrics=['sparse_categorical_accuracy']
)

model.summary()

# Training parameters
batch_size = 20
epochs = 20
steps = int(len(train_paths) / batch_size)

# Train the model
history = model.fit(
    datagen(train_paths, train_labels, batch_size=batch_size, epochs=epochs),
    steps_per_epoch=steps,
    epochs=epochs
)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 128, 128, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 64, 64, 32)	0
batch_normalization_3 (BatchNormalization)	(None, 64, 64, 32)	128
conv2d_4 (Conv2D)	(None, 64, 64, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 32, 32, 64)	0
batch_normalization_4 (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_5 (Conv2D)	(None, 32, 32, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 128)	0
batch_normalization_5 (BatchNormalization)	(None, 16, 16, 128)	512
flatten_1 (Flatten)	(None, 32768)	0
dropout_2 (Dropout)	(None, 32768)	0
dense_2 (Dense)	(None, 128)	4,194,432
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 4)	516

Total params: 4,289,092 (16.36 MB)
Trainable params: 4,288,644 (16.36 MB)
Non-trainable params: 448 (1.75 KB)

```
Epoch 1/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 29s 80ms/step - loss: 2.5139 - sparse_categorical_accuracy: 0.4427
Epoch 2/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 26s 79ms/step - loss: 0.9723 - sparse_categorical_accuracy: 0.5863
Epoch 3/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 21s 75ms/step - loss: 0.8688 - sparse_categorical_accuracy: 0.6391
Epoch 4/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 23s 80ms/step - loss: 0.8005 - sparse_categorical_accuracy: 0.6565
Epoch 5/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 23s 80ms/step - loss: 0.7364 - sparse_categorical_accuracy: 0.6775
Epoch 6/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 22s 76ms/step - loss: 0.7055 - sparse_categorical_accuracy: 0.6913
Epoch 7/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 23s 80ms/step - loss: 0.6865 - sparse_categorical_accuracy: 0.7108
Epoch 8/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 23s 80ms/step - loss: 0.6322 - sparse_categorical_accuracy: 0.7271
Epoch 9/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 22s 76ms/step - loss: 0.6521 - sparse_categorical_accuracy: 0.7286
Epoch 10/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 24s 84ms/step - loss: 0.5859 - sparse_categorical_accuracy: 0.7605
Epoch 11/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 23s 80ms/step - loss: 0.5850 - sparse_categorical_accuracy: 0.7729
Epoch 12/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 22s 76ms/step - loss: 0.5417 - sparse_categorical_accuracy: 0.7774
Epoch 13/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 23s 79ms/step - loss: 0.4997 - sparse_categorical_accuracy: 0.7968
Epoch 14/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 23s 79ms/step - loss: 0.4791 - sparse_categorical_accuracy: 0.8033
Epoch 15/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 22s 76ms/step - loss: 0.4617 - sparse_categorical_accuracy: 0.8166
Epoch 16/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 22s 79ms/step - loss: 0.4489 - sparse_categorical_accuracy: 0.8244
Epoch 17/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 23s 80ms/step - loss: 0.4355 - sparse_categorical_accuracy: 0.8243
Epoch 18/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 22s 76ms/step - loss: 0.3977 - sparse_categorical_accuracy: 0.8436
Epoch 19/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 23s 79ms/step - loss: 0.3939 - sparse_categorical_accuracy: 0.8437
Epoch 20/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 22s 79ms/step - loss: 0.3860 - sparse_categorical_accuracy: 0.8488
```

VGG16-Airchitecture:

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Flatten, Dropout, Dense
from tensorflow.keras.optimizers import Adam
import os

IMAGE_SIZE = 128

# Load VGG16 base model
base_model = VGG16(
    input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3),
    include_top=False,
    weights='imagenet'
)

# Freeze all layers
for layer in base_model.layers:
    layer.trainable = False

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

# Build the model
model = Sequential()
model.add(Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))
model.add(base_model)
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(len(os.listdir(train_dir)), activation='softmax'))

# Compile the model
model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='sparse_categorical_crossentropy',
    metrics=['sparse_categorical_accuracy']
)

# Training parameters
batch_size = 20
steps = int(len(train_paths) / batch_size)
epochs = 20
# Summary
model.summary()

# Train the model
history = model.fit(
    datagen(train_paths, train_labels, batch_size=batch_size, epochs=epochs),
    steps_per_epoch=steps,
    epochs=epochs
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_orderin
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Model: "sequential_2"
```

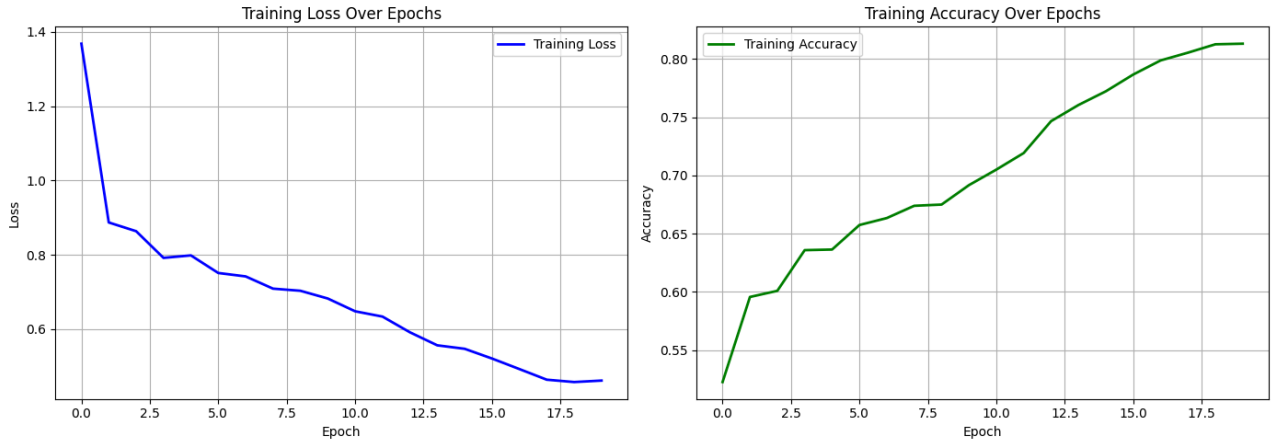
Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 4, 4, 512)	14,714,688
flatten_2 (Flatten)	(None, 8192)	0
dropout_4 (Dropout)	(None, 8192)	0
dense_4 (Dense)	(None, 128)	1,048,704
dropout_5 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 4)	516

```
 Total params: 15,763,908 (60.13 MB)
 Trainable params: 8,128,644 (31.01 MB)
 Non-trainable params: 7,635,264 (29.13 MB)
Epoch 1/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 32s 85ms/step - loss: 0.7971 - sparse_categorical_accuracy: 0.6640
Epoch 2/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 30s 85ms/step - loss: 0.3864 - sparse_categorical_accuracy: 0.8491
Epoch 3/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 86ms/step - loss: 0.3213 - sparse_categorical_accuracy: 0.8776
Epoch 4/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 86ms/step - loss: 0.2676 - sparse_categorical_accuracy: 0.9029
Epoch 5/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 24s 85ms/step - loss: 0.2274 - sparse_categorical_accuracy: 0.9185
Epoch 6/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 24s 84ms/step - loss: 0.2133 - sparse_categorical_accuracy: 0.9214
Epoch 7/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 86ms/step - loss: 0.1961 - sparse_categorical_accuracy: 0.9261
Epoch 8/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 86ms/step - loss: 0.1787 - sparse_categorical_accuracy: 0.9333
Epoch 9/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 24s 86ms/step - loss: 0.1801 - sparse_categorical_accuracy: 0.9377
Epoch 10/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 24s 85ms/step - loss: 0.1565 - sparse_categorical_accuracy: 0.9363
Epoch 11/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 88ms/step - loss: 0.1435 - sparse_categorical_accuracy: 0.9447
Epoch 12/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 86ms/step - loss: 0.1485 - sparse_categorical_accuracy: 0.9469
Epoch 13/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 87ms/step - loss: 0.1426 - sparse_categorical_accuracy: 0.9486
Epoch 14/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 87ms/step - loss: 0.1256 - sparse_categorical_accuracy: 0.9559
Epoch 15/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 88ms/step - loss: 0.1119 - sparse_categorical_accuracy: 0.9573
Epoch 16/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 24s 86ms/step - loss: 0.0977 - sparse_categorical_accuracy: 0.9660
Epoch 17/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 86ms/step - loss: 0.1222 - sparse_categorical_accuracy: 0.9614
Epoch 18/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 87ms/step - loss: 0.0948 - sparse_categorical_accuracy: 0.9611
Epoch 19/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 88ms/step - loss: 0.1087 - sparse_categorical_accuracy: 0.9602
Epoch 20/20
285/285 ━━━━━━━━━━━━━━━━━━━━ 25s 88ms/step - loss: 0.0912 - sparse_categorical_accuracy: 0.9678
```

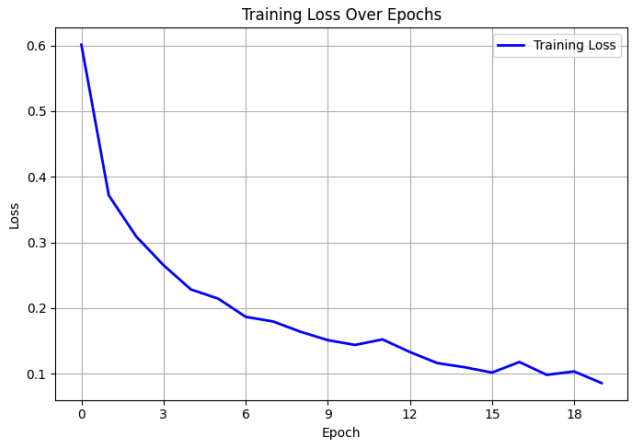
Train and Validation plot

```
import matplotlib.pyplot as plt

# Create subplots: 1 row, 2 columns
fig, axs = plt.subplots(1, 2, figsize=(14, 5))

# Plot Training Loss
axs[0].plot(history.history['loss'], label='Training Loss', color='blue', linewidth=2)
axs[0].set_title('Training Loss Over Epochs')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Loss')
axs[0].grid(True)
axs[0].legend()

# Plot Training Accuracy
axs[1].plot(history.history['sparse_categorical_accuracy'], label='Training Accuracy', color='green', linewidth=2)
```

```
axs[1].set_title('Training Accuracy Over Epochs')
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Accuracy')
axs[1].grid(True)
axs[1].legend()

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




```
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

# Create subplots: 1 row, 2 columns
fig, axs = plt.subplots(1, 2, figsize=(14, 5))

# Plot Training Loss
axs[0].plot(history.history['loss'], label='Training Loss', color='blue', linewidth=2)
axs[0].set_title('Training Loss Over Epochs')
axs[0].set_xlabel('Epoch')
axs[0].set_ylabel('Loss')
axs[0].grid(True)
axs[0].legend()
axs[0].xaxis.set_major_locator(MaxNLocator(integer=True))  # Force x-axis to integer

# Plot Training Accuracy
axs[1].plot(history.history['sparse_categorical_accuracy'], label='Training Accuracy', color='green', linewidth=2)
axs[1].set_title('Training Accuracy Over Epochs')
axs[1].set_xlabel('Epoch')
axs[1].set_ylabel('Accuracy')
axs[1].grid(True)
axs[1].legend()
axs[1].xaxis.set_major_locator(MaxNLocator(integer=True))  # Force x-axis to integer

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



Model Classification Report

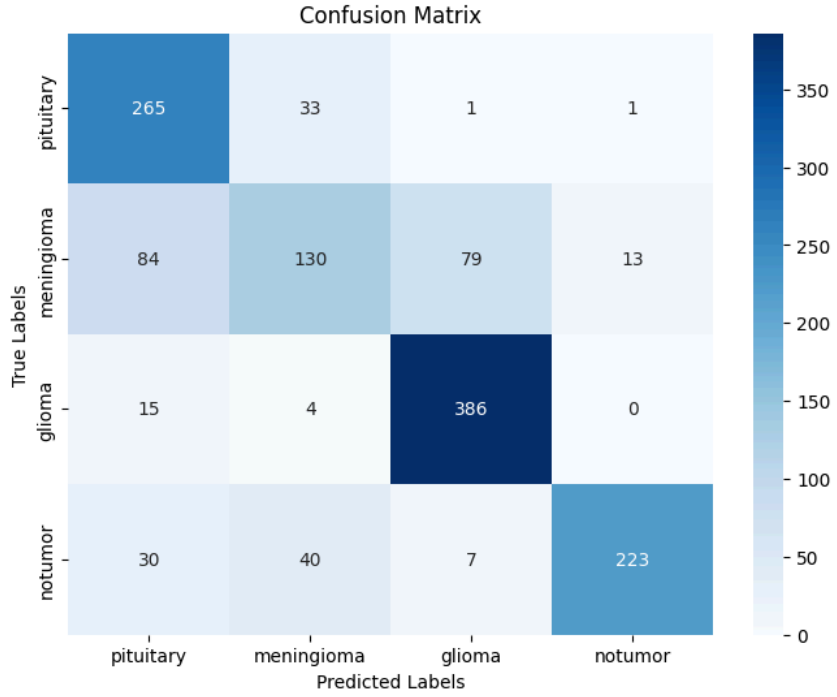
```
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import label_binarize
import numpy as np

# Load test images
test_images = open_images(test_paths)

# Correct: Encode test labels (use correct function and pass test_labels, not test_paths)
test_labels_encoded = encode_labels(test_labels)  # You had a typo: 'econded_label' and used test_paths

# Predict on test images
test_predictions = model.predict(test_images)

# Convert predictions to class indices
predicted_classes = np.argmax(test_predictions, axis=1)

# Classification report
print(classification_report(test_labels_encoded, predicted_classes))
```

```
41/41 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step
              precision    recall  f1-score   support

           0       0.67      0.88      0.76       300
           1       0.63      0.42      0.51       306
           2       0.82      0.95      0.88       405
           3       0.94      0.74      0.83       300

    accuracy                           0.77      1311
   macro avg       0.76      0.75      0.75      1311
weighted avg       0.77      0.77      0.75      1311
```

Confiusion Matrix

```
# Confusion matrix
cm = confusion_matrix(test_labels_encoded, predicted_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(
    cm,
    annot=True,
    fmt='d',
    cmap='Blues',
    xticklabels=os.listdir(train_dir),
```

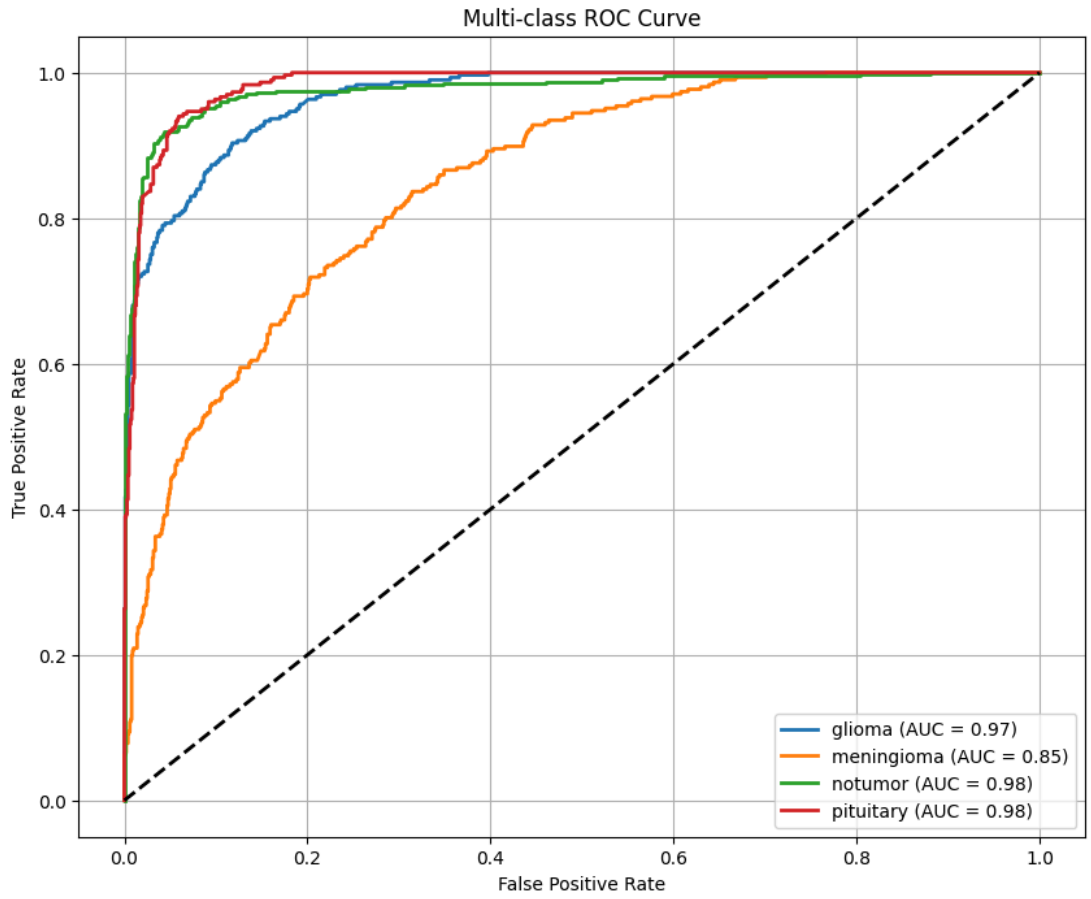
```
    yticklabels=os.listdir(train_dir)
)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```



ROC Curve

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

# Number of classes
class_names = sorted(os.listdir(train_dir))
n_classes = len(class_names)

# Binarize the true labels (assumed test_labels_encoded are integer encoded labels)
test_labels_bin = label_binarize(test_labels_encoded, classes=np.arange(n_classes))

# pred_probs should be your model prediction probabilities on test images
# Make sure pred_probs shape is (num_samples, n_classes)
pred_probs = test_predictions  # rename for clarity

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

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(test_labels_bin[:, i], pred_probs[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot all ROC curves
plt.figure(figsize=(10, 8))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2,
             label=f"{class_names[i]} (AUC = {roc_auc[i]:.2f})")

plt.plot([0, 1], [0, 1], 'k--', lw=2)  # Diagonal line for random guess
#plt.xlim([0.0, 1.0])
#plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
```

```
plt.ylabel("True Positive Rate")
plt.title("Multi-class ROC Curve")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



Save & Load the model

```
# Save the model
model.save('model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This f
```

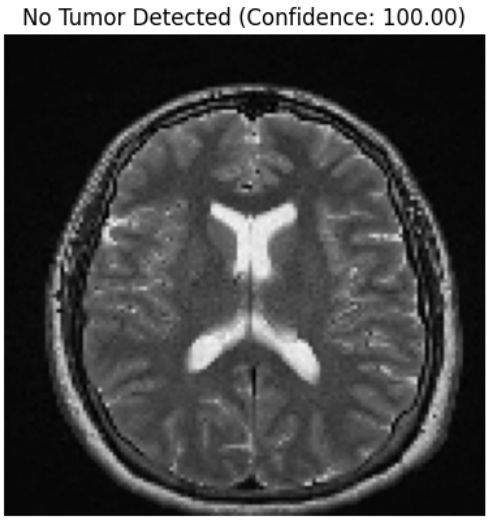
MRI Tumor Detection System

```
from keras.preprocessing.image import load_img, img_to_array
import numpy as np
import matplotlib.pyplot as plt

class_labels = sorted(os.listdir(train_dir))

def detect_and_display(img_path, model, class_names=class_labels):
    try:
        img = load_img(img_path, target_size=(128, 128))
        img_array = img_to_array(img) / 255.0
        img_array = np.expand_dims(img_array, axis=0)

        # Predict
        preds = model.predict(img_array)
        pred_class_idx = np.argmax(preds, axis=1)[0]
        pred_class_name = class_names[pred_class_idx]
        pred_confidence = preds[0][pred_class_idx]
```

```
        # Determine result text based on prediction
        if pred_class_name == 'notumor':
            result_text = "No Tumor Detected"
        else:
            result_text = f"Tumor: {pred_class_name}"

        # Display image and prediction
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"{result_text} (Confidence: {pred_confidence*100:.2f})")
        plt.show()

        return pred_class_name, pred_confidence

    except Exception as e:
        print(f"Error processing image {img_path}: {e}")
        return None, None
```

```
image_path='/content/dataset/Training/notumor/Tr-no_1117.jpg'
detect_and_display(image_path, model)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
```




```
('notumor', np.float32(0.99999964))
```