Brain Tumor Classification Based on Deep Convolutional Neural Network using MRI images

Prepared By:

Momen Albawarid 21110399

Supervised By:

Eng. Ayah Karajeh

Project Submitted in partial fulfillment for the degree of Bachelor of Science

……

Fall-2025

Table of Contents

1. Application Description 3
1.1 Brief overview 3
1.2 Detailed diagrams with all components labeled. 4
1.3 Stages of Implementation. 6
Phase 1: Data Preparation and Baseline Model Training 6
Phase 2: Explainable AI (XAI) Development 7
Phase 3: Graphical User Interface (GUI) Development 7
Phase 4: System Integration and Parameter Tuning 7
2. Test Plan 8
2.1 Testing Procedures and Validation Strategy 8
2.1.1 Software Functional Testing 8
2.2 Analysis of Results and Optimization 10
2.2.1 Optimization Strategy: Fine-Tuning and Class Weighting 10
2.2.2 Quantitative Performance Metrics 11
2.2.3 Resource Efficiency Analysis 11
3. Reflection 12
3.1 Evaluation of Project Performance 12
3.2 Professional Development: My Journey 13
3.3 Future Work 13

Table of Figures

Figure 1 - End-to-End System Workflow. 4
Figure 2 - Software Component Structure. 4
Figure 3 - Application Execution Sequence. 5
Figure 4 - This diagram explains how the explainable AI (XAI) generates heatmap. 6
Figure 5 Automated Unit Test Execution. 8

Table of Tables

Table 1 - Manual System Test Log 9
Table 2 - Evolutionary Analysis of Model Optimization and Performance. 10
Table 3 - Classification Report for the Final Optimized MobileNetV2 Model. 11

1. Application Description

1.1 Brief overview

The developed application is a Python desktop tool for brain tumor classification, representing the practical evolution of the findings from my Computer Research Project (CRP). While the CRP focused on evaluating deep learning architectures identifying MobileNetV2 as the optimal balance between high accuracy (~93%) and computational efficiency, this application operationalizes those findings into a deployable clinical support tool.

The system addresses the "black box" problem in medical AI by integrating Explainable AI (XAI). It allows users to upload an MRI scan and receive a dual output: (1) a classification prediction (Glioma, Meningioma, Pituitary, or No Tumor) and (2) a Score-CAM saliency map. Unlike traditional gradient-based methods, Score-CAM was implemented to provide cleaner, noise-free visualizations by weighting activation maps based on their contribution to the target class score.

The software architecture is engineered into two decoupled layers to ensure performance and responsiveness:

- **User Interface Layer (CustomTkinter GUI):**
  Designed with a modern "radiology-style" dark theme, this layer manages user interactions, image browsing, and results visualization. Crucially, the application utilizes multi-threading (threading module) to offload heavy inference tasks to a background worker. This prevents the interface from freezing (the "Not Responding" error) and provides real-time feedback via a custom loading modal during analysis.

- **Inference & XAI Layer (TensorFlow Backend):** This engine powers the core logic defined in xai.py. It loads the fine-tuned MobileNetV2 model and executes a precise preprocessing pipeline (including bilinear resizing and contrast adjustment with a factor of 2.0) to match the training conditions. It then computes the Score-CAM overlay by analyzing the top 32 feature channels, ensuring the generated heatmap accurately localizes the tumor region without obscuring anatomical details.

1.2 Detailed diagrams with all components labeled.

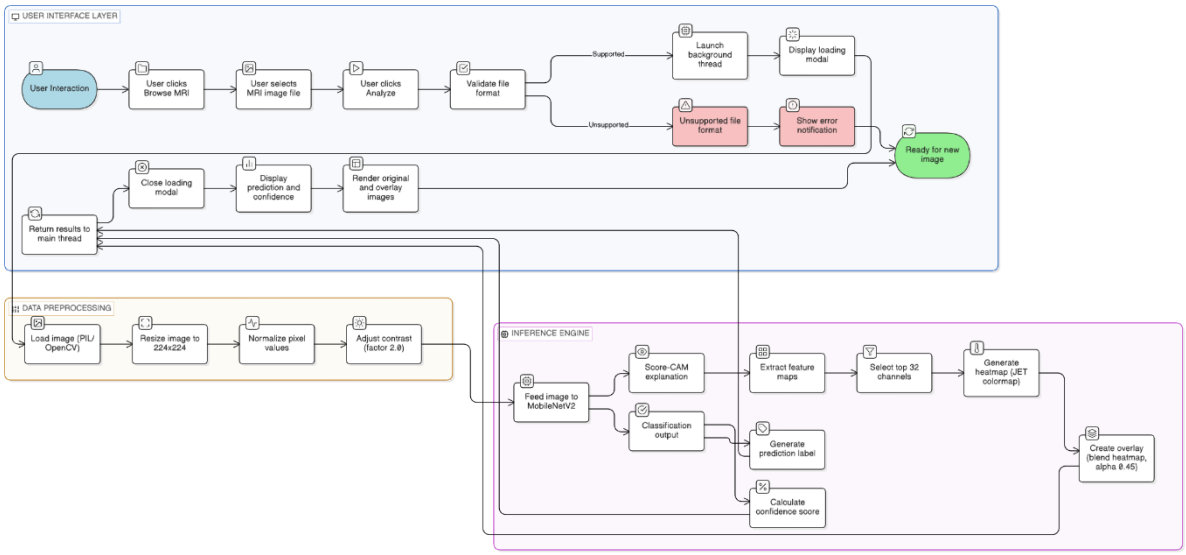

Figure 1 - End-to-End System Workflow. This diagram illustrates the complete journey of an MRI image from user upload to final result. It shows how the system is organized into three distinct stages: the User Interface (which manages user interaction and application responsiveness), Data Preprocessing (which prepares the image for analysis), and the Inference Engine (which generates both the diagnosis and the visual explanation).

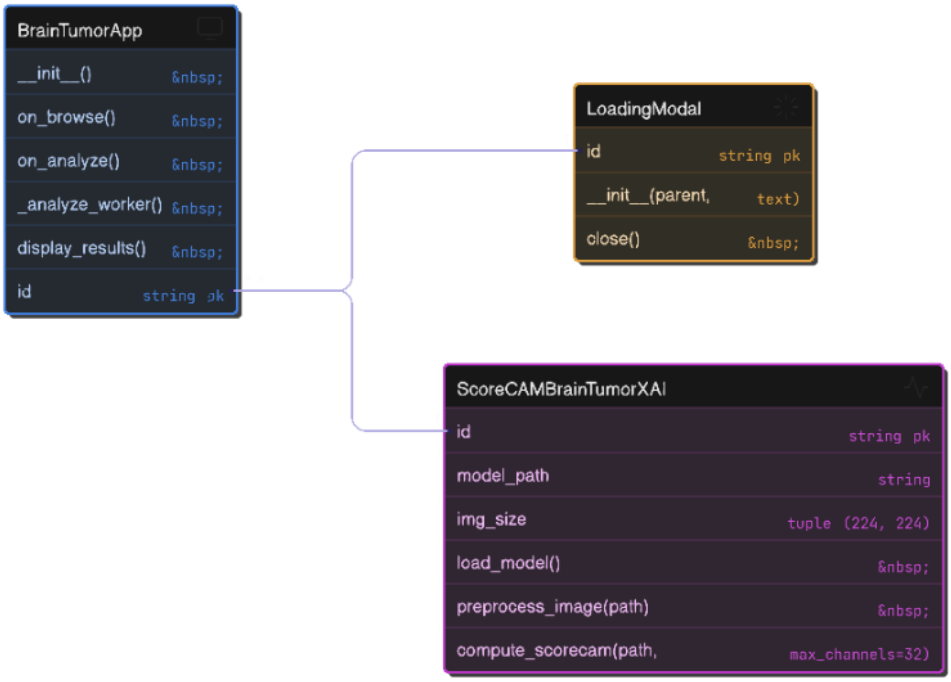

Figure 2 - Software Component Structure. This diagram illustrates the modular design of the application code. The BrainTumorApp acts as the central controller, managing the user interface and interactions. It delegates tasks to two specialized components: the LoadingModal (which provides visual feedback to the user during processing) and the ScoreCAMBrainTumorXAI (which contains all the logic for image preprocessing and AI analysis), ensuring clean separation between the interface and the intelligent engine

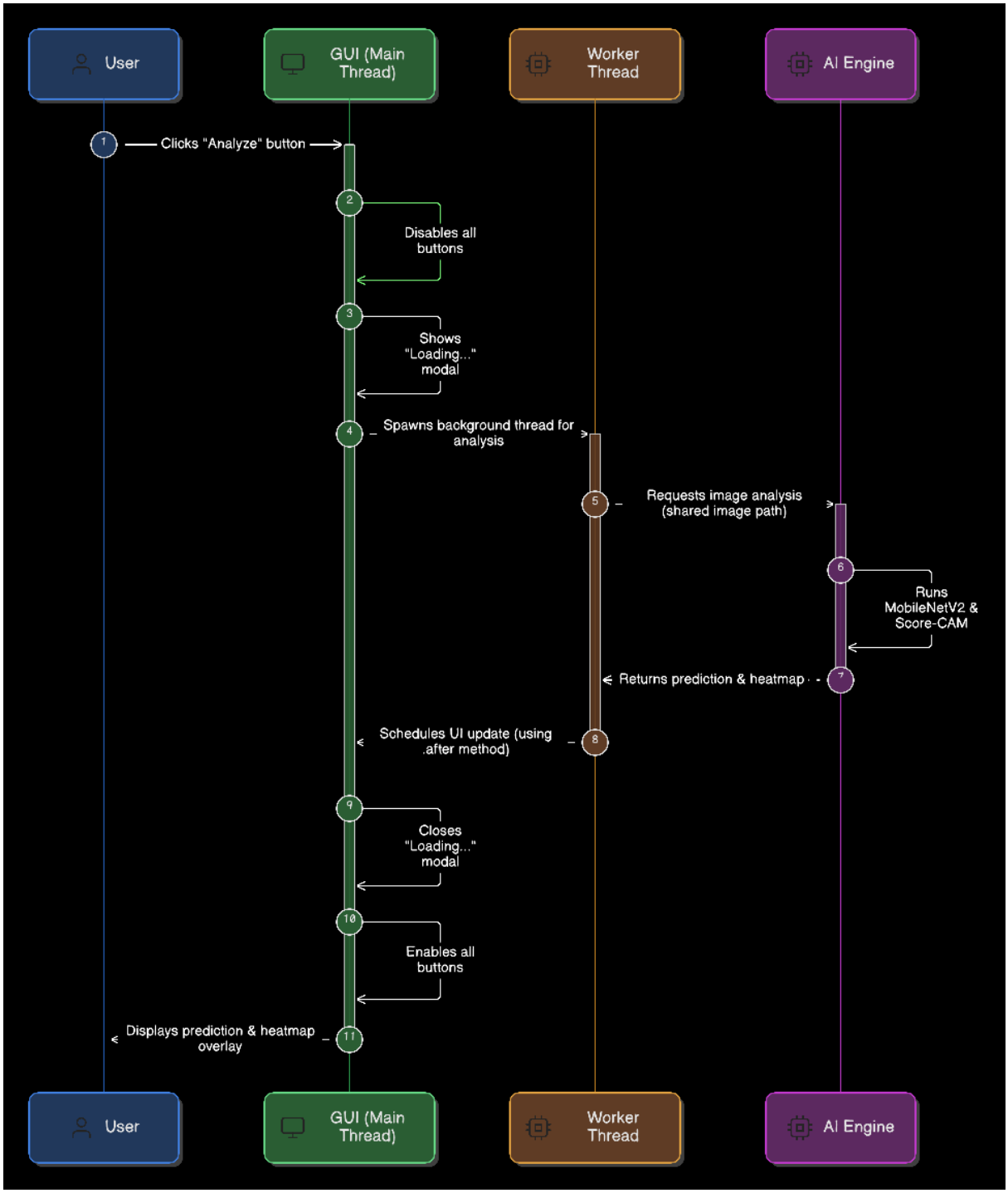

Figure 3 - Application Execution Sequence. This diagram illustrates the timeline of events when a user clicks "Analyze." To ensure the application does not freeze, the GUI (Main Thread) immediately launches a separate Worker Thread to handle the heavy comp.

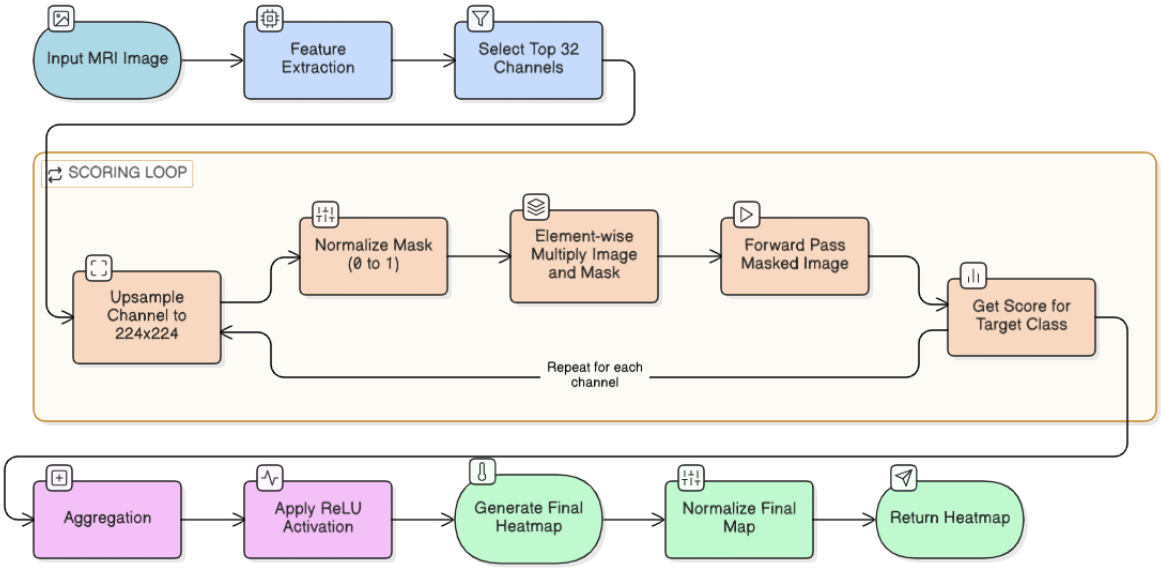

Figure 4 - This diagram explains how the explainable AI (XAI) generates heatmap. The process starts by extracting the 32 most relevant feature patterns from the AI model. It then enters a loop where it uses each pattern as a "mask" over the original image and re-tests the model to see how important that specific area is for the final diagnosis. Finally, it combines these important scores into a single heatmap that highlights the critical tumor regions.

1.3 Stages of Implementation.

The project implementation was executed in four distinct phases, moving from data research to model optimization, and finally to software application development.

Phase 1: Data Preparation and Baseline Model Training

The initial phase focused on securing a robust dataset and establishing a baseline for performance. The "Brain MRI ND-5" dataset was selected for its diversity, containing 17,891 images across axial, coronal, and sagittal planes.

- **Preprocessing Experiments:** Initial training attempts using standard data augmentation (rotation/flipping) on the training set resulted in generalization issues when tested against non-augmented validation data.
- **Enhancement Strategy:** The strategy was shifted to use Image Enhancement (histogram equalization) without geometric augmentation. This approach yielded stable learning curves and higher validation accuracy.
- **Model Selection:** MobileNetV2 was selected as the backbone architecture due to its balance of high accuracy (~90%) and low memory footprint (~200MB), making it ideal for the target deployment environment.

Phase 2: Explainable AI (XAI) Development

To address the "black box" nature of deep learning in healthcare, an explainability module was developed.

- **Score-CAM Implementation:** Instead of the traditional Grad-CAM, the project implemented Score-CAM. This gradient-free method weights activation maps by their contribution to the target class score, resulting in cleaner heatmaps with less noise.
- **Overlay Optimization:** The visualization pipeline was tuned to overlay the heatmap on the original MRI using a JET colormap with a transparency alpha of 0.45, ensuring the tumor region is highlighted without obscuring the underlying anatomical structure.

Phase 3: Graphical User Interface (GUI) Development

The application layer was built using Python and `CustomTkinter` to provide a modern, user-friendly experience for non-technical medical staff.

- **Asynchronous Architecture:** To prevent the application from freezing during heavy inference tasks, the analysis engine was decoupled from the UI using the Python `threading` module.
- **User Feedback:** A custom `LoadingModal` was implemented to provide visual feedback during the inference process, improving the perceived performance of the application.
- **Visualization:** The UI was designed to render the original MRI and the XAI overlay side-by-side, allowing for immediate visual verification of the model's prediction.

Phase 4: System Integration and Parameter Tuning

The final phase involved integrating the inference engine with the GUI and defining runtime parameters for optimal performance.

- **Runtime Configuration:** The engine was configured with specific hyperparameters (`max_channels=32` for Score-CAM and `contrast_factor=2.0` for preprocessing) to balance speed and visual clarity.
- **Deployment:** The system was packaged as a desktop application, capable of loading the `.h5` model once at startup to minimize inference latency for subsequent checks.

2. Test Plan

2.1 Testing Procedures and Validation Strategy

To ensure the reliability of the system, a comprehensive testing strategy was adopted, divided into two distinct layers: Software Functional Testing and Model Performance Validation. This dual-layer approach ensures that the application is not only robust and responsive for the end-user but also powered by a highly accurate and generalizable AI engine.

2.1.1 Software Functional Testing

The software testing phase focused on the stability of the Graphical User Interface (GUI) and the efficiency of the backend inference engine.

- **Automated Unit Testing:** A dedicated test suite (test_suite.py) was developed to validate the internal logic of the inference engine. This automated test verified that image preprocessing (resizing to 224 * 224), model loading, and Score-CAM heatmap generation functioned correctly without runtime errors.

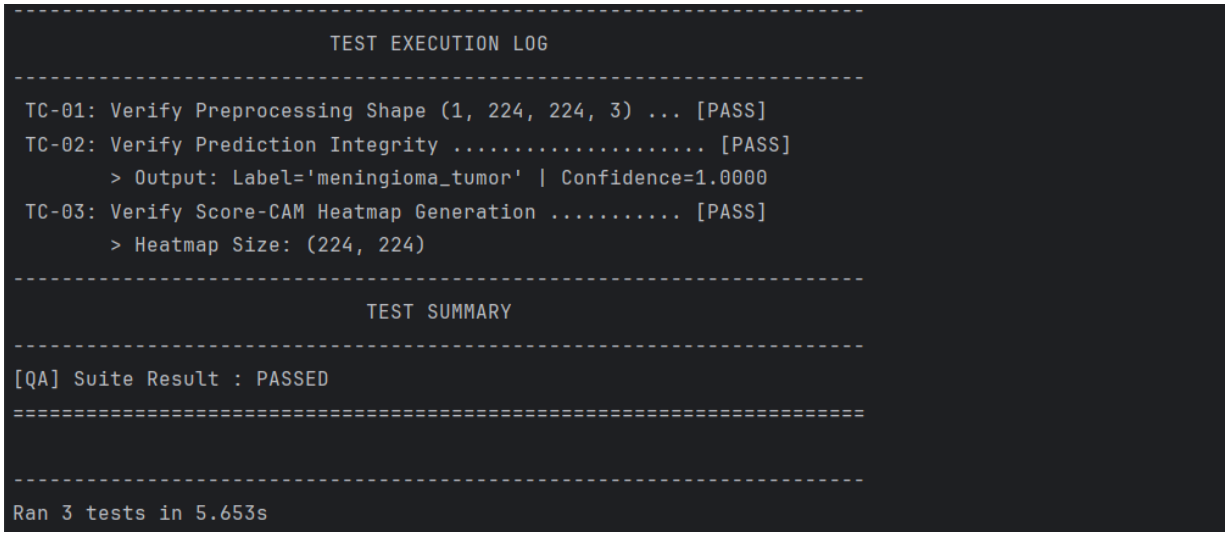

Figure 5 Automated Unit Test Execution. The test suite validates preprocessing shapes, prediction ranges, and heatmap generation logic.

- **Manual System Testing:** The desktop application underwent rigorous manual testing to verify user workflows, exception handling (unsupported file formats), and concurrency. A key focus was ensuring the application remained responsive during heavy inference tasks by utilizing background threading.

Test ID	Scenario	Action Performed	Expected Result	Status
01	**Standard Workflow**	Load a valid .jpg MRI image and click "Analyze".	Prediction and Heatmap appear side-by-side. Status updates to "Done".	**PASS**
02	**File Compatibility**	Load a valid .png MRI image.	Image loads and analyzes correctly (same as JPG).	**PASS**
03	**Invalid File Type**	Attempt to load a .txt or .pdf file.	File dialog automatically filters view to only show supported formats (.jpg, .png), preventing invalid selection.	**PASS**
04	**Concurrency**	Click "Analyze", then immediately drag the window.	Window moves smoothly; no "Not Responding" freeze.	**PASS**
05	**Empty State**	Click "Analyze" **without** selecting an image first.	The 'Analyze' button remains disabled (grayed out) until a valid image is loaded.	**PASS**
06	**Rapid Clicks**	Click "Analyze" button 5 times quickly.	Button disables after 1st click; only one analysis runs.	**PASS**
07	**Session Reset**	Analyze Image A, then immediately load Image B.	Previous results (prediction/heatmap) clear; new image loads ready for analysis.	**PASS**

Table 1 - Manual System Test Log.

2.2 Analysis of Results and Optimization

Following the functional validation of the software, the core AI engine was subjected to performance analysis. The initial baseline models (documented in the preliminary research) struggled with generalization, achieving approximately 77% accuracy. Based on this data, a targeted optimization phase was executed.

2.2.1 Optimization Strategy: Fine-Tuning and Class Weighting

To address the generalization gap, the MobileNetV2 model was retrained with an improved pipeline:

- **Optimization:** The final model utilized **Class Weighting** (`cw_v2`) to handle dataset imbalances, ensuring the model did not bias towards the majority class.
- **Outcome:** This retraining phase resulted in a significant performance leap. The final model achieved a **Test Accuracy of 93%** on the hold-out test set (3,961 images), demonstrating robust generalization capability.

Our testing process had two main phases. The results from the initial research (CRP) helped us find problems in our data strategy, which we fixed for the final application.

Phase	What We Did	What Happened (Result)	How We Fixed It
Phase 1: Initial Research (CRP)	We trained the model on one dataset (ND-5) and tested it on a totally different one (Figshare).	**Accuracy was low (77%).** The model struggled because the test images were too different from the training images.	**Lesson Learned:** The model needs to see a variety of image styles during training to learn effectively.
Phase 2: Experimentation	We tried to fix the problem by rotating and flipping the images (Augmentation).	**It didn't help much.** The validation results were unstable, and the model started memorizing the wrong details.	**Action:** We removed the complex rotations and focused on simple image enhancement (making brightness consistent).
Phase 3: Final Product	We used a standard "Train/Test Split" on a complete dataset.	**Accuracy reached 93%.** Because the training data was representative of the test data, the model performed much better.	**Final Result:** We deployed this high-accuracy model in the Desktop App.

Table 2 - Evolutionary Analysis of Model Optimization and Performance.

2.2.2 Quantitative Performance Metrics

The classification report below details the model's precision and recall for each tumor category. The high precision scores (>90% for most classes) indicate a low False Positive rate, which is critical in medical diagnostics.

Class	Precision	Recall	F1-Score
Glioma	**0.94**	**0.96**	**0.95**
Meningioma	**0.90**	**0.86**	**0.88**
No Tumor	**0.92**	**0.95**	**0.93**
Pituitary	**0.96**	**0.98**	**0.97**
Weighted Avg	**0.93**	**0.93**	**0.93**

Table 3 - Classification Report for the Final Optimized MobileNetV2 Model.

Data Source: Final Retraining Notebook Validation Results (capstone_project.ipynb)

2.2.3 Resource Efficiency Analysis

Despite the high accuracy, the model retained its efficiency profile.

- **Inference Speed:** Average inference time remained under 1.5 seconds on a standard CPU.
- **Memory Footprint:** The MobileNetV2 backbone consumed significantly less memory (~200MB) compared to heavier architectures like Xception (~1400MB), validating the architectural choice for a deployable desktop application.

3. Reflection

3.1 Evaluation of Project Performance

The main goal of this project was to prove that Artificial Intelligence can **replace human radiologists** in the initial diagnosis of brain tumors. By using simple, low-cost resources (a standard laptop), I built a desktop application that helps diagnose tumors from 2D MRI images with speed and accuracy that rivals manual human analysis.

- **Functional Success:** The application successfully meets all requirements. It allows a user to load an MRI image, and within seconds, the AI predicts the tumor type (Glioma, Meningioma, Pituitary, or No Tumor) and displays the result alongside a visual explanation.
- **Performance Metrics:**
    - **Accuracy:** The final model achieved **93% accuracy**, proving that a properly trained AI is highly reliable.
    - **Efficiency:** The system runs effectively on consumer hardware without needing expensive hospital servers.
- **Challenges (The "Grid" Issue):**
    - The hardest technical challenge was developing the Explainable AI (XAI) feature. Initially, when I tried to generate the heatmap, it looked like a "blocky grid" that didn't match the brain shape.
    - **The Problem:** The AI model sees the image in a very low resolution (a 7×7 grid of features), but the real MRI image is high resolution (224×224). When I tried to overlay them directly using standard methods like Grad-CAM, the "feature map" size didn't match the "image space," causing the heatmap to be misaligned or meaningless.
    - **The Solution:** I switched to a method called **Score-CAM**. Instead of relying on the raw grid, Score-CAM up samples the features carefully and tests them against the image. This fixed the alignment issue and produced smooth, accurate heatmaps that correctly highlighted the tumor.

3.2 Professional Development: My Journey

This project was a continuous evolution. It started as a research idea and grew into a complete software product.

- **Phase 1: Research (The CRP Phase):**
    - I started with the *Capstone Research Project (CRP)*. At this stage, I was learning the basics. I trained my first MobileNet model, but the results were only satisfactory (~77% accuracy). I realized that just running code wasn't enough; I needed to understand the *data*.
- **Phase 2: Optimization (The Notebook Phase):**
    - I went back to the drawing board with my Python notebooks. I analyzed why the first model failed and realized the training data wasn't diverse enough. I rewrote the training pipeline (as seen in capstone_project.ipynb), applied "Class Weighting," and removed bad data augmentation. This analysis pushed the accuracy from 77% to 93%.
- **Phase 3: Engineering (The Application Phase):**
    - The final step was turning that code into a real product. I learned to build a professional GUI using customtkinter and threading. I moved from writing simple scripts to building a complex system with "Object-Oriented Programming," separating the Brain (xai.py) from the Interface (app.py).

3.3 Future Work

This project proves that the future of radiology is automated. If I had the resources to continue, I would expand this into a system that fully automates the diagnostic workflow:

1. **Direct MRI Integration:** The software would connect directly to the MRI machine. As soon as the scan is done, the AI would process it instantly, removing the waiting time for a doctor.
2. **3D Volumetric View:** Instead of looking at flat 2D images, the AI would analyze the full 3D shape of the brain to calculate the exact volume of the tumor.
3. **Automated Reporting:** The final system would not just find the tumor it would write the full medical report describing the patient's case integrating a Large Language Model (LLM). This would effectively replace the manual reporting task of the radiologist, saving time, reducing human error, and ensuring every patient gets an accurate diagnosis immediately.