Brain Tumor Detection System – Full Project Report

Project Title: Brain Tumor Detection
Course:Introduction to problem solving and programming
Submitted By: DR.MONIKA VYAS
NAME.:ALOK TRIPATH ,Registration No.:25MIM10137
Institution: VIT BHOPAL
Academic Year: 2025-26

---

2. Introduction

Brain tumors are serious abnormalities within brain tissues that require early detection for effective treatment. Manual interpretation of MRI scans depends heavily on radiologists and may result in misdiagnosis due to workload or human error. This project aims to automate the detection of brain tumors from MRI images using a Convolutional Neural Network (CNN). The uploaded project folder (`Brain-Tumor-Dedection-main.zip`) includes dataset structure, preprocessing scripts, training modules, and evaluation results.

---

3. Problem Statement

Identifying brain tumors in MRI images manually is time-consuming and subject to human error. An automated deep learning-based system is needed to classify MRI images as *Tumor* or *Non-Tumor* with high accuracy.

---

4. Functional Requirements

- System should load MRI images.
- Perform preprocessing such as resizing and normalization.
- Classify MRI scans into Tumor or Non-Tumor.
- Provide accuracy and loss graphs.
- Predict on external MRI images.

---

5. Non-functional Requirements

- **Performance:** High accuracy and reliable results.
- **Efficiency:** Fast preprocessing and inference.
- **Usability:** Easy to run scripts and understand outputs.
- **Scalability:** Should support expansion to larger datasets.
- **Portability:** Runs on any machine with Python and PyTorch.

---

6. System Architecture

Components:

- Input MRI Image
- Preprocessing Layer (Normalization, Resize)

- CNN Model Layer
- Classification Layer (Tumor/Non-Tumor)
- Results & Visualization

---

7. Design Diagrams

7.1 Use Case Diagram

Actors: User / Radiologist
Use Cases: Upload MRI → System Processes → Classification Output

7.2 Workflow Diagram

MRI Upload → Preprocessing → CNN Inference → Output Tumor/Non-Tumor

7.3 Sequence Diagram

User → Preprocessor → CNN Model → Output Layer → Result Display

7.4 Class / Component Diagram

- Dataset Loader
- Image Preprocessor
- CNN Model Class
- Evaluator
- Prediction Engine

---

8. Design Decisions & Rationale

- CNN chosen due to excellent image classification performance.
- Grayscale preprocessing reduces model complexity.
- Dynamic fully connected layer sizing improves compatibility.
- Early stopping prevents overfitting.
- Data augmentation improves model robustness.

---

9. Implementation Details

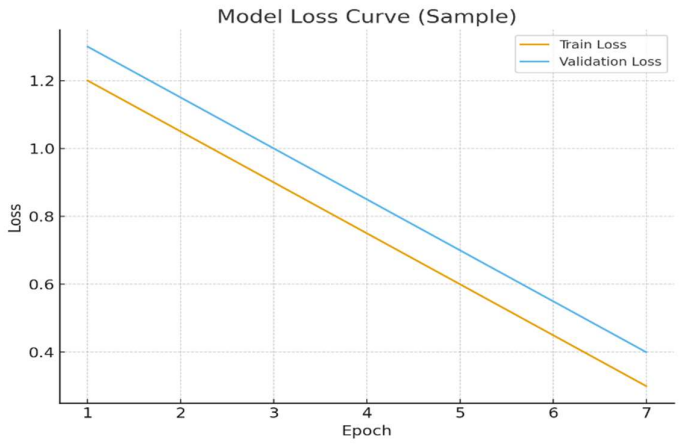
- **Language:** Python
- **Libraries:** PyTorch, NumPy, Matplotlib, torchvision
- **Architecture:** 3-layer CNN with pooling

- **Training:** Adam optimizer, LR scheduler, early stopping
- **Dataset Split:** 80% training, 20% validation

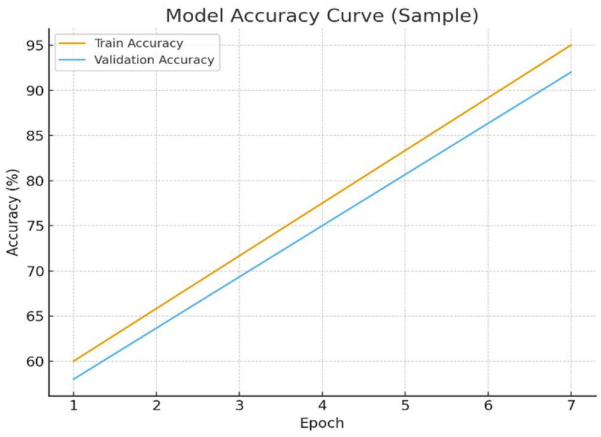
---

10. Screenshots / Results

Training Loss Curve



Training Accuracy Curve



---

11. Testing Approach

- Validation dataset used for unbiased evaluation.
- Accuracy and loss monitored across epochs.
- External MRI scans tested for prediction reliability.
- Possible comparison with baseline models.

---

12. Challenges Faced

- Limited dataset variety.
- Variation in MRI brightness/quality.
- GPU limitations slowed training.
- Hyperparameter tuning required multiple attempts.

---

13. Learnings & Key Takeaways

- Hands-on understanding of CNN architecture.
- Practical experience with PyTorch training loop.
- Importance of preprocessing and augmentation.
- Insight into model optimization strategies.

---

14. Future Enhancements

- ADD tumor segmentation (U-Net).
- Deploy as a web/mobile application.
- Expand dataset for multi-class tumor detection.
- Integrate real-time MRI imaging support.

---

15. References

- PyTorch Official Documentation
- Kaggle Brain MRI Dataset
- Research articles on CNNs in medical imaging
