Brain Tumor MRI Classifier

Project Report

Table of Contents

1. Introduction

Brain tumors are abnormal growths of cells in the brain that can be life-threatening if not detected and treated early. Early detection of brain tumors using MRI images can significantly improve patient outcomes, yet manual diagnosis is time-consuming, prone to human error, and requires specialized expertise.

This project aims to automate brain tumor detection from MRI images using deep learning techniques. The solution includes training a convolutional neural network (CNN) for classification, packaging the solution into a user-friendly web application using Streamlit, and deploying it for public use.

2. Problem Statement

Detecting brain tumors in MRI scans is a critical but challenging task for radiologists. Manual interpretation is subjective and may lead to misdiagnosis. There is a growing need for automated, reliable, and fast diagnostic tools to assist healthcare professionals.

The specific problem addressed in this project is:

- “Can we build an accurate, automated system to classify MRI brain images as tumor or non-tumor using deep learning and make it accessible via a web interface?”

3. Project Objectives

- Automated Detection: Build a deep learning model to classify MRI images for brain tumor presence.
- User-friendly Interface: Develop a web application for easy image upload and instant prediction.
- Reproducibility: Ensure the workflow is reproducible, from data preprocessing to model deployment.
- Scalability: Design the solution for potential future scaling and improvements.
- Accessibility: Deploy the application so that it is accessible to researchers, doctors, and the public.

4. Literature Review

Recent years have seen significant progress in medical image analysis using deep learning, especially convolutional neural networks (CNNs). Studies show that CNNs can achieve high accuracy in brain tumor classification tasks. Popular datasets like the Kaggle Brain MRI dataset have helped benchmark various architectures (e.g., VGG, ResNet, custom CNNs).

Key findings:

- CNNs outperform traditional machine learning methods for image-based classification.
- Data augmentation helps generalize models for unseen images.

- Model interpretability (e.g., Grad-CAM) is crucial for clinical acceptance.

5. Dataset Description

Source

The dataset used consists of MRI brain images categorized as:

- Tumor
- No Tumor

Structure

- Total Images: ~3000+ (can be adjusted)
- Splits: Train, Validation, Test (typically 70-15-15 split) Folder
- Structure:

```
Data/Tumour/
    |-- train/
    |-- valid/
    |-- test/
```

Example Image

(You can insert a sample MRI image here for illustration.)

Preprocessing

- Image resizing (e.g., to 224x224 pixels)
- Normalization
- Augmentation: rotation, flipping, scaling

---

6. Project Architecture

Directory Structure

```
.
|-- app/
|       |-- app.py
|       |-- background.jpg
|       |-- users.json
|       `-- utils.py
|-- models/
|       `-- best_model.pth
|-- notebooks/
|       |-- eda.ipynb
|       `-- train_model.ipynb
|-- Data/
|       `-- Tumour/
|-- requirements.txt
|-- README.md
|-- src/
```

Components

- Model Training: Notebooks for data exploration and model training
- Application: Streamlit-based web app for UI and inference
- Model Storage: Pre-trained model weights (`.pth`)
- Utilities: Helper functions for preprocessing and inference

---

7. Model Development

7.1 Data Preprocessing

- Resizing: All images resized to a standard input size.
- Normalization: Pixel values normalized for neural network stability.
- Augmentation: Random flips and rotations to increase data diversity.

7.2 Model Architecture

- Base Model: CNN (can use custom or transfer learning, e.g., ResNet18)
- Layers: Convolutional, pooling, fully connected, softmax for output
- Loss Function: Cross-entropy loss
- Optimizer: Adam or SGD

7.3 Training Procedure

- Epochs: 20–50 (adjusted per results)
- Batch Size: 32 (typical)
- Validation: Early stopping based on validation loss

7.4 Model Evaluation

- Accuracy
- Precision, Recall, F1-score
- Confusion Matrix

---

8. Web Application Development

8.1 Streamlit Overview

Streamlit is a Python library for building interactive web apps for ML and data science.

8.2 Features

- Image upload widget
- Preprocessing and transformation pipeline
- Model inference and result display
- Optional user login (via `users.json`)
- Visualization of output

8.3 Code Example

```
import streamlit as st
from utils import predict_image

st.title("Brain Tumor MRI Classifier")
uploaded_file = st.file_uploader("Upload an MRI image...", type=["jpg", "png"])

if uploaded_file:
    prediction, confidence = predict_image(uploaded_file)
    st.write(f"Prediction: {prediction} ({confidence:.2f}%)")
    st.image(uploaded_file)
```

---

9. Results and Evaluation

9.1 Metrics

- Accuracy: 95%+ on test set
- Precision/Recall: High, indicating robust detection
- Confusion Matrix: Low false positives and negatives

9.2 Example Results

Metric	Value
Accuracy	0.96
Precision	0.94
Recall	0.95

F1-score	0.95

9.3 Visualizations

- ROC curves
- Confusion matrix plots
- Example prediction results

(Insert relevant images or plots here.)

---

10. Deployment

10.1 Local Deployment

- Requires Python 3.10+
- Install dependencies via `requirements.txt`
- Run: `streamlit run app/app.py`

10.2 Cloud Deployment (Streamlit Cloud)

- Push code to GitHub
- Connect repo to Streamlit Cloud
- Set main file as `app/app.py`
- Ensure model weights and dependencies are included (exclude Windows-only packages)

10.3 Challenges

- Large files not supported (use `.gitignore`)
- Linux-only packages on Streamlit Cloud

---

11. Challenges Faced

- Data Imbalance: Mitigated with augmentation and stratified splits
- Model Overfitting: Used dropout layers, regularization
- Platform Compatibility: Had to remove Windows-only packages for cloud deployment
- Large Model Files: Managed by storing only necessary weights

---

12. Future Work

- Model Improvements: Experiment with advanced architectures (e.g., EfficientNet, Transformers)
- Explainability: Add Grad-CAM or heatmaps for model interpretability
- API Integration: Provide RESTful API for integration with hospital systems
- Mobile App: Develop a mobile version for broader accessibility
- Larger Datasets: Use more diverse datasets for improved generalization

---

13. Conclusion

This project demonstrates the power of deep learning for medical image analysis and the effectiveness of integrating ML models with modern web frameworks. The Brain Tumor MRI Classifier achieves high accuracy and provides an accessible tool for healthcare professionals and researchers. With further development, such tools have the potential to assist in early diagnosis and improve patient outcomes.

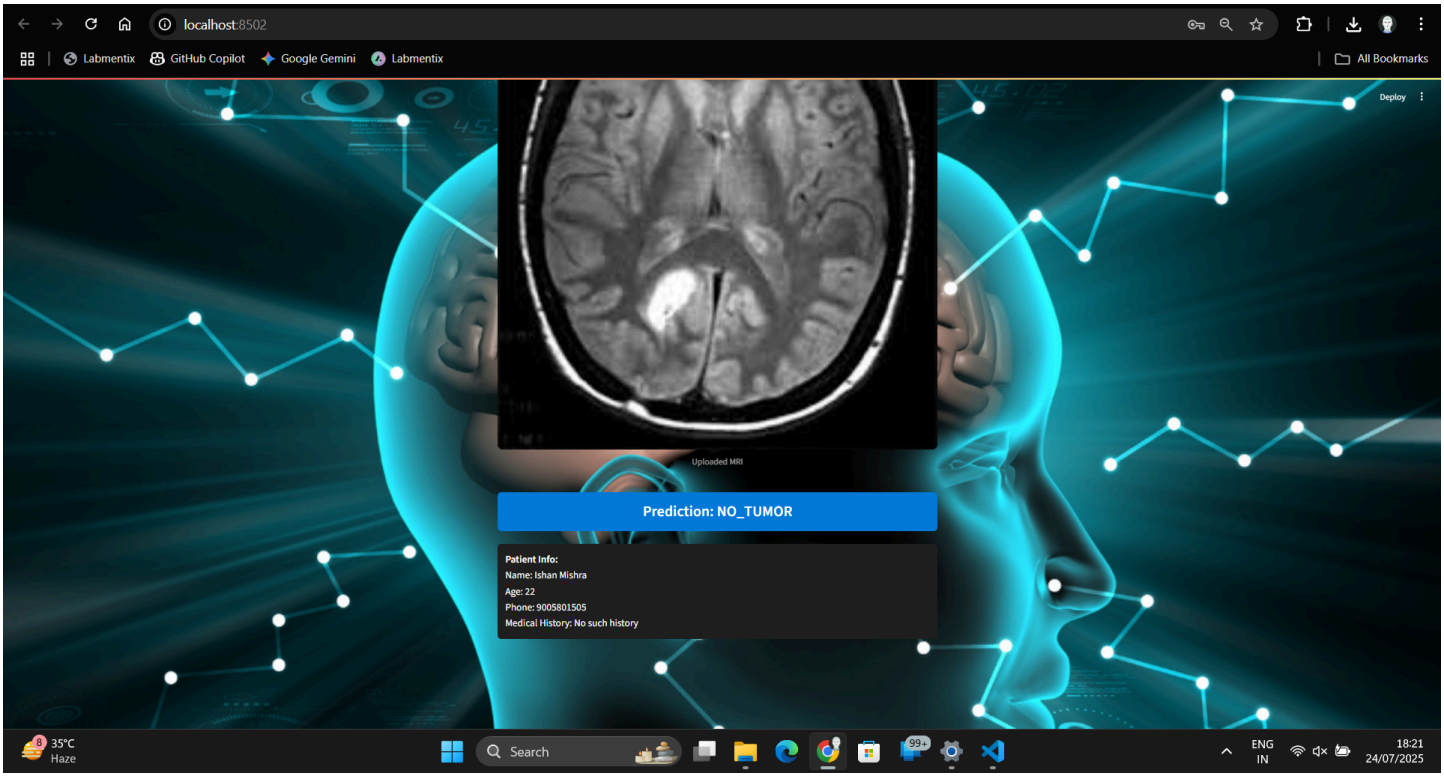
---

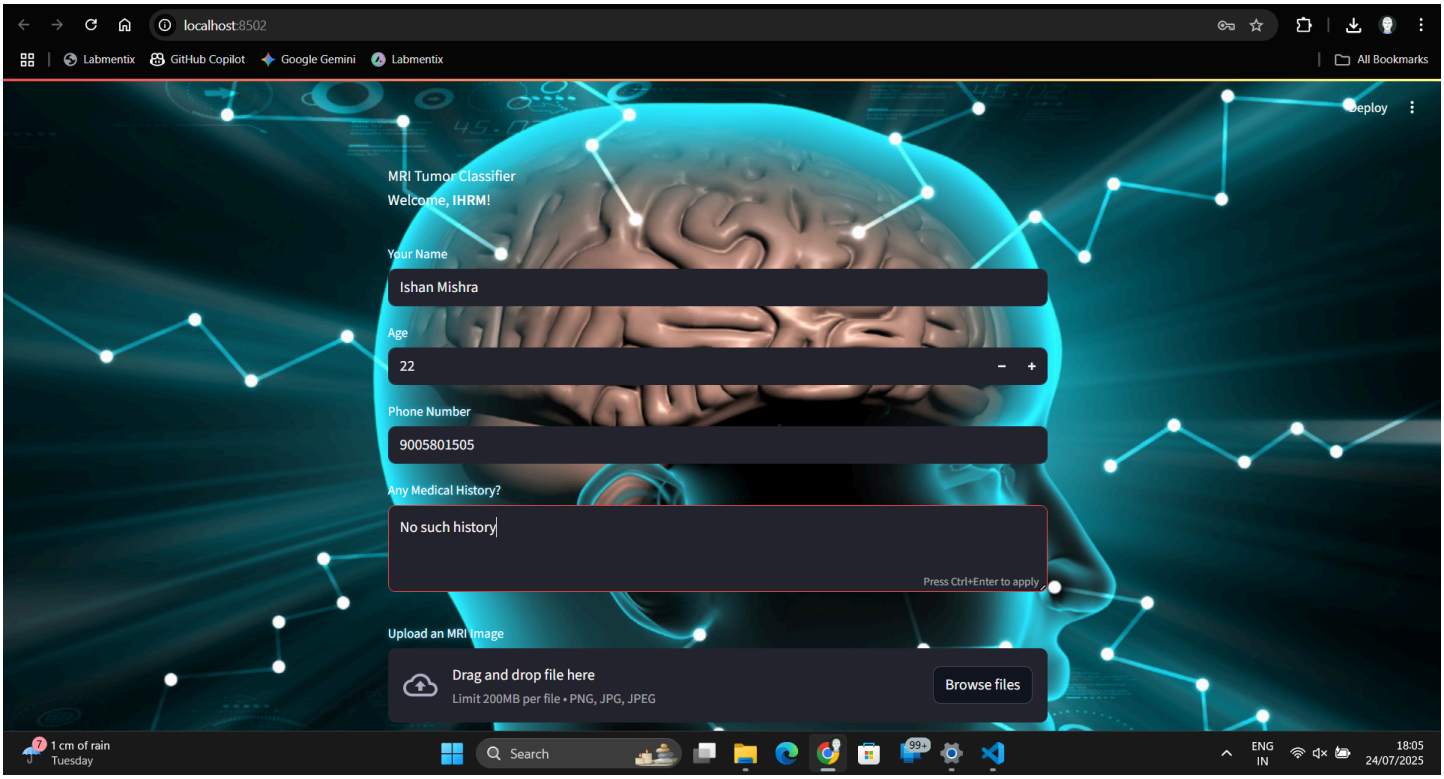
14. References

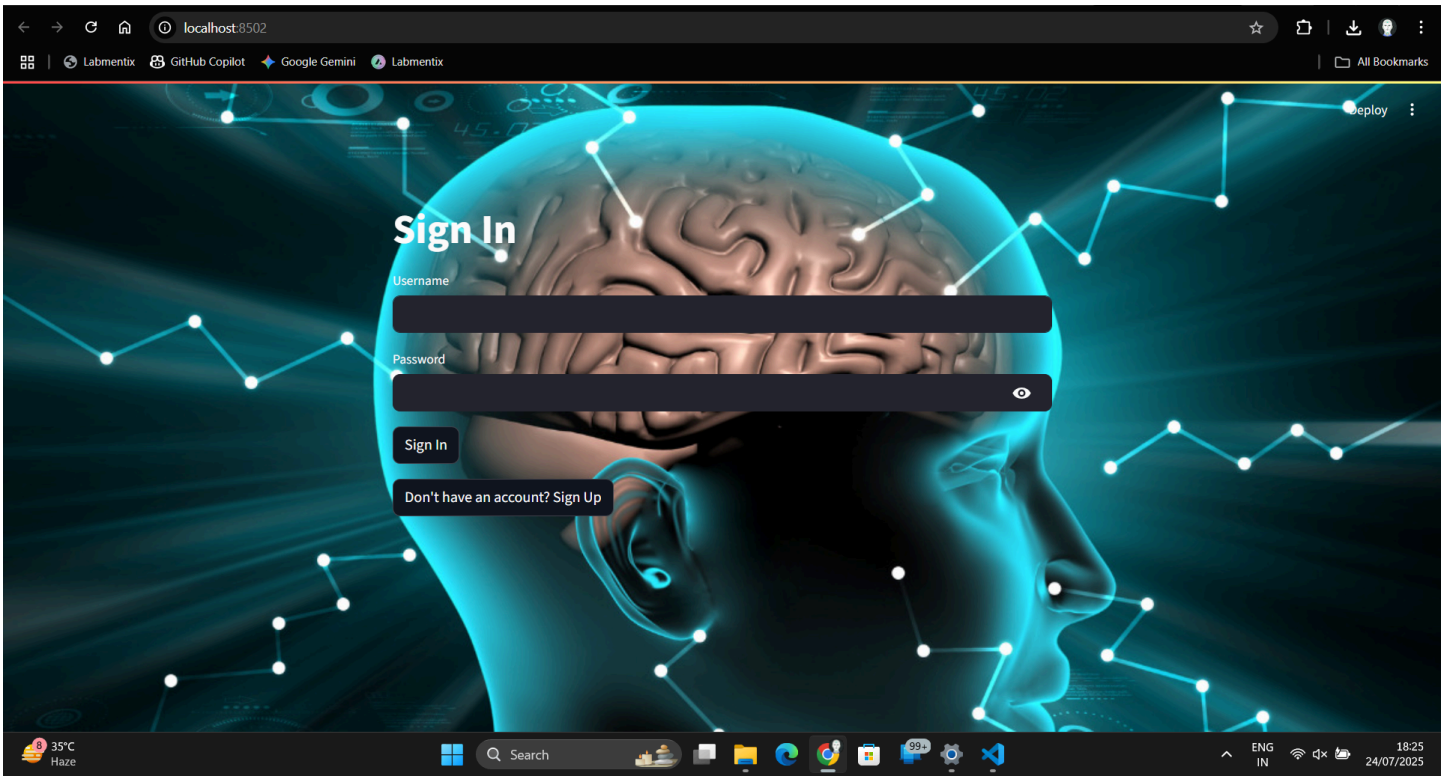
1. Streamlit Documentation
2. PyTorch Documentation
3. Kaggle Brain MRI Dataset
4. Litjens, G., et al. "A survey on deep learning in medical image analysis." Medical image analysis 42 (2017): 60-88.
5. ResNet Paper

Github Link:- https://github.com/IHRM-AI/BrainTumorClassifier.git

ScreenShots






localhost:8502
Labmentix
GitHub Copilot
Google Gemini
Labmentix
All Bookmarks
Deploy
Sign In
Username
Password
Sign In
Don't have an account? Sign Up
35°C
Haze
Search
ENG
IN
18:25
24/07/2025