# Brain MRI Classification for Tumour Detection

by

Rebecca Holland

24/25 Methods 4: Deep Learning and Big Data Integration: Technologies and Tools

18th June 2025

# Abstract

This project explores the use of artificial intelligence in MRI image-based brain tumour diagnosis. To determine if an MRI image had a tumour or not, a convolutional neural network built on the ResNet18 architecture was trained. The system, which was created in Google Colab and built in PyTorch, was eventually made available to users via a simple Streamlit interface that allowed anyone to input photographs and get predictions and confidence scores. The motivation behind this project is personal, inspired by the loss of a loved one to a brain tumour that was detected too late. The technical implementation, decision-making procedure, and practical utility of the system are described in this report.

# Contents

## 0.1 Project Idea

The project's goal is to create a binary image classifier that can recognise brain cancers in MRI pictures. It utilises transfer learning with the ResNet18 architecture to classify images into two categories: "Tumour" or "No Tumour." This concept directly applies to various topics covered in the module, specifically sessions 3,4, and 7. The technology supports early diagnosis choices, especially for non-specialist evaluation or in low-resource situations, offering a useful application of AI in healthcare.

## 0.2 What I Made

The system developed includes the following components:

- A fully commented and functional Python codebase written in Google Colab
- A custom PyTorch Dataset class for loading and transforming images
- A transfer learning setup using pretrained ResNet18
- Data augmentation strategies (rotation, flipping) to enhance generalisability
- A training and evaluation pipeline with confusion matrix and classification report
- A prediction function for newly uploaded MRI images
- An interactive Streamlit app that allows real-time image upload and classification
- Output of predictions alongside a confidence score to increase interpretability

The model achieved around 74% accuracy and showed reliable performance across both tumour and non-tumour classifications. The Streamlit interface was designed to simulate a potential deployment use case.

## 0.3 How I Made It

### 0.3.1 Realisation (Scope, Feasibility, Tools)

I used a publically accessible dataset from Kaggle to choose a project that strikes a mix between practicality and scope (Chakrabarty 2019). I used transfer learning using the ResNet18 architecture (He et al. 2016), pretrained on ImageNet, rather than creating a neural network from the ground up. I was able to save time and money by utilising already-learned functionalities. For cloud-based GPU access, Google Colab was used for all development, and local preparations were made for further testing and deployment.

### 0.3.2 Process (Software Engineering Decisions)

I followed a modular and accessible approach from the standpoint of software engineering. I used PyTorch DataLoaders for batching, made a special dataset class to control the loading of MRI images, and organised the model architecture by freezing early layers and redefining the last fully linked layer. The Adam optimiser with cross-entropy loss was employed by the system. The robustness of the model was enhanced by data augmentation.

A real-time diagnostic interface was then constructed using the trained model, which was saved as a '.pth' file and used in a different context. Because of its simplicity and speed, Streamlit was selected. In order to load the model, convert incoming user photos, and display the outcome along with the confidence score, I wrote an 'app.py' script. The tool is intended for non-technical users and can be accessed through a local browser.

### 0.3.3 Communication (Presentation and Documentation)

A detailed Colab notebook with inline annotations, prediction visualisations, and classification metric output is used to convey the project. Each design decision and the context of the model are fully documented in this study.

## 0.4 Code Explanation

The solution consists of two main components: a training pipeline developed in Google Colab and a deployed prediction interface using Streamlit. Below is a breakdown of the logic and structure of the code, along with how each part functions.

### 0.4.1 Google Colab Training Pipeline

The project begins with mounting Google Drive to ensure access to the image dataset stored in a specific directory. All required Python packages such as PyTorch, torchvision, scikit-learn, and matplotlib are installed and imported.

**Data Preparation** The images are located in two folders: one for tumour-positive scans and another for non-tumour scans. A Python script recursively loads the file paths and assigns class labels based on the folder name ('yes' = tumour, 'no' = no tumour). The dataset is split into training and testing sets using train_test_split from scikit-learn, stratified by label for balance.

A custom PyTorch Dataset class (BrainMRIDataset) is implemented to streamline loading and transforming images. The transforms applied include resizing to 128×128 pixels, random horizontal flipping, slight random rotations, and normalisation. These augmentations improve the model's ability to generalise by introducing variability during training.

**Model Architecture** The model I used to create this is ResNet18, a well-known convolutional neural network architecture pretrained on ImageNet. To fine-tune it for binary classification, the base layers are frozen (preventing their weights from updating during training), and the final fully connected layer is replaced with a new classifier consisting of:

- A linear layer reducing features to 128 units
- ReLU activation
- Dropout for regularisation
- A final linear layer mapping to two output classes

This configuration preserves general image recognition features while allowing training of the task-specific classifier.

**Training and Evaluation** The model is trained over 10 epochs using cross-entropy loss and the Adam optimiser. During each epoch, the model processes images in mini-batches, updating only the new classifier layers. After training, the model is evaluated on the test set using metrics like accuracy, precision, recall, and confusion matrix- all computed using classification_report and confusion_matrix from scikit-learn.

Additionally, a custom function predict_image() allows testing individual images by displaying them alongside their prediction and confidence score. Another function, show_predictions(), displays a few random predictions from the test set for visual verification.

Finally, the trained model's weights are saved to a .pth file using torch.save() for later deployment.

### 0.4.2 Streamlit Deployment App

The second part of the project is a web-based deployment using Streamlit, allowing non-technical users to interact with the trained model.

**Interface Design** The Streamlit script (app.py) begins by importing the necessary libraries including Streamlit, PIL for image handling, and PyTorch modules. A title and short description are displayed to the user.

**Model Loading** A function load_model() reconstructs the same ResNet18 architecture used during training and loads the saved weights from model.pth. It is wrapped with @st.cache_resource to prevent reloading on every interaction. If the model fails to load, an error message would show.

**Image Input and Prediction** Users can upload an image in .jpg, .jpeg, or .png format. Once uploaded, it is converted to RGB format and displayed. The image is transformed using the same preprocessing pipeline as the training data to ensure consistency.

The image is passed through the model, and a probability distribution over the two classes is produced using softmax. The highest value determines the prediction, and the associated confidence score is shown.

**Output** Streamlit then displays the uploaded image, the prediction (e.g., "Tumour"), and the model's confidence (e.g., "95.32%"). This real-time feedback makes the model accessible and interpretable for a broader audience.

## 0.5 Why I Made It

This project has personal significance in addition to being technically intriguing. My own experience of losing a family member to a brain tumour motivated me to create a technology that might aid in early diagnosis. My goal was to turn classroom learning into something impactful. The availability of free tools like Google Colab and Streamlit for the prototyping and deployment of such models demonstrates how accessible AI has become for tackling significant medical issues.

## 0.6 Impact and Real-World Use Case

Practically speaking, this model could help physicians, particularly in places where radiologists are hard to come by. It is intended to be user-friendly, lightweight, and interpretable. An MRI scan might be uploaded, and a prediction and confidence level could be obtained instantly by a technician or physician. This is useful for early screening in community clinics or for triage in crowded hospitals. The Streamlit interface's ease of use guarantees that the tool can be utilised as an instructional tool or linked into more extensive healthcare platforms.

# Bibliography

Chakrabarty, N. (2019), 'Brain mri images for brain tumor detection', https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection. Accessed: 2025-05-06.

He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)', pp. 770–778.