Brain Tumour Detection

Reem Hamieh
rah127@mail.aub.edu
202301773

Yasmin Halabi
ynh07@mail.aub.edu
202305100

Hanan Zwaihed
hfz02@mail.aub.edu
202301690

Abstract

This project presents a deep learning pipeline for simultaneous tumor classification and localization from grayscale MRI/CT brain scans. A multi-output convolutional neural network (CNN) is developed to predict both a binary tumor presence label and a bounding box indicating tumor position. The system processes pre-annotated medical images with label files for training and testing. The goal is to provide a lightweight, interpretable model to support clinical workflows.

1 Problem Statement

The goal of this project is to develop a deep learning model capable of detecting the presence of brain tumors in grayscale MRI/CT images. The system is designed to solve a binary classification problem, where each input image is labeled as either containing a tumor or not.

The input to the system is a preprocessed grayscale image of standardized size (224×224). The expected output is a binary prediction: 1 indicating the presence of a tumor, and `0` indicating no tumor. Model performance is primarily evaluated based on classification accuracy on a held-out test set, along with training and validation accuracy during the learning process.

2 Libraries and Tools

- **TensorFlow/Keras:** Used for building, training, and evaluating the convolutional neural network (CNN) model.
- **OpenCV and Pillow:** Used for loading, resizing, and preprocessing medical image data.
- **scikit-learn:** Employed for splitting the dataset into training, validation, and test sets using `train_test_split`.
- **NumPy:** Used for numerical operations and handling image arrays.

3 Implementation Details

3.1 Data Preprocessing

(1) **Image Loading**:
- Loaded grayscale MRI/CT scans in JPG format using `load_img()` from Keras
- Each image was resized to a standardized resolution of 224×224 pixels, ensuring consistent input dimensions for the CNN model.
- Loaded images were converted to NumPy arrays with `img_to_array()`

(2) **Intensity Normalization**:
- Pixel values were normalized to the $[0, 1]$ range by dividing all values by 255.0 which helped in stabilizing learning and ensuring pixel intensity consistency.
- Maintained single channel (grayscale) to reflect real-world clinical imaging formats.

(3) **Label Processing**:
- Parsed accompanying TXT files containing annotations
- Extracted binary labels (0=no tumor, 1=tumor) from first value
- Implemented error handling for missing/invalid files

3.2 Data Splitting

- The dataset was split into 80% training, 10% validation, and 10% testing using two successive `train_test_split()` calls.
- Class distribution was kept balanced using shuffling and a fixed random seed (42).
- This allowed proper monitoring of model generalization during training.

3.3 CNN Model Architecture

- **Input Layer**: Accepts 224×224×224×1 grayscale images matching the standardized preprocessing applied to all input data.
- **Convolutional Blocks**:
  - 3 sequential Conv2D layers with 32, 64, and 128 filters respectively (3×3 kernels)
  - Each followed by MaxPooling2D (2×2) to reduce spatial dimensions and control overfitting.
  - A shared Flatten layer feeds into two branches:
    - **Classification head:** Dense layer with sigmoid activation for binary output.
    - **Bounding box head:** Dense layer with linear activation to predict 4 bounding box coordinates.

Classification Head:
- A Flatten layer is used to convert the 2D feature maps into a 1D vector before passing it to dense layers.
- A fully connected Dense layer with 128 units and ReLU activation is used to learn high-level patterns.
- Dropout (0.5) for regularization
- The final output layer uses a single neuron with a sigmoid activation function to produce a binary classification output (tumor or no tumor).

3.4 Training Configuration

- **Loss Functions:** Binary cross-entropy for classification and MSE for bounding box regression.
- **Metrics:** Accuracy (classification) and MAE (bounding box).
- **Callbacks:** EarlyStopping and ReduceLROnPlateau were used to improve generalization.
- **Batch size:** 64
- **Epochs:** 100

Student report made for CMPS297AC
Cmps 261, Deadline: 2025-Apr-10,

3.5 Visualizations

- Plots of training/validation accuracy over epochs.
- Loss curves for both outputs.

4 Results

Include brief discussions about the results. Present the results using visualizations, Tables for quantitative results, images, etc.

4.1 What went right:

- The final model achieved a test accuracy of **0.82**, reflecting strong generalization despite the model's complexity.
- A more expressive CNN architecture with three convolutional blocks (32, 64, and 128 filters) was used to allow the model to capture more complex spatial features.
- Incorporating **early stopping** and **ReduceLROnPlateau** improved training stability and generalization by dynamically adjusting learning behavior and halting training at optimal points.
- The grayscale preprocessing and resizing to `224×224` ensured consistent input shape while maintaining important spatial features relevant to medical imaging.

4.2 What didn't go as expected:

- Applying **L2 regularization** resulted in significantly worse accuracy (~0.42), likely due to excessive constraint on the model's capacity in a small-data setting.
- Initial attempts at **data augmentation** led to poor results, potentially because the transformations (e.g., shear, zoom, rotation) distorted medically relevant features like tumor shape or location.
- Although the more complex CNN (with 32, 64, and 128 filters) had a higher risk of overfitting, it ultimately produced better performance than the simplified versions.

4.3 Challenges faced during implementation:

- Managing the custom dataset with `.jpg` image files and corresponding `.txt` label files required careful handling of missing or malformed annotations.
- Multiple training setups were tested to balance underfitting and overfitting, which made tuning hyperparameters like batch size, architecture depth, and callbacks an iterative and time-consuming process.
- The model's performance was difficult to interpret without additional tools like confusion matrices or class-wise metrics, which would offer deeper insights into false positives vs. false negatives.

4.4 Code

This is the link to our code.

5 Conclusion

Key Findings: The final CNN model achieved a test accuracy of 82%, demonstrating strong performance in distinguishing between tumor and non-tumor brain images. A more expressive architecture with three convolutional layers (32, 64, and 128 filters) allowed the model to learn complex spatial features. Although deeper networks typically risk overfitting on small datasets, this issue was mitigated through regularization techniques such as dropout, early stopping, and dynamic learning rate adjustment.

Lessons Learned: Through this project, we learned the importance of balancing model complexity with dataset size, especially in medical image analysis where annotated data is limited. We also experienced how design decisions—such as data normalization, validation splitting, and training callbacks—can significantly affect a model's performance. While data augmentation did not yield immediate improvements, it highlighted the need for domain-aware augmentation strategies when working with clinical images. Overall, the project provided valuable experience in developing, tuning, and evaluating deep learning models in a healthcare context.

6 Contributions

Each group member contributed equally to this project, fully understanding the code.

Student Declaration

Have you used any generative AI tools to complete this work:
YES □ NO ■

Details: Example: I used ChatGPT to debug my convolution implementation