Brain Tumor Analysis Pipeline: Segmentation & Classification

Project Documentation

November 23, 2025

1 Project Summary

This project implements a complete Deep Learning pipeline for the automated analysis of Brain Tumor MRI scans (Meningioma, Glioma, Pituitary). It utilizes a **Two-Stage Approach**:

1. **Segmentation (Stage 1):** A **U-Net** model isolates the tumor region from the MRI slice, generating a binary mask.
2. **Classification (Stage 2):** A **Custom CNN** classifies the tumor type. Crucially, the classifier is trained and tested on **cropped** tumor regions (derived from the masks) rather than the full MRI. This significantly reduces background noise and improves robustness.

Key Features

- **Advanced Preprocessing:** Anisotropic Diffusion for denoising and CLAHE for contrast enhancement.
- **Robust Evaluation:** Includes Pixel-level confusion matrices, Boundary Error Heatmaps, and Dice/IoU tracking.
- **Explainability:** Integrated **Grad-CAM** (Class Activation Mapping) to visualize CNN focus areas.
- **Professional Visualization:** Generates a comprehensive 6-panel A4 report for every inference, including overlays, probability maps, and confidence bars.

2 Project Structure

```
CS663_Project_Final/
|-- noise_accuracy.py           # Test classifier robustness against noise
|-- test_preprocessing.py       # Visualize the effects of denoising/CLAHE
|
|-- Data/
|   |-- converted_npy/          # Place your .npy dataset files here
|
|-- Models/
|   |-- unet_model.py           # U-Net architecture definition
|   |-- custom_cnn.py           # Custom CNN architecture for classification
|
|-- Segmentation/
|   |-- train_unet.py           # Training loop for U-Net
|   |-- eval_segmentation.py       # Comprehensive metrics (Dice, Boundary Error, etc
    .)
|   |-- inference_unet.py          # Run just the segmentation model
|   |-- postprocessing.py          # Morphological cleanup (removes small noise blobs)
|
|-- Classification/
|   |-- train_classifier.py        # Training loop for CNN (auto-crops tumors)
|   |-- inference_classifier.py    # Run just the classifier (with Grad-CAM)
|
|-- Utils/
|   |-- full_pipeline_viz.py       # MASTER SCRIPT: Full Seg -> Crop -> Classify -> A4
     Report
|   |-- dataset_loader.py          # Loads .npy files (Figshare format)
|   |-- image_preprocessing.py     # Anisotropic diffusion, CLAHE, Normalization
|   |-- visualization.py           # Plotting overlays, grids, and confusion matrices
|   |-- metrics.py                 # Calc functions: Dice, IoU, F1, etc.
|   |-- gradcam.py                 # Gradient Class Activation Mapping logic
|   |-- eval.py                    # Evaluation suite for the classifier
|
|-- Outputs/                       # Stores trained weights (.keras), logs, and plots
```

3 File Summaries

3.1 Root Directory

- `noise_accuracy.py`: A stress-test script that adds Gaussian noise to images to see how classification accuracy degrades.
- `test_preprocessing.py`: A utility to visually compare "Raw" vs "Preprocessed" (Denoised) images side-by-side.

3.2 Models/

- `unet_model.py`: Defines a standard U-Net with Encoder-Decoder blocks and skip connections for binary segmentation.
- `custom_cnn.py`: Defines a 3-block Convolutional Neural Network with a dense head for 3-class classification.

3.3 Segmentation/

- `train_unet.py`: Trains the U-Net using Binary Crossentropy + Dice Loss. Saves the best model to `Outputs/`.
- `eval_segmentation.py`: Generates detailed metrics (Dice/IoU histograms, Boundary Error maps, Sensitivity/Specificity) for the trained U-Net.
- `postprocessing.py`: Contains `keep_largest_component` and morphological operations to clean up noisy U-Net predictions.

3.4 Classification/

- `train_classifier.py`: Trains the Custom CNN. **Logic:** It automatically crops tumors using Ground Truth masks during training to teach the model to focus only on the relevant anatomy.

- `inference_classifier.py`: Runs inference on the classifier and generates Grad-CAM heatmaps to show "where the model is looking".

3.5 Utils/

- `full_pipeline_viz.py`: The primary inference script. It loads an image, runs the full segmentation-cropping-classification pipeline, and generates a detailed 6-panel A4 visualization report.
- `dataset_loader.py`: Handles loading the Figshare `.npy` dataset and splitting it into Train/Val/Test.
- `image_preprocessing.py`: Implements the `anisotropic_diffusion_denoise` and `apply_CLAHE` functions.
- `gradcam.py`: TensorFlow implementation of Grad-CAM to generate heatmaps from the last convolutional layer.

4 How to Run the Project

4.1 Step 0: Prerequisites

Ensure you have the required libraries installed. Note that `medpy` is required for anisotropic diffusion.

```
pip install tensorflow numpy opencv-python matplotlib scikit-learn seaborn medpy tqdm
```

4.2 Step 1: Data Setup

Ensure your `.npy` dataset files (images, masks, and labels) are located in:

`Data/converted_npy/`

4.3 Step 2: Train Segmentation (U-Net)

Train the U-Net to learn how to find tumors.

```
python3 Segmentation/train_unet.py
```

- **Output:** Best weights saved to `Outputs/UNet_preprocessed/best_unet.keras`.
- **Log:** `history.npy` and sample predictions saved in `Outputs/`.

4.4 Step 3: Train Classifier (Custom CNN)

Train the classifier. This script will automatically use GT masks to crop the training data, ensuring the CNN learns from clean inputs.

```
python3 Classification/train_classifier.py
```

- **Output:** Best weights saved to `Outputs/Classifier_Cropped/folds/fold_1/best_model.keras`.
- **Log:** Cross-validation summary saved to `cv_summary.json`.

4.5 Step 4: Run the Full Pipeline

This is the main inference script. It takes a raw image, passes it through the U-Net to find the tumor, crops it, and passes it to the CNN for classification. It produces a professional A4 diagnostic report.

```
python3 Utils/full_pipeline_viz.py --input Data/converted_npy/130_image.npy --out
    Final_Results/report.png --seg Outputs/UNet_preprocessed/best_unet.keras --clf
    Outputs/Classifier_Cropped/folds/fold_1/best_model.keras
```

Arguments:

- `-input`: Path to a `.npy` or standard image file.
- `-seg`: Path to U-Net weights.
- `-clf`: Path to CNN weights.
- `-gt_mask`: (Optional) Path to ground truth mask for comparison.

4.6 Step 5: Advanced Evaluation

Evaluate Segmentation Performance: Generates histograms, boxplots, and boundary error maps.

```
python3 Segmentation/eval_segmentation.py --out Evaluation/Segmentation_Results
```

Evaluate Classifier Performance: Generates ROC curves, PR curves, and Confusion Matrices.

```
python3 Utils/eval.py --model Outputs/Classifier_Cropped/folds/fold_1/best_model.keras
```

Test Noise Robustness: Check how the model handles noisy inputs.

```
python3 noise_accuracy.py --data Data/converted_npy --clf Outputs/Classifier_Cropped/
    folds/fold_1/best_model.keras
```

5 Technical Pipeline Logic

The system processes MRI scans through a strictly sequential two-stage pipeline designed to minimize background noise influence on classification.

1. **Input Handling and Standardization**
   - The system accepts MRI slices in various formats (`.npy`, `.jpg`, `.png`).
   - Inputs are automatically converted to grayscale. If the input is a 3-channel RGB image, it is converted using standard luminance weights to ensure channel consistency before processing.
2. **Advanced Preprocessing (Signal Enhancement)** Before segmentation, the raw MRI undergoes a rigorous enhancement pipeline:
   - **Resizing:** Images are standardized to 256×256 using `cv2.INTER_AREA`.
   - **Normalization:** Intensity values are normalized using Z-score standardization and then scaled.
   - **Anisotropic Diffusion:** A specialized denoising filter is applied ($\kappa = 25$, $\gamma = 0.05$, 5 iterations). This smooths homogeneous regions (tissue) while preserving sharp boundaries (edges/tumors).

- **CLAHE:** Contrast Limited Adaptive Histogram Equalization is applied (Clip Limit $= 1.2$, Tile Grid $= 14 \times 14$) to enhance local contrast.

3. **Stage 1: Semantic Segmentation (U-Net)** The preprocessed image is fed into a U-Net architecture to identify the tumor region:
   - **Inference:** The model outputs a pixel-wise probability map of shape $(256, 256, 1)$ with a `sigmoid` activation.
   - **Post-processing:** The raw probability map is cleaned by thresholding ($p \geq 0.5$) and finding the largest connected component to remove noise artifacts.
4. **Region of Interest (ROI) Extraction** To prepare for classification, the system extracts a clean view of the tumor:
   - **Bounding Box Calculation:** A bounding box is computed around the segmentation mask with a safety padding of 16-32 pixels.
   - **Cropping:** The original high-resolution image is cropped using these coordinates.
5. **Stage 2: Multi-Class Classification (CNN)** The cropped ROI is preprocessed again (Z-score normalization) and passed to the classifier:
   - **Architecture:** A custom 3-block CNN (32-64-128 filters) with a dense head (512 units).
   - **Output:** The model predicts the probability distribution across three classes: *Meningioma, Glioma, and Pituitary.*
6. **Visualization (A4 Report)** The pipeline finalizes by generating a 6-panel composite visualization:
   - **Panel A:** Original Image.
   - **Panel B:** Overlay (GT vs Prediction).
   - **Panel C:** Predicted Binary Mask.
   - **Panel D:** Segmentation Probability Heatmap (Jet colormap).
   - **Panel E:** The Cropped ROI used for classification.
   - **Panel F:** Horizontal Bar Chart comparing Predicted probabilities vs Ground Truth label.