SILESIAN UNIVERSITY OF TECHNOLOGY
FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

BIOLOGICALLY INSPIRED ARTIFICIAL INTELLIGENCE

PROJECT REPORT
TOPIC: BRAIN TUMOR DETECTION FROM MRI IMAGES

Supervisor: mgr inż. Marcin Wierzchanowski

Student: Illia Karpenko

Gliwice 2024

Contents

1. Introduction

There are many life-threatening health conditions that if diagnosed late, can lead to death. Brain tumors are one of those diseases that require timely diagnosis to be effectively treated.

Magnetic Resonance Imagin (MRI) is the most valuable tool in the diagnosis of brain tumors. It provides detailed images of brain structure without ionizing radiation. However, the detection and segmentation of brain tumors from MRI images can be challenging. It is a process that requires a certain medical expertise.

This project addresses the task of brain tumor detection with the help of artificial intelligence.

2. Objectives

The goal of this project is to create an application that can help doctors to identify tumor regions on the MRI scans of the brain. This application will speed up the diagnostic process, aid treatment planning and improve patient care.

3. Analysis of the task

The task of brain tumor detection has a diverse range of potential approaches, each with its own set of advantages and disadvantages.

3.1 Possible approaches

Traditional image processing methods:

- Thresholding
  Pros: simple, computationally fast, suitable for well-defined tumors with distinct intensity differences.

Cons: sensitive to noise and intensity variations within tumors and healthy tissue, may require manual threshold adjustment, limited accuracy for complex tumor shapes.

- Region growing
  Pros: more flexible than thresholding, can handle some intensity variations, able to segment connected regions.
  Cons: sensitive to noise, struggles with tumors that merge with healthy tissue, computationally more expensive than thresholding.

- Morphological operations
  Pros: can enhance tumor boundaries, remove noise, and fill holes in segmented regions.
  Cons: highly dependent on parameter tuning, can introduce artifacts.

Machine learning methods:

- Support vector machines (SVM)
  Pros: good generalization capability, effective for high-dimensional feature spaces.
  Cons: requires careful feature engineering, training can be slow for large datasets, may not capture intricate tumor structures.

- Random forests
  Pros: robust to noise and outliers, less prone to overfitting than individual decision trees, can handle missing data.
  Cons: computationally expensive, difficult to interpret, may not capture fine-grained details in tumor boundaries.

Deep learning methods:

- Convolutional neural networks (CNN)

Pros: automatic feature learning from raw data, powerful representation capacity, excels at image classification and object detection.
Cons: requires substantial amounts of annotated data, may lose spatial information due to pooling, can be prone to overfitting.

- U-Net
  Pros: specifically designed for biomedical image segmentation, excellent at capturing spatial information, effective with limited training data.
  Cons: can be computationally expensive, requires careful hyperparameter tuning, may struggle with highly imbalanced datasets.

3.2 Selected methodology

For this project, we selected a convolutional neural network U-Net. It was developed specifically for biomedical image segmentation at the Computer Science Department of the University of Freiburg.

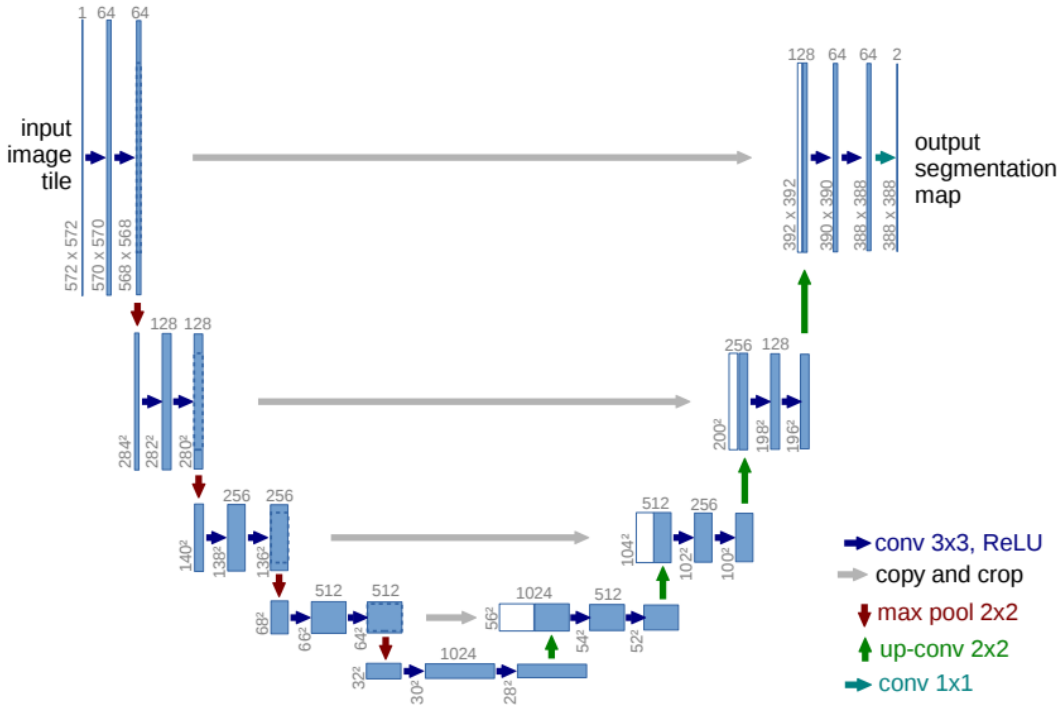

Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

The U-Net architecture illustrated in Figure 1 resembles a “U” shape. It consists of two main paths:

1. Contracting path (Encoder): this path follows a typical CNN structure and is responsible for feature extraction. It comprises four blocks, each containing two 3x3 convolutional layers followed by batch normalization and ReLU activation. A 2x2 max-pooling operation with stride 2 is applied after each block to downsample the feature maps, effectively doubling the number of feature channels while reducing spatial dimensions.

2. Expanding path (Decoder): this path is responsible for upsampling the feature maps back to the original image size and refining the segmentation details. It consists of four blocks, each containing a 2x2 up-convolution that halves the number of feature channels, followed by concatenation with the corresponding cropped feature map from the contracting path (skip connection). This is followed by two 3x3 convolutional layers and a ReLU activation.

Key components of U-Net architecture:

- Skip connections: these are the defining features of U-Net. They connect the feature maps from the contracting path to the corresponding layers in the expanding path. This allows the network to recover fine-grained details lost during downsampling, resulting in more accurate and precise segmentations.
- Convolutional layers: these layers apply filters to the input image (or feature maps) to extract features like edges, textures, and shapes.
- Batch normalization: this normalizes the activations of each layer, accelerating training and improving model performance.
- ReLU activation: the rectified linear unit (ReLU) introduces non-linearity into the network, enabling it to learn complex relationships in the data.
- Max pooling: this downsamples the feature maps, reducing the computational burden and increasing the receptive field.

- Transposed convolution (Up-convolution): this upsamples the feature maps, increasing their spatial resolution.

3.3 Dataset

For this project, we utilized the LGG (lower-grade glioma) Segmentation Dataset available on Kaggle. This dataset provides a valuable resource for training and evaluating brain tumor segmentation models. It comprises brain MRI images and corresponding manual FLAIR (fluid-attenuated inversion recovery) abnormality segmentation masks, offering a comprehensive representation of LGG tumors.

Dataset details:

- Source: the images were obtained from The Cancer Imaging Archive (TCIA), specifically from the lower-grade glioma collection of The Cancer Genome Atlas (TCGA).
- Patients: the dataset includes data from 110 patients, each with at least a fluid-attenuated inversion recovery (FLAIR) sequence and available genomic cluster data. Tumor genomic clusters and patient information are provided in a data.csv file.
- Image format: all images are in .tif format with three channels per image.
- Mask format: the masks are binary, single-channel images that segment the FLAIR abnormality present in the FLAIR sequence.
- Total number of images: 7858 (3939 pairs of MRI images with masks)
- Collection methodology: the segmentation masks were created and approved by a board-certified radiologist at Duke University, ensuring the quality and reliability of the annotations.
- Organization: the dataset is organized into 110 folders, one for each patient, each named after a case ID and containing information about the source institution. Each folder includes MRI images with a standardized naming convention: TCGA_<institution-

code>_<patient-id>_<slice-number>.tif. The corresponding masks have a _mask suffix.

This dataset was chosen for several reasons:

- The masks are created and approved by a medical expert, ensuring high-quality annotations.
- The dataset encompasses a variety of tumor shapes, sizes, and locations across different patients, contributing to a robust training process.
- It is publicly available on Kaggle, facilitating reproducibility and further research

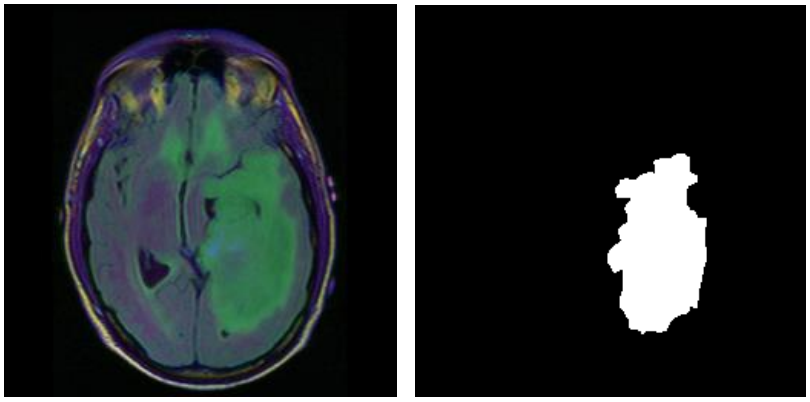

Fig. 2. Example of an MRI image and a corresponding mask from the dataset

3.4 Data preprocessing

Given that the chosen dataset already provides preprocessed images and masks, our preprocessing steps focused on preparing the data for training and inference within our deep learning model. The following operations were performed:

- Train, validation, test split: the dataset was randomly split into three subsets:
  - training set (72%): used to train the U-Net model.
  - validation set (18%): used to monitor model performance during training and adjust hyperparameters if needed.

  - test set (10%): used for final evaluation of the trained model's performance.
- Data augmentation: to increase the diversity of the training data and improve the model's ability to generalize, we applied various data augmentation techniques to the training set. These techniques included random rotations, width and height shifts, shearing, zooming, and horizontal flipping.
- Image normalization: both input images and masks were normalized by dividing their pixel values by 255. This scales the pixel values to the range [0, 1], which is often beneficial for neural network training.
- Mask normalization: masks were binarized by setting pixel values above 0.5 to 1 (tumor) and pixel values below or equal to 0.5 to 0 (background). This ensured consistency for training and evaluation.
- Batch generation: images and masks were organized into batches for efficient training.

3.5 Tools and frameworks

In the realm of deep learning and medical image processing, a variety of tools and frameworks are available for development. These include:

- TensorFlow: a comprehensive open-source platform developed by Google, renowned for its flexibility, scalability, and production-ready capabilities. TensorFlow offers a vast ecosystem of tools and libraries, including Keras, for building and deploying machine learning models.
- Keras: a high-level neural networks API written in Python, running on top of TensorFlow. Keras simplifies the process of defining and training models with its user-friendly interface and modular building blocks.
- PyTorch: an open-source machine learning framework developed by Meta AI, known for its dynamic computation graph and strong research community.
- SimpleITK (Insight Segmentation and Registration Toolkit): a simplified layer built upon the ITK (Insight Toolkit), specifically designed for medical image analysis. SimpleITK provides tools for image I/O, filtering, registration, and segmentation.

We chose to use Keras as our high-level API on top of TensorFlow. Keras' intuitive syntax and modular design significantly simplified the process of building the U-Net architecture.

4. Specifications of the software solution

4.1 Internal specifications

The core components of our software solution are structured as follows.

Python scripts:

- main.py: the primary script of the entire workflow. It handles:
  - loading the dataset and performing the train/validation/test split.
  - setting up data generators for efficient training and data augmentation.
  - instantiating and compiling the U-Net model.
  - training the model with specified parameters and callbacks.
  - evaluating the model's performance on the validation and test sets.
- unet.py: defines the U-Net model architecture. It handles:
  - the initialization of the model layers and their connections.
  - the core function *unet(input_size)* constructs and returns a Keras model object representing the U-Net architecture.
- utils.py: provides utility functions for various tasks:
  - *dice_coefficients* function calculates a metric for evaluating the similarity between two samples.
  - *dice_coefficients_loss* function calculates the negative of the dice coefficient, as loss functions are typically minimized during training.
  - *intersection_over_union* function calculates the Intersection over Union (IoU) score, a metric for evaluating segmentation accuracy.

Data Structures:

- Pandas DataFrames: used to organize and manage image and mask file paths for the dataset. It facilitates splitting and batch generation.
- NumPy arrays: used to represent images and masks as numerical matrices, enabling efficient processing and manipulation.

4.2 External specifications

The external interface for interacting with the solution is a web application built using Flask, a lightweight Python web framework chosen for its simplicity and ease of development.

Web application structure:

- User interface (HTML): the main template for the web application that includes a form for file uploading, a button that triggers the image analysis and placeholders for the input and prediction images.
- Styling (CSS): the stylesheet defines a simple visual appearance of the web page.
- Backend processing (Flask): handles file uploads and saves the image to the *static/uploads* folder, calls the *process_image* function to perform brain tumor detection with the trained model.

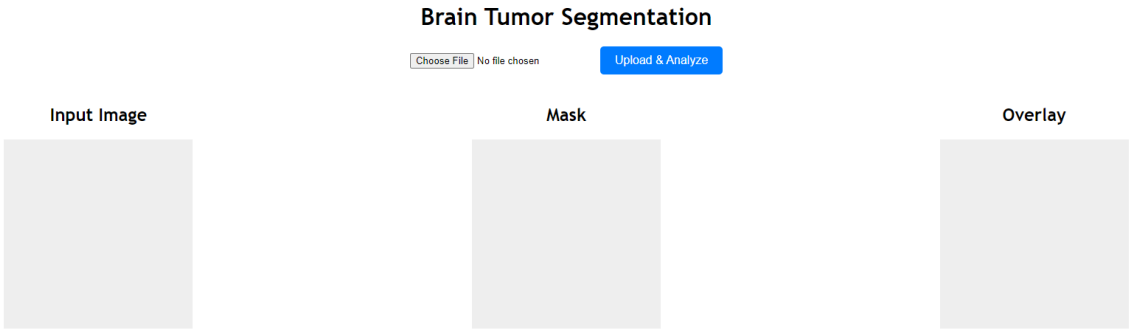

Fig 3. "The application user interface after launch"

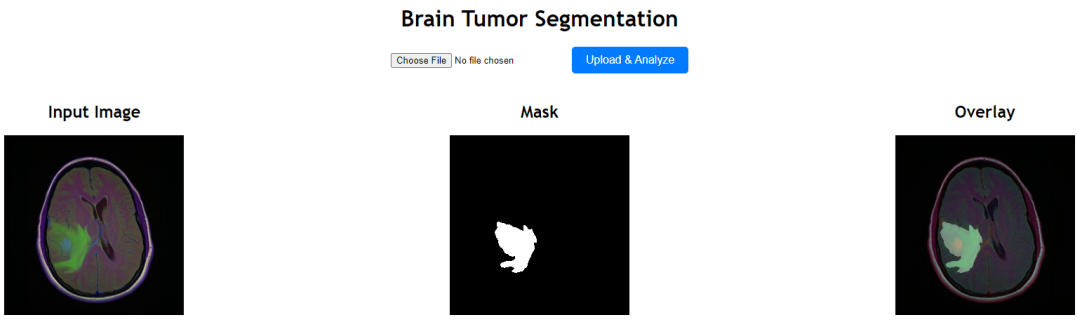

Fig. 4 "The application user interface after uploading an image and analyzing it"

5. Experiments

5.1 Experimental background

In this project, we conducted a series of experiments to train and evaluate a U-Net model for brain tumor detection and segmentation. The primary objectives of these experiments were to:

- Strive for high performance of the model with IoU and dice coefficient above 0.80 in test results.
- Validate model performance across diverse data samples.
- Address and resolve issues related to GPU processing.
- Find insights into the model's performance and areas for improvement.

Initially, we attempted to train the model on a standard computer. However, due to the computational demands of deep learning and the size of the dataset, the training process was prohibitively slow – 4 hours per epoch.

To overcome this limitation, we transitioned to the Kaggle Kernel cloud platform, which provides access to powerful NVIDIA Tesla P100 GPUs for accelerated training.

To quantify the performance of each epoch, we used different evaluation metrics:

- Dice Coefficient - a measure of overlap between two samples.

$$Dice\ Coefficient = \frac{2 \cdot |A \cap B| + smooth}{|A| + |B| + smooth}$$

- Intersection over Union (IoU)

$$IoU = \frac{|A \cap B| + smooth}{|A \cup B| + smooth}$$

- Dice Coefficient Loss – represents the negative of Dice Coefficient. In our solution it ranges from 0 (no overlap) to -1 (perfect overlap). Most optimization algorithms are designed to minimize a function rather than maximize it.

  A – ground truth binary mask
  B – predicted binary mask
  smooth – smoothing factor to avoid division by zero.

To ensure we capture the optimal model parameters during training, we utilized Keras' ModelCheckpoint callback, which automatically saves the model weights whenever the validation loss reaches a new minimum value.

5.2 Presentation of experiments and results

We performed multiple training runs with varying hyperparameter to find the optimal configuration.

Learning rate:

An initial learning rate of 0.0001 is a standard starting point for many deep learning tasks, including image segmentation. We used a polynomial learning rate scheduler to gradually decrease the learning rate over the course of training. The initial learning rate was set at 0.0001, and it decreased to 0.000001 to by the end of the training process. This approach allowed the model to learn quickly in the initial stages and then fine-tune its parameters as training progressed.

$$LR(epoch) = \left((initial_{lr} - end_{lr}) \cdot \left(1 - \frac{epoch}{EPOCHS}\right)\right) + end_{lr}$$

LR(epoch) - the learning rate at a given epoch.
$initial_{lr}$ – starting learning rate
end_{lr} – ending learning rate
epoch – current epoch number
EPOCHS – total number of epochs

We decided to train the model for 150 epochs and batch size 32. The statistics gathered during the first 3 epochs and overall progress of the model training is displayed on Figure 5.

```
Epoch 1: LearningRateScheduler setting learning rate to 0.0001.
Epoch 1/150
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 674ms/step - binary_accuracy: 0.8166 - dice_coefficients: 0.0728 - intersection_over_union: 0.0385 - loss: -0.0728Found 708 validated ima
ge filenames.
Found 708 validated image filenames.

Epoch 1: val_loss improved from inf to -0.03642, saving model to unet.keras
88/88 ━━━━━━━━━━━━━━━━━━━━ 84s 781ms/step - binary_accuracy: 0.8177 - dice_coefficients: 0.0733 - intersection_over_union: 0.0388 - loss: -0.0733 - val_binary_accurac
y: 0.9882 - val_dice_coefficients: 0.0364 - val_intersection_over_union: 0.0186 - val_loss: -0.0364 - learning_rate: 1.0000e-04

Epoch 2: LearningRateScheduler setting learning rate to 9.934e-05.
Epoch 2/150
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 673ms/step - binary_accuracy: 0.9845 - dice_coefficients: 0.1802 - intersection_over_union: 0.1003 - loss: -0.1806
Epoch 2: val_loss improved from -0.03642 to -0.04877, saving model to unet.keras
88/88 ━━━━━━━━━━━━━━━━━━━━ 72s 759ms/step - binary_accuracy: 0.9845 - dice_coefficients: 0.1803 - intersection_over_union: 0.1003 - loss: -0.1807 - val_binary_accurac
y: 0.9863 - val_dice_coefficients: 0.0475 - val_intersection_over_union: 0.0247 - val_loss: -0.0488 - learning_rate: 9.9340e-05

Epoch 3: LearningRateScheduler setting learning rate to 9.868000000000001e-05.
Epoch 3/150
88/88 ━━━━━━━━━━━━━━━━━━━━ 0s 673ms/step - binary_accuracy: 0.9858 - dice_coefficients: 0.2283 - intersection_over_union: 0.1309 - loss: -0.2280
Epoch 3: val_loss improved from -0.04877 to -0.09296, saving model to unet.keras
88/88 ━━━━━━━━━━━━━━━━━━━━ 66s 750ms/step - binary_accuracy: 0.9858 - dice_coefficients: 0.2284 - intersection_over_union: 0.1309 - loss: -0.2281 - val_binary_accurac
y: 0.9900 - val_dice_coefficients: 0.0922 - val_intersection_over_union: 0.0492 - val_loss: -0.0930 - learning_rate: 9.8680e-05
```

Fig. 5 "The process of model training"

The best results were achieve on the 119th epoch. The model was automatically saved to the *unet.keras* file. We obtained the following results on the test set:

Loss: -0.88.

Dice coefficient: 0.78

IoU: 0.99

```
Found 393 validated image filenames.
Found 393 validated image filenames.
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 230ms/step - binary_accuracy: 0.9974 - dice_coefficients: 0.8856 - intersection_over_union: 0.7978 - loss: -0.8856
Test Loss  -0.8806064128875732
Test IoU  0.9976422190666199
Test Dice Coefficient  0.7893004417419434
```

Fig. 6 "Results of the trained model on the test set"

The plot in Figure 7 displays the training (red line) and validation loss (blue line) over 150 epochs. Both the training and validation loss decrease

significantly within the first 20 epochs. This indicates that the model is quickly learning the underlying patterns in the data and improving its performance.

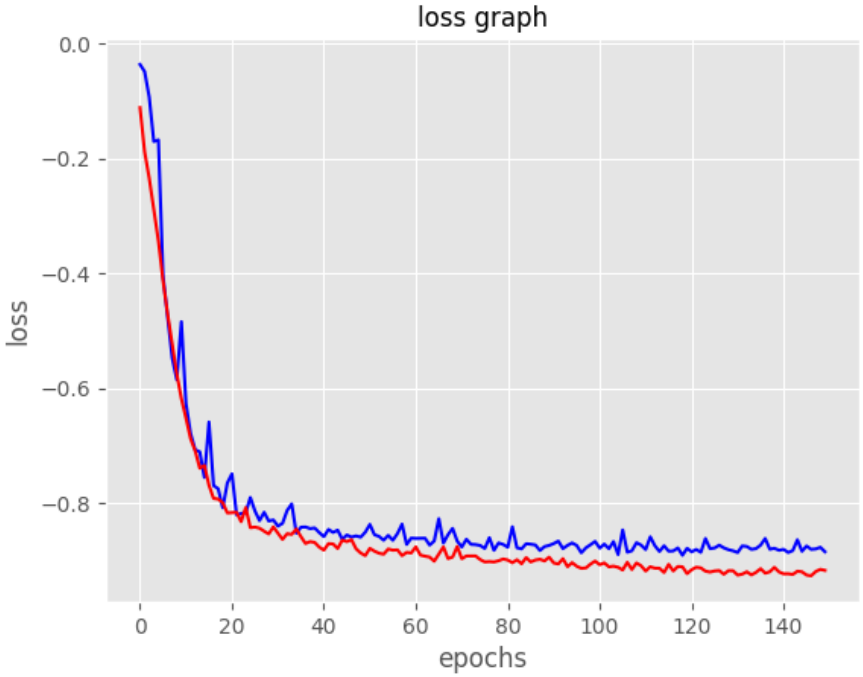

Fig. 7 “Training and test loss of the model”

From about epoch 20, the training and validation losses converge and remain relatively stable with minor fluctuations. The model does not exhibit significant overfitting as the validation loss does not increase while the training loss decreases.

The plot in Figure 8 illustrates the training and validation Dice Coefficient over 150 epochs. The plot shows a rapid increase in both training and validation Dice Coefficients within the first 20 epochs, followed by stabilization around epoch 40, indicating effective learning and good generalization.

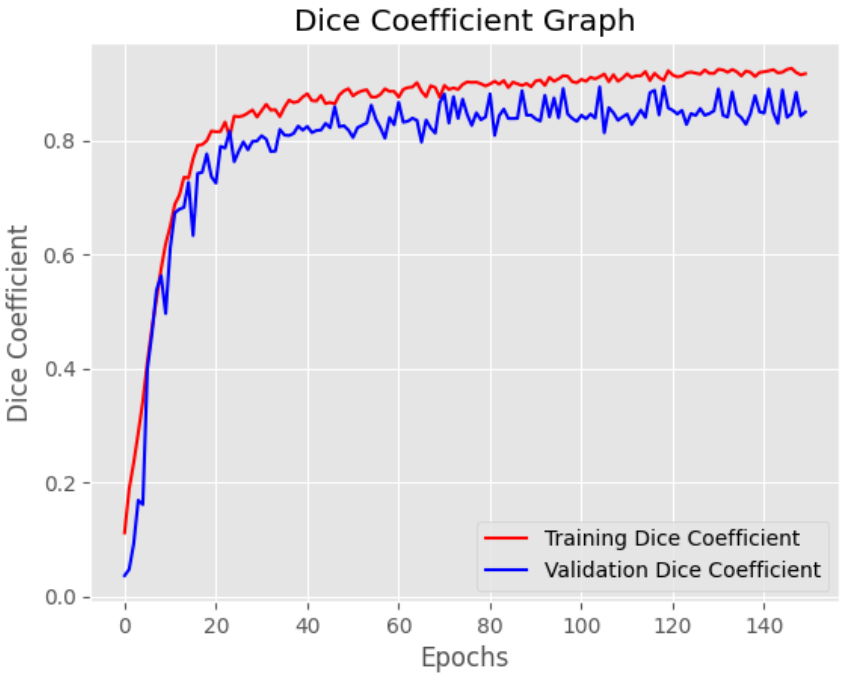

Fig. 8 "Dice Coefficient throughout the model training"

The IoU plot in Figure 9 exhibits a similar pattern to the Dice Coefficient plot, with a rapid increase during the initial epochs and stabilization around epoch 40, indicating consistent performance.

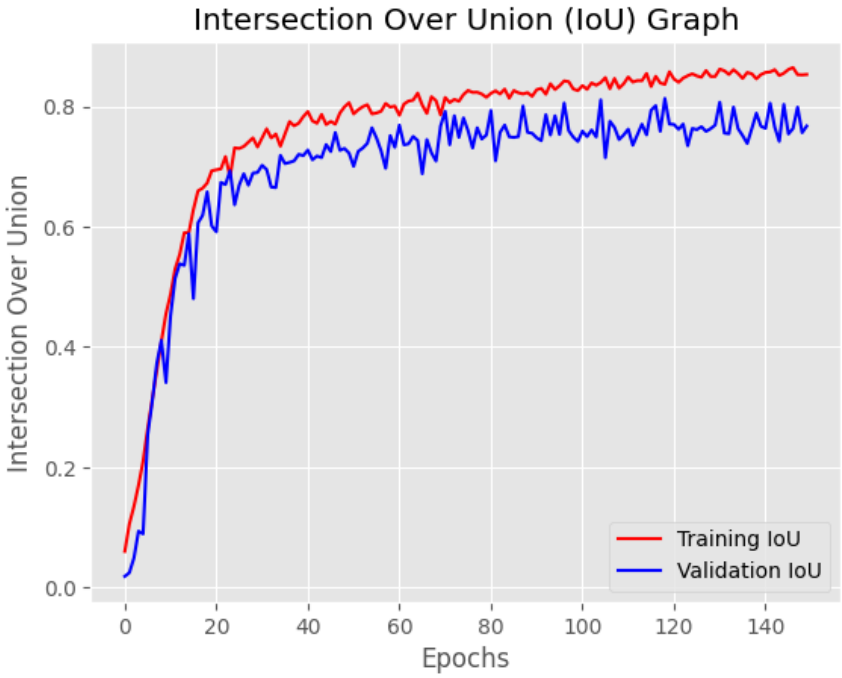

Fig. 9 "IoU throughout the model training"

5.3 Application demonstration

The final look of the application is demonstrated on Figures 10 to 12

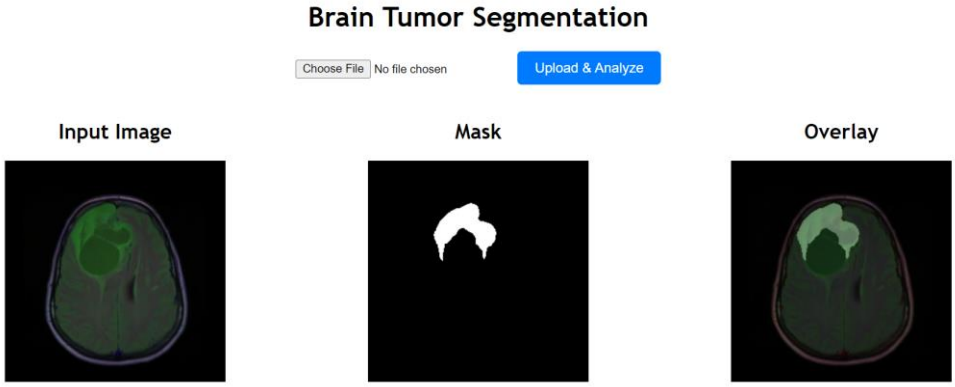

Fig. 10 "Application demonstration"

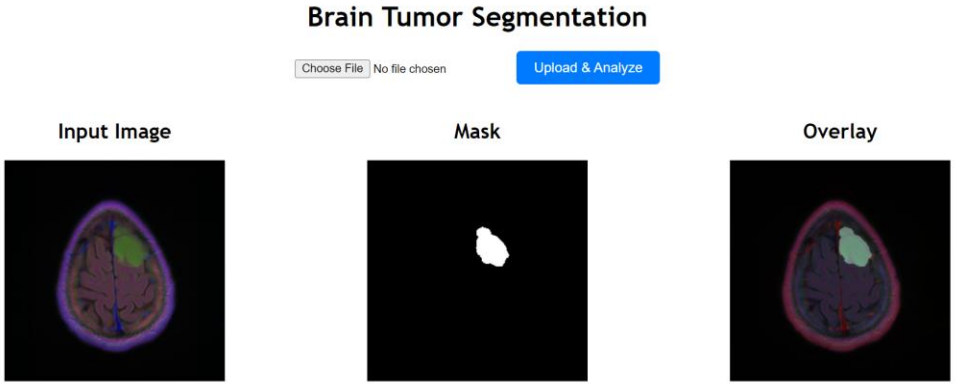

Fig. 11 "Application demonstration"

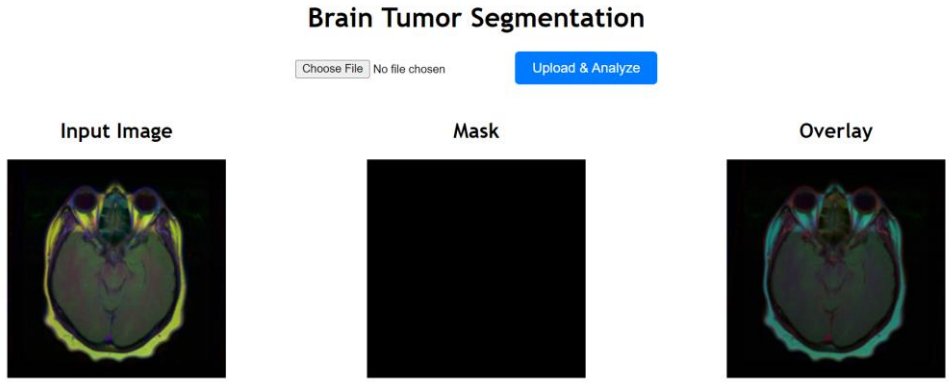

Fig. 12 "Application demonstration"

6. Summary

In this project, we successfully developed a brain tumor detection system leveraging the power of the U-Net deep learning architecture. Our system achieved good results in identifying and segmenting tumor regions within MRI scans. The utilization of the LGG Segmentation Dataset, combined with appropriate preprocessing techniques and hyperparameter tuning, led to a robust model capable of generalizing to unseen data.

However, there are several areas where the system could be further enhanced. Post-processing techniques like morphological operations or conditional random fields (CRF) could be used in future versions of the application to refine the segmentation results by smoothing boundaries and reducing false positives. Expanding the dataset with a greater diversity of tumor types would likely improve the model's robustness and applicability in real-world clinical settings.

The integration of such biologically inspired AI solutions into clinical workflows could lead to more efficient and accurate healthcare for this critical medical condition.

References

1. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." *In Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015)*. Springer, Cham, 2015. 234-241. https://arxiv.org/abs/1505.04597

2. Keras 3 API documentation https://keras.io/api/

3. Mateusz Buda, Research Associate at Duke University. "Brain MRI Segmentation." Kaggle, 2019 https://www.kaggle.com/datasets/mateuszbuda/lgg-mri-segmentation/data

4. Flask documentation https://flask.palletsprojects.com/en/3.0.x/api/

5. Kaggle Kernel documentation https://www.kaggle.com/docs/notebooks

6. Project files https://github.com/IlliaKarpenko/BrainTumorDetection