PROJECT REPORT

Brain tumor classification and detection

Prepared by

Menna mahmoud | Nour Yahia

Abdullah Yasser | Amr Ibrahim

Ahmed Abolenaga | Sara Alaa

Supervision: Eng. Abdullah Wagih

Introduction

Brain tumor diagnosis is a critical task in medical imaging, requiring both accurate classification and precise localization to support effective treatment planning. In this project, we used a private MRI dataset comprising T1-weighted, contrast-enhanced T1, and T2-weighted scans. The dataset is divided into 44 folders across 14 brain tumor types, including astrocytoma, carcinoma, ependymoma, ganglioglioma, germinoma, glioblastoma, granuloma, medulloblastoma, meningioma, neurocytoma, oligodendroglioma, papilloma, schwannoma, and tuberculoma. All images were de-identified, interpreted by radiologists, and made available for research purposes.

After evaluating various deep learning models, we finalized our approach using EfficientNet-B5 for tumor classification and YOLOv8 for detection. EfficientNet-B5 was selected for its strong performance in multi-class classification tasks, while YOLOv8 provided an efficient and accurate solution for localizing tumors within the MRI images.

To evaluate our models, we used standard performance metrics. For classification, we relied on precision, recall, F1-score, and accuracy. For detection, we used metrics common in object detection tasks, including mean Average Precision (mAP), Intersection over Union (IoU), precision-recall curves, and model fitness.

The system was deployed using (insert deployment platform here), providing an interactive interface for uploading scans, performing classification, and visualizing tumor detection results.

This project demonstrates how combining advanced classification and detection models can significantly enhance diagnostic workflows in neuro-oncology.

Dataset Description

This project utilizes a private collection of brain MRI images, encompassing various imaging modalities such as T1-weighted, contrast-enhanced T1 (T1C), and T2-weighted scans. The dataset is organized into 44 folders, each corresponding to a specific brain tumor type. All images are anonymized, devoid of any patient identifiers, and have been interpreted by qualified radiologists, ensuring their suitability for research and study purposes.

Tumor Types

The dataset includes images representing the following 14 distinct brain tumor types: Papiloma, Tuberculoma, Granuloma, Carcinoma, _NORMAL, Ganglioglioma, Neurocitoma, Schwannoma, Germinoma, Meduloblastoma, Meningioma, Astrocitoma, Oligodendroglioma, Glioblastoma, and Ependimoma. As shown in fig.1.

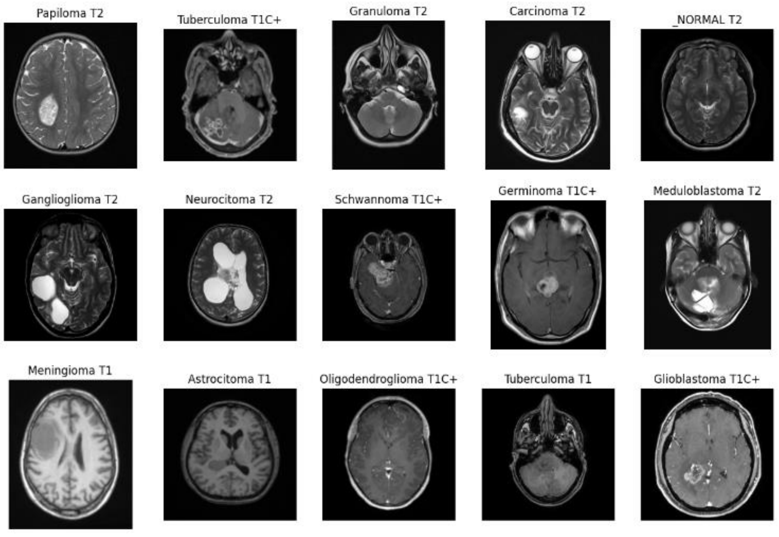


Fig 1: samples of dataset classes

Data Structure

The dataset is structured into 44 directories, each named after a specific tumor type. Within each directory, there are multiple MRI images corresponding to that tumor. This hierarchical organization facilitates straightforward data loading and preprocessing for machine learning tasks.

Anonymization and Labeling

All MRI images have been thoroughly anonymized to remove any patient-specific information, ensuring compliance with privacy standards. The labeling of images into respective tumor categories has been performed by experienced radiologists, providing a reliable ground truth for supervised learning algorithms.

Data Challenges: Class Imbalance

One of the main challenges in this dataset was class imbalance some tumor classes contained hundreds of images, while others had only a few dozen. This imbalance could lead to biased models that perform well only in majority classes, potentially overlooking rare but clinically important tumor types.

To mitigate this, we explored and implemented several balancing techniques:

- Data Augmentation: For underrepresented classes, we applied a range of augmentation techniques including random rotation, zoom, shifts (horizontal/vertical), Brightness and contrast adjustments and Horizontal/vertical flips
- Oversampling: We increased the effective representation of minority classes in training by duplicating or augmenting their data to balance class distribution.
- Class Weights: In our loss functions (especially during classification training), we assigned higher weights to minority classes, penalizing misclassifications more strongly than for majority classes.
- Stratified Splitting: While dividing the dataset into training, validation, and testing sets, we used stratified sampling to preserve class distribution proportions across all subsets.

These strategies collectively helped improve model generalization across all 15 tumor categories.

Image Preprocessing for Model Compatibility

Since pretrained convolutional neural networks like EfficientNet-B5 expect 3-channel RGB images, we developed a custom image generator that converted grayscale MRI images to RGB by duplicating the single channel across all three RGB channels. This ensured compatibility with the input expectations of the classification model.

Additionally, all the images were:

- Resized to the input size expected by the model
- Normalized (pixel values scaled between 0 and 1 or standardized as needed)
- Batch-loaded using a custom data generator to manage memory usage and augmentations on the fly

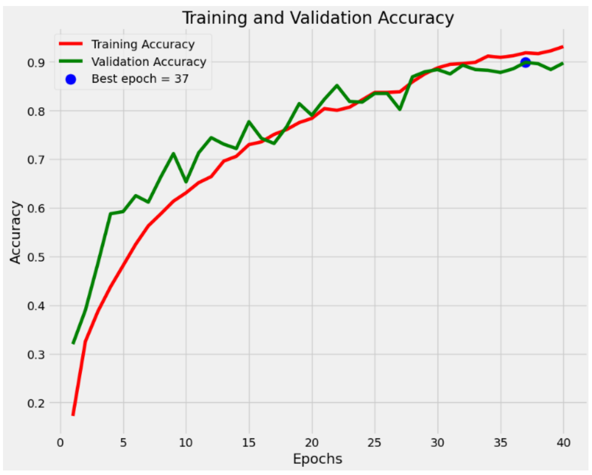
This careful preprocessing, combined with balancing strategies, helped us create a more robust pipeline for training both classification and detection models, even in the presence of data skew.

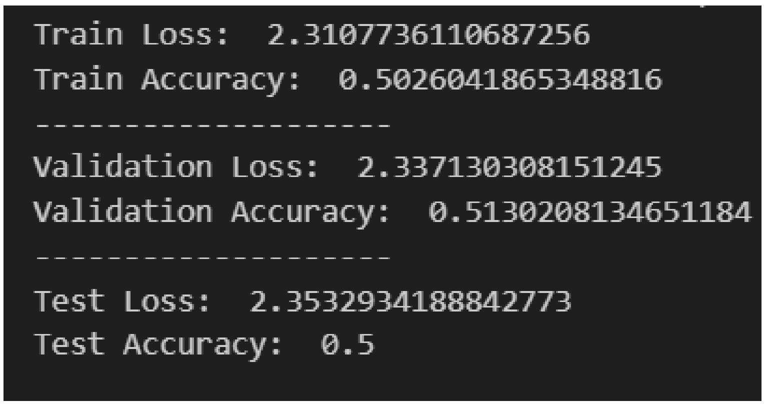
Classification Model Selection and Architecture

For classification, there were many options available—we could either build a convolutional neural network (CNN) from scratch or use a pretrained model. Since pretrained models offer the advantage of transfer learning and often outperform shallow models on small datasets, we chose the second path. However, among pretrained models, there are numerous architectures to choose from, so we had to experiment to determine which one would best suit our task.

We initially started with **ResNet**, a deep residual network known for introducing *residual connections* or *skip connections* that help mitigate the vanishing gradient problem in deep networks. These connections allow the model to learn identity functions and ease the training of very deep architectures.

We tested several ResNet versions, **ResNet50, ResNet50v2,ResNet101**, and **ResNet152**. However, we found that ResNet was not well-suited to our dataset. The earlier versions tended to **underfit** the data as shown in fig.2., while the deeper versions (ResNet101 and ResNet152) were **too complex** for our dataset, which is relatively small in size. This led to **overfitting**, where the model performed well on training data but poorly on validation data as shown in fig 3.





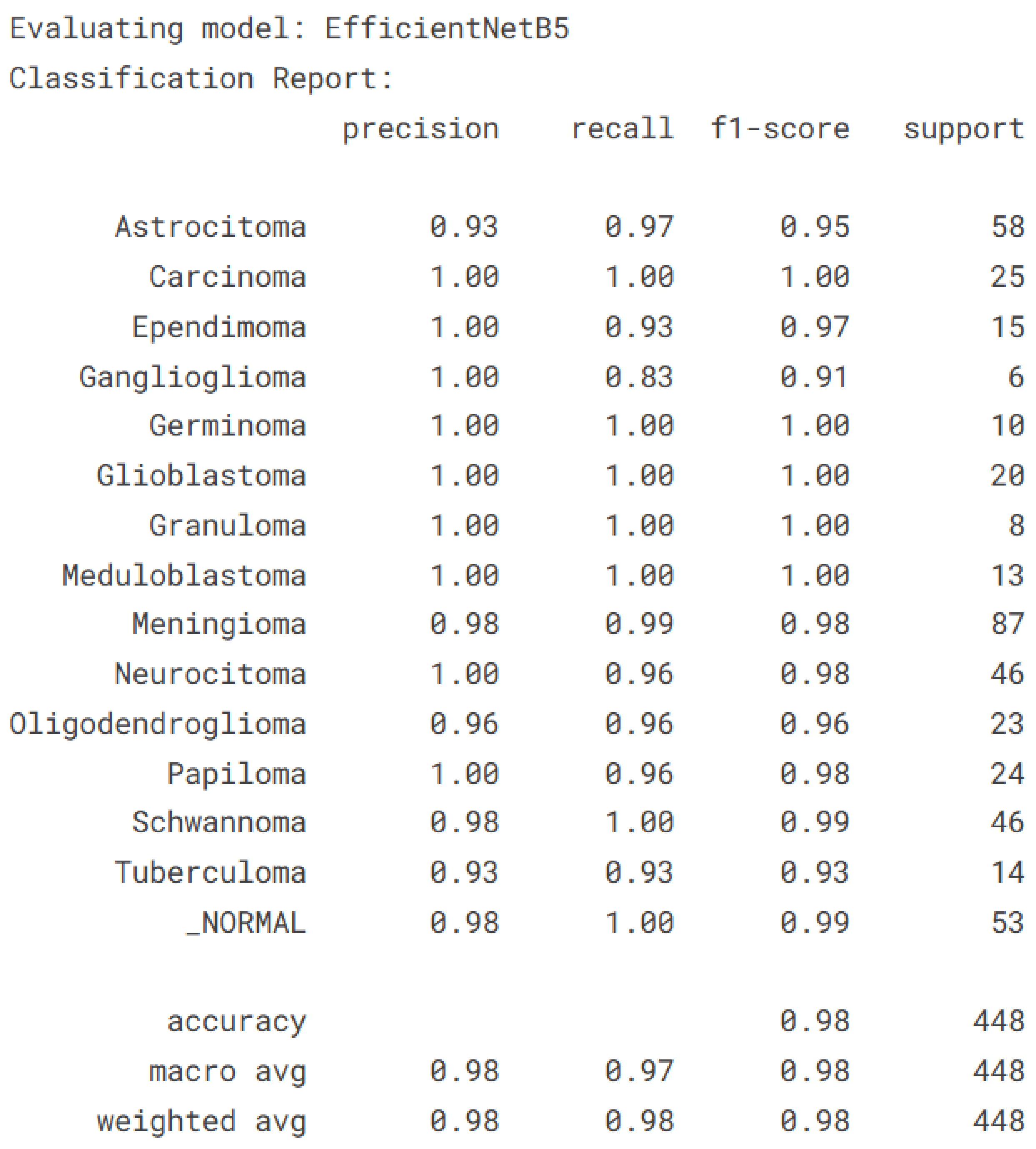
```
Train Loss:  2.3107736110687256
Train Accuracy:  0.5026041865348816
--------------------
Validation Loss:  2.337130308151245
Validation Accuracy:  0.5130208134651184
--------------------
Test Loss:  2.3532934188842773
Test Accuracy:  0.5
```

Fig 3: accuracy over the training of ResNet 50v2

Fig 2: accuracy over the training of ResNet 50

We then experimented with **DenseNet**, which connects each layer to every other layer in a feed-forward fashion. While DenseNet provided better feature reuse and was less prone to vanishing gradients, the performance metrics we achieved using it were not as promising as expected.

Finally, we moved on to **EfficientNet**, a family of models that uniformly scale depth, width, and resolution using a compound coefficient. After testing multiple versions, **EfficientNetB0, B1, B3, B4**, and **B5**, we found that **EfficientNetB5** yielded the most **balanced and robust performance** on our dataset. It offered high accuracy and generalization without significant overfitting, making it the optimal choice for our classification task, as shown in *Fig. 4*.



```
Evaluating model: EfficientNetB5
Classification Report:
                   precision    recall  f1-score   support

      Astrocitoma       0.93      0.97      0.95        58
        Carcinoma       1.00      1.00      1.00        25
       Ependimoma       1.00      0.93      0.97        15
     Ganglioglioma       1.00      0.83      0.91         6
        Germinoma       1.00      1.00      1.00        10
      Glioblastoma       1.00      1.00      1.00        20
        Granuloma       1.00      1.00      1.00         8
    Meduloblastoma       1.00      1.00      1.00        13
       Meningioma       0.98      0.99      0.98        87
      Neurocitoma       1.00      0.96      0.98        46
 Oligodendroglioma       0.96      0.96      0.96        23
         Papiloma       1.00      0.96      0.98        24
       Schwannoma       0.98      1.00      0.99        46
      Tuberculoma       0.93      0.93      0.93        14
          _NORMAL       0.98      1.00      0.99        53

         accuracy                           0.98       448
        macro avg       0.98      0.97      0.98       448
     weighted avg       0.98      0.98      0.98       448
```

Fig 4: accuracy over the training of EffiecentNet

Classification Model Structure

The core model structure was composed as shown in fig.5 of the following components:

Base Model: EfficientNetB5 (without top layer, with global max pooling)

Custom Head:

- Flatten() layer to reduce spatial dimensions.
- BatchNormalization() to stabilize and speed up training.
- A fully connected Dense(256, activation='relu') layer.
- Dropout(rate=0.45) to reduce overfitting.
- Final Dense(num_classes, activation='softmax') layer for multiclass classification.

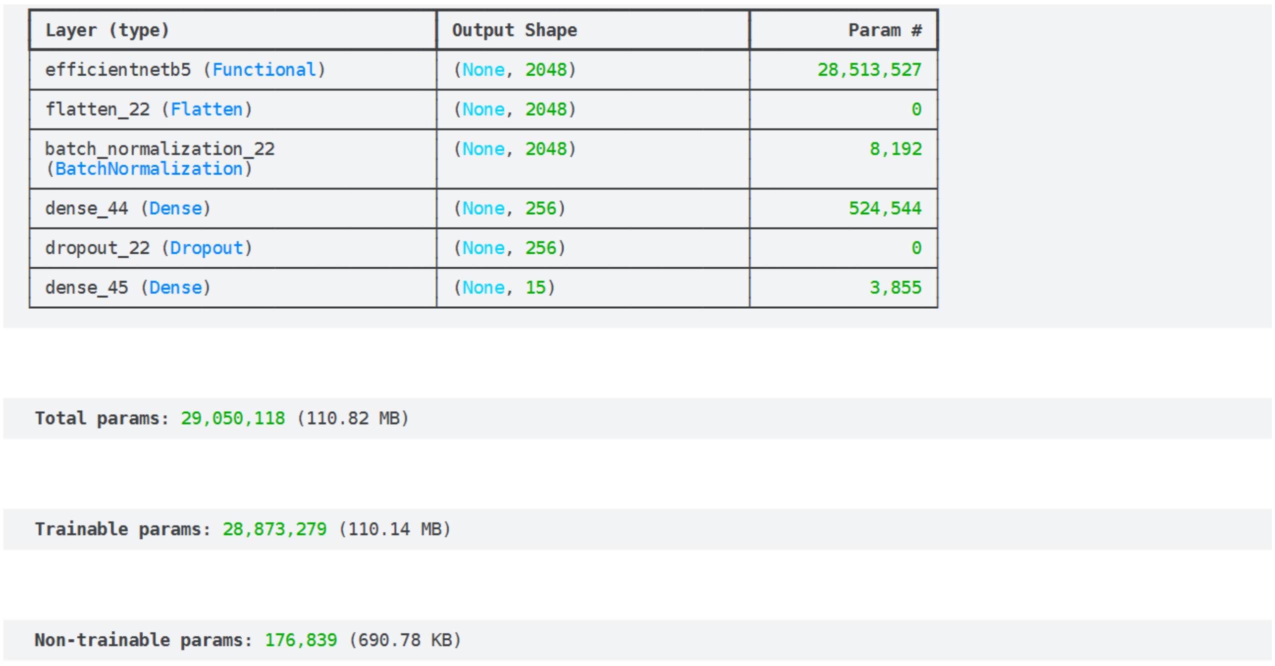
The model was compiled with the following settings:

- **Loss Function**: categorical_crossentropy – suitable for multiclass classification problems.
- **Optimizer**: Adam with a learning rate of 0.001 – a widely used optimizer that adapts learning rates during training.
- **Evaluation Metric**: Accuracy was tracked during both training and validation phases.

We trained the model for up to 40 epochs with a batch size of 32, though early stopping mechanisms were implemented through a custom callback class (MyCallback). This callback introduced a set of advanced training controls:

- **Patience-based Early Stopping**: Training would halt if no improvement was seen over multiple epochs.
- **Learning Rate Adjustment**: The learning rate was halved if the model performance plateaued, based on a defined threshold of 0.9.
- **Interactive Interruption**: The training process prompted for user input every few epochs (ask_epoch=5) to optionally stop training manually.

All models were trained on RGB images of shape (224, 224, 3), resized accordingly to meet the input requirements of pretrained models.



Layer (type)	Output Shape	Param #
efficientnetb5 (Functional)	(None, 2048)	28,513,527
flatten_22 (Flatten)	(None, 2048)	0
batch_normalization_22 (BatchNormalization)	(None, 2048)	8,192
dense_44 (Dense)	(None, 256)	524,544
dropout_22 (Dropout)	(None, 256)	0
dense_45 (Dense)	(None, 15)	3,855

```
Total params: 29,050,118 (110.82 MB)

Trainable params: 28,873,279 (110.14 MB)

Non-trainable params: 176,839 (690.78 KB)
```

Fig 5: classification model structure

Model Challenges: Unstable Accuracy Spikes and Partial Overfitting

During the classification model training phase, we observed an unusual pattern: both training and validation accuracy would spike significantly at a certain epoch, reaching unexpectedly high values, only to drop back to baseline levels and stabilize for the remaining epochs. This inconsistent behavior suggested a form of partial overfitting, where the model temporarily memorized certain patterns in the training data without truly learning generalized features.

This instability made it difficult to rely on traditional early stopping or fixed learning rate schedules, as the performance spikes were misleading and not reflective of real progress.

Solution

To address this, we developed a custom callback mechanism (MyCallback) that provided fine-grained control over the training loop:

- It monitored accuracy, loss, and validation behavior at each epoch and within batches.
- Implemented early stopping and learning rate adjustment based on real thresholds rather than just accuracy curves.
- Included logic to ask for user input at certain checkpoints (epochs) and allowed controlled intervention to halt or continue training.

This approach helped in stabilizing training, reducing the risk of overfitting, and maintaining model generalization throughout the epoch.

```
Evaluating model: EfficientNetB5
--------------------------------------------------
```

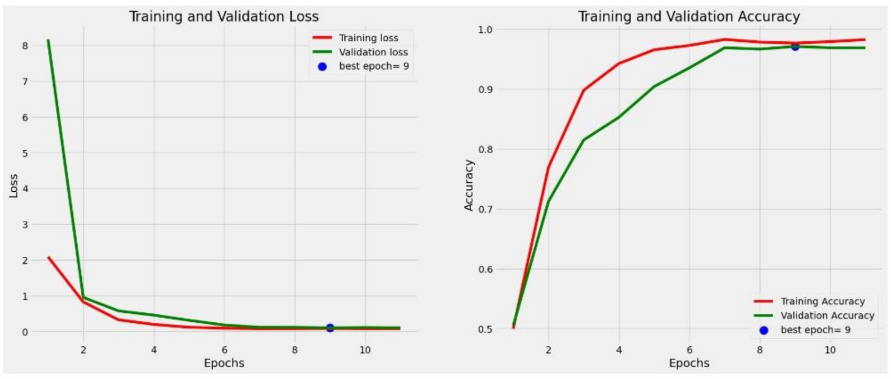


Fig 6: After Custom Callback Modification

Object Detection Model

In addition to classifying brain tumor types, we aimed to localize tumor regions within MRI scans. For this purpose, we implemented an object detection pipeline using **YOLOv8** (You Only Look Once – version 8), a state-of-the-art real-time object detector developed by Ultralytics. YOLOv8 is known for its balance between speed and accuracy, making it suitable for both academic and production-level medical imaging tasks.

YOLOv8 was selected after experimenting with other detection approaches due to its:

- Modern anchor-free architecture, which simplifies bounding box prediction.
- High performance on small objects, crucial for detecting tumors of varying sizes.
- Ease of integration, with robust support from the Ultralytics ecosystem.

Training and Evaluation
The model was trained on manually annotated tumor regions using bounding boxes. Images were converted to RGB and resized as required. The dataset was split into training and validation sets while preserving class distributions.

To evaluate the performance of our detection model, we used standard object detection metrics as shown in fig 6, with the following results:

- **Precision**: 91.24% — indicating that the majority of predicted tumor regions were correct.
- **Recall**: 89.63% — demonstrating the model's effectiveness in identifying true tumor regions.
- **mAP@0.5**: 96.27% — high accuracy of bounding box predictions at a standard IoU threshold.
- **mAP@0.5:0.95**: 71.39% — strong performance across stricter IoU thresholds.
- **Fitness Score**: 73.88% — a composite indicator of overall model robustness.

These results confirm that YOLOv8 performed reliably in localizing tumors across the diverse 15 tumor categories derived from the original 44-class dataset, making it a strong candidate for real-world deployment in clinical or research settings.
Visual results with bounding boxes and class confidence were generated and analyzed. Finally, the trained model was exported in ONNX format for deployment flexibility, and the best.pt weights were delivered to the deployment phase for integration into the target system.

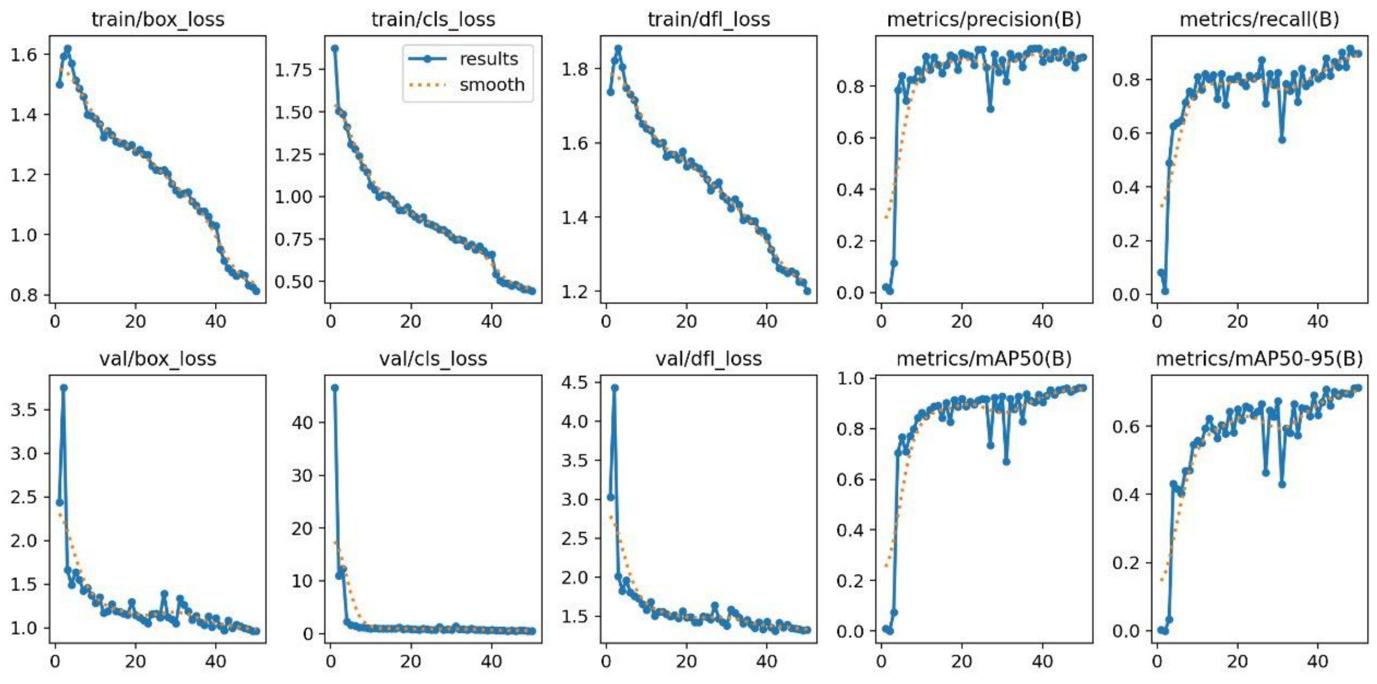


Fig 7: standard object detection metrics

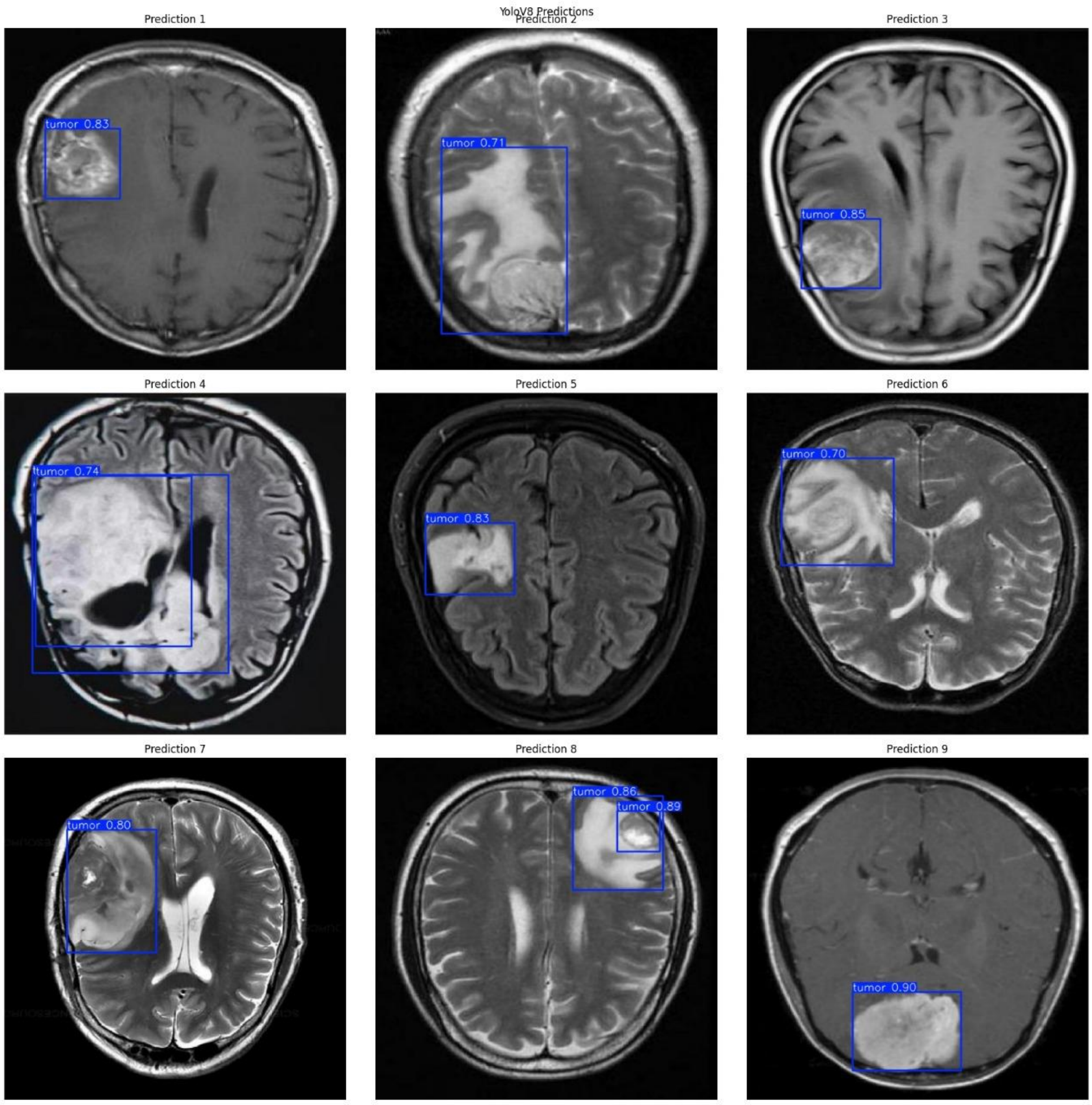


Fig 8: samples of detecting cancer in human brain MRI

Deployment

This Python script is a Flask web application that provides a user interface for classifying brain tumor images using a pre-trained deep learning model based on EfficientNetB5. The app loads the model, processes uploaded images, performs predictions, and returns the tumor type along with the prediction probability. Users can upload images via a web interface, where the backend resizes and preprocesses the images before running them through the model. The prediction results are then displayed on a results page along with the uploaded image, which is encoded in base64 for embedding in the HTML.

Steps of the Brain Tumor Classification Flask App

1. Import Required Libraries:

Imports libraries such as Flask for the web framework, TensorFlow for loading the model, PIL for image handling, and others like NumPy, io, and base64 for data processing.

2. Initialize the Flask App:

Creates a Flask application instance using app = Flask(_name_).

3. Load the Pre-trained Model:

Loads a pre-trained EfficientNetB5 model from a .h5 file which is trained to classify different types of brain tumors.

4. Define Class Labels:

A list of class names (brain tumor types), such as Glioblastoma, Meningioma, Astrocitoma, and _NORMAL for healthy images.

5. Preprocess Uploaded Image:

Resizes the image to 224x224 pixels.

Converts the image into a tensor (numerical array) suitable for the neural network.

6. Classify the Image:

Passes the processed image to the model to get predictions.

Retrieves the predicted class label with the highest probability.

Converts the probability to a float for display.

7. Define Web Routes:

/ Route: Renders the home page (HTML form for image upload).

/classify Route:

Receives the uploaded image.

Convert it to base64 format for display in HTML.

Calls the classification function and renders the result (predicted label and probability) on a new page.

8. Run the Application:

The app runs on host 0.0.0.0 and port 7860, making it accessible on the network.

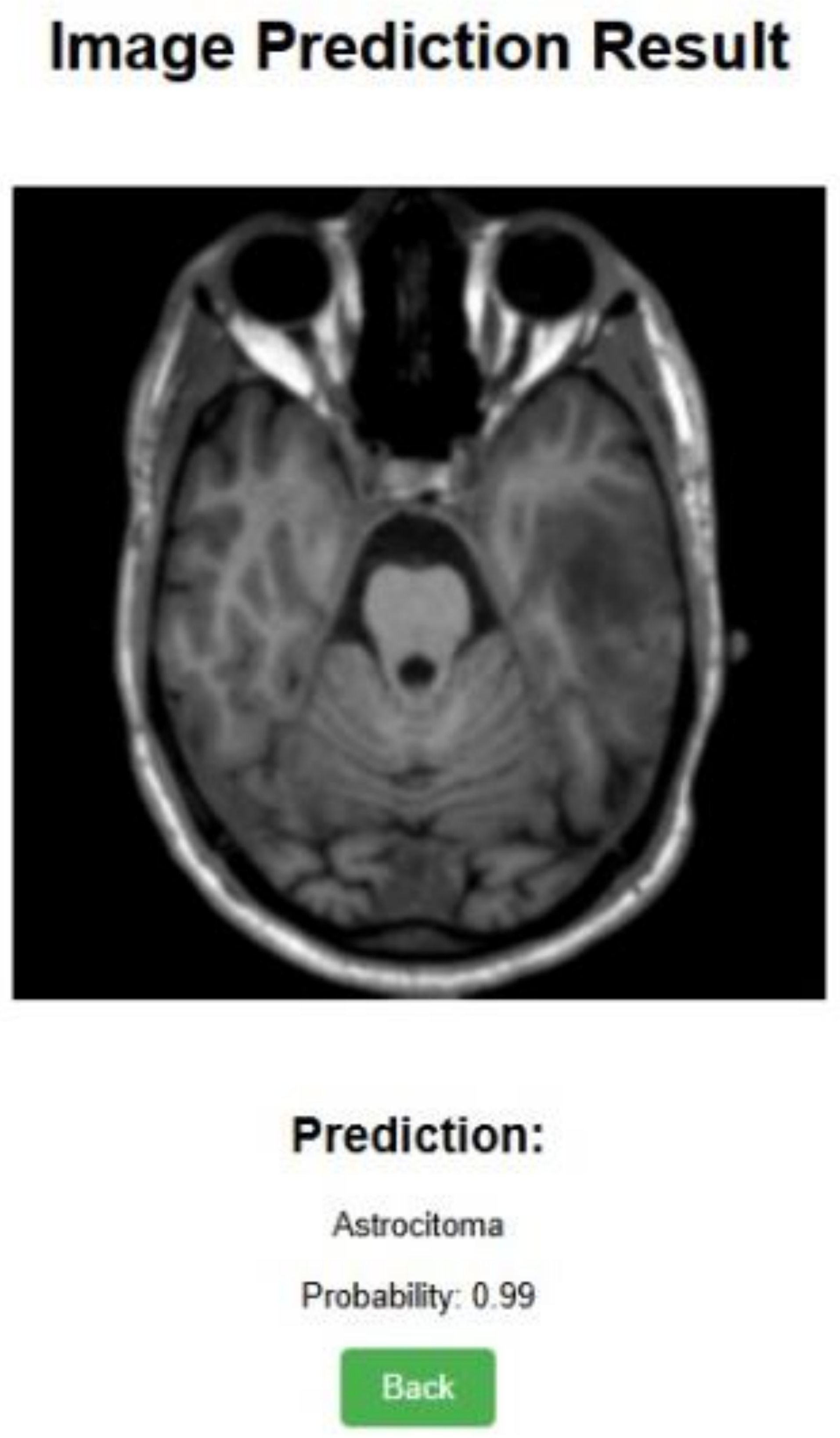


Image Prediction Result

Prediction:

Astrocitoma

Probability: 0.99

After evaluating several architectures, **EfficientNetB5** was selected for classification due to its strong performance and generalization capabilities. For tumor localization, **YOLOv8** was implemented and achieved high evaluation metrics, including strong precision and recall scores, as well as excellent mAP values. Together, these models provided a robust solution for both identifying and localizing brain tumors.

Key strategies such as custom callbacks, data augmentation, and tailored evaluation metrics ensured the model was both accurate and stable. This project not only demonstrated the effectiveness of combining classification and detection tasks in medical imaging but also highlighted the importance of carefully handling preprocessing, model selection, and training strategy when working with real-world healthcare data.

Future work may focus on expanding the dataset, automating annotations for detection, and optimizing deployment in clinical environments.

Work closely with radiologists and oncologists to validate predictions against expert annotations, ensuring the model's applicability in real-world settings.

References

1. Kaggle. (n.d.). *Brain tumor MRI images (44 classes)*. Retrieved from https://www.kaggle.com/datasets/fernando2rad/brain-tumor-mri-images-44c
2. TensorFlow. (n.d.). *TensorFlow: An end-to-end open-source machine learning platform*. Retrieved from https://www.tensorflow.org/
3. Ultralytics. (n.d.). *YOLOv8 Documentation*. Retrieved from https://docs.ultralytics.com/
4. Tan, M., & Le, Q. V. (2019). *EfficientNet: Rethinking model scaling for convolutional neural networks*. *arXiv preprint* arXiv:1905.11946. https://arxiv.org/abs/1905.11946
5. He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep residual learning for image recognition*. *arXiv preprint* arXiv:1512.03385. https://arxiv.org/abs/1512.03385
6. Chai, Y., Zhang, Y., Xie, J., & Zhang, J. (2024). *A high-performance deep learning pipeline for brain tumor classification and localization*. *npj Precision Oncology, 8*, Article 41. https://www.nature.com/articles/s41698-024-00575-0
7. Keras. (n.d.). *Keras Applications*. Retrieved from https://keras.io/api/applications/