# Learning Module

## Workshop

"Deep Learning and Its Applications in Assisting Human"

2024

# Contents

# CHAPTER 1
# Introduction

## 1.1 OVERVIEW OF MEDICAL IMAGING AND BRAIN TUMOR DETECTION

Medical imaging plays a crucial role in diagnosing and treating brain tumors. Early detection and accurate localization of brain tumors are critical for improving patient outcomes. With advances in deep learning and computer vision, automatic detection and segmentation of brain tumors from medical images have become increasingly feasible.

### 1.1.1 Object Detection in Medical Imaging

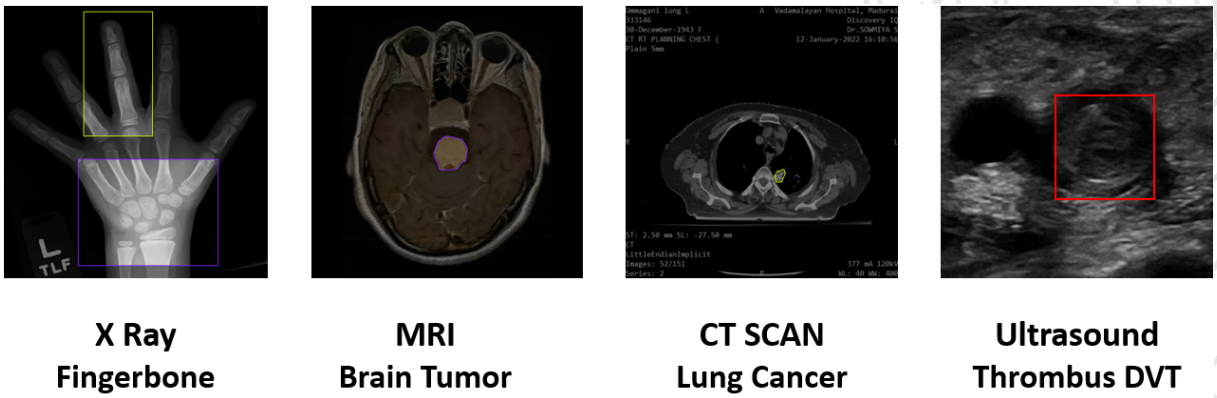



Figure 1.1: Medical Image

Medical imaging plays a crucial role in diagnosing and treating brain tumors. Early detection and accurate localization of brain tumors are critical for improving patient outcomes. With advances in deep learning and computer vision, automatic detection and segmentation of brain tumors from medical images have become increasingly feasible. Object detection can be applied in various types of medical imaging, such as MRI, CT scans, X-rays, and ultrasound, each serving specific diagnostic purposes. MRI is effective for viewing soft tissues like the brain and internal organs, CT scans provide detailed images of organs and bones, X-rays are used to examine hard structures like bones, and ultrasound is useful for visualizing internal organs and blood flow. By detecting objects in these different types of medical images, algorithms can help identify anomalies such as tumors or vascular blockages, supporting more accurate diagnosis and treatment planning. Together, these

technologies enhance the precision and efficiency of brain tumor detection and other critical diagnostic processes in medical imaging.

### 1.1.2 Importance of Accurate Brain Tumor Detection

Accurate brain tumor detection is essential for several reasons:

- **Early Diagnosis:** Detecting tumors at an early stage can significantly improve the chances of successful treatment.
- **Treatment Planning:** Accurate detection helps in planning surgery, radiation therapy, and other treatment modalities.
- **Reduction of Human Error:** Automated detection reduces the likelihood of human error, particularly in cases where tumors are small or obscured by surrounding tissues.

### 1.1.3 Challenges in Medical Imaging

While medical imaging techniques like MRI (Magnetic Resonance Imaging) provide detailed images of the brain, there are several challenges in automating brain tumor detection:

- **Variability in Tumor Appearance:** Tumors can vary significantly in size, shape, and texture, making it difficult to create a one-size-fits-all detection model.
- **Class Imbalance:** The occurrence of tumors in medical datasets is relatively rare, leading to class imbalance issues.
- **Noise and Artifacts:** Medical images often contain noise and artifacts that can obscure the features of the tumor.

## 1.2 INTRODUCTION TO YOLOV8

### 1.2.1 What is YOLOv8?

YOLOv8 (You Only Look Once, version 8) is a state-of-the-art object detection model that builds upon the success of earlier YOLO models. It is designed to balance speed and accuracy, making it suitable for real-time applications. YOLOv8 uses a single neural network to predict bounding boxes and class probabilities simultaneously, which allows it to process images more efficiently than many other models.

### 1.2.2 Why Choose YOLOv8 for Brain Tumor Detection?

YOLOv8 is particularly well-suited for brain tumor detection due to the following reasons:

- **Real-time Detection:** YOLOv8's architecture allows for fast inference, making it ideal for real-time applications, such as in medical imaging workflows where quick decision-making is essential.
- **High Accuracy:** Despite its speed, YOLOv8 maintains high detection accuracy, which is crucial for detecting small and irregularly shaped brain tumors.
- **Flexibility:** YOLOv8 can be fine-tuned for specific medical datasets, enabling it to adapt to the unique characteristics of medical images.

CHAPTER 2

# Fundamentals of Object Detection

## 2.1 UNDERSTANDING OBJECT DETECTION

Object detection is a crucial computer vision task that involves identifying and localizing objects within an image. This process provides both class labels and spatial coordinates (bounding boxes) for each detected object.

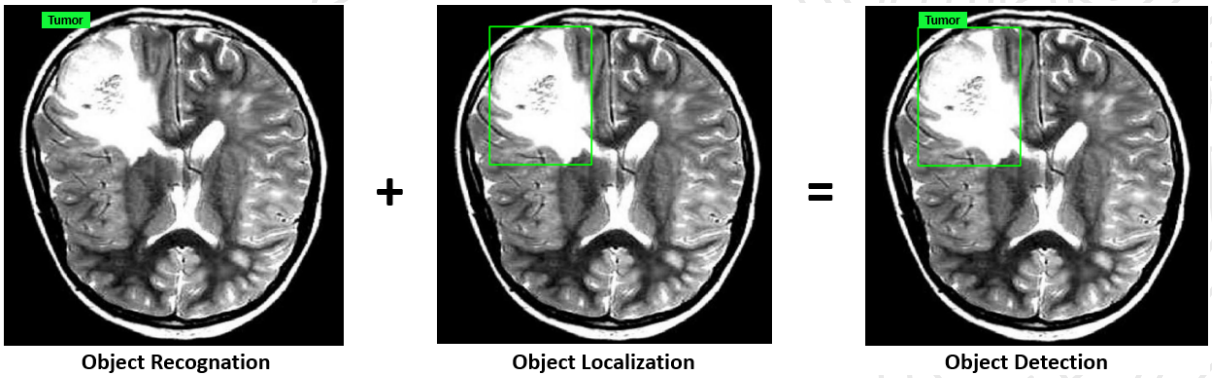



Figure 2.1: The View of roboflow

- **Classification vs. Detection:**
  - **Classification:** Predicts a single label describing the entire image. Output: one class label.
  - **Detection:** Predicts multiple labels with their locations, including class labels, bounding box coordinates, and confidence scores for each object.
- **Object Detection Task:**
  - Combines classification and localization to identify objects and determine their positions within an image.
  - Provides both the type of object (e.g., ”organ,” ”tumor,” ”anomaly”) and the location of each object, marked by bounding boxes.
- **Applications:** Essential for autonomous driving, medical imaging, and security systems, as it enables recognition of object types and precise spatial positions, supporting various real-world applications.

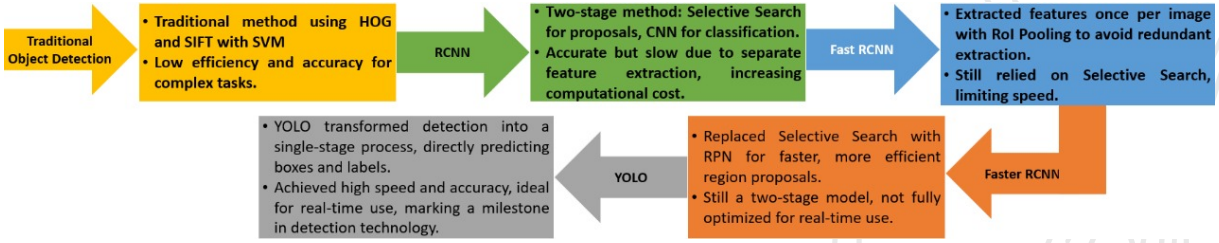



Figure 2.2: The Journey of Object Detection

### 2.1.1 The Journey of Object Detection

Since traditional object detection methods, such as HOG (Histogram of Oriented Gradients) and SIFT (Scale-Invariant Feature Transform) combined with classifiers like SVM (Support Vector Machine), lacked efficiency and accuracy for complex tasks, the introduction of R-CNN marked a significant evolution in object detection technology. R-CNN used a two-stage approach with Selective Search to generate region proposals and CNN for classification; however, this process was very slow. Fast R-CNN improved efficiency by extracting image features only once and using Region of Interest (RoI) Pooling, but it still relied on Selective Search. Faster R-CNN later introduced the Region Proposal Network (RPN), which replaced Selective Search, making object detection significantly faster and more efficient. Nonetheless, Faster R-CNN still operated in two stages and was less suited for real-time applications. YOLO (You Only Look Once) transformed object detection into a single-stage process that directly predicts bounding boxes and class labels simultaneously, making it very fast and ideal for real-time applications. YOLO's efficient approach has made it a top choice for applications requiring high speed without sacrificing accuracy.

### 2.1.2 Two-Stage Detection VS One-Stage Detection

Two-Stage Detection and One-Stage Detection methods are approaches focused on artificial neural network architectures specifically designed for object detection tasks in computer vision. These methods are based on different approaches for integrating object detection and classification processes, according to the number of stages required to recognize objects in an image.

#### Two-Stage Detection

- **Based on Region Proposal Architecture**
  The *Two-Stage Detection* method is based on the idea that object detection can achieve higher accuracy by dividing it into two stages: first, generating a number of candidate regions that most likely contain objects, and second, classifying and refining the location of each region. This algorithm uses a *Region Proposal Network* (RPN) or other region proposal methods.

- **Approach That Separates Detection and Classification**
  *Two-Stage Detection* models, such as *Faster R-CNN*, are designed to prioritize accuracy since each candidate region is analyzed more thoroughly. By separating detection and classification processes into two distinct stages, this method focuses only on relevant regions, resulting in better accuracy.

**One-Stage Detection**

- **Based on Single Pass Detection Architecture**
  The *One-Stage Detection* method uses a direct, single-pass object detection approach. Algorithms like *YOLO* (*You Only Look Once*) and *SSD* (*Single Shot MultiBox Detector*) perform detection and classification simultaneously across the entire image without dividing the process into two stages.

- **Grid-Based Prediction Approach**
  In *One-Stage Detection*, the architecture divides the image into a grid, with each cell in the grid predicting the location and class of the detected object. This approach enables faster processing, as the entire image is processed in a single calculation.

## 2.2 KEY CONCEPTS

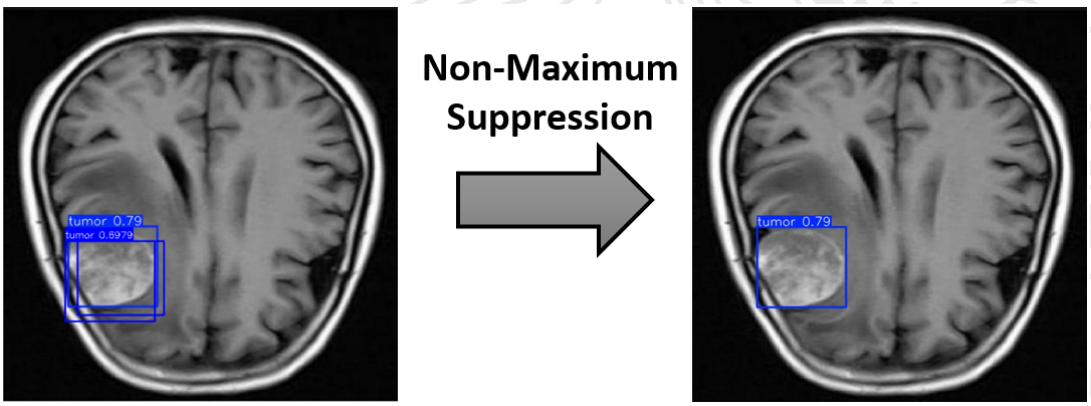



Figure 2.3: The View of roboflow

### 2.2.1 Bounding Boxes

- **Definition:** A bounding box is a rectangle used to enclose an object within an image.

- **Purpose:** Explain how bounding boxes allow the model to localize objects within an image.

- **Diagram:** Include a diagram showing an object within a bounding box and illustrate its coordinates.

In YOLO (You Only Look Once), a **bounding box parameter file label** is a file that contains information about the *position, size*, and *label* of objects detected in an image or video. Each detected object is marked with a **bounding box**, which is a box that surrounds the object. In the label file for YOLO, each line usually represents one bounding box with the following format:

1. **Class ID**: The index or ID of the object class (e.g., 0 for " Metacarpal," 1 for "Proximal Phalanx").

2. **x_center**: The x-coordinate of the center of the bounding box (as a ratio of the image width, usually between 0 and 1).
3. **y_center**: The y-coordinate of the center of the bounding box (as a ratio of the image height, usually between 0 and 1).
4. **width**: The width of the bounding box (as a ratio of the image width).
5. **height**: The height of the bounding box (as a ratio of the image height).

For example:

1. Bounding Box on Hand Bones and Joints


Figure 2.4: Bounding Box on Hand Bones and Joints

- The bounding boxes help clearly mark each specific bone or joint area.
- **X-ray of the hand** with **bounding boxes** around each joint and bone.
- Each bone or joint is labeled with a specific color:
  - **MC** = Metacarpal (bones in the palm of the hand).
  - **PP** = Proximal Phalanx (first segment of the fingers).
  - **MP** = Middle Phalanx (middle segment of the fingers).
  - **DP** = Distal Phalanx (tip segment of the fingers).

2. Bounding Box Coordinates on the Carpal Bone

- Shows **bounding box coordinates** for the ”Carpal” bone.
- **X** and **Y**: Center position of the bounding box within the image.
- **W (Width)** and **H (Height)**: Width and height of the bounding box.
- This information indicates the location and size of the bounding box, making it easier to identify the Carpal bone area in the X-ray.

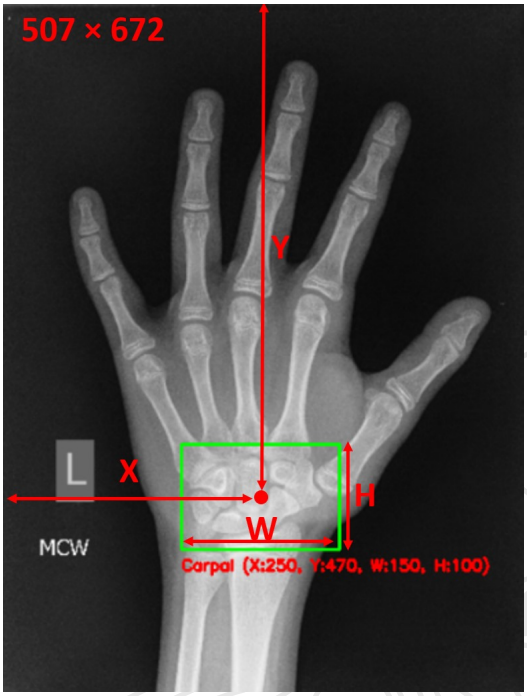



Figure 2.5: Bounding Box Coordinates on the Carpal Bone

### 2.2.2 Confidence Scores

- **Definition:** Confidence scores indicate the model's confidence level in its prediction of an object's class.
- **Purpose:** Discuss the significance of high-confidence scores and how they reflect the model's certainty.
- **Diagram:** Display an image with multiple objects, each labeled with its confidence score.

### 2.2.3 Non-Maximum Suppression (NMS)

- **Definition:** NMS is a technique to eliminate redundant bounding boxes for the same object.
- **Purpose:** Explain how NMS retains the box with the highest confidence score and suppresses others.
- **Diagram:** Show a before-and-after example of NMS, where multiple overlapping boxes reduce to a single box.

## 2.3 THE YOLO FRAMEWORK FOR OBJECT DETECTION

### 2.3.1 YOLO Grid System

- **Definition:** NMS is a technique to eliminate redundant bounding boxes for the same object.

- **Purpose:** Each grid cell predicts the presence of an object.
- **Diagram:** Include an example of an image split into a grid, with cells highlighting detected objects.

### 2.3.2 Predictions per Grid Cell

- **Explanation:** Each cell predicts bounding boxes, confidence scores, and class probabilities.
- **Diagram:** A sample grid cell illustration showing predictions for object class, bounding box coordinates, and confidence scores.

### 2.3.3 Intersection over Union (IoU)

- **Explanation:** IoU measures the overlap between predicted and actual bounding boxes.
- **Purpose:** Explain how IoU is used to evaluate model accuracy.
- **Formula and Diagram:** Show the IoU formula and an illustration of overlapping bounding boxes.

### 2.3.4 Anchor Boxes

- **Explanation:** Anchor boxes help YOLO handle different object shapes and sizes within a grid cell.
- **Diagram:** Include examples of different anchor boxes on an image and how they fit various object shapes.

### 2.3.5 Step-by-Step Object Detection Process with YOLO

- **Step 1:** Image input and resizing.
- **Step 2:** Grid division and prediction for each cell.
- **Step 3:** Applying NMS to finalize object predictions.
- **Illustration:** A flowchart depicting the YOLO process from input to final object detection.

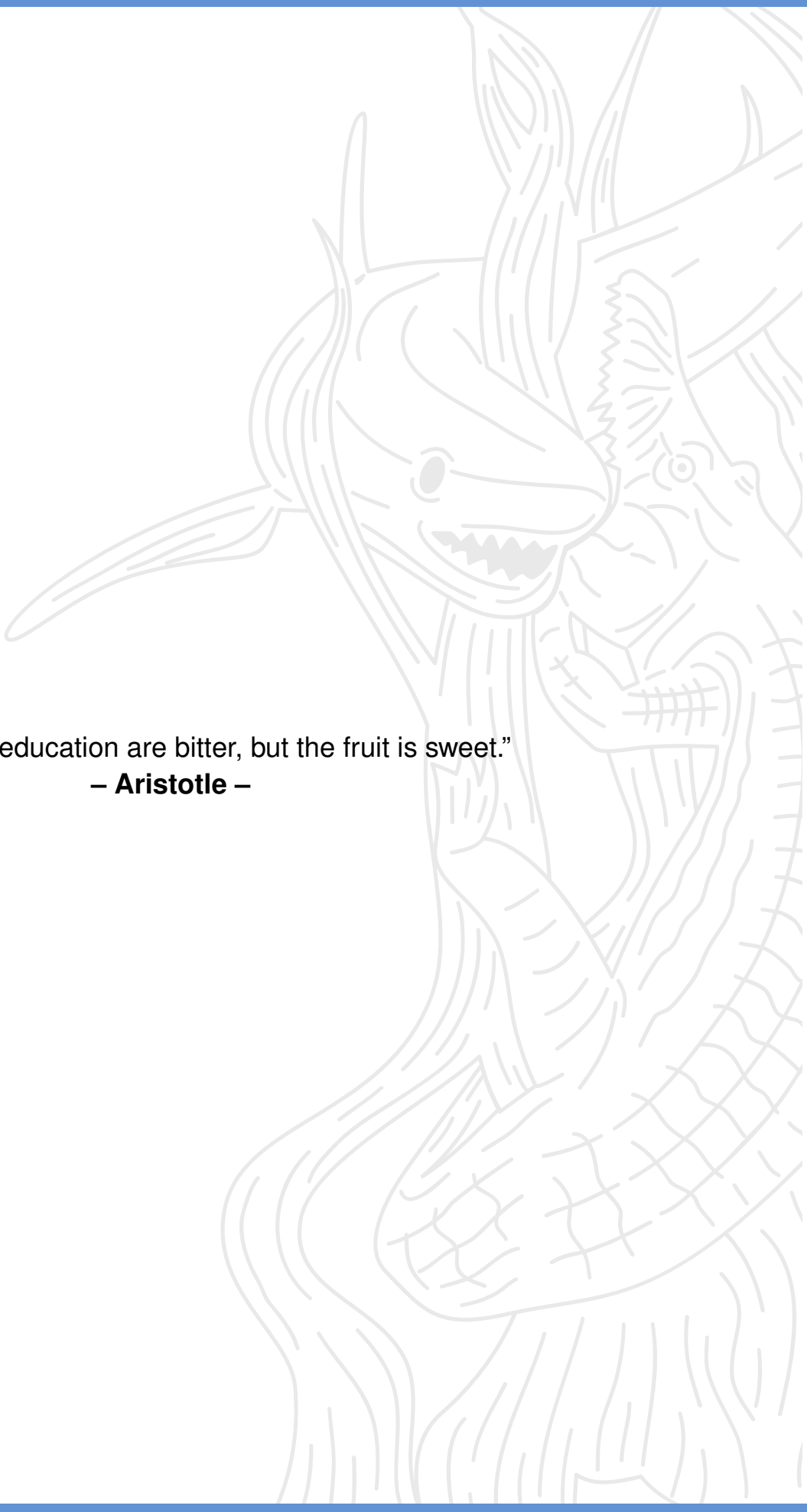
### 2.3.6 Step-by-Step Object Detection Process with YOLO

- **Step 1:** Image input and resizing.
- **Step 2:** Grid division and prediction for each cell.
- **Step 3:** Applying NMS to finalize object predictions.
- **Illustration:** A flowchart depicting the YOLO process from input to final object detection.

## 2.4 KEY ADVANCEMENTS IN YOLOV8

YOLOv8 introduces several new features and improvements:

- **Anchor-Free Detection:** YOLOv8 moves away from anchor-based predictions, simplifying the process and reducing the number of hyperparameters.
- **Improved Speed and Accuracy:** YOLOv8 enhances both speed and accuracy, making it highly suitable for real-time applications without compromising performance.
- **Better Generalization:** YOLOv8 is designed to generalize better across a variety of tasks and datasets, making it more robust for use cases in diverse environments, such as medical image detection.



“The roots of education are bitter, but the fruit is sweet.”
**– Aristotle –**

# CHAPTER 3
# Brain Tumor Detection Using YOLOv8

## 3.1 INTRODUCTION

Medical imaging and machine learning have revolutionized the diagnosis of conditions such as brain tumors. **YOLOv8** (You Only Look Once version 8) is a robust model for object detection that we will use to detect brain tumors from MRI or CT scan images. This chapter will guide you through setting up the model, training it, and preparing it for deployment.

## 3.2 SETTING UP THE ENVIRONMENT

1. **GPU Verification, Code** click here

```
!nvidia-smi
```

2. **Directory Check, Code** click here

```
import os
HOME = os.getcwd()
print(HOME)
```

3. **Installing ultralytics and Roboflow Code** click here

```
!pip install ultralytics==8.2.103 -q
!pip install roboflow==1.1.48 --quiet
```

4. **Verification of Installation Code** click here

```
import ultralytics
ultralytics.checks()
```

## 3.3 PREPARING A CUSTOM DATASET

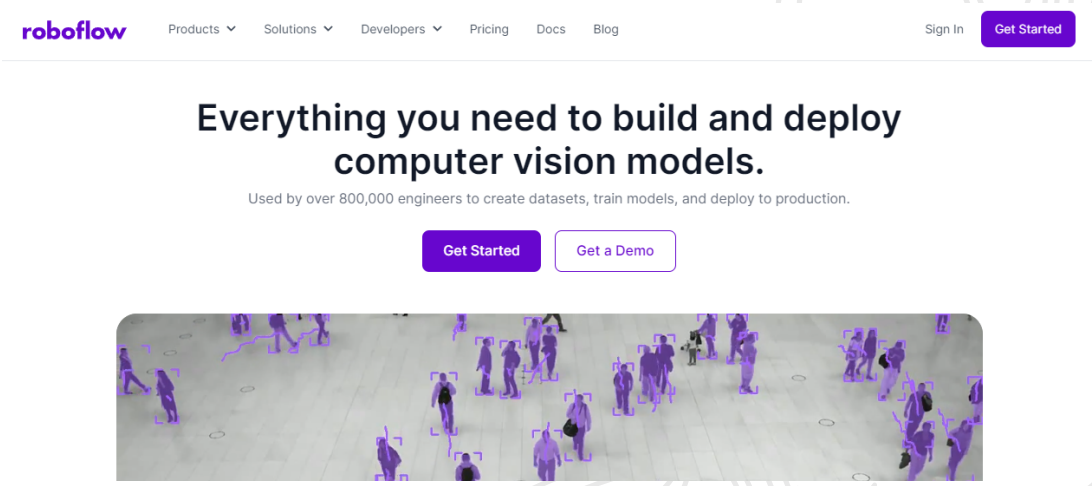



Figure 3.1: The View of roboflow

Building a custom dataset can be a time-consuming and labor-intensive process. Collecting images, accurately labelling them, and exporting them in the correct format for model training can take hours, or even hundreds of hours, to complete. Each step, from data collection to labelling and formatting, requires precision to ensure a high-quality dataset that is ready for training. Fortunately, tools like **Roboflow** are available to streamline and accelerate this process, making it more structured and efficient, thus reducing the significant workload involved. This section provides a step-by-step guide to preparing a custom dataset using Roboflow, specifically for brain tumor detection using the YOLOv8 model. This guide will help you understand each stage in preparing an optimal dataset, ultimately enhancing the quality of the model's detection results.

- **Step 1: Creating a Project : Make account → Dasborad → New Project**

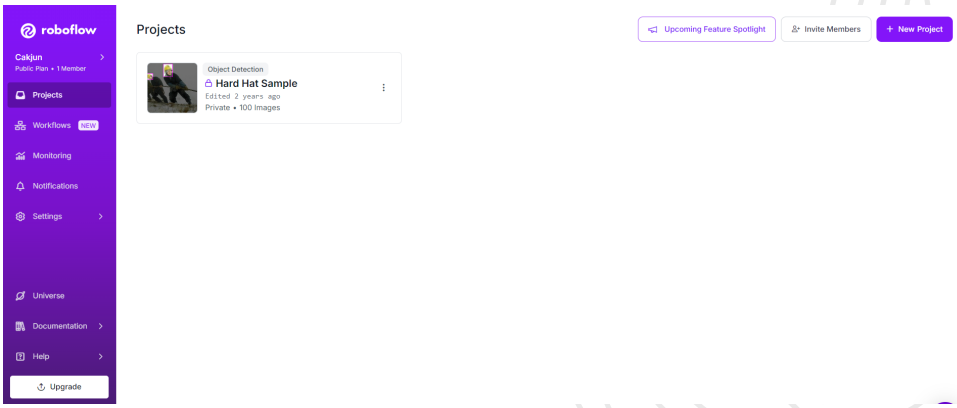



Figure 3.2: Main Menu View of Roboflow

When creating the project, make sure to select the appropriate **Project Type**. Since our task is brain tumor detection using YOLOv8, we will choose **Object Detection** as the project type. Object detection projects allow for the identification and localization of tumors within medical images, which is crucial for this tutorial.

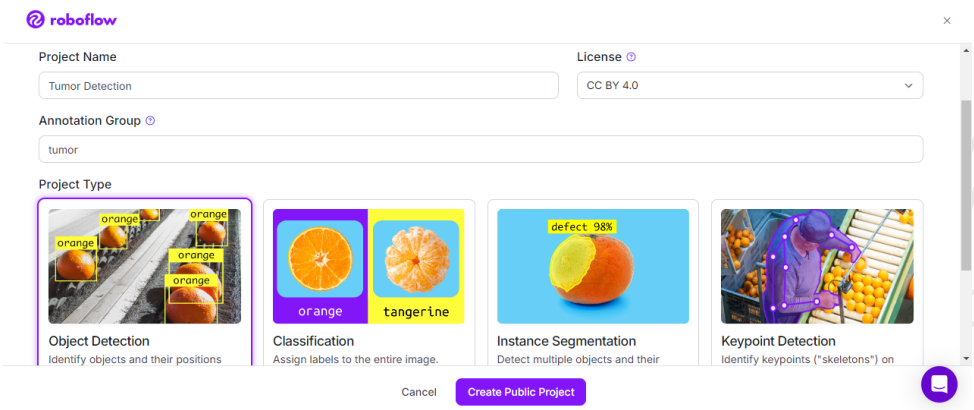



Figure 3.3: Create Project in Roboflow

- **Step 2: Uploading Images**

Once the project is created, you need to upload the data (i.e., images) to your project. **Klik Upload data → Select file or folder image → save and continue**

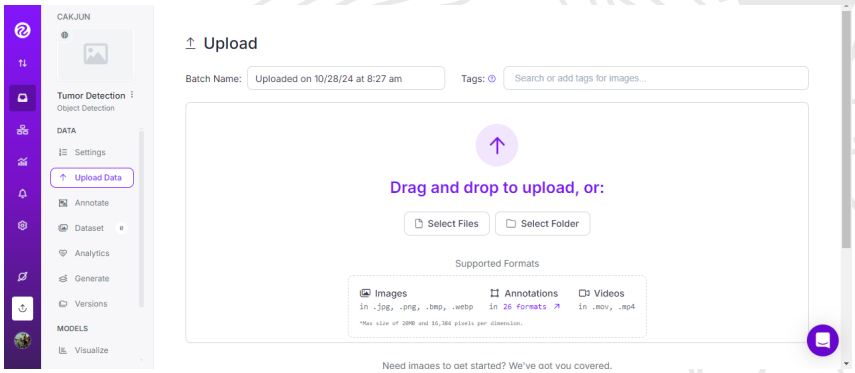



Figure 3.4: Upload Images in Roboflow

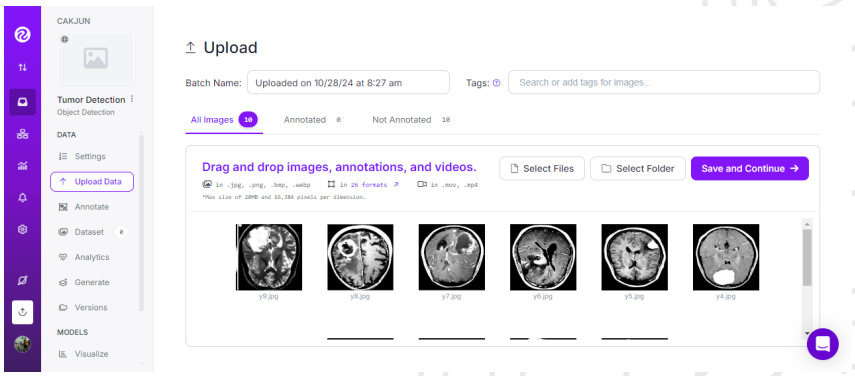



Figure 3.5: Example Images Upload in Roboflow

- **Step 3: labelling Images**

Labeling in Roboflow is a streamlined process where you can draw bounding boxes around regions of interest and then you can use manual labeling.
Roboflow labelling provides an easy-to-use interface, enabling you to manually labelling any number of images for the object detection model.

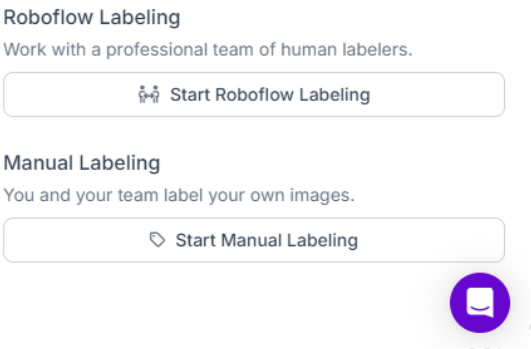



Figure 3.6: Image Labelling Method Selection in Roboflow

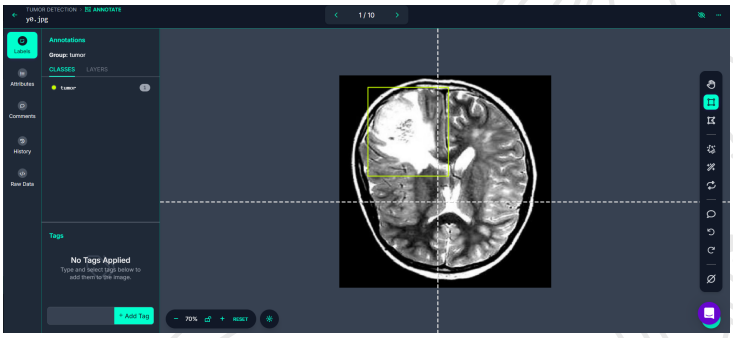


Figure 3.7: Labelling Using Bounding Box Tool

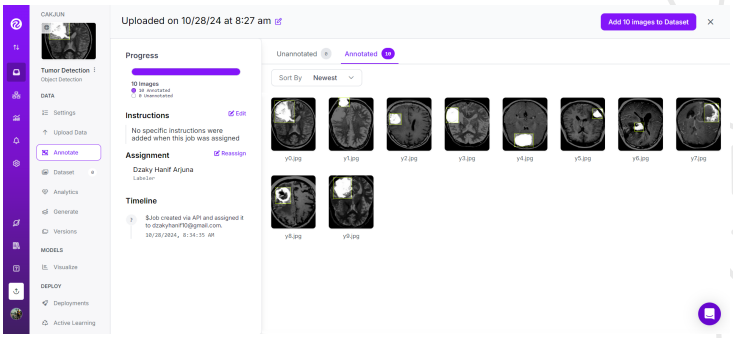



Figure 3.8: Image After Labelling

- **Step 4: Generating a New Dataset Version**

Once your images and annotations are prepared, you can **Generate a Dataset Version**. When generating a new version, Roboflow offers the option to apply preprocessing and augmentation techniques. These steps, while optional, can significantly enhance the robustness of your model.

**Train/Test Split**: At this stage, a division will be made for the train set, valid set, and test set. usually the percentage of the train set compared to the others must be greater.

**Preprocessing**: This includes steps like resizing images, normalizing pixel values, and adjusting contrast or brightness. Preprocessing ensures that the dataset is uniform and suitable for model training.

**Augmentation**: Data augmentation techniques, such as flipping, rotating, or cropping images, artificially expand the dataset by creating new variations of the existing data. This can help improve the model's ability to generalize across different scenarios.Generating a new dataset version with augmentations can improve the performance of your YOLOv8 model, particularly when working with limited data, as is often the case in medical imaging.

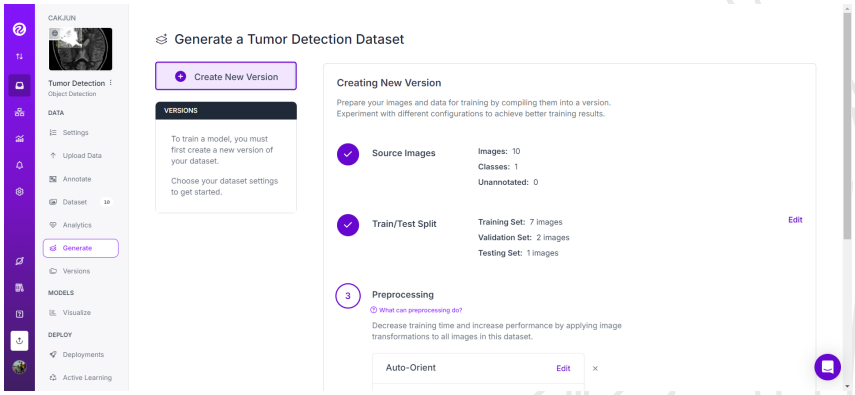



Figure 3.9: Generate a Dataset

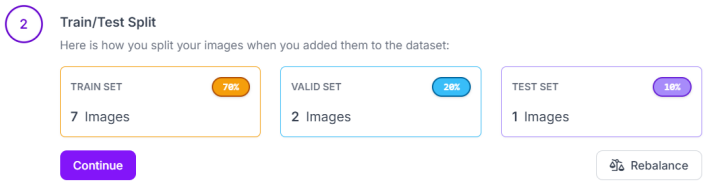



Figure 3.10: Train/Test Split

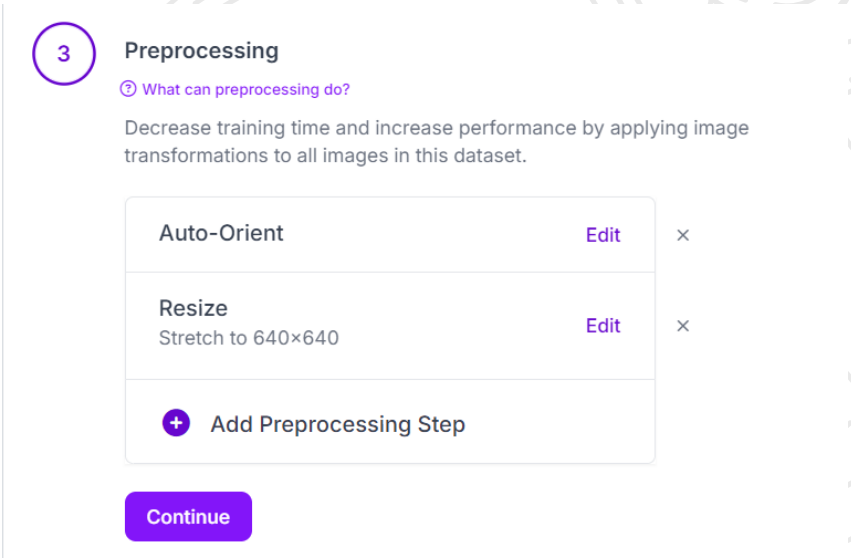



Figure 3.11: Preprocessing Dataset

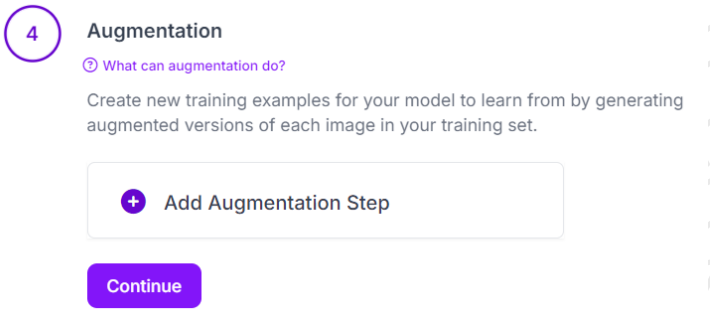



Figure 3.12: Augmentation Dataset

- **Step 5: Exporting the Dataset**

Once the dataset version is generated, the dataset is hosted on Roboflow's servers and can be directly loaded into your notebook for training. To export the dataset, click **Export** and choose the format compatible with your object detection model.

For this tutorial, select the **YOLOv8** format. Roboflow will provide you with the necessary links and code to easily integrate the dataset into your YOLOv8 training pipeline. In previous versions, we used YOLOv5, but for this tutorial, YOLOv8 offers better performance and faster processing.

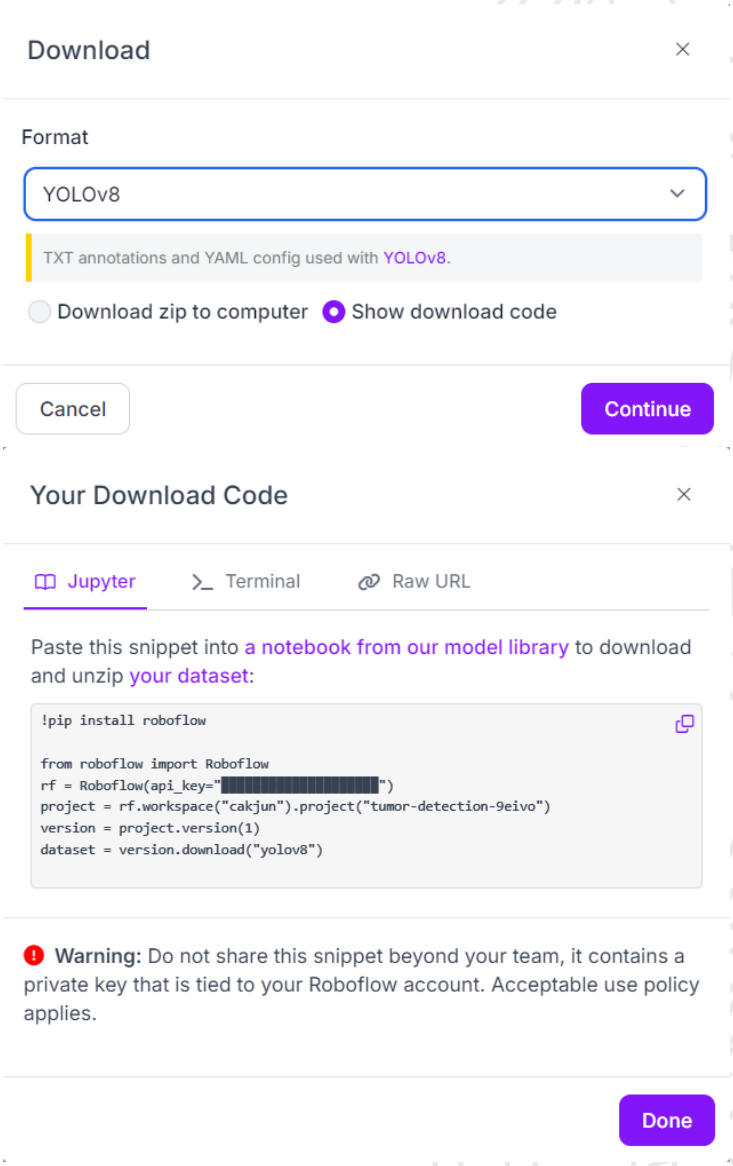



Figure 3.13: Export Database

By following these steps, you will have successfully prepared a custom dataset for brain tumor detection using Roboflow. The process of creating a project, uploading images, labelling, and exporting the dataset can be completed in a streamlined and efficient manner. This dataset can now be used in the next stages of model training and evaluation with YOLOv8.

## 3.4 DOWNLOADING THE DATASET

Code click here
We will use a **brain tumor detection dataset** hosted on Roboflow. This dataset contains labeled images that we will use for training YOLOv8. Ensure you have a Roboflow account and generate an API key for access. First, create a directory for storing the dataset:

```
!mkdir -p {HOME}/datasets
```

Next, navigate to the newly created dataset directory:

```
%cd {HOME}/datasets
```

Use the Roboflow API to download the dataset:

```
from roboflow import Roboflow
rf = Roboflow(api_key="YOUR_API_KEY")
project = rf.workspace("YOUR_WORKSPACE").project("brain-tumors-detection")
version = project.version(2)
dataset = version.download("yolov8")
```

Finally, navigate back to the main directory:

```
%cd {HOME}
```

### 3.4.1 Code Explanation

- `!mkdir -p {HOME}/datasets`: This command creates a new directory named `datasets` in the home directory. The `-p` option ensures that no error is thrown if the directory already exists and creates any necessary parent directories.
- `%cd {HOME}/datasets`: This command changes the current working directory to the newly created datasets directory, preparing the environment for downloading the dataset.
- `from roboflow import Roboflow`: This line imports the Roboflow library, which provides tools to interact with the Roboflow API for dataset management.
- `rf = Roboflow(api_key="YOUR_API_KEY")`: This line initializes the Roboflow object using your API key. Replace `YOUR_API_KEY` with the actual API key obtained from your Roboflow account.
- `project = rf.workspace("YOUR_WORKSPACE").project("brain-tumors-detection")`: This line accesses the specific project within your Roboflow workspace. You must replace `YOUR_WORKSPACE` with the name of your actual workspace.
- `version = project.version(2)`: This line selects the version of the dataset you want to download. In this case, version 2 of the brain tumor detection project is being accessed.
- `dataset = version.download("yolov8")`: This line downloads the dataset in a format compatible with YOLOv8, which will be used for training.

- `%cd {HOME}`: This command returns to the main directory after the dataset has been downloaded, ensuring that the subsequent commands operate in the correct context.

### 3.4.2 Visual Explanation

The steps outlined in this section illustrate how to download the brain tumor detection dataset from Roboflow. By following these commands, you create a structured directory for the dataset, facilitating easier data management.

The utilization of the Roboflow API enables seamless access to high-quality labeled images, which are critical for training YOLOv8. This dataset serves as the foundation for the model's learning process, providing it with the necessary examples to learn from. By ensuring that the dataset is downloaded correctly and stored in a designated directory, you set the stage for an organized and efficient training workflow.

Additionally, navigating back to the main directory after downloading the dataset helps maintain a clean working environment, making it easier to execute subsequent training commands without confusion or errors. This systematic approach is essential for efficient project management in machine learning workflows.

## 3.5 TRAINING THE YOLOV8 MODEL

With the dataset prepared, we can now train our YOLOv8 model. We will fine-tune a pre-trained `yolov8s.pt` model on our dataset. The following command trains the model for 25 epochs. Adjust the epochs as needed based on your data complexity and computational resources:

Code click here

```
!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=25 imgsz=800 plots=True
```

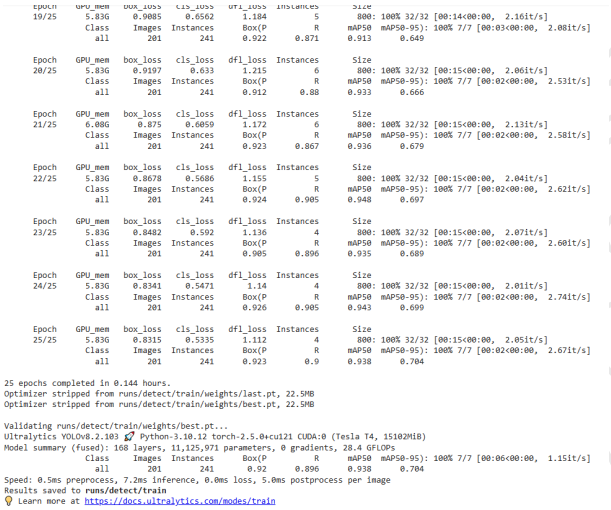



Figure 3.14: Training YOLOv8

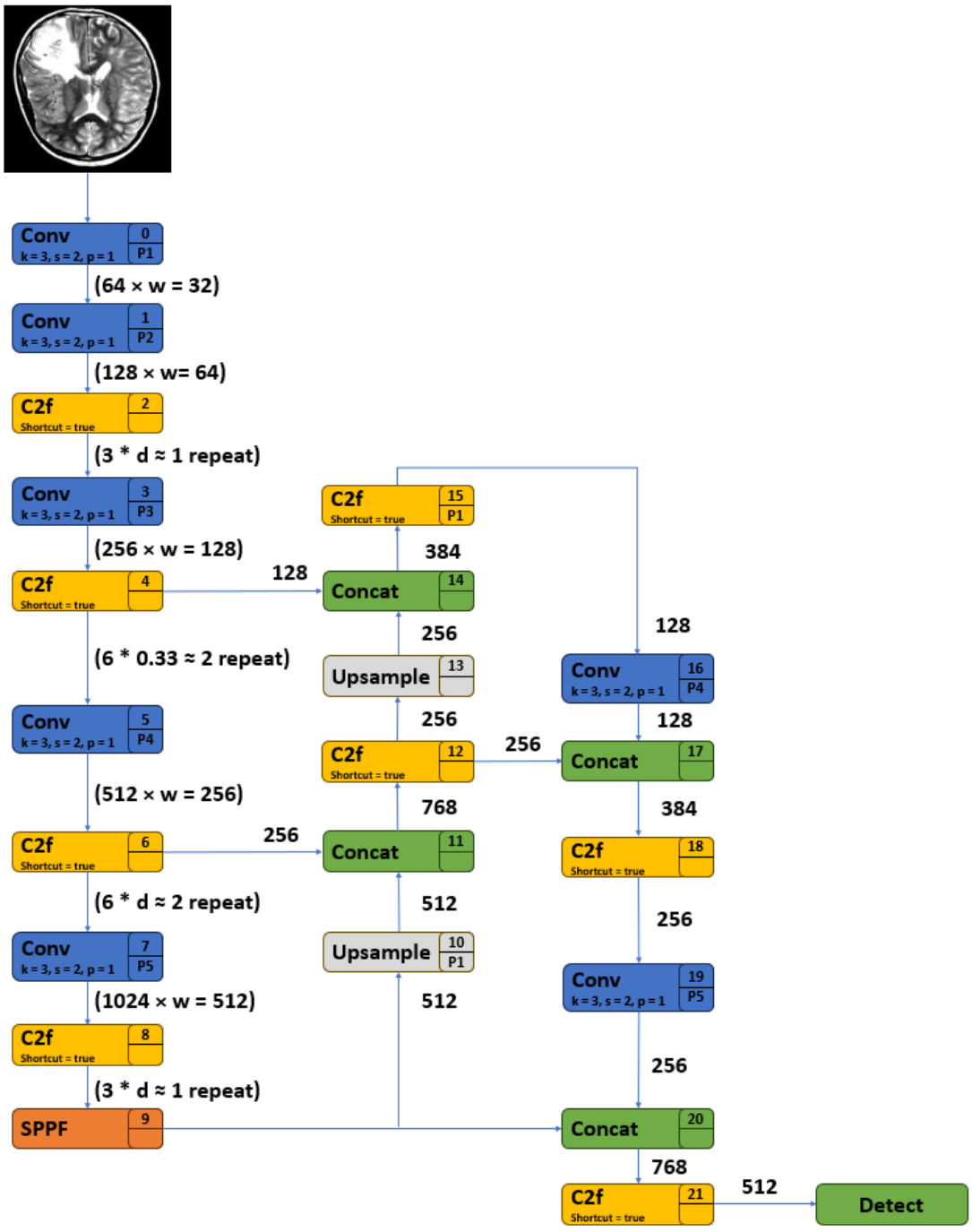



Figure 3.15: Arsitektur YOLOv8s.pt

### 3.5.1 Code Explanation

- `!yolo task=detect`: This part of the command specifies that the task at hand is object detection. YOLO is designed to detect and classify objects in images.

- `mode=train`: This parameter indicates that the model should be trained, as opposed to validating or testing.

- `model=yolov8s.pt`: This specifies the use of the pre-trained YOLOv8 small model. Leveraging a pre-trained model allows us to benefit from transfer learning, where the model has already learned features from a large dataset.

- `data={dataset.location}/data.yaml`: This parameter points to the dataset's configuration file, which contains essential information about the dataset, such as paths to the training and validation images and the classes to be detected.
- `epochs=25`: This parameter sets the number of training epochs to 25. An epoch represents one complete pass through the training dataset. Adjusting the number of epochs allows you to control how long the model is trained.
- `imgsz=800`: This sets the input image size to 800x800 pixels. A larger image size can lead to better accuracy, but it also increases computational demands.
- `plots=True`: This parameter enables plotting, allowing visualization of the training progress, including metrics like loss and accuracy over epochs. Visual feedback is crucial for assessing the model's performance during training.

### 3.5.2 Training Parameters Explanation

The training parameters are critical for configuring how the model learns from the dataset. Each parameter influences the training process in different ways:

- `task=detect`: Directs the YOLO model to focus on detecting objects in images.
- `model=yolov8s.pt`: Utilizes a pre-trained model, which provides a solid starting point for fine-tuning on our specific dataset.
- `data=`: Ensures the model accesses the correct dataset configuration, essential for loading data correctly during training.
- `epochs=25`: Defines the duration of training; more complex datasets may require additional epochs for optimal performance.
- `imgsz=800`: Affects input size; larger images generally yield better detection results but require more memory and processing power.
- `plots=True`: Facilitates monitoring of training progress, which is vital for diagnosing issues and adjusting training strategies as necessary.

By understanding and adjusting these parameters, we can effectively tailor the training process to enhance model performance in detecting brain tumors.

## 3.6 EXPECTED OUTCOMES AND MODEL EVALUATION

After training, your YOLOv8 model will be fine-tuned for detecting brain tumors on your dataset. The model can now be used for inference on new images. In the following chapter, we will evaluate the trained model's performance, analyze its predictions, and discuss deployment options for real-time detection.

# CHAPTER 4
# Evaluating YOLOv8 for Brain Tumor Detection

Once the YOLOv8 model has been trained on the brain tumor dataset, the next step is to evaluate its performance. In this chapter, we will cover the key performance metrics used for object detection tasks and how to visualize the results of the model on brain tumor images.

## 4.1 PERFORMANCE METRICS

Evaluating the performance of an object detection model like YOLOv8 requires using specific metrics that reflect the accuracy and reliability of the model. Below are the key metrics used for evaluating YOLOv8 on brain tumor detection:

### 4.1.1 Precision, Recall, F1-Score

- **Precision:** Precision measures the proportion of predicted positive detections (i.e., detected tumors) that are actually correct. It is defined as:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

  A high precision score indicates that when the model predicts a tumor, it is likely to be correct.

- **Recall:** Recall, or sensitivity, measures the proportion of actual tumors that are correctly detected by the model. It is defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

  A high recall score means that the model is successfully detecting most tumors in the images.

- **F1-Score:** The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is useful when you want to account for both false positives and false negatives:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

### 4.1.2 Mean Average Precision (mAP)

Mean Average Precision (mAP) is one of the most important metrics for evaluating object detection models. It summarizes the precision and recall across all classes and provides a single score to represent the model's performance. The steps for calculating mAP are:

- **Intersection over Union (IoU):** IoU is a measure of overlap between the predicted bounding box and the ground truth bounding box. IoU is defined as:

  $$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

  A prediction is considered correct if the IoU between the predicted and actual bounding boxes exceeds a predefined threshold (e.g., 0.5).

- **Average Precision (AP):** AP is the area under the precision-recall curve for each class. It is calculated by averaging the precision values at different recall levels.

- **Mean Average Precision (mAP):** mAP is the average of the AP values for all classes. Since brain tumor detection typically involves a single class, the mAP will be the AP value for the tumor class.

## 4.2 CONFUSION MATRIX

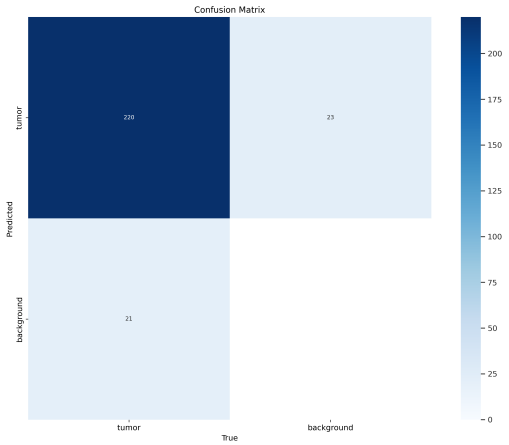



Figure 4.1: Confusion Matrix

The confusion matrix is a common tool used in classification tasks to evaluate the performance of a machine learning model. It provides a comprehensive summary of the number of correct and incorrect predictions made by the model for each class. In the context of brain tumor detection, the confusion matrix helps us understand how effectively the model distinguishes between tumor and non-tumor regions.
To display the confusion matrix after training, use the following code click here
:

```
%cd{HOME}
Image(filename=f'{HOME}/runs/detect/train/confusion_matrix.
    png', width=600)
```

### 4.2.1 Code Explanation

- `%cd {HOME}`: This command changes the working directory to the home directory, which is essential for ensuring that the file paths specified in the code are accurate. This allows the program to locate the necessary files without any issues.
- `Image(filename=f'{HOME}/runs/detect/train/confusion_matrix.png', width=600)`: This line of code loads and displays an image file that contains the confusion matrix. The specified PNG file was generated during the training process and illustrates how well the model classified the different classes. The `width=600` parameter ensures that the image is displayed with a width of 600 pixels for clarity.

### 4.2.2 Visual Explanation

Figure 4.1 presents the confusion matrix generated during the training process. This matrix visually summarizes the model's performance in classifying tumor and non-tumor regions. The confusion matrix displays true positive, true negative, false positive, and false negative counts for each class, allowing us to assess the model's accuracy and identify potential areas for improvement. For example, a high number of false positives may indicate that the model incorrectly identifies non-tumor regions as tumors, while a high number of false negatives suggests that the model misses actual tumors. Analyzing the confusion matrix is essential for fine-tuning the model and enhancing its classification capabilities, ultimately leading to improved diagnostic accuracy in brain tumor detection.

## 4.3 VISUALIZING RESULTS

Upon completing the training process, it is important to evaluate the model's results by visualizing key performance metrics such as the confusion matrix and other relevant images. The following code blocks help in generating and displaying these visuals.

### 4.3.1 Training Results Overview

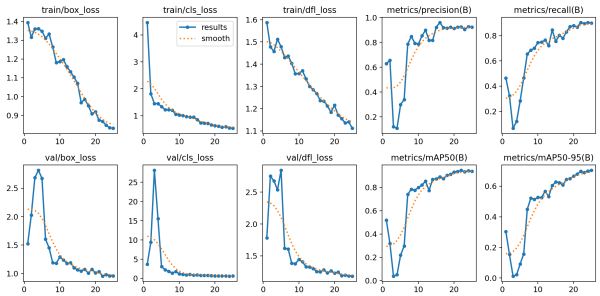



Figure 4.2: Results Overview

The results of the training process, such as loss curves and precision-recall metrics, can be visualized using the following code click here
:

```
%cd {HOME}
Image(filename=f'{HOME}/runs/detect/train/results.png', width
    =600)
```

**Code Explanation**

- `%cd {HOME}`: This command changes the working directory to the home directory. This is crucial for ensuring that the paths specified in the code are correct and that the program can locate the necessary files without issues.
- `Image(filename=f'{HOME}/runs/detect/train/results.png', width=600)`: This line loads and displays an image file that contains the training results. Specifically, it refers to a PNG file generated during training that includes loss curves and precision-recall metrics. The parameter `width=600` sets the width of the displayed image to 600 pixels, ensuring clarity and proper scaling for visibility.

**Visual Explanation**

Figure 4.2 provides an overview of the model's performance over the training epochs. The visual representation of the loss curves and precision-recall metrics is essential for tracking the learning process of the model.
By analyzing the loss curves, we can observe how the model's error decreases over time, indicating improvements in its performance. Similarly, the precision-recall metrics illustrate the model's ability to correctly identify positive instances while minimizing false positives. These visualizations are vital for diagnosing potential issues in the training process, such as overfitting or underfitting, and they help inform further adjustments in model training and parameter tuning to enhance performance.

### 4.3.2 Sample Predictions from Validation Set

To check the model's predictions on the validation dataset, we can visualize examples of the model's predictions. The following code displays a sample prediction from the validation set code click here :

```
%cd {HOME} Image(filename=f'{HOME}/runs/detect/train/
    val_batch0_pred.jpg', width=600)
```

**Code Explanation**

- `%cd {HOME}`: This command changes the working directory to the home directory. This is important to ensure that the file paths used in the code are relative to the correct directory, allowing the program to locate the necessary files seamlessly.
- `Image(filename=f'{HOME}/runs/detect/train/val_batch0_pred.jpg', width=600)`: This line of code loads and displays an image file from the specified path. The image in question is the first batch of predicted results from the validation set. The `width=600` parameter sets the displayed image width to 600 pixels, ensuring that the image is properly sized for visibility.

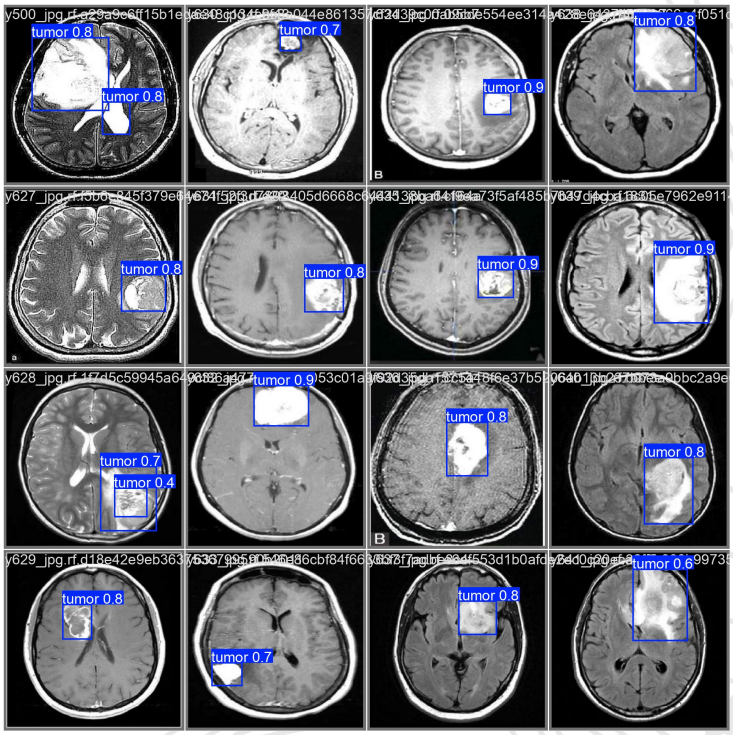



Figure 4.3: Result

### Visual Explanation

Figure 4.3 visualizes the bounding boxes predicted by YOLOv8 on a sample image from the validation dataset. This output allows us to qualitatively assess how well the model identifies brain tumors within the images.
By examining the predicted bounding boxes, we can determine whether they accurately encompass the tumors and identify any potential false positives or missed detections. This qualitative assessment is crucial, as it helps us understand the model's performance and effectiveness in real-world applications. Additionally, visual inspections can guide further improvements in the model's training and fine-tuning processes, highlighting areas where the model may require additional training or adjustments to improve its detection accuracy.

## 4.4 MODEL VALIDATION

Model validation is an essential step in evaluating the model's performance on unseen data. The primary objective of this phase is to assess how well the trained model generalizes to new examples not encountered during training. By using the trained weights and running validation mode, we can obtain important performance metrics that provide insights into the model's accuracy, precision, recall, and overall robustness.
The following command runs the YOLOv8 model in validation mode on the validation set, utilizing the best weights saved during training. The output of this process includes key performance indicators that allow us to understand how the model performs on the validation dataset. Code click here.

```
%cd {HOME}
!yolo task=detect mode=val model={HOME}/runs/detect/train/
   weights/best.pt data={dataset.location}/data.yaml
```

### 4.4.1 Code Explanation

- `%cd {HOME}`: Changes the working directory to the home folder. This is essential for ensuring that the YOLO model can locate the necessary files, including the saved weights and the configuration data.
- `task=detect`: Specifies that the task is object detection. YOLOv8 can handle multiple tasks such as segmentation and classification, but in this case, detection is the focus.
- `mode=val`: Sets the model to validation mode, meaning that it will use the validation dataset to assess the performance of the trained model. Validation helps determine how well the model generalizes to new, unseen data.
- `model={HOME}/runs/detect/train/weights/best.pt`: Points to the weight file generated from the training process, specifically the best-performing weights. These weights are used to make predictions during validation, ensuring that the model's best performance is evaluated.
- `data={dataset.location}/data.yaml`: Specifies the path to the dataset's configuration file. This file contains metadata about the dataset, such as the locations of the images and annotations, and it guides the model in loading and validating the correct data.

By executing this command, the model evaluates the validation dataset and provides key performance metrics, such as precision, recall, F1 score, and mean average precision (mAP). These metrics are critical for understanding the model's ability to detect objects accurately across different categories and assessing its overall reliability. Additionally, running validation allows us to detect overfitting, where the model may perform well on the training set but struggles with unseen data.

```
/content
Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11,125,971 parameters, 0 gradients, 28.4 GFLOPs
val: Scanning /content/datasets/Brain-Tumors-Detection-2/valid/labels.cache... 201 images, 0 backgrounds, 0 corrupt: 100% 201/201 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 13/13 [00:05<00:00,  2.20it/s]
                   all        201        241      0.923        0.9      0.938      0.705
Speed: 3.9ms preprocess, 13.9ms inference, 0.0ms loss, 4.6ms postprocess per image
Results saved to runs/detect/val
💡 Learn more at https://docs.ultralytics.com/modes/val
```

Figure 4.4: Results Overview

### 4.4.2 Code Explanation

Figure 4.4 displays an overview of the results generated by the validation process. This visualization includes important metrics such as the precision-recall curve and confusion matrix, which illustrate how well the model performs on the validation dataset. The precision-recall curve helps in understanding the trade-off between precision (how many positive predictions are correct) and recall (how many actual positives are detected), while the confusion matrix provides a breakdown of true positives, false positives, and false negatives. This graphical representation of the validation results is essential for identifying areas of improvement in the model. If the precision is high but the recall is low, for example, it might indicate that the model is overly conservative, missing some objects that should have been detected. Conversely, if the recall is high but the precision is low, the model may be producing too many false positives, detecting objects that are not present.

Overall, validation helps in fine-tuning the model's performance and ensuring that it generalizes well to real-world data. The results displayed in the figure guide further adjustments in the training process, such as modifying the architecture, adjusting hyperparameters, or augmenting the training dataset to improve the model's accuracy.

## 4.5 MAKING PREDICTIONS ON TEST DATA

After completing the validation phase, we can proceed to make predictions on the test dataset using the trained model. This step is crucial for evaluating the model's performance on unseen data, providing insights into its ability to generalize. By running the trained model on test images, we can observe how well it detects objects under conditions similar to those in real-world applications.
The following code block executes the YOLOv8 model on the test dataset, utilizing the best weights saved during training to generate predictions. This is done with a confidence threshold to filter out low-confidence detections, ensuring only the most likely predictions are visualized and saved for future analysis. Code click here.

```
%cd {HOME}
!yolo task=detect mode=predict model={HOME}/runs/detect/train
   /weights/best.pt conf=0.25 source={dataset.location}/test/
   images save=True
```

### 4.5.1 Code Explanation

- `%cd HOME`: This line changes the working directory to the home directory. This setup is necessary to ensure that all required paths are relative to the home directory, allowing the YOLO model to correctly locate the saved weights and the test dataset.

- `task=detect`: Specifies that the task for YOLO is object detection, as opposed to other tasks like segmentation or classification. This parameter is critical as it tells the model to use the detection framework during inference.

- `mode=predict`: This parameter sets the model mode to "predict," which means that the model will use the test dataset for inference rather than training or validation.

- `model=HOME/runs/detect/train/weights/best.pt`: Points to the specific weight file (the best model from training) saved in the training directory. These weights represent the trained model's parameters, optimized to detect objects effectively.

- `conf=0.25`: Sets the confidence threshold to 0.25, filtering out detections with confidence scores below this value. A threshold of 0.25 means the model will only display bounding boxes for detections with at least a 25% confidence level, thus reducing noise from low-confidence predictions.

- `source=dataset.location/test/images`: Specifies the location of the test images, which will be processed by the model for object detection. This allows for flexibility in specifying various test datasets for different applications.

- `save=True`: Enables the model to save the prediction results. This parameter is essential for later stages, as it allows for visual verification and analysis of model performance on the test images.

Using this code, the model generates bounding boxes for detected objects in the test images and saves the annotated images for further inspection. This visualization step is invaluable in determining the model's performance and identifying any potential misdetections or areas where the model could be further refined.

```
%cd {HOME}
!yolo task=detect mode=predict model={HOME}/runs/detect/train/weights/best.pt conf=0.25 source={dataset.location}/test/images save=True
```

```
/content
Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11,125,971 parameters, 0 gradients, 28.4 GFLOPs

image 1/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y701_jpg.rf.2ecfd5634c3ca52c5ad1f4d726f30a40.jpg: 800x800 1 tumor, 22.8ms
image 2/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y702_jpg.rf.30fed4a6ad98b30c51bdd5c86e6f2d1e.jpg: 800x800 1 tumor, 22.9ms
image 3/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y703_jpg.rf.1789cf62115797babacf49c8be81d74c.jpg: 800x800 1 tumor, 22.8ms
image 4/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y704_jpg.rf.81aca4af861646a66b6e282879b13488.jpg: 800x800 2 tumors, 22.8ms
image 5/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y705_jpg.rf.adec9252865c857a7f8520623e883ef1.jpg: 800x800 1 tumor, 22.8ms
image 6/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y706_jpg.rf.42d69cea985670c99ad77e84c6fa6e31.jpg: 800x800 1 tumor, 22.8ms
image 7/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y707_jpg.rf.53f064e6179f2c00d8fad8d9a2f83446.jpg: 800x800 1 tumor, 22.8ms
image 8/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y708_jpg.rf.352301bb8ca7574f64aebc61a51dda64.jpg: 800x800 2 tumors, 22.8ms
image 9/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y709_jpg.rf.e369f03584e81e83c5c27ed1fd73d3da.jpg: 800x800 1 tumor, 17.0ms
image 10/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y710_jpg.rf.a988cef9b79034c91fa4a89f580acfc6.jpg: 800x800 1 tumor, 14.8ms
image 11/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y711_jpg.rf.ad6ff9bc3830ec089f6587a948f6f3a8.jpg: 800x800 1 tumor, 14.8ms
image 12/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y712_jpg.rf.ccd32a734708e8096b712afb448c8e5e.jpg: 800x800 1 tumor, 14.9ms
image 13/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y713_jpg.rf.997d94c0d53c59d4076ab21035f27aa8.jpg: 800x800 1 tumor, 14.6ms
image 14/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y714_jpg.rf.4b9a5ec559ae1f00412607d1820d22d3.jpg: 800x800 1 tumor, 14.4ms
image 15/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y715_jpg.rf.b2894642ae02353cf812f6641cfc85d9.jpg: 800x800 1 tumor, 14.3ms
image 16/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y716_jpg.rf.ab41883f69109506f7cee7319bb32963.jpg: 800x800 2 tumors, 14.3ms
image 17/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y717_jpg.rf.a63b0c08e146120a882016c773f21e74.jpg: 800x800 1 tumor, 13.7ms
image 18/100 /content/datasets/Brain-Tumors-Detection-2/test/images/y718_jpg.rf.541f3b0e827d4fdbe4e8077473ac2770.jpg: 800x800 1 tumor, 11.7ms
```

Figure 4.5: Predictions on Test Data

### 4.5.2 Visual Explanation

Figure 4.5 illustrates the predictions made by the trained YOLOv8 model on the test dataset. In this image, bounding boxes are drawn around the detected objects, each with an associated confidence score reflecting the model's certainty. This visualization demonstrates the practical application of the model's training, as it attempts to identify objects in new, unseen data.

The use of a confidence threshold of 0.25 ensures that only predictions with a reasonable level of certainty are displayed. This threshold strikes a balance between capturing all relevant objects and avoiding false positives, enhancing the reliability of the predictions.

By saving the prediction results, the model allows us to conduct a detailed post-inference analysis, examining the quality of bounding boxes and the accuracy of object classifications. This step provides a comprehensive view of the model's detection capabilities and helps highlight any potential areas for improvement. Visual inspections such as this are essential for assessing the model's real-world applicability and robustness in detecting objects accurately.

## 4.6 DISPLAYING PREDICTIONS

After training the object detection model and generating predictions, the next critical step is to visualize these predictions for verification purposes. Visualizing the predictions allows us to evaluate how well the model has detected objects and whether the bounding boxes produced align accurately with the objects in the images. It also serves as a useful tool for

identifying potential areas where the model may need improvement, such as misdetections or missed objects.
The following code dynamically locates the latest folder containing the prediction images and displays a few of them. We utilize the Python libraries 'glob' to search for images in the relevant folder and 'IPython.display' to render these images within the notebook. This code dynamically identifies the most recent folder, generated during the model inference process, and displays the first three predicted images. Code click here

```
import glob
from IPython.display import Image, display
# Define the base path where the folders are located
base_path = '/content/runs/detect/'
# List all directories that start with 'predict' in the base path
subfolders = [os.path.join(base_path, d) for d in os.listdir(base_path)
if os.path.isdir(os.path.join(base_path, d)) and d.startswith('predict')]
    # Find the latest folder by modification time
latest_folder = max(subfolders, key=os.path.getmtime)
image_paths = glob.glob(f'{latest_folder}/*.jpg')[:3]
    # Display each image
    for image_path in image_paths:
     display(Image(filename=image_path, width=600))
     print("\n")
```

### 4.6.1 Code Explanation

- `base_path`: Defines the root directory where the detection results are stored. In this case, it is '/content/runs/detect/', which is the standard location where YOLO or other object detection models store their predictions.
- `subfolders`: This line searches for all directories in the base path that start with the word "predict." Each time the model detects objects in new images, it typically creates a new directory with the name starting with "predict." Only directories are considered, ensuring that other non-folder items are ignored.
- `latest_folder`: This line finds the latest folder based on the modification time (the most recent activity). This is useful when there are multiple prediction results from different experiments, allowing the script to automatically select the latest output.
- `image_paths`: Retrieves the first three images in the latest folder found. These images will be displayed for evaluation purposes.
- `display(Image(...))`: This function from 'IPython.display' is used to render the images directly in the notebook. Each image is displayed with a width of 600 pixels, providing clarity without overwhelming the layout.

Visualizing the prediction results is crucial as it allows for a better understanding of the model's performance beyond numerical accuracy or loss values. By looking at the bounding boxes overlaid on the images, we can determine how well the model is detecting objects and whether there are any objects that were missed or incorrectly labeled.

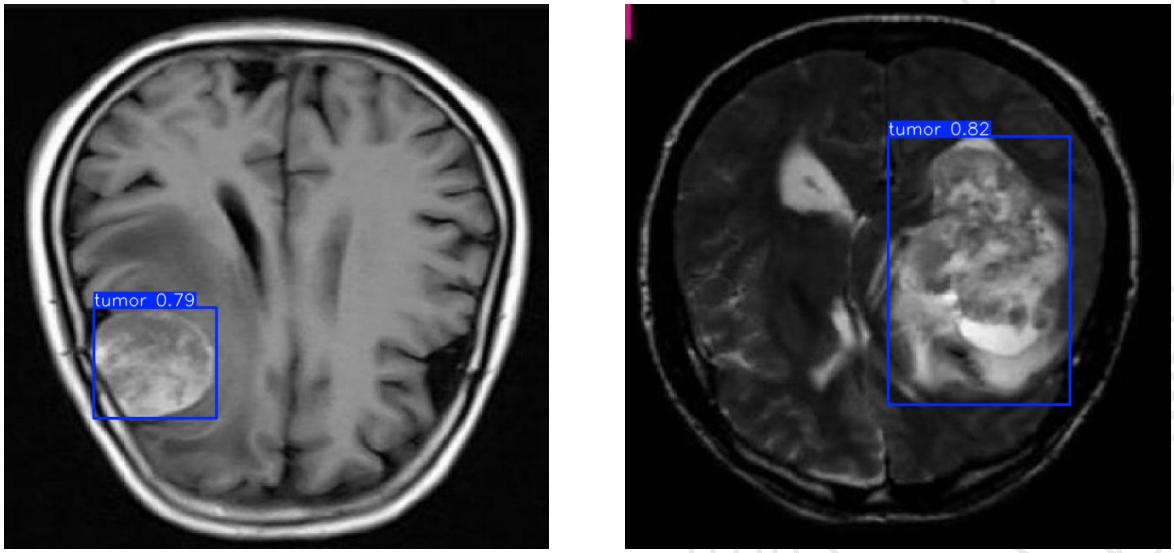



Figure 4.6: Results Overview for Both Images

### 4.6.2 Visual Explanation

In Figure 4.6, we present two side-by-side images displaying the predicted outcomes generated by the object detection model. The first image (*Hasil 1*) and the second image (*Hasil 2*) each show the results of the model's predictions. The images have been processed by the model, and bounding boxes have been added to highlight the detected objects.
The space between the images is essential to prevent them from appearing too close or overlapping, ensuring a clean and organized presentation. This allows for individual analysis of each image while still displaying them in a single row for direct comparison.
The caption below provides a concise description of the content in the images, summarizing the detected objects. By placing the caption in the middle, we give a unified explanation for both images, offering clarity and context without redundancy.
Overall, visualizing predictions in this manner provides a comprehensive overview of the model's performance and serves as a tool for identifying areas that may require further refinement in the model's training.

### 4.6.3 Interpreting Detection Results

Interpreting the detection results involves reviewing the predicted bounding boxes and their associated confidence scores. A typical YOLOv8 output includes:

- **Bounding Box Coordinates:** The coordinates define the location of the detected tumor.
- **Confidence Score:** The confidence score represents the model's certainty about the detection. A higher confidence score indicates greater certainty that the detected region contains a tumor.

When reviewing the results, it is important to consider:

- **False Positives:** Instances where the model detects a tumor that does not exist.
- **False Negatives:** Instances where the model fails to detect an actual tumor.

A good way to evaluate these results is to overlay the predicted bounding boxes on the images and manually inspect the areas where the model made mistakes (false positives or false negatives). This helps in diagnosing model performance and understanding areas for improvement.