Name : Sacin Jadhav
Div : DL2
PRN : 202201040080

Sub : Deep Learning

Lab Assignment NO.03: Object Detection and Multi Object Classification

Ultralitycs Platform Documentation

Objective

This document serves as a guide for implementing the YOLOv11 model on the Ultralitycs platform for real-time brain tumor detection. The objective is to set up the environment, prepare the dataset, train the model, run inference, and evaluate performance.

Task 1: Environment Setup and YOLOv11 Installation

Installing Python and Dependencies

To run YOLOv11, a compatible Python environment must be set up. This includes installing Python version 3.8 or higher. Additionally, essential libraries such as Torch (for deep learning computations), Ultralytics (for YOLO implementation), OpenCV (for image processing), NumPy (for numerical operations), and Matplotlib (for visualization) are required.

Installing YOLOv11

YOLOv11, an advanced object detection model, needs to be obtained from the official repository. After acquiring the model files, the dependencies specified in its requirements must be installed. This ensures that all necessary components, such as pre-trained weights, image processing modules, and training utilities, are correctly configured.

Verifying the Installation

After setup, it is crucial to test whether YOLOv11 runs properly. This can be done by executing a simple object detection task using sample images or webcam feeds. If the model successfully detects objects and produces output images with bounding boxes, the installation is considered successful.

Task 2: Dataset Preparation

Downloading the Dataset

The dataset used for training the YOLOv11 model must consist of brain tumor images labeled for detection. Publicly available datasets from platforms such as Kaggle or medical imaging repositories can be utilized.

Organizing Data in YOLO Format

For YOLOv11 to work effectively, the dataset must be structured in a specific format. Images should be separated into training and validation sets. Corresponding annotation files should be created in YOLO format, where each labeled tumor instance is represented by a class identifier and bounding box coordinates. Proper organization ensures smooth processing during model training.

Task 3: Training the Model

Initiating Model Training

Once the dataset is prepared, the YOLOv11 model must be trained using the provided images and labels. During training, the model learns to detect tumors by optimizing its parameters over multiple iterations. Hyperparameters such as image size, batch size, and the number of training epochs influence the training process.

Generating Trained Weights

After successful training, the model produces a set of optimized weights. These weights are stored in a designated directory and represent the trained version of YOLOv11 specific to brain tumor detection. These weights will later be used for inference on unseen data.

Task 4: Running Inference

Performing Tumor Detection

Once the model is trained, it can be used to analyze new brain scans. The trained YOLOv11 model processes test images and detects potential tumors by marking them with bounding boxes. The model utilizes the trained weights to make predictions, and the detected objects are displayed visually.

Interpreting Results

The output of the inference stage consists of detected tumor locations with confidence scores indicating the certainty of each detection. These results can be further analyzed by medical professionals for diagnosis and assessment.

Task 5: Evaluating Model Performance

Assessing Accuracy and Metrics

The effectiveness of the trained YOLOv11 model is determined by evaluating its performance on the validation dataset. Key metrics such as precision, recall, mean Average Precision (mAP), and inference speed are computed to measure how well the model detects brain tumors. Higher accuracy and precision indicate better detection capability.

Logging and Analysis

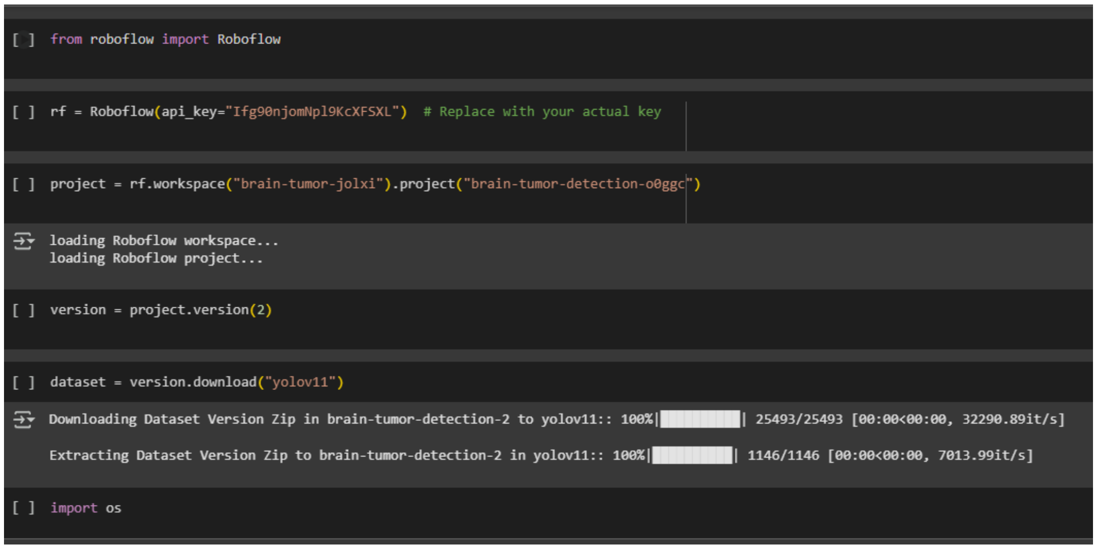
All performance results are logged for further analysis. By reviewing these logs, improvements can be made to the model by fine-tuning parameters, increasing dataset size, or applying data augmentation techniques to enhance detection accuracy.

Code file :
https://colab.research.google.com/drive/1aptunOu78o3BuhL9ag996Q0M2wV3pcoM

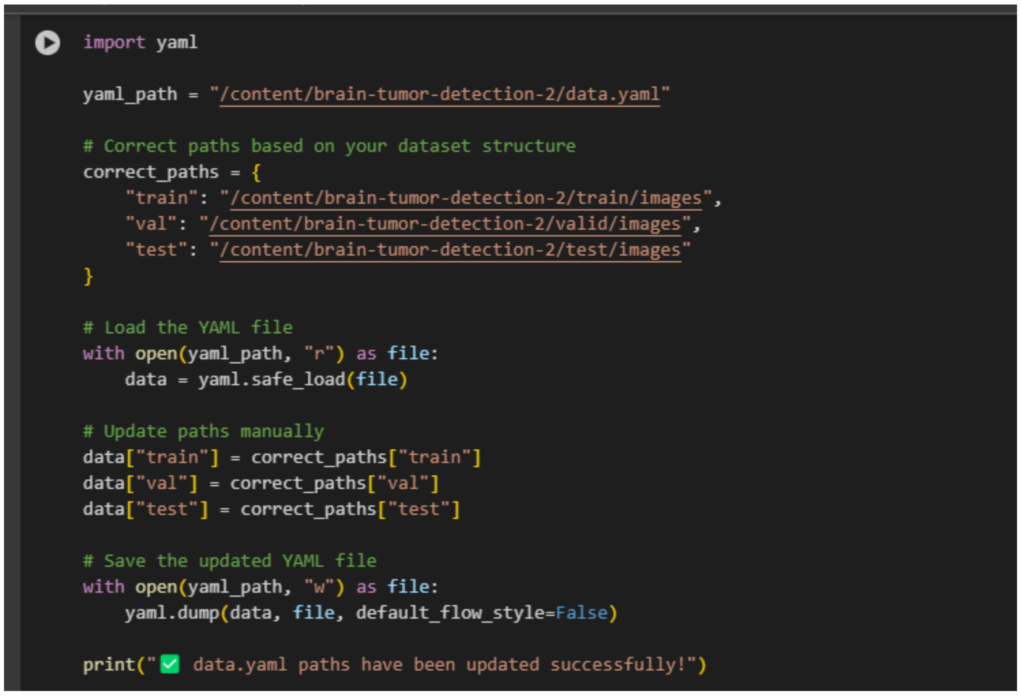
Dataset Link : https://universe.roboflow.com/brain-tumor-jolxi/brain-tumor-detection-o0ggc/dataset/2

GITHUB Link : https://github.com/SACHIN9637/Deep-Learning/tree/main/Lab%20Assignment_3

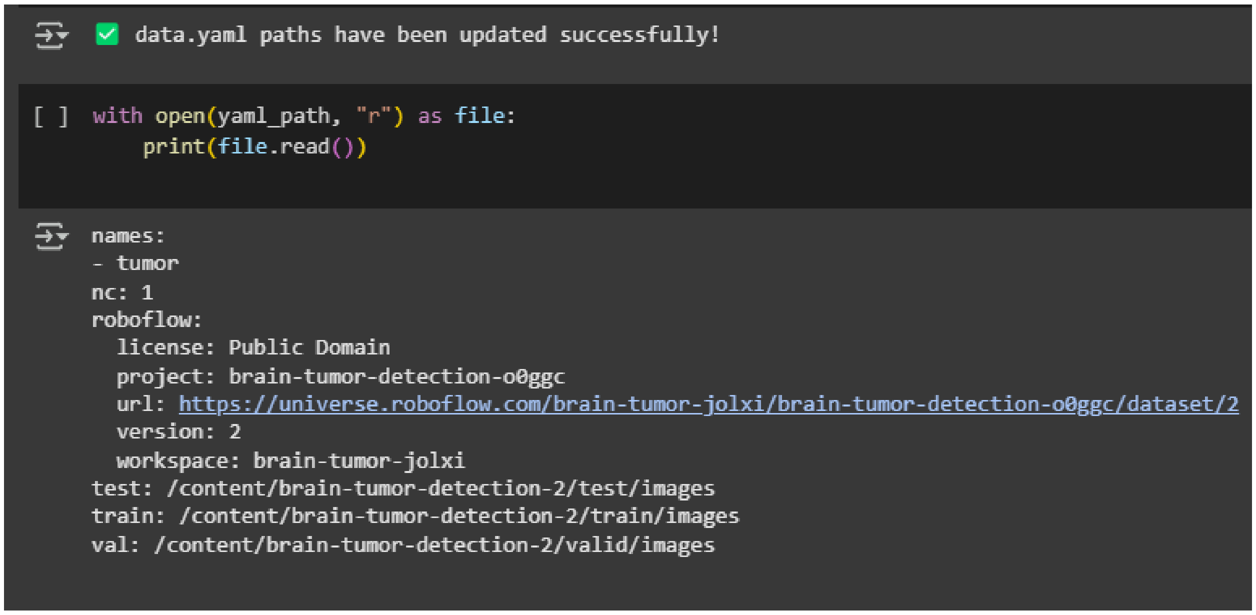
Screenshots of model performance metrics



```
[ ] from roboflow import Roboflow

[ ] rf = Roboflow(api_key="Ifg90njomNpl9KcXFSXL")  # Replace with your actual key

[ ] project = rf.workspace("brain-tumor-jolxi").project("brain-tumor-detection-o0ggc")

    loading Roboflow workspace...
    loading Roboflow project...

[ ] version = project.version(2)

[ ] dataset = version.download("yolov11")

    Downloading Dataset Version Zip in brain-tumor-detection-2 to yolov11:: 100%|██████████| 25493/25493 [00:00<00:00, 32290.89it/s]

    Extracting Dataset Version Zip to brain-tumor-detection-2 in yolov11:: 100%|██████████| 1146/1146 [00:00<00:00, 7013.99it/s]

[ ] import os
```



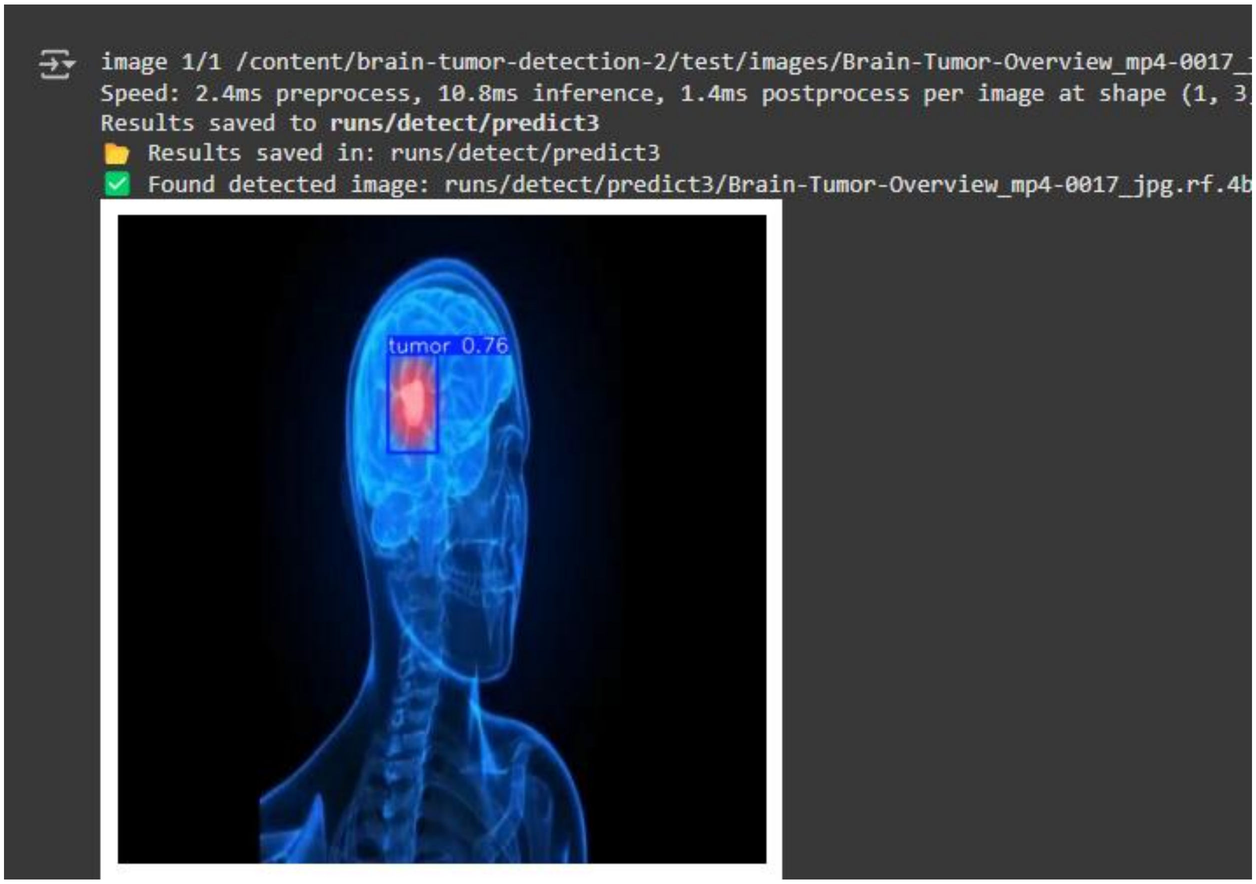
```
import yaml

yaml_path = "/content/brain-tumor-detection-2/data.yaml"

# Correct paths based on your dataset structure
correct_paths = {
    "train": "/content/brain-tumor-detection-2/train/images",
    "val": "/content/brain-tumor-detection-2/valid/images",
    "test": "/content/brain-tumor-detection-2/test/images"
}

# Load the YAML file
with open(yaml_path, "r") as file:
    data = yaml.safe_load(file)

# Update paths manually
data["train"] = correct_paths["train"]
data["val"] = correct_paths["val"]
data["test"] = correct_paths["test"]

# Save the updated YAML file
with open(yaml_path, "w") as file:
    yaml.dump(data, file, default_flow_style=False)

print("✅ data.yaml paths have been updated successfully!")
```



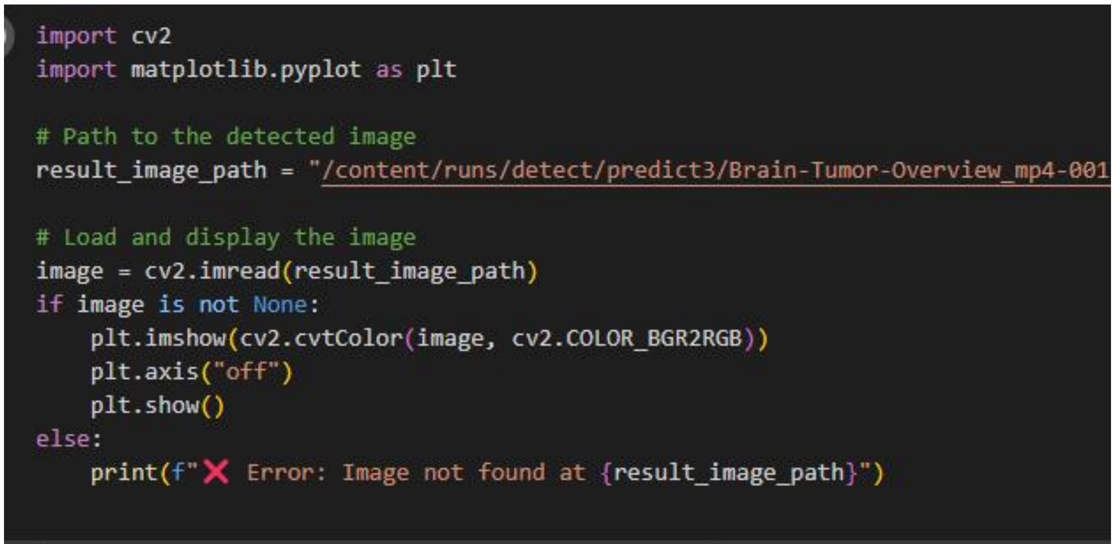
```
✅ data.yaml paths have been updated successfully!
```

```
[ ] with open(yaml_path, "r") as file:
        print(file.read())
```

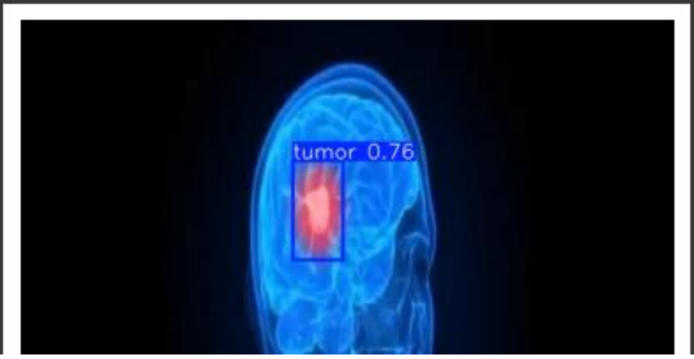
```
names:
- tumor
nc: 1
roboflow:
  license: Public Domain
  project: brain-tumor-detection-o0ggc
  url: https://universe.roboflow.com/brain-tumor-jolxi/brain-tumor-detection-o0ggc/dataset/2
  version: 2
  workspace: brain-tumor-jolxi
test: /content/brain-tumor-detection-2/test/images
train: /content/brain-tumor-detection-2/train/images
val: /content/brain-tumor-detection-2/valid/images
```

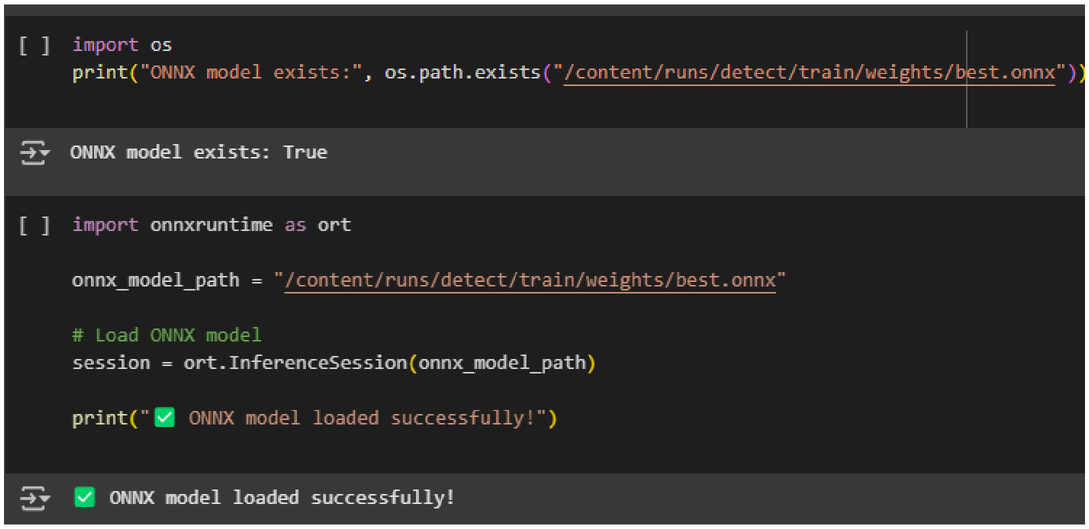


```
image 1/1 /content/brain-tumor-detection-2/test/images/Brain-Tumor-Overview_mp4-0017_
Speed: 2.4ms preprocess, 10.8ms inference, 1.4ms postprocess per image at shape (1, 3
Results saved to runs/detect/predict3
📁 Results saved in: runs/detect/predict3
✅ Found detected image: runs/detect/predict3/Brain-Tumor-Overview_mp4-0017_jpg.rf.4b
```



```python
import cv2
import matplotlib.pyplot as plt

# Path to the detected image
result_image_path = "/content/runs/detect/predict3/Brain-Tumor-Overview_mp4-001

# Load and display the image
image = cv2.imread(result_image_path)
if image is not None:
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis("off")
    plt.show()
else:
    print(f"❌ Error: Image not found at {result_image_path}")
```



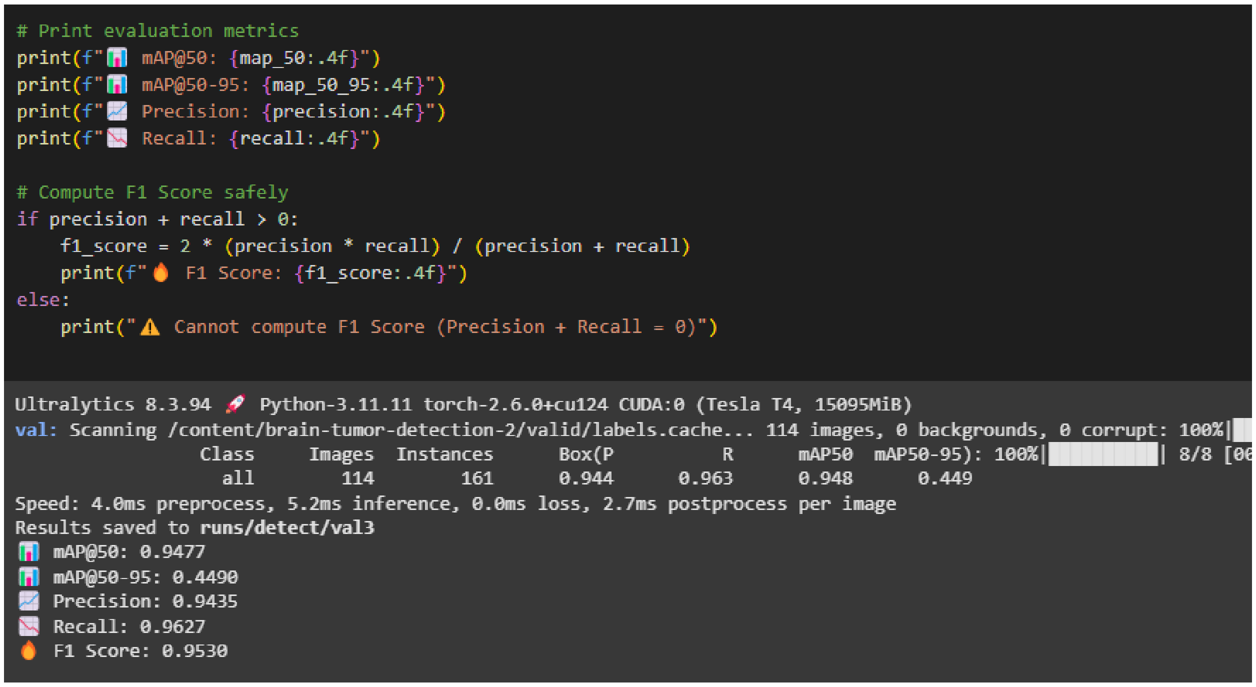


```python
[ ]  import os
     print("ONNX model exists:", os.path.exists("/content/runs/detect/train/weights/best.onnx"))
```

```
ONNX model exists: True
```

```python
[ ]  import onnxruntime as ort

     onnx_model_path = "/content/runs/detect/train/weights/best.onnx"

     # Load ONNX model
     session = ort.InferenceSession(onnx_model_path)

     print("✅ ONNX model loaded successfully!")
```

```
✅ ONNX model loaded successfully!
```



```
# Print evaluation metrics
print(f"📊 mAP@50: {map_50:.4f}")
print(f"📊 mAP@50-95: {map_50_95:.4f}")
print(f"📈 Precision: {precision:.4f}")
print(f"📉 Recall: {recall:.4f}")

# Compute F1 Score safely
if precision + recall > 0:
    f1_score = 2 * (precision * recall) / (precision + recall)
    print(f"🔥 F1 Score: {f1_score:.4f}")
else:
    print("⚠ Cannot compute F1 Score (Precision + Recall = 0)")
```

```
Ultralytics 8.3.94 🚀 Python-3.11.11 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
val: Scanning /content/brain-tumor-detection-2/valid/labels.cache... 114 images, 0 backgrounds, 0 corrupt: 100%|
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 8/8 [0
                   all        114        161      0.944      0.963      0.948      0.449
Speed: 4.0ms preprocess, 5.2ms inference, 0.0ms loss, 2.7ms postprocess per image
Results saved to runs/detect/val3
📊 mAP@50: 0.9477
📊 mAP@50-95: 0.4490
📈 Precision: 0.9435
📉 Recall: 0.9627
🔥 F1 Score: 0.9530
```

Evaluation Metrics Explanation and Discussion

1. Mean Average Precision (mAP)

- **mAP@50: 0.9477**
  - This indicates the average precision when using an Intersection over Union (IoU) threshold of 0.50. A high value (close to 1) suggests that the model performs well in detecting brain tumors with reasonable confidence.
- **mAP@50-95: 0.4490**
  - This measures the average precision across multiple IoU thresholds (from 0.50 to 0.95 in steps of 0.05). A lower value suggests that the model struggles with more precise bounding box overlaps.

2. Precision: 0.9435

- Precision measures how many of the predicted tumor detections are actually correct. A high precision value (94.35%) indicates a low false positive rate, meaning the model is confident about the detections it makes.

3. **Recall: 0.9627**
    - Recall measures how many actual tumors are correctly detected. A high recall value (96.27%) means that most tumors in the dataset are being successfully identified, with very few false negatives.
4. **F1 Score: 0.9530**
    - The F1 Score is the harmonic mean of precision and recall, balancing false positives and false negatives. A score of 95.30% indicates strong overall performance.
5. **Speed Metrics (per image)**
    - Preprocessing: 4.0 ms
    - Inference: 5.2 ms
    - Post-processing: 2.7 ms
    - These values indicate the model's efficiency in processing images. Faster inference times (5.2 ms per image) make the model suitable for real-time applications.

Discussion

- The high mAP@50 and recall values indicate that the model is highly capable of detecting tumors, with very few missing detections.
- The lower mAP@50-95 suggests that the model's bounding box predictions may need improvement in precise localization.
- Precision vs. Recall Tradeoff: The model has slightly higher recall than precision, meaning it is slightly more aggressive in detecting tumors. While this minimizes missed detections (false negatives), it could lead to some false positives.

- The speed metrics suggest that the model is fast enough for real-time or near-real-time processing, making it useful for medical imaging applications.

Potential Improvements

- Bounding Box Refinement: Improve localization by using higher-resolution images or fine-tuning anchor box sizes.

- Data Augmentation: Enhance training with rotation, scaling, or contrast adjustments to improve generalization.

- Hyperparameter Tuning: Experiment with learning rate, batch size, and epochs for optimized results.