

ScanInsight

December 8, 2024

1 ScanInsight: Tool for Brain Tumor Detection & Classification

Authors:
[Zain Ali](#), [Hallie Kinsey](#), and [Liam Richardson](#)

Contents

1 ScanInsight: Tool for Brain Tumor Detection & Classification	1
1.1 Introduction	2
1.2 Imports and environmental variables	2
1.3 EDA	3
1.3.1 Parsing Data	3
1.3.2 Creating Dataframe to better understand	5
1.3.3 Image Quantity	7
1.3.4 Extracting Metadata	8
1.3.5 Understanding Slices	8
1.3.6 Understanding Labels	24
1.4 Models	30
1.4.1 Model Selection and Initial Experiments	30
1.4.2 YOLOv8 - Initial Test	30
1.4.3 YOLOv11 - Main Run	69
1.4.4 Model Comparison - YOLOv8 vs YOLOv11	97
1.5 Dashboard UI API	100
1.5.1 Creating a models file	101
1.5.2 API information	124
1.6 Code for UI	127
1.7 Refrences	135

1.1 Introduction

ScanInsight is a prototype framework designed to assist medical professionals in detecting and classifying brain tumors from MRI scans using state-of-the-art YOLO-based object detection models. The overarching goal is to expedite the diagnostic process, enhance accuracy, and provide a user-friendly interface that medical staff can use to visualize and interpret model predictions.

In this project, we leverage a publicly available dataset of MRI images sourced from Kaggle ([Medical Image Dataset for Brain Tumor Detection](#)). The dataset includes a comprehensive collection of annotated MRI scans, pre-partitioned into training, validation, and test sets. It is ideally structured for training and evaluating object detection models, allowing us to focus on model selection, tuning, and result interpretation.

Our approach involves:

- **Exploratory Data Analysis (EDA):**

We start by exploring the dataset structure, image distributions, label integrity, and bounding box configurations. This exploration guides us in making informed decisions on model selection and hyperparameter tuning.

- **Model Training with YOLOv8 and YOLOv11:**

We fine-tune two state-of-the-art YOLO models—YOLOv8 and YOLOv11—on the given dataset. Both models predict the presence of three distinct tumor classes and generate bounding boxes around regions of interest. By comparing performance metrics (precision, recall, mAP), we identify the model that best fits the intended clinical application.

- **Performance Evaluation:**

We delve into precision-recall trade-offs, confusion matrices, and class-wise metrics to understand the model's strengths and weaknesses. This granular analysis ensures that we choose a model configuration that generalizes well and can guide effective clinical decisions.

- **User Interface & Deployment:**

As a final step, we integrate the trained model into a prototype UI dashboard, making it accessible to non-technical users. The interface, along with its visualizations and annotation features, can streamline the workflow in a clinical environment.

For further details and code, please refer to our GitHub repository: [AAI-521-Final](#). The repository contains scripts, notebooks, and documentation to replicate or extend this work. As a proof-of-concept, **ScanInsight** lays the groundwork for more extensive validation, refinement, and possible integration into clinical decision-support systems.

1.2 Imports and environmental variables

```
[2]: import cv2
import hashlib
import itertools
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import pickle
```

```

import random
import re
import seaborn as sns
import shutil
import tkinter as tk
import torch
import torchvision.transforms as transforms
import yaml
import math

from IPython.display import display, clear_output
from PIL import Image, ImageTk
from ipywidgets import Dropdown, FloatSlider, IntSlider, interact, interactive
from scipy.stats import pearsonr
from skimage.metrics import structural_similarity as ssim
from tkinter import Scrollbar, filedialog, messagebox, ttk, Text
from tqdm import tqdm
from ultralytics import YOLO

```

```

[4]: # Set base data paths - You can change these paths for your run
current_path = os.getcwd()
data_path = 'data'

data__full_path = os.path.join(current_path, data_path)

brain_yolo9_dataset = 'BrainTumorDetectionYolov9/BrainTumorDetectionYolov9'
brain_yolo8_dataset = 'TumorDetectionYolov8/OD8/Brain Tumor Detection'

dataset_base_url = os.path.join(current_path, data_path, brain_yolo9_dataset)
data_yaml_path = os.path.join(current_path, dataset_base_url, 'data.yaml')

# Predefined colors for labels (cycling colors if labels exceed this list)
label_colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0),
                 (255, 0, 255), (0, 255, 255), (128, 0, 128), (128, 128, 0)]

```

1.3 EDA

1.3.1 Parsing Data

```

[ ]: def resolve_paths_from_yaml(yaml_file_path):
    yaml_dir = os.path.dirname(yaml_file_path)
    with open(yaml_file_path, 'r') as file:
        yaml_data = yaml.safe_load(file)

    resolved_paths = {}
    for key, relative_path in yaml_data.items():
        if isinstance(relative_path, str):

```

```

        img_dir = os.path.normpath(os.path.join(yaml_file_path, relative_path))
        label_dir = img_dir.replace('/images', '/labels')

        if not os.path.exists(img_dir):
            raise FileNotFoundError(f"Image path does not exist: {img_dir}")
        if not os.path.exists(label_dir):
            raise FileNotFoundError(f"Label path does not exist: {label_dir}")

        resolved_paths[key] = {'images': img_dir, 'labels': label_dir}

    return resolved_paths

resolved_paths = resolve_paths_from_yaml(data_yaml_path)

for key, paths in resolved_paths.items():
    print(f"{key.capitalize()} Paths: {paths}")

```

Train Paths: {'images': '/root/notebooks/xray/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images', 'labels': '/root/notebooks/xray/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/labels'}

Val Paths: {'images': '/root/notebooks/xray/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/valid/images', 'labels': '/root/notebooks/xray/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/valid/labels'}

Test Paths: {'images': '/root/notebooks/xray/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/test/images', 'labels': '/root/notebooks/xray/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/test/labels'}

```
[ ]: def parse_data_yaml(yaml_path):
    with open(yaml_path, 'r') as file:
        data_yaml = yaml.safe_load(file)
    return data_yaml
```

```
[ ]: # Parsing the data.yaml file
data_yaml = parse_data_yaml(data_yaml_path)
print("Parsed data.yaml content:")
for key, value in data_yaml.items():
    print(f"{key}: {value}")
```

Parsed data.yaml content:

train: ../train/images

val: ../valid/images

test: ../test/images

nc: 3

names: ['label0', 'label1', 'label2']

1.3.2 Creating Dataframe to better understand

Processing the YAML dataset, matching images with labels, extracting metadata, and outputting a DataFrame.

```
[ ]: def create_dataset_df(data_yaml_path, data_yaml):
    data_entries = []
    yaml_dir = data_yaml_path
    counts = {'train': {'images': 0, 'labels': 0},
              'val': {'images': 0, 'labels': 0},
              'test': {'images': 0, 'labels': 0}}

    images_without_labels = []
    labels_without_images = []

    for data_type in ['train', 'val', 'test']:
        if data_type in data_yaml:
            rel_image_path = data_yaml[data_type]
            image_dir = os.path.normpath(os.path.join(yaml_dir, rel_image_path))
            label_dir = image_dir.replace('images', 'labels')

            if not os.path.exists(image_dir):
                print(f"Image directory does not exist: {image_dir}")
                continue
            if not os.path.exists(label_dir):
                print(f"Label directory does not exist: {label_dir}")
                continue

            # Get all image and label files and process them
            image_files = [f for f in os.listdir(image_dir) if f.lower().
             ↪endswith('.jpg', '.jpeg', '.png')]

            label_files = [f for f in os.listdir(label_dir) if f.lower().
             ↪endswith('.txt')]

            for image_file in image_files:
                abs_image_path = os.path.join(data_path, image_dir, image_file)
                abs_label_path = os.path.join(label_dir, os.path.
                 ↪splitext(image_file)[0] + '.txt')
                counts[data_type]['images'] += 1

                label_exists = os.path.exists(abs_label_path)
                if label_exists:
                    counts[data_type]['labels'] += 1
                else:
                    images_without_labels.append(rel_image_path)

            rel_image_path = os.path.relpath(abs_image_path, current_path)
```

```

        rel_label_path = os.path.relpath(abs_label_path, current_path)
        ↵if label_exists else None

            # Regex to match "volume_<number>" and "slice_<number>". We
            ↵want to extract the metadata
            volume, slice_num = None, None
            volume_match = re.search(r'volume_(\d+)', image_file, re.
            ↵IGNORECASE)
            slice_match = re.search(r'slice_(\d+)', image_file, re.
            ↵IGNORECASE)

            if volume_match:
                volume = volume_match.group(1)
            if slice_match:
                slice_num = slice_match.group(1)

            unique_hash_id = hashlib.md5(rel_image_path.encode()).
            ↵hexdigest()

            data_entries.append({
                'type': data_type,
                'image_path': rel_image_path,
                'label_path': rel_label_path,
                'volume': volume,
                'slice': slice_num,
                'id': unique_hash_id
            })

        # Check for labels without images
        for label_file in label_files:
            corresponding_image = os.path.join(image_dir, label_file.
            ↵replace('.txt', '.jpg'))
            if not os.path.exists(corresponding_image):
                labels_without_images.append(os.path.join(label_dir, ↵
            ↵label_file))

        df = pd.DataFrame(data_entries)

        # Print counts for each data type
        for data_type, count in counts.items():
            print(f"{data_type.capitalize()} - Total Images: {count['images']}, ↵
        ↵Total Labels: {count['labels']}")

        # Print images without labels
        if images_without_labels:
            print("\nImages without labels:")

```

```

    for img in images_without_labels:
        print(f" - {img}")

    # Print labels without images
    if labels_without_images:
        print("\nLabels without images:")
        for lbl in labels_without_images:
            print(f" - {lbl}")

    return df, images_without_labels, labels_without_images

```

1.3.3 Image Quantity

```
[ ]: dataset_df, images_without_labels, labels_without_images = create_dataset_df(data_yaml_path, data_yaml)

# Just validation in case of some messup:
if dataset_df.empty:
    print("Dataset DataFrame is empty. Please check the dataset paths.")
else:
    display(dataset_df.head())

Train - Total Images: 6931, Total Labels: 6930
Val - Total Images: 1980, Total Labels: 1980
Test - Total Images: 990, Total Labels: 990

Images without labels:
- BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/volume_244_slic
e_53.jpg.rf.21e910587f9329e96fa00056018c1eb1.jpg

      type          image_path \
0  train  BrainTumorDetectionYolov9/BrainTumorDetectionY...
1  train  BrainTumorDetectionYolov9/BrainTumorDetectionY...
2  train  BrainTumorDetectionYolov9/BrainTumorDetectionY...
3  train  BrainTumorDetectionYolov9/BrainTumorDetectionY...
4  train  BrainTumorDetectionYolov9/BrainTumorDetectionY...

                    label_path  volume  slice \
0  BrainTumorDetectionYolov9/BrainTumorDetectionY...     46     61
1  BrainTumorDetectionYolov9/BrainTumorDetectionY...   198     79
2  BrainTumorDetectionYolov9/BrainTumorDetectionY...   323     31
3  BrainTumorDetectionYolov9/BrainTumorDetectionY...   151     60
4  BrainTumorDetectionYolov9/BrainTumorDetectionY...   320    105

      id
0  508ff881e6ee8c6010f040a98e0b6ffa
1  cfbafb37ece154765249a28b00272560
2  5ec689cac88f4ad262d8648b7d94afad
```

```
3 d7f202943c6646502c6ef087781629a5
4 764366c3d1640db6cf0a55e6ee23a55
```

9901 total images, with one without any label, so a total of 9900 images with 6930 in training.

1.3.4 Extracting Metadata

```
[ ]: # Making sure its a number, because will be graphing this later
dataset_df['volume'] = dataset_df['volume'].astype(int)
dataset_df['slice'] = dataset_df['slice'].astype(int)

[10]: print("Total Number of Volumes(different patients):", len(dataset_df["volume"].
    ↴unique()))
```

Total Number of Volumes(different patients): 369

1.3.5 Understanding Slices

A slice is one image instance.

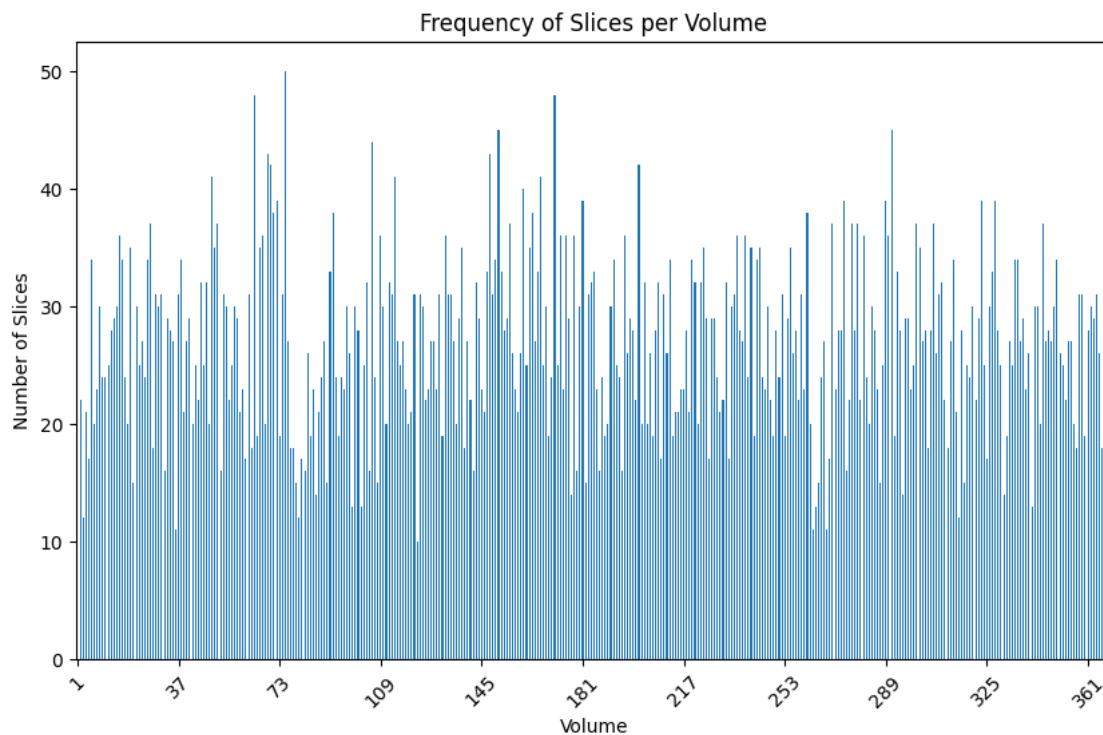
Quantity of slices per volume

```
[ ]: # Function to plot the frequency of slices for a specific volume or all volumes
def plot_slice_frequency(df, volume_number = None):
    slice_counts = df.groupby('volume')['slice'].nunique()
    if volume_number is not None:
        if volume_number in slice_counts.index:
            slice_counts = slice_counts.loc[[volume_number]]
            title = f"Frequency of Slices for Volume {volume_number}"
        else:
            print(f"Volume {volume_number} not found in the dataset.")
            return
    else:
        all_volumes = range(slice_counts.index.min(), slice_counts.index.max() ↴
        + 1)
        slice_counts = slice_counts.reindex(all_volumes, fill_value=0)
        title = "Frequency of Slices per Volume"

    plt.figure(figsize=(10, 6))
    slice_counts.plot(kind='bar')
    plt.title(title)
    plt.xlabel("Volume")
    plt.ylabel("Number of Slices")
    plt.xticks(rotation=45)
    if len(slice_counts) > 20:
        plt.xticks(ticks=range(0, len(slice_counts), len(slice_counts) // 10))

    plt.show()
```

```
plot_slice_frequency(dataset_df)
```



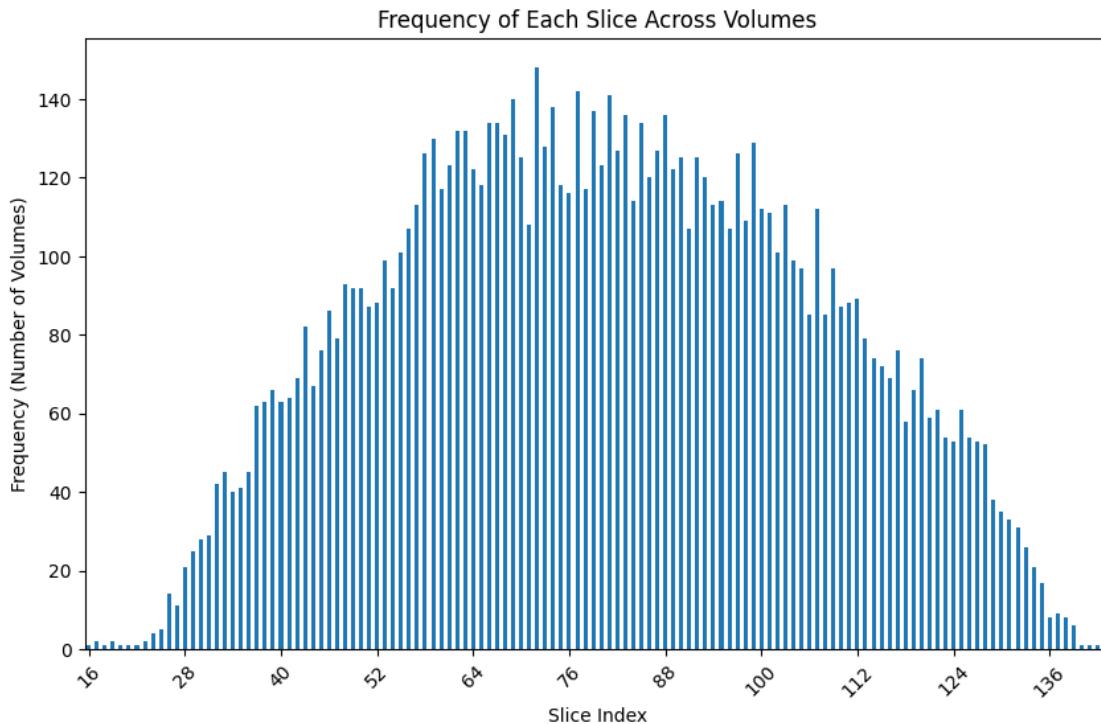
Some volumes have higher slices and some low.

Frequency of slices over all

```
[ ]: # Count the occurrences of each slice across all volumes
def plot_slice_occurrence_frequency(df):
    slice_occurrence_counts = df['slice'].value_counts().sort_index()
    plt.figure(figsize=(10, 6))
    slice_occurrence_counts.plot(kind='bar')

    plt.title("Frequency of Each Slice Across Volumes")
    plt.xlabel("Slice Index")
    plt.ylabel("Frequency (Number of Volumes)")
    plt.xticks(rotation=45)
    if len(slice_occurrence_counts) > 20:
        plt.xticks(ticks=range(0, len(slice_occurrence_counts), max(1, len(slice_occurrence_counts) // 10)))
    plt.show()

plot_slice_occurrence_frequency(dataset_df)
```



The middle of the brain, if we assume that the area represented by the same slice number is the same. This assumption was later proven wrong, even though the slices seem similar, there is enough evidence to say they are not the exact same region. They are close by.

Order of Slices Another thing we are trying to check is if it's safe to delete the images without any labels.

```
[ ]: # Function to count slices for a specific volume or all volumes if none is specified
def count_slices_per_volume(df, volume_number = None, detailed = False):
    volume_groups = df.groupby('volume')['slice'].apply(list)
    slice_counts = df.groupby('volume')['slice'].nunique()

    if volume_number is not None:
        if volume_number in slice_counts.index:
            print(f"Volume {volume_number}: {slice_counts[volume_number]} slices")
            if detailed:
                print("Detailed slice numbers within this volume:")
                print(f" - Slices: {sorted(volume_groups[volume_number])}")
        else:
            print(f"Volume {volume_number} not found in the dataset.")
    else:
```

```

print("Number of slices within each volume:")
for volume, count in slice_counts.items():
    print(f"Volume {volume}: {count} slices")
    if detailed:
        print(f" - Slices: {sorted(volume_groups[volume])}")
print()

# Print slice count for volume 46
count_slices_per_volume(dataset_df, volume_number=244, detailed = True)

```

Volume 244: 35 slices
Detailed slice numbers within this volume:
- Slices: [41, 46, 47, 49, 53, 54, 55, 59, 62, 65, 72, 73, 76, 77, 79, 82, 83, 87, 89, 91, 92, 96, 97, 99, 101, 103, 104, 107, 109, 111, 113, 116, 117, 118, 121]

So if we delete “Images without labels”, we should be fine, as the slices are not equadistance. Initially the idea was if the slices are in order, then we might run into issue of missing data in the middle, which might lead to deleting a full volume if there is only few values. Which in our case was only one image, and it would have mean deleting only one volume.

Deleting the unneeded images:

```
[ ]: print("Images to delete:", images_without_labels)
dataset_df = dataset_df[dataset_df['label_path'].notna()].reset_index(drop=True)
```

Images to delete: ['BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/volume_244_slice_53.jpg.rf.21e910587f9329e96fa00056018c1eb1.jpg']

Breakpoint - dataset_df The breakpoints are given to give any interested body to interact with our code, without have to run the full notebook. This breakpoint is for `dataset_df` which contains all the paths for the images and labels relative to `current_path`. It also includes volume and slice number and the unique id of the image/label.

```
[6]: # Always run this cell
file_path = os.path.join(current_path, data_path, ↴'brain_tumor_detection_path_data.csv')
```

```
[ ]: # Run this to save your performance.
# Saving in case if have to pull info. Run this cell to save
dataset_df.to_csv(file_path, index=False)
```

```
[7]: # Run this cell to read the data and continue after this point. You might also want to run the first import cell first.
dataset_df = pd.read_csv(file_path)
```

Slice Visualization

```
[ ]: def load_image(filepath, grayscale = False, size= False):
    image = cv2.imread(filepath)

    if image is None:
        raise FileNotFoundError(f"Image not found at path: {filepath}")

    if grayscale:
        image = convert_to_grayscale(image)

    if size:
        return cv2.resize(image, size)
    return image

# creating the 3D Tumor Mask with Color Coding by Class
def load_label(filepath):
    labels = []
    if not os.path.exists(filepath):
        return labels
    with open(filepath, 'r') as file:
        for line in file:
            parts = line.strip().split()
            if len(parts) == 5:
                labels.append({
                    'class_id': int(parts[0]),
                    'x_center': float(parts[1]),
                    'y_center': float(parts[2]),
                    'width': float(parts[3]),
                    'height': float(parts[4])
                })
    return labels

def draw_labels(image, labels):
    image_h, image_w = image.shape[:2]

    for label in labels:
        class_id = label['class_id']
        x_center = label['x_center'] * image_w
        y_center = label['y_center'] * image_h
        width = label['width'] * image_w
        height = label['height'] * image_h

        x1 = int(x_center - width / 2)
        y1 = int(y_center - height / 2)
        x2 = int(x_center + width / 2)
        y2 = int(y_center + height / 2)

        color = label_colors[class_id % len(label_colors)]
```

```

        cv2.rectangle(image, (x1, y1), (x2, y2), color, 1)
        cv2.putText(image, f"Class {class_id}", (x1, y1 - 5),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.4, color, 1)
    return image

def convert_to_grayscale(image):

    grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    grayscale_image_bgr = cv2.cvtColor(grayscale_image, cv2.COLOR_GRAY2BGR)
    return grayscale_image_bgr

def display_images_labels_grey(image, labels, show_both= True, figsize=(12, 6)):

    annotated_color_image = draw_labels(image.copy(), labels)

    if show_both:
        grayscale_image = convert_to_grayscale(image)
        annotated_grayscale_image = draw_labels(grayscale_image.copy(), labels)

        plt.figure(figsize=figsize)
        plt.subplot(1, 2, 1)
        plt.imshow(cv2.cvtColor(annotated_color_image, cv2.COLOR_BGR2RGB))
        plt.title("Color Image with Labels")
        plt.axis('off')
        plt.subplot(1, 2, 2)
        plt.imshow(cv2.cvtColor(annotated_grayscale_image, cv2.COLOR_BGR2RGB))
        plt.title("Grayscale Image with Labels")
        plt.axis('off')
        plt.show()
    else:
        plt.figure(figsize=(figsize[0] // 2, figsize[1]))
        plt.imshow(cv2.cvtColor(annotated_color_image, cv2.COLOR_BGR2RGB))
        plt.title("Image with Labels")
        plt.axis('off')
        plt.show()

def select_entry(dataset_df, location):
    if dataset_df.empty:
        raise ValueError("Dataset is empty.")
    if location == "random":
        sample_entry = dataset_df.sample(n=1).iloc[0]
    else:
        if location < 0 or location >= len(dataset_df):
            raise IndexError("Location index out of range.")
        sample_entry = dataset_df.iloc[location]
    return {
        'image_path': sample_entry['image_path'],

```

```

        'label_path': sample_entry['label_path']
    }

def display_sample_entry(image_path, label_path, show_both=False, ↴
    ↪grayscale=False):
    print("\nSelected sample entry:")
    print(f"Image path: {image_path}")
    print(f"Label path: {label_path}")

    image = load_image(image_path, grayscale=grayscale)
    labels = load_label(label_path) if label_path else []
    display_images_labels_grey(image, labels, show_both=show_both)

def check_image(dataset_df, location="random", show_both=False, ↴
    ↪grayscale=False):
    try:
        sample_entry = select_entry(dataset_df, location)
        image_path = os.path.join(data_path, sample_entry['image_path'])
        label_path = os.path.join(data_path, sample_entry['label_path'])
        display_sample_entry(image_path, label_path, show_both=show_both, ↴
            ↪grayscale=grayscale)
    except (ValueError, IndexError) as e:
        print(e)

```

Lets look at what these images look like with label and data on top of it.

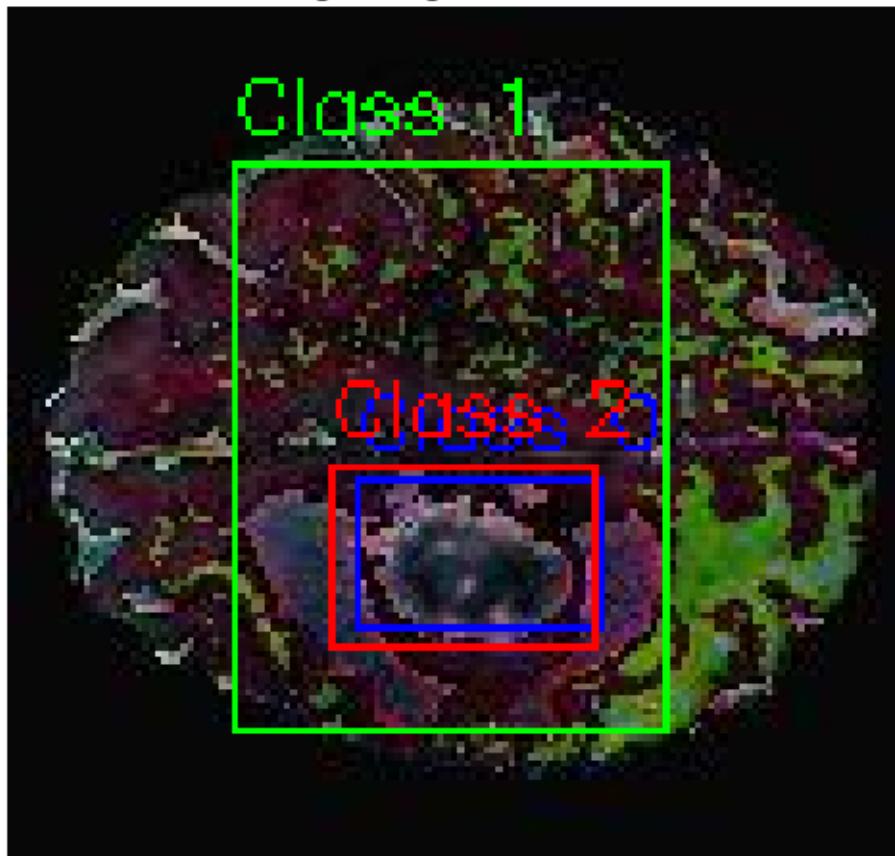
[170]: check_image(dataset_df)

```

Selected sample entry:
Image path: BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/vol
ume_25_slice_109.jpg.rf.d7f6efafbc604a469359493a59601f8b.jpg
Label path: BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/labels/vol
ume_25_slice_109.jpg.rf.d7f6efafbc604a469359493a59601f8b.txt

```

Image Image with Labels



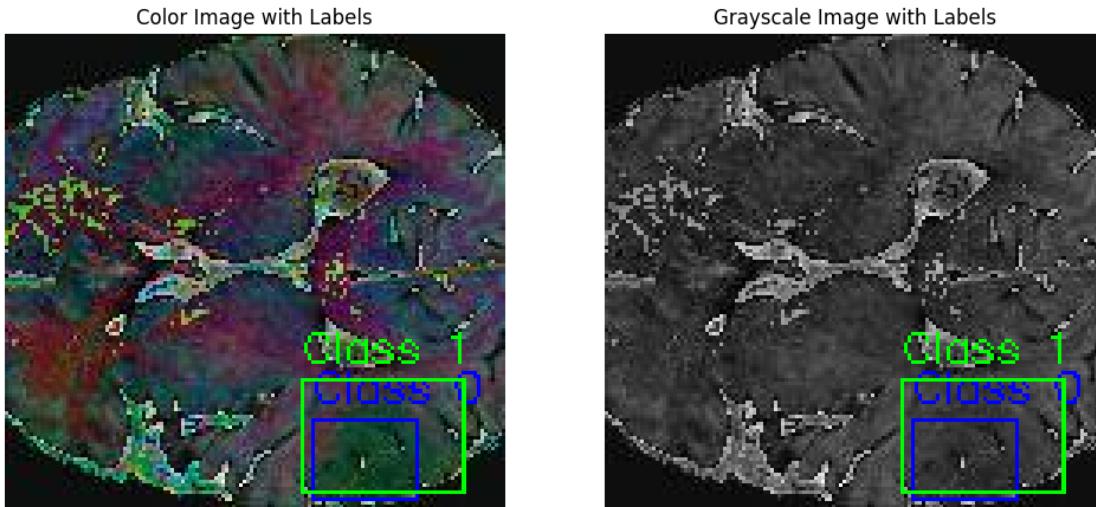
Want to check the image within grey scale as well.

```
[ ]: check_image(dataset_df, show_both=True)
```

Selected sample entry:

Image path: BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/volume_87_slice_77.jpg.rf.9d34931fa6a819b251b6579931c1f0b3.jpg

Label path: BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/labels/volume_87_slice_77.jpg.rf.9d34931fa6a819b251b6579931c1f0b3.txt



Color does seem to contain more information and at a glance it seems to be important.

```
[ ]: def reconstruct_volume(dataset_df, volume_number):
    volume_df = dataset_df[dataset_df['volume'] == volume_number].
    ↪sort_values(by='slice').reset_index(drop=True)
    slices = []
    for _, row in volume_df.iterrows():
        image_path = os.path.join(data_path, row['image_path'])
        image = load_image(image_path)
        slices.append(image)
    volume = np.stack(slices, axis=0)
    return volume, volume_df

def create_tumor_mask(volume_df, volume_shape):
    tumor_mask = np.zeros(volume_shape[:3], dtype=np.uint8)
    class_masks = {}

    for slice_index, row in volume_df.iterrows():
        label_path = os.path.join(data_path, row['label_path'])
        labels = load_label(label_path)
        for label in labels:
            class_id = label['class_id']
            x_center = label['x_center']
            y_center = label['y_center']
            width = label['width']
            height = label['height']
            img_height, img_width = volume_shape[1], volume_shape[2]
            x1 = int((x_center - width / 2) * img_width)
            y1 = int((y_center - height / 2) * img_height)
            x2 = int((x_center + width / 2) * img_width)
```

```

        y2 = int((y_center + height / 2) * img_height)
        x1, y1, x2, y2 = max(0, x1), max(0, y1), min(img_width - 1, x2), ↵
        ↵min(img_height - 1, y2)

    if class_id not in class_masks:
        class_masks[class_id] = np.zeros(volume_shape[:3], dtype=np.
    ↵uint8)
        class_masks[class_id][slice_index, y1:y2, x1:x2] = 1

    return tumor_mask, class_masks

def display_slice(slice_index, show_overlay=True, image_size=10.0):
    mri_slice = volume[slice_index]
    overlay = mri_slice.copy()
    if show_overlay:
        for class_id, class_mask in class_masks.items():
            color = label_colors[class_id % len(label_colors)]
            color_bgr = (color[2], color[1], color[0])
            red_overlay = np.zeros_like(mri_slice)
            red_overlay[..., 0] = color[2]
            red_overlay[..., 1] = color[1]
            red_overlay[..., 2] = color[0]
            alpha = 0.5
            mask_slice = class_mask[slice_index]
            overlay = np.where(mask_slice[..., None],
                               (1 - alpha) * overlay + alpha * red_overlay,
                               overlay)

    plt.figure(figsize=(image_size, image_size))
    plt.imshow(cv2.cvtColor(overlay.astype(np.uint8), cv2.COLOR_BGR2RGB))
    plt.title(f'Volume {selected_volume}, Slice {slice_index}')
    plt.axis('off')
    legend_patches = [plt.Line2D([0], [0], marker='o', color='w', ↵
    ↵markerfacecolor=(color[0]/255, color[1]/255, color[2]/255), markersize=10, ↵
    ↵label=f'Class {class_id}')
                      for class_id, color in enumerate(label_colors) if ↵
    ↵class_id in class_masks]
    plt.legend(handles=legend_patches, loc='upper right', title="Classes")
    plt.show()

```

```
[ ]: unique_volumes = sorted(dataset_df['volume'].unique())
volume_dropdown = Dropdown(
    options=unique_volumes,
    value=unique_volumes[0],
    description='Select Volume:'
)
```

```

# Slider for controlling image size
size_slider = FloatSlider(
    value=10.0,
    min=5.0,
    max=20.0,
    step=0.5,
    description='Image Size:'
)

def update_volume(volume_number):
    global volume, volume_df, tumor_mask, class_masks, selected_volume, max_slice
    selected_volume = volume_number
    volume, volume_df = reconstruct_volume(dataset_df, volume_number)
    tumor_mask, class_masks = create_tumor_mask(volume_df, volume.shape)
    max_slice = volume.shape[0] - 1

    # Clear the output to remove the previous interactive widget
    clear_output(wait=True)

    # Recreating the slice slider with the updated max_slice value
    slice_slider = IntSlider(min=0, max=max_slice, step=1, value=0, description='Slice Index')
    display(volume_dropdown)
    display(interactive(display_slice,
        slice_index=slice_slider,
        show_overlay=True,
        image_size=size_slider))

# Set initial values
selected_volume = volume_dropdown.value
volume, volume_df = reconstruct_volume(dataset_df, selected_volume)
tumor_mask, class_masks = create_tumor_mask(volume_df, volume.shape)
max_slice = volume.shape[0] - 1

# Displaying dashboard
volume_dropdown.observe(lambda change: update_volume(change.new), names='value')
display(volume_dropdown)
display(interactive(display_slice,
    slice_index=IntSlider(min=0, max=max_slice, step=1, value=0),
    show_overlay=True,
    image_size=size_slider))

```

```

Dropdown(description='Select Volume:', index=9, options=(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16..)

interactive(children=(IntSlider(value=0, description='Slice Index', max=23), Checkbox(value=True, description=...

```

A recording of this visual dashboard is saved in `/data/data-exploration-visual.mov`.

This was essentially build for all of the group members to have some visuals to the data, and also to understand what will need to be predicted and what does the end result need to look like.

This was also where we thought about stacked solution, there rather than just processing one image at a time, we process a volume and set of stack images. This can be useful for physicians to understand the scope of the issue in 3d.

Question: Do all the slice numbers (not index) contain the same subsection of the brain? As we were doing this EDA, we ran into another question: Do all the slice numbers (not index) contain the same subsection of the brain?

This was a followup from the “Frequency of slices over all” section, where we wanted to confirm the assumption.

```
[ ]: def display_slice_comparisons(dataset_df, slice_number, top_n=6):
    slice_df = dataset_df[dataset_df['slice'] == slice_number]
    top_slice_images = slice_df.sort_values(by='volume').head(top_n)
    n_cols = 3
    n_rows = math.ceil(top_n / n_cols)
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5 * n_rows)) #_
    ↪Adjust the height as needed

    if n_rows == 1:
        axes = np.array([axes]) if isinstance(axes, plt.Axes) else axes #_
    ↪Ensure axes is an array
        axes = axes.flatten()

    for i, (_, row) in enumerate(top_slice_images.iterrows()):
        image_path = os.path.join(data_path, row['image_path'])
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # Convert BGR to RGB_
    ↪for proper display with matplotlib

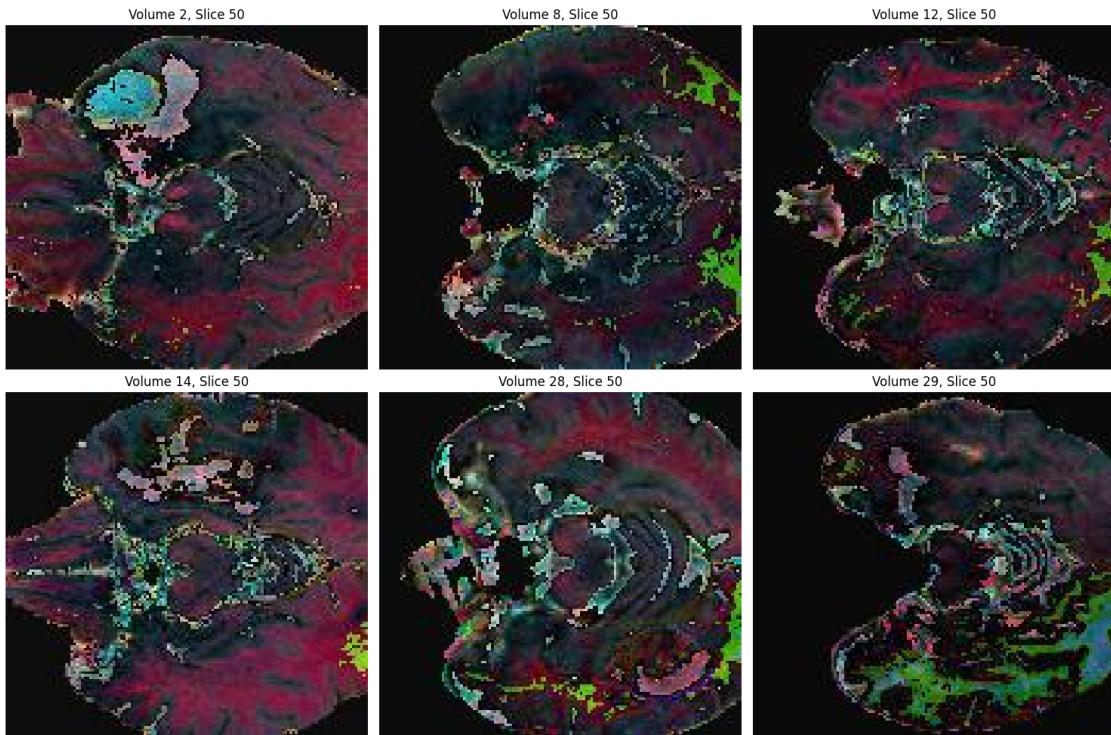
        row_idx = i // n_cols
        col_idx = i % n_cols

        if n_rows == 1:
            axes[col_idx].imshow(image)
            axes[col_idx].set_title(f"Volume {row['volume']}, Slice_"
    ↪{row['slice']}")_
            axes[col_idx].axis('off')
        else:
            axes[row_idx, col_idx].imshow(image)
            axes[row_idx, col_idx].set_title(f"Volume {row['volume']}, Slice_"
    ↪{row['slice']}")_
            axes[row_idx, col_idx].axis('off')
```

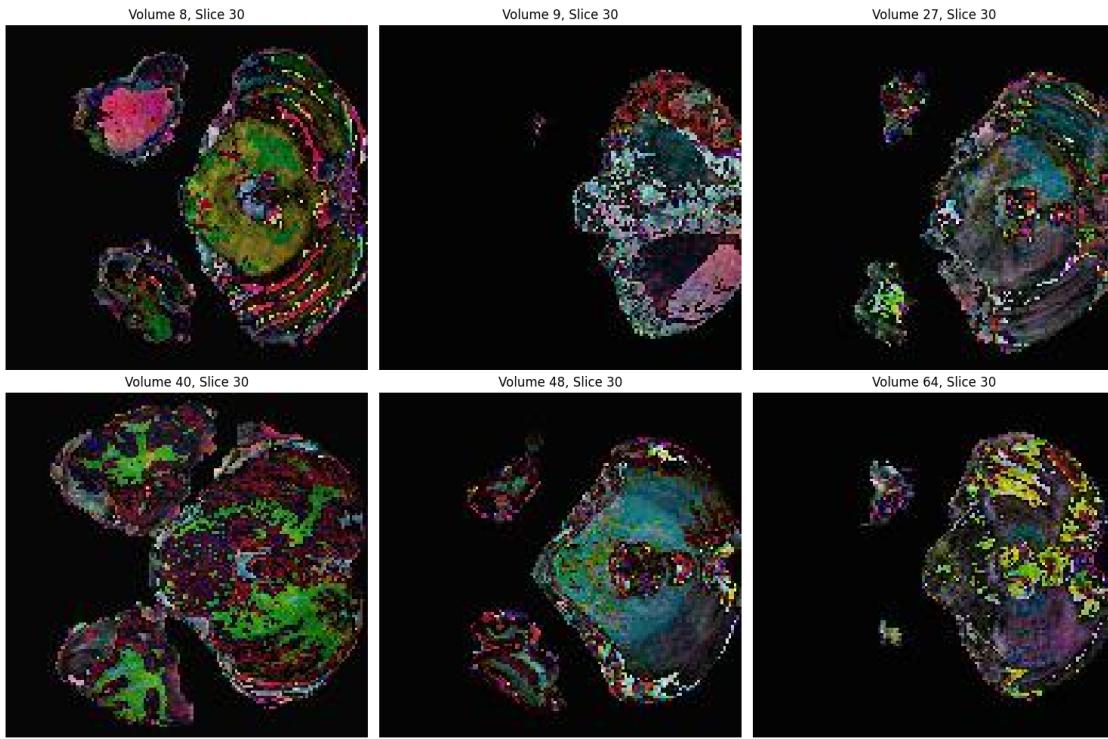
```
for j in range(i + 1, n_rows * n_cols):
    if n_rows == 1:
        axes[j].axis('off')
    else:
        row_idx = j // n_cols
        col_idx = j % n_cols
        axes[row_idx, col_idx].axis('off')

plt.tight_layout()
plt.show()
```

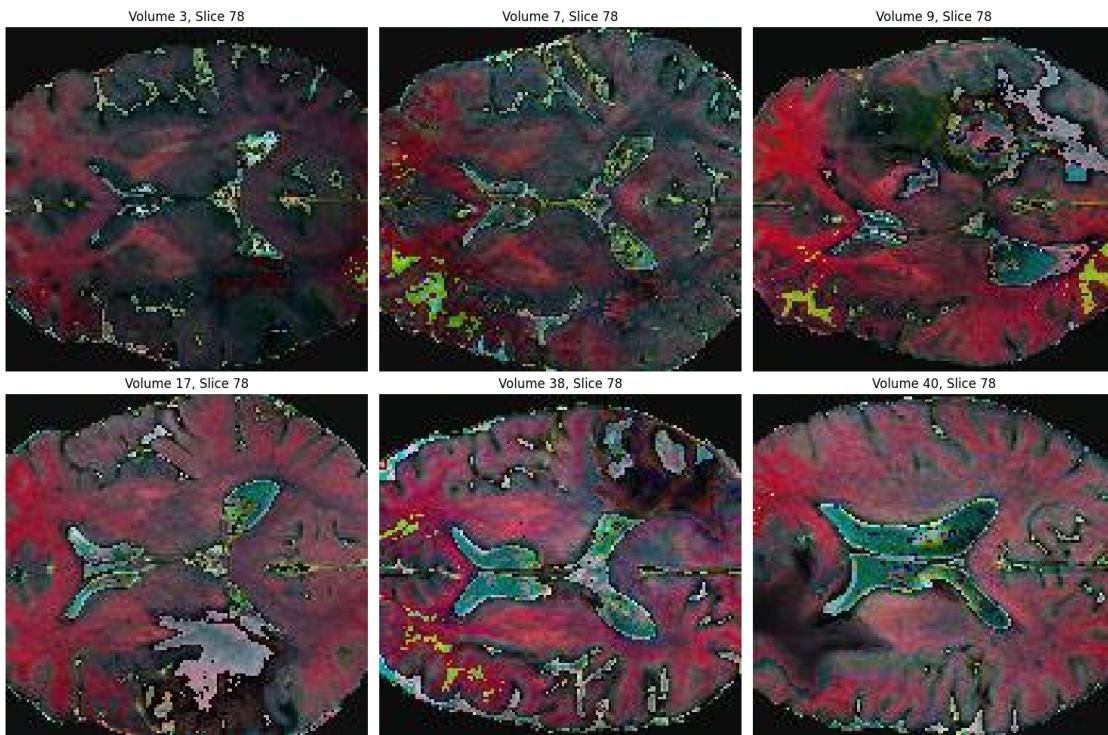
```
[68]: display_slice_comparisons(dataset_df, slice_number = 50, top_n= 6)
```



```
[72]: display_slice_comparisons(dataset_df, slice_number = 30, top_n= 6)
```



```
[73]: display_slice_comparisons(dataset_df, slice_number = 78, top_n= 6)
```



From looking at visually, there seems to be a similarity within same `slice_number`, but one issue is the distribution of frequency of Slices can introduce some bias.

```
[ ]: def calculate_ssim(image1, image2):
    return ssim(image1, image2)

def calculate_histogram_similarity(image1, image2):
    hist1 = cv2.calcHist([image1], [0], None, [256], [0, 256])
    hist2 = cv2.calcHist([image2], [0], None, [256], [0, 256])
    hist1 = cv2.normalize(hist1, hist1).flatten()
    hist2 = cv2.normalize(hist2, hist2).flatten()
    return pearsonr(hist1, hist2)[0] # Returns the Pearson correlation coefficient

def analyze_random_slices_for_brain_subsections(dataset_df, num_samples=500, ↴
    ↴downsample_size=False, ssim_threshold=0.5):
    ssim_scores = []
    hist_scores = []
    unique_slices = dataset_df['slice'].unique().tolist()

    while len(ssim_scores) < num_samples:
        slice_number = random.choice(unique_slices)
        slice_df = dataset_df[dataset_df['slice'] == slice_number]
        image_paths = slice_df['image_path'].tolist()

        if len(image_paths) < 2:
            continue

        images = [cv2.cvtColor(load_image(path, size=downsample_size), cv2.
    ↴COLOR_BGR2GRAY) for path in image_paths[:2]]

        hist_score = calculate_histogram_similarity(images[0], images[1])

        if hist_score > ssim_threshold:
            ssim_score = calculate_ssim(images[0], images[1])
            ssim_scores.append(ssim_score)
            hist_scores.append(hist_score)
        else:
            ssim_scores.append(0) # Mark as low similarity
            hist_scores.append(hist_score)

    # Calculate statistics for the scores
    avg_ssim = np.mean(ssim_scores)
    std_ssim = np.std(ssim_scores)
    avg_hist = np.mean(hist_scores)
```

```

    std_hist = np.std(hist_scores)

    print("Analysis Summary:")
    print(f"  Average SSIM: {avg_ssim:.4f} (Std Dev: {std_ssim:.4f})")
    print(f"  Average Histogram Similarity: {avg_hist:.4f} (Std Dev: {std_hist:.
        .4f})")

# Plotting the distribution of similarity scores
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(ssim_scores, bins=10, edgecolor='black')
plt.title("SSIM Score Distribution")
plt.xlabel("SSIM Score")
plt.ylabel("Frequency")

plt.subplot(1, 2, 2)
plt.hist(hist_scores, bins=10, edgecolor='black')
plt.title("Histogram Similarity Distribution")
plt.xlabel("Histogram Similarity")
plt.ylabel("Frequency")

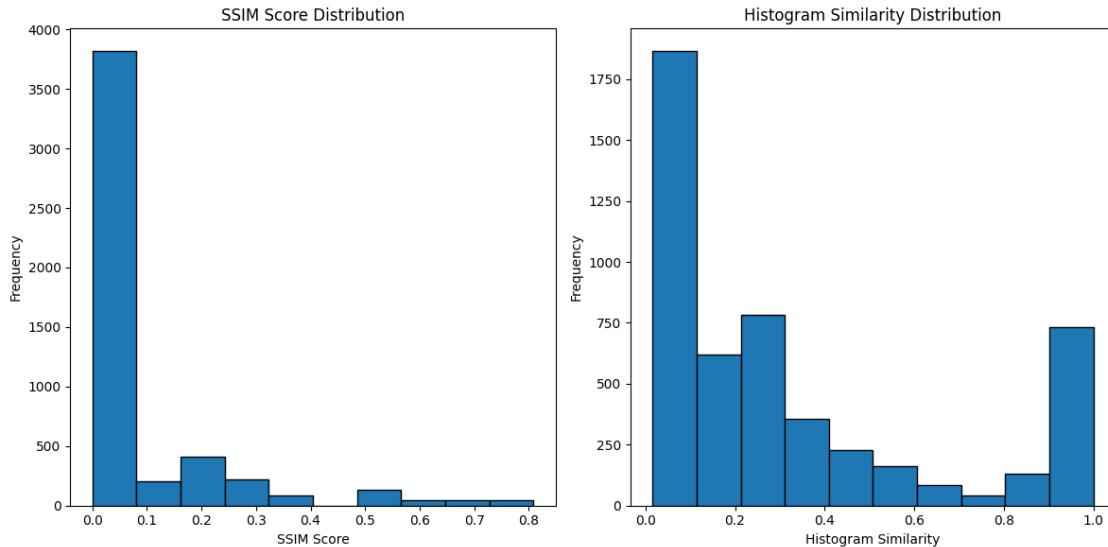
plt.tight_layout()
plt.show()

analyze_random_slices_for_brain_subsections(dataset_df, num_samples=5000)

```

Analysis Summary:

Average SSIM: 0.0745 (Std Dev: 0.1601)
Average Histogram Similarity: 0.3342 (Std Dev: 0.3304)



The analysis indicates that the slices likely represent different subsections of the brain rather than a consistent, single region. It also can mean each image even at same slice is quite different even if it is the same region. The low average SSIM (0.0745) and high variability in both SSIM and histogram similarity scores suggest substantial structural and intensity differences between same slices among different volumes. The distribution of scores, with most values concentrated near zero, further supports the conclusion that these slices capture varying anatomical sections rather than the same subsection across different slice numbers.

Another reason behind this analysis beside checking for assumptions, was to check if there is a way to take a controled image and do some ML comparision. If a slice is the same part of the brain, it could just be compared with a healthy brain and we should be able to know if there is a tumor. This would have been easier to implement if we had a high comparison, but we do not.

1.3.6 Understanding Labels

Max label amount

```
[ ]: dataset_df['class_nums'] = dataset_df['label_path'].apply(
    lambda path: len(set(int(line.split()[0])) for line in open(path) if os.path.
    exists(path)))
)

# Find the maximum number of unique classes across all label files
max_class_count = dataset_df['class_nums'].max()
print(f"Maximum number of unique classes across all labels: {max_class_count}")
```

Maximum number of unique classes across all labels: 3

Creating full dataframe dataset We decided to collect all the information about the labels and overlay it within our `dataset_df`.

```
[ ]: expanded_rows = []

for _, row in tqdm(dataset_df.iterrows(), total=len(dataset_df), u
    desc="Processing images"):
    image_path = os.path.join(data_path, row['image_path'])
    label_path = os.path.join(data_path, row['label_path'])
    volume = row['volume']
    slice_num = row['slice']
    data_category = row['type']

    # Check if label file exists and proceed if it does
    if os.path.exists(label_path):
        with open(label_path, 'r') as file:
            lines = file.readlines()
            # For each class label line, extract details and add a new row to
            expanded_rows
            for line in lines:
                parts = line.strip().split()
```

```

    if len(parts) != 5:
        continue # Skip malformed lines

    # Parse the class information and bounding box data
    class_id = int(parts[0])
    x_center = float(parts[1])
    y_center = float(parts[2])
    width = float(parts[3])
    height = float(parts[4])

    # Create a dictionary representing a single row for this class
    row_data = {
        'data_category': data_category,
        'image_path': image_path,
        'volume': volume,
        'slice': slice_num,
        'class_id': class_id,
        'x_center': x_center,
        'y_center': y_center,
        'width': width,
        'height': height,
    }

    # Append the row data to expanded_rows
    expanded_rows.append(row_data)

expanded_df = pd.DataFrame(expanded_rows)

```

Processing images: 100% | 9900/9900 [00:00<00:00, 12449.98it/s]

```

[141]: # Add one-hot encoded columns for each class (outside the main loop)
for cid in range(max_class_count + 1):
    expanded_df[f'class_{cid}'] = (expanded_df['class_id'] == cid).astype(int)

# Display the expanded DataFrame
print("Expanded DataFrame with one row per class and bounding box details:")
display(expanded_df.head())

```

Expanded DataFrame with one row per class and bounding box details:

	data_category	image_path	volume	\
0	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	46	
1	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	198	
2	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	198	
3	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	198	
4	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	323	

	slice	class_id	x_center	y_center	width	height	class_0	class_1	\
0	61	1	0.428058	0.674242	0.525180	0.318182	0	1	

```

1      79      0  0.467626  0.219697  0.187050  0.151515      1      0
2      79      1  0.500000  0.219697  0.467626  0.393939      0      1
3      79      2  0.460432  0.174242  0.230216  0.272727      0      0
4      31      1  0.366906  0.215909  0.043165  0.053030      0      1

  class_2  class_3
0      0      0
1      0      0
2      0      0
3      1      0
4      0      0

```

Breakpoint - expanded_df This breakpoint is for `expanded_df` which contains all the data from the `dataset_df`, and overlays the data for each label. Each row is associated to one label per image. If an image has multiple labels, there will be multiple rows of data.

```
[43]: file_path_expanded_df = os.path.join(current_path, data_path, 'expanded_brain_tumor_annotations.csv')
```

```
[10]: # Saving in case if have to pull info. Run this cell to save
expanded_df.to_csv(file_path_expanded_df, index=False)
```

```
[44]: # Run this cell to read the data from the CSV
expanded_df = pd.read_csv(file_path_expanded_df)

# Check the contents to ensure it's loaded correctly
print(expanded_df.head())
```

	data_category	image_path	volume	\
0	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	46	
1	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	198	
2	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	198	
3	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	198	
4	train	BrainTumorDetectionYolov9/BrainTumorDetectionY...	323	

	slice	class_id	x_center	y_center	width	height	class_0	class_1	\
0	61	1	0.428058	0.674242	0.525180	0.318182	0	1	
1	79	0	0.467626	0.219697	0.187050	0.151515	1	0	
2	79	1	0.500000	0.219697	0.467626	0.393939	0	1	
3	79	2	0.460432	0.174242	0.230216	0.272727	0	0	
4	31	1	0.366906	0.215909	0.043165	0.053030	0	1	

	class_2	class_3
0	0	0
1	0	0
2	0	0
3	1	0
4	0	0

Removing bias Moving forward, we wanted to use `train` data to do the EDA, so there is no bias being introduced within our analysis from test of validation.

```
[45]: # Separate DataFrames by data_category type
train_df = expanded_df[expanded_df['data_category'] == 'train'].
    ↪reset_index(drop=True)
test_df = expanded_df[expanded_df['data_category'] == 'test'].
    ↪reset_index(drop=True)
val_df = expanded_df[expanded_df['data_category'] == 'val'].
    ↪reset_index(drop=True)
```

```
[53]: class_counts = train_df['class_id'].value_counts()

total_count = class_counts.sum()
class_percentages = (class_counts / total_count) * 100

# Print statements for displaying the results
print("Class counts and percentages in train_df:")
for class_id, count in class_counts.items():
    percentage = class_percentages[class_id]
    print(f"Label Class {class_id}: {count} instances ({percentage:.2f}%)")
```

Class counts and percentages in train_df:
Label Class 1: 6745 instances (44.84%)
Label Class 0: 4238 instances (28.18%)
Label Class 2: 4058 instances (26.98%)

```
[56]: max_count = class_counts.max()
less_than_highest_percentage = ((max_count - class_counts) / max_count) * 100

# Difference from the highest class count (in percentage)
print("\nDifference from the highest class count (in percentage):")
for class_id, difference_percentage in less_than_highest_percentage.items():
    print(f"Label Class {class_id}: {difference_percentage:.2f}% fewer
    ↪instances than the highest class count ({max_count})")
```

Difference from the highest class count (in percentage):
Label Class 1: 0.00% fewer instances than the highest class count (6745)
Label Class 0: 37.17% fewer instances than the highest class count (6745)
Label Class 2: 39.84% fewer instances than the highest class count (6745)

Label one is

Distribution Analysis: Functioning to plot distributions of `x_center`, `y_center`, `width`, and `height` for each class. We want to see how these values differ among the values of data.

```
[ ]: def plot_distributions_by_class(df):
    metrics = ['x_center', 'y_center', 'width', 'height']
```

```

unique_classes = sorted(df['class_id'].unique())
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
fig.suptitle("Comparative Distributions of Bounding Box Metrics by Class", fontweight='bold', fontsize=16)
# I was unable to use my own pallet so had to use the husl fo this graph
palette = sns.color_palette("husl", len(unique_classes))
for i, metric in enumerate(metrics):
    ax = axes[i // 2, i % 2]

    for idx, class_id in enumerate(unique_classes):
        sns.kdeplot(df[df['class_id'] == class_id][metric], ax=ax,
                    color=palette[idx], label=f'Class {class_id}' if i == 0 else "")

    # Plot total distribution across all classes for reference
    sns.kdeplot(df[metric], ax=ax, color='black', linestyle='--',
                label='Total' if i == 0 else "")

    ax.set_title(f'Distribution of {metric.capitalize()} by Class')
    ax.set_xlabel(metric.capitalize())
    ax.set_ylabel('Density')

handles, labels = axes[0, 0].get_legend_handles_labels()
sorted_handles_labels = sorted(zip(handles, labels), key=lambda x: x[1])
sorted_handles, sorted_labels = zip(*sorted_handles_labels)

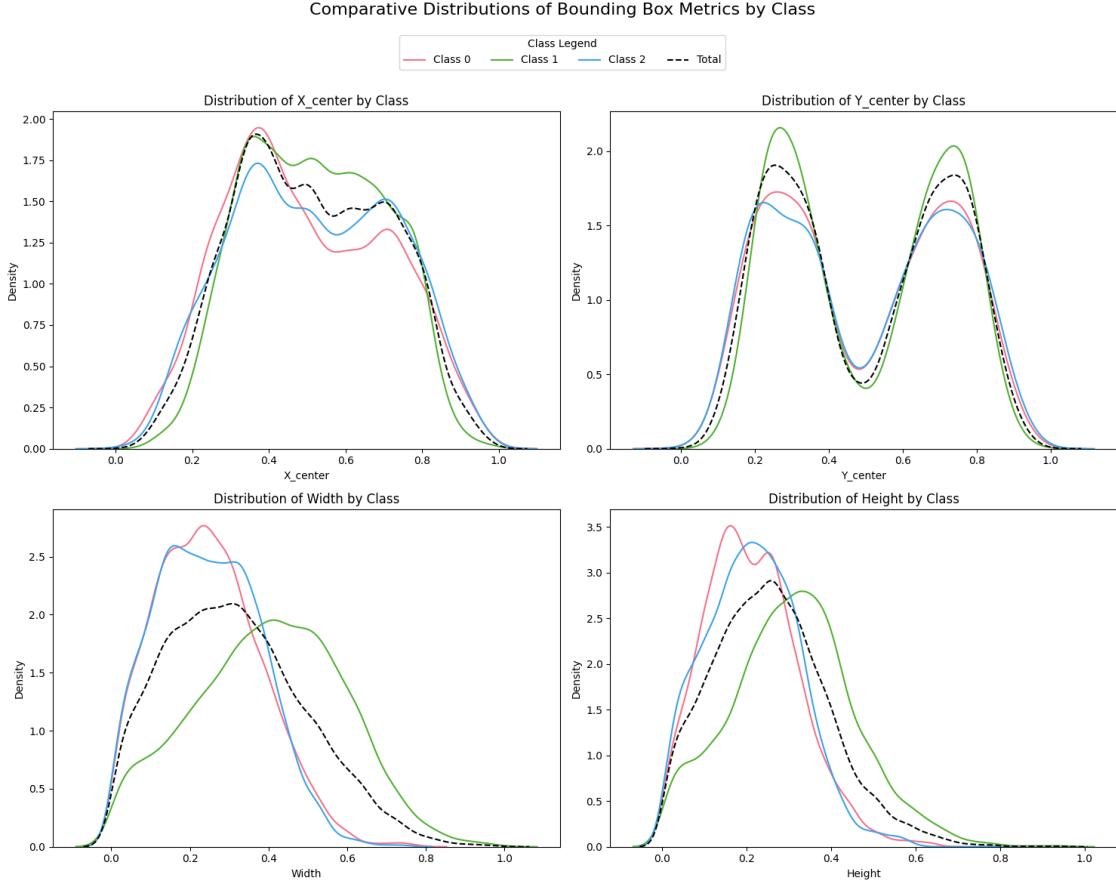
fig.legend(sorted_handles, sorted_labels, loc='upper center',
           bbox_to_anchor=(0.5, 0.95),
           ncol=len(unique_classes) + 1, title="Class Legend")

plt.tight_layout(rect=[0, 0, 1, 0.92])
plt.show()

```

Analysis

[]: plot_distributions_by_class(train_df)



Distribution of X_{center} by Class:

The X_{center} distribution shows that horizontal centers are mostly concentrated between 0.1 and 0.8, with each class demonstrating slight variations. Class 1 displays a more left-centered distribution with a peak around 0.2, while Classes 0 and 2 follow similar but slightly different shapes. This suggests that objects in Class 1 are generally positioned more towards the left side of the frame. The overall similarity in distribution shapes indicates commonalities across classes, but the differences in peak positions may reflect subtle distinctions in object placement.

Distribution of Y_{center} by Class:

The Y_{center} metric has a pronounced bimodal distribution across all classes, with peaks near 0.2 and 0.8. Class 1 shows the highest peaks at both positions, suggesting a strong preference for objects in this class to be located near the top or bottom of the frame. This consistent vertical pattern across classes implies that objects are often positioned in two distinct areas within the image, potentially indicating structural layout patterns. Class 1's prominent peaks also suggest more consistent object placements in this class compared to the others.

Distribution of Width by Class:

Width distributions vary significantly by class, with Class 0 featuring a high-density peak at smaller width values around 0.2, indicating narrower objects. Class 1's peak is slightly higher, around 0.3, but has a broader spread than Class 0. Class 2 shows a more diverse range of widths with a lower density peak, indicating that objects in this class come in a variety of sizes. This range highlights

that Class 2 likely represents more variability in object sizes, which could influence how the model learns size-based distinctions across classes.

Distribution of Height by Class:

The height distributions similarly show that smaller height values are more common, particularly around 0.2. Class 0 has a high-density peak at smaller heights, indicating that its objects tend to be shorter. Class 2, on the other hand, has a broader distribution, suggesting more variability in object height within this class. The dashed total line aligns closely with Class 1's distribution, implying that Class 1's height distribution may be representative of the overall dataset. This variability across classes in height distribution may affect the model's interpretation of object height distinctions.

1.4 Models

The dataset is already well-organized for use with the YOLO model, so there is no need to process data from a DataFrame. Instead, we utilize the .txt label files provided in the dataset directly.

1.4.1 Model Selection and Initial Experiments

We began with **YOLOv8** as a preliminary experiment. However, after analyzing the results, we decided to proceed with **YOLOv11** for the main training and evaluation. The YOLOv8 run was purely experimental, and fragments of its code are retained here for documentation purposes. This approach ensures a complete record of the workflow, even though YOLOv11 became the primary focus.

For a detailed overview of the initial experiment, refer to [YOLOv8 - Initial Test](#).

For the main run details, see [YOLOv11 - Main Run](#).

To compare the metrics between YOLOv8 and YOLOv11, refer to [Model Comparison - YOLOv8 vs YOLOv11](#).

Data Preparation for YOLO Models The dataset is already well-organized for use with the YOLO models, so there is no need to preprocess data from a DataFrame. Instead, we directly use the .txt label files provided in the dataset.

1.4.2 YOLOv8 - Initial Test

We initially used subset of data to make sure everything was running fine. Below is the code for creating the subset:

```
[ ]: train_images = os.listdir(resolved_paths['train']['images'])

# Sample 10 images randomly for the subset
subset_train_images = random.sample(train_images, 10)

resolved_paths['subset_train'] = {
    'images': os.path.normpath(os.path.join(resolved_paths['train']['images'],
    '..../subset_train/images')),
    'labels': os.path.normpath(os.path.join(resolved_paths['train']['labels'],
    '..../subset_train/labels'))
}
```

```

os.makedirs(resolved_paths['subset_train']['images'], exist_ok=True)
os.makedirs(resolved_paths['subset_train']['labels'], exist_ok=True)

# Copying selected images and their corresponding labels to subset directories
for image_name in subset_train_images:
    image_path = os.path.join(resolved_paths['train']['images'], image_name)
    label_name = image_name.replace('.jpg', '.txt')
    label_path = os.path.join(resolved_paths['train']['labels'], label_name)
    shutil.copy(image_path, os.path.
        ↪join(resolved_paths['subset_train']['images'], image_name))
    shutil.copy(label_path, os.path.
        ↪join(resolved_paths['subset_train']['labels'], label_name))

print(f"Subset images and labels copied to: "
    ↪{resolved_paths['subset_train']['images']} and
    ↪{resolved_paths['subset_train']['labels']}")

```

Subset images and labels copied to: /root/notebooks/xray/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/subset_train/images and /root/notebooks/xray/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/subset_train/labels

YOLOv8n Results Here's a revised and polished version of your text:

Once the setup was finalized, we utilized a cloud instance to perform a complete initial training and testing run with YOLOv8n. The trained model is saved at the following location:
results/yolov8n_brain_tumor_detection_v1/yolov8n_brain_tumor_detection_v1.pt.

Note: The code for running and testing the model was unintentionally overwritten during one of our commits. However, we preserved the run data and the trained model for reference and further analysis.

Below is the detailed run information:

Metrics

Metric	All	Label 0	Label 1	Label 2
Precision (P)	0.646	0.606	0.733	0.599
Recall (R)	0.516	0.400	0.674	0.475
mAP@50	0.553	0.443	0.707	0.511
mAP@50-95	0.276	0.188	0.407	0.232

Speed

Stage	Time (ms/image)
Preprocessing	0.2
Inference	1.8
Postprocessing	0.8

The parameters for both YOLOv8 and YOLOv11 runs were identical, as follows:

Parameters

Parameter	Value
Number of epochs	3
Image size (imgsz)	640
Batch size	4

The following are the YOLOv8 validation and test results. For YOLOv11, we improved the process by including only the relevant information and presenting it using graphics.

Note: The code in the next cell will not execute and is retained as a leftover from the YOLOv8 post-run analysis.

```
[ ]: # # Validate the model
# print("\nRunning Validation on Validation Dataset...")
# results_val = model.val()
# print("Validation Results:")
# print(results_val)

# # Test the model
# print("\nRunning Testing on Test Dataset...")
# results_test = model.val(split='test')
# print("Test Results:")
# print(results_test)
```

```
Running Validation on Validation Dataset...
Ultralytics 8.3.38 Python-3.10.15 torch-2.5.1+cu118 CUDA:0 (NVIDIA
A100-SXM4-40GB, 40514MiB)
Model summary (fused): 168 layers, 3,006,233 parameters, 0 gradients, 8.1 GFLOPs

val: Scanning /root/notebooks/xray/data/BrainTumorDetectionYolov9/B
rainTumorDetectionYolov9/valid/labels.cache... 1980 images, 17 backgrounds, 0
corrupt: 100% | 1980/1980 [00:00<?, ?it/s]
      Class    Images Instances   Box(P)      R    mAP50
mAP50-95): 100% | 495/495 [00:09<00:00, 51.80it/s]
               all     1980     4380    0.669    0.508    0.537
0.261           label0    1246     1246    0.604    0.376    0.401
0.162           label1    1944     1944    0.763    0.68     0.716
0.402           label2    1190     1190    0.638    0.469    0.495
0.22
Speed: 0.2ms preprocess, 1.5ms inference, 0.0ms loss, 0.7ms postprocess per
image
```

```

Results saved to runs/detect/train122
Validation Results:
ultralytics.utils.metrics.DetMetrics object with attributes:

ap_class_index: array([0, 1, 2])
box: ultralytics.utils.metrics.Metric object
confusion_matrix: <ultralytics.utils.metrics.ConfusionMatrix object at
0x7f0e1c08e4a0>
curves: ['Precision-Recall(B)', 'F1-Confidence(B)', 'Precision-Confidence(B)', 'Recall-Confidence(B)']
curves_results: [[array([
    0, 0.001001, 0.002002, 0.003003,
    0.004004, 0.005005, 0.006006, 0.007007, 0.008008, 0.009009,
    0.01001, 0.011011, 0.012012, 0.013013, 0.014014, 0.015015,
    0.016016, 0.017017, 0.018018, 0.019019, 0.02002, 0.021021,
    0.022022, 0.023023,
    0.024024, 0.025025, 0.026026, 0.027027, 0.028028,
    0.029029, 0.03003, 0.031031, 0.032032, 0.033033, 0.034034,
    0.035035, 0.036036, 0.037037, 0.038038, 0.039039, 0.04004,
    0.041041, 0.042042, 0.043043, 0.044044, 0.045045, 0.046046,
    0.047047,
    0.048048, 0.049049, 0.05005, 0.051051, 0.052052,
    0.053053, 0.054054, 0.055055, 0.056056, 0.057057, 0.058058,
    0.059059, 0.06006, 0.061061, 0.062062, 0.063063, 0.064064,
    0.065065, 0.066066, 0.067067, 0.068068, 0.069069, 0.07007,
    0.071071,
    0.072072, 0.073073, 0.074074, 0.075075, 0.076076,
    0.077077, 0.078078, 0.079079, 0.08008, 0.081081, 0.082082,
    0.083083, 0.084084, 0.085085, 0.086086, 0.087087, 0.088088,
    0.089089, 0.09009, 0.091091, 0.092092, 0.093093, 0.094094,
    0.095095,
    0.096096, 0.097097, 0.098098, 0.099099, 0.1001,
    0.1011, 0.1021, 0.1031, 0.1041, 0.10511, 0.10611,
    0.10711, 0.10811, 0.10911, 0.11011, 0.11111, 0.11211,
    0.11311, 0.11411, 0.11512, 0.11612, 0.11712, 0.11812,
    0.11912,
    0.12012, 0.12112, 0.12212, 0.12312, 0.12412,
    0.12513, 0.12613, 0.12713, 0.12813, 0.12913, 0.13013,
    0.13113, 0.13213, 0.13313, 0.13413, 0.13514, 0.13614,
    0.13714, 0.13814, 0.13914, 0.14014, 0.14114, 0.14214,
    0.14314,
    0.14414, 0.14515, 0.14615, 0.14715, 0.14815,
    0.14915, 0.15015, 0.15115, 0.15215, 0.15315, 0.15415,
    0.15516, 0.15616, 0.15716, 0.15816, 0.15916, 0.16016,
    0.16116, 0.16216, 0.16316, 0.16416, 0.16517, 0.16617,
    0.16717,
    0.16817, 0.16917, 0.17017, 0.17117, 0.17217,
    0.17317, 0.17417, 0.17518, 0.17618, 0.17718, 0.17818,
    0.17918, 0.18018, 0.18118, 0.18218, 0.18318, 0.18418,
])]]]

```

0.18519,	0.18619,	0.18719,	0.18819,	0.18919,	0.19019,
0.19119,	0.19219,	0.19319,	0.19419,	0.1952,	0.1962,
0.1972,	0.1982,	0.1992,	0.2002,	0.2012,	0.2022,
0.2032,	0.2042,	0.20521,	0.20621,	0.20721,	0.20821,
0.20921,	0.21021,	0.21121,	0.21221,	0.21321,	0.21421,
0.21522,	0.21622,	0.21722,	0.21822,	0.21922,	0.22022,
0.22122,	0.22222,	0.22322,	0.22422,	0.22523,	0.22623,
0.22723,	0.22823,	0.22923,	0.23023,	0.23123,	0.23223,
0.23323,	0.23423,	0.23524,	0.23624,	0.23724,	0.23824,
0.23924,	0.24024,	0.24124,	0.24224,	0.24324,	0.24424,
0.24525,	0.24625,	0.24725,	0.24825,	0.24925,	0.25025,
0.25125,	0.25225,	0.25325,	0.25425,	0.25526,	0.25626,
0.25726,	0.25826,	0.25926,	0.26026,	0.26126,	0.26226,
0.26326,	0.26426,	0.26527,	0.26627,	0.26727,	0.26827,
0.26927,	0.27027,	0.27127,	0.27227,	0.27327,	0.27427,
0.27528,	0.27628,	0.27728,	0.27828,	0.27928,	0.28028,
0.28128,	0.28228,	0.28328,	0.28428,	0.28529,	0.28629,
0.28729,	0.28829,	0.28929,	0.29029,	0.29129,	0.29229,
0.29329,	0.29429,	0.2953,	0.2963,	0.2973,	0.2983,
0.2993,	0.3003,	0.3013,	0.3023,	0.3033,	0.3043,
0.30531,	0.30631,	0.30731,	0.30831,	0.30931,	0.31031,
0.31131,	0.31231,	0.31331,	0.31431,	0.31532,	0.31632,
0.31732,	0.31832,	0.31932,	0.32032,	0.32132,	0.32232,
0.32332,	0.32432,	0.32533,	0.32633,	0.32733,	0.32833,
0.32933,	0.33033,	0.33133,	0.33233,	0.33333,	0.33433,
0.33534,	0.33634,	0.33734,	0.33834,	0.33934,	0.34034,
0.34134,	0.34234,	0.34334,	0.34434,	0.34535,	0.34635,
0.34735,	0.34835,	0.34935,	0.35035,	0.35135,	0.35235,
0.35335,	0.35435,	0.35536,	0.35636,	0.35736,	0.35836,
0.35936,	0.36036,	0.36136,	0.36236,	0.36336,	0.36436,
0.36537,	0.36637,	0.36737,	0.36837,	0.36937,	0.37037,
0.37137,	0.37237,	0.37337,	0.37437,	0.37538,	0.37638,
0.37738,	0.37838,	0.37938,	0.38038,	0.38138,	0.38238,
0.38338,	0.38438,	0.38539,	0.38639,	0.38739,	0.38839,
0.38939,	0.39039,	0.39139,	0.39239,	0.39339,	0.39439,
0.3954,	0.3964,	0.3974,	0.3984,	0.3994,	0.4004,
0.4014,	0.4024,	0.4034,	0.4044,	0.40541,	0.40641,
0.40741,	0.40841,	0.40941,	0.41041,	0.41141,	0.41241,

0.41341,	0.41441,	0.41542,	0.41642,	0.41742,	0.41842,
0.41942,	0.42042,	0.42142,	0.42242,	0.42342,	0.42442,
0.42543,	0.42643,	0.42743,	0.42843,	0.42943,	0.43043,
0.43143,					
	0.43243,	0.43343,	0.43443,	0.43544,	0.43644,
0.43744,	0.43844,	0.43944,	0.44044,	0.44144,	0.44244,
0.44344,	0.44444,	0.44545,	0.44645,	0.44745,	0.44845,
0.44945,	0.45045,	0.45145,	0.45245,	0.45345,	0.45445,
0.45546,					
	0.45646,	0.45746,	0.45846,	0.45946,	0.46046,
0.46146,	0.46246,	0.46346,	0.46446,	0.46547,	0.46647,
0.46747,	0.46847,	0.46947,	0.47047,	0.47147,	0.47247,
0.47347,	0.47447,	0.47548,	0.47648,	0.47748,	0.47848,
0.47948,					
	0.48048,	0.48148,	0.48248,	0.48348,	0.48448,
0.48549,	0.48649,	0.48749,	0.48849,	0.48949,	0.49049,
0.49149,	0.49249,	0.49349,	0.49449,	0.4955,	0.4965,
0.4975,	0.4985,	0.4995,	0.5005,	0.5015,	0.5025,
0.5035,					
	0.5045,	0.50551,	0.50651,	0.50751,	0.50851,
0.50951,	0.51051,	0.51151,	0.51251,	0.51351,	0.51451,
0.51552,	0.51652,	0.51752,	0.51852,	0.51952,	0.52052,
0.52152,	0.52252,	0.52352,	0.52452,	0.52553,	0.52653,
0.52753,					
	0.52853,	0.52953,	0.53053,	0.53153,	0.53253,
0.53353,	0.53453,	0.53554,	0.53654,	0.53754,	0.53854,
0.53954,	0.54054,	0.54154,	0.54254,	0.54354,	0.54454,
0.54555,	0.54655,	0.54755,	0.54855,	0.54955,	0.55055,
0.55155,					
	0.55255,	0.55355,	0.55455,	0.55556,	0.55656,
0.55756,	0.55856,	0.55956,	0.56056,	0.56156,	0.56256,
0.56356,	0.56456,	0.56557,	0.56657,	0.56757,	0.56857,
0.56957,	0.57057,	0.57157,	0.57257,	0.57357,	0.57457,
0.57558,					
	0.57658,	0.57758,	0.57858,	0.57958,	0.58058,
0.58158,	0.58258,	0.58358,	0.58458,	0.58559,	0.58659,
0.58759,	0.58859,	0.58959,	0.59059,	0.59159,	0.59259,
0.59359,	0.59459,	0.5956,	0.5966,	0.5976,	0.5986,
0.5996,					
	0.6006,	0.6016,	0.6026,	0.6036,	0.6046,
0.60561,	0.60661,	0.60761,	0.60861,	0.60961,	0.61061,
0.61161,	0.61261,	0.61361,	0.61461,	0.61562,	0.61662,
0.61762,	0.61862,	0.61962,	0.62062,	0.62162,	0.62262,
0.62362,					
	0.62462,	0.62563,	0.62663,	0.62763,	0.62863,
0.62963,	0.63063,	0.63163,	0.63263,	0.63363,	0.63463,
0.63564,	0.63664,	0.63764,	0.63864,	0.63964,	0.64064,
0.64164,	0.64264,	0.64364,	0.64464,	0.64565,	0.64665,

0.64765,					
	0.64865,	0.64965,	0.65065,	0.65165,	0.65265,
0.65365,	0.65465,	0.65566,	0.65666,	0.65766,	0.65866,
0.65966,	0.66066,	0.66166,	0.66266,	0.66366,	0.66466,
0.66567,	0.66667,	0.66767,	0.66867,	0.66967,	0.67067,
0.67167,					
	0.67267,	0.67367,	0.67467,	0.67568,	0.67668,
0.67768,	0.67868,	0.67968,	0.68068,	0.68168,	0.68268,
0.68368,	0.68468,	0.68569,	0.68669,	0.68769,	0.68869,
0.68969,	0.69069,	0.69169,	0.69269,	0.69369,	0.69469,
0.6957,					
	0.6967,	0.6977,	0.6987,	0.6997,	0.7007,
0.7017,	0.7027,	0.7037,	0.7047,	0.70571,	0.70671,
0.70771,	0.70871,	0.70971,	0.71071,	0.71171,	0.71271,
0.71371,	0.71471,	0.71572,	0.71672,	0.71772,	0.71872,
0.71972,					
	0.72072,	0.72172,	0.72272,	0.72372,	0.72472,
0.72573,	0.72673,	0.72773,	0.72873,	0.72973,	0.73073,
0.73173,	0.73273,	0.73373,	0.73473,	0.73574,	0.73674,
0.73774,	0.73874,	0.73974,	0.74074,	0.74174,	0.74274,
0.74374,					
	0.74474,	0.74575,	0.74675,	0.74775,	0.74875,
0.74975,	0.75075,	0.75175,	0.75275,	0.75375,	0.75475,
0.75576,	0.75676,	0.75776,	0.75876,	0.75976,	0.76076,
0.76176,	0.76276,	0.76376,	0.76476,	0.76577,	0.76677,
0.76777,					
	0.76877,	0.76977,	0.77077,	0.77177,	0.77277,
0.77377,	0.77477,	0.77578,	0.77678,	0.77778,	0.77878,
0.77978,	0.78078,	0.78178,	0.78278,	0.78378,	0.78478,
0.78579,	0.78679,	0.78779,	0.78879,	0.78979,	0.79079,
0.79179,					
	0.79279,	0.79379,	0.79479,	0.7958,	0.7968,
0.7978,	0.7988,	0.7998,	0.8008,	0.8018,	0.8028,
0.8038,	0.8048,	0.80581,	0.80681,	0.80781,	0.80881,
0.80981,	0.81081,	0.81181,	0.81281,	0.81381,	0.81481,
0.81582,					
	0.81682,	0.81782,	0.81882,	0.81982,	0.82082,
0.82182,	0.82282,	0.82382,	0.82482,	0.82583,	0.82683,
0.82783,	0.82883,	0.82983,	0.83083,	0.83183,	0.83283,
0.83383,	0.83483,	0.83584,	0.83684,	0.83784,	0.83884,
0.83984,					
	0.84084,	0.84184,	0.84284,	0.84384,	0.84484,
0.84585,	0.84685,	0.84785,	0.84885,	0.84985,	0.85085,
0.85185,	0.85285,	0.85385,	0.85485,	0.85586,	0.85686,
0.85786,	0.85886,	0.85986,	0.86086,	0.86186,	0.86286,
0.86386,					
	0.86486,	0.86587,	0.86687,	0.86787,	0.86887,
0.86987,	0.87087,	0.87187,	0.87287,	0.87387,	0.87487,

0.87588,	0.87688,	0.87788,	0.87888,	0.87988,	0.88088,
0.88188,	0.88288,	0.88388,	0.88488,	0.88589,	0.88689,
0.88789,					
	0.88889,	0.88989,	0.89089,	0.89189,	0.89289,
0.89389,	0.89489,	0.8959,	0.8969,	0.8979,	0.8989,
0.8999,	0.9009,	0.9019,	0.9029,	0.9039,	0.9049,
0.90591,	0.90691,	0.90791,	0.90891,	0.90991,	0.91091,
0.91191,					
	0.91291,	0.91391,	0.91491,	0.91592,	0.91692,
0.91792,	0.91892,	0.91992,	0.92092,	0.92192,	0.92292,
0.92392,	0.92492,	0.92593,	0.92693,	0.92793,	0.92893,
0.92993,	0.93093,	0.93193,	0.93293,	0.93393,	0.93493,
0.93594,					
	0.93694,	0.93794,	0.93894,	0.93994,	0.94094,
0.94194,	0.94294,	0.94394,	0.94494,	0.94595,	0.94695,
0.94795,	0.94895,	0.94995,	0.95095,	0.95195,	0.95295,
0.95395,	0.95495,	0.95596,	0.95696,	0.95796,	0.95896,
0.95996,					
	0.96096,	0.96196,	0.96296,	0.96396,	0.96496,
0.96597,	0.96697,	0.96797,	0.96897,	0.96997,	0.97097,
0.97197,	0.97297,	0.97397,	0.97497,	0.97598,	0.97698,
0.97798,	0.97898,	0.97998,	0.98098,	0.98198,	0.98298,
0.98398,					
	0.98498,	0.98599,	0.98699,	0.98799,	0.98899,
0.98999,	0.99099,	0.99199,	0.99299,	0.99399,	0.99499,
0.996,	0.997,	0.998,	0.999,	1], array([[
0.92958,	0.92958,	0.92958, ... , 0.00015716,	7.8582e-05,	0],	
[1,	1,	1, ... , 0.0002829,	0.00014145,	
0],					
	[1,	1,	1, ... , 0.00015905,	7.9525e-05,
0]], 'Recall', 'Precision'], [array([0,	0.001001,	0.002002,
0.003003,	0.004004,	0.005005,	0.006006,	0.007007,	0.008008,
0.009009,	0.01001,	0.011011,	0.012012,	0.013013,	0.014014,
0.015015,	0.016016,	0.017017,	0.018018,	0.019019,	0.02002,
0.021021,	0.022022,	0.023023,			
	0.024024,	0.025025,	0.026026,	0.027027,	0.028028,
0.029029,	0.03003,	0.031031,	0.032032,	0.033033,	0.034034,
0.035035,	0.036036,	0.037037,	0.038038,	0.039039,	0.04004,
0.041041,	0.042042,	0.043043,	0.044044,	0.045045,	0.046046,
0.047047,					
	0.048048,	0.049049,	0.05005,	0.051051,	0.052052,
0.053053,	0.054054,	0.055055,	0.056056,	0.057057,	0.058058,
0.059059,	0.06006,	0.061061,	0.062062,	0.063063,	0.064064,
0.065065,	0.066066,	0.067067,	0.068068,	0.069069,	0.07007,
0.071071,					
	0.072072,	0.073073,	0.074074,	0.075075,	0.076076,
0.077077,	0.078078,	0.079079,	0.08008,	0.081081,	0.082082,
0.083083,	0.084084,	0.085085,	0.086086,	0.087087,	0.088088,

0.089089,	0.09009,	0.091091,	0.092092,	0.093093,	0.094094,
0.095095,	0.096096,	0.097097,	0.098098,	0.099099,	0.1001,
0.1011,	0.1021,	0.1031,	0.1041,	0.10511,	0.10611,
0.10711,	0.10811,	0.10911,	0.11011,	0.11111,	0.11211,
0.11311,	0.11411,	0.11512,	0.11612,	0.11712,	0.11812,
0.11912,	0.12012,	0.12112,	0.12212,	0.12312,	0.12412,
0.12513,	0.12613,	0.12713,	0.12813,	0.12913,	0.13013,
0.13113,	0.13213,	0.13313,	0.13413,	0.13514,	0.13614,
0.13714,	0.13814,	0.13914,	0.14014,	0.14114,	0.14214,
0.14314,	0.14414,	0.14515,	0.14615,	0.14715,	0.14815,
0.14915,	0.15015,	0.15115,	0.15215,	0.15315,	0.15415,
0.15516,	0.15616,	0.15716,	0.15816,	0.15916,	0.16016,
0.16116,	0.16216,	0.16316,	0.16416,	0.16517,	0.16617,
0.16717,	0.16817,	0.16917,	0.17017,	0.17117,	0.17217,
0.17317,	0.17417,	0.17518,	0.17618,	0.17718,	0.17818,
0.17918,	0.18018,	0.18118,	0.18218,	0.18318,	0.18418,
0.18519,	0.18619,	0.18719,	0.18819,	0.18919,	0.19019,
0.19119,	0.19219,	0.19319,	0.19419,	0.1952,	0.1962,
0.1972,	0.1982,	0.1992,	0.2002,	0.2012,	0.2022,
0.2032,	0.2042,	0.20521,	0.20621,	0.20721,	0.20821,
0.20921,	0.21021,	0.21121,	0.21221,	0.21321,	0.21421,
0.21522,	0.21622,	0.21722,	0.21822,	0.21922,	0.22022,
0.22122,	0.22222,	0.22322,	0.22422,	0.22523,	0.22623,
0.22723,	0.22823,	0.22923,	0.23023,	0.23123,	0.23223,
0.23323,	0.23423,	0.23524,	0.23624,	0.23724,	0.23824,
0.23924,	0.24024,	0.24124,	0.24224,	0.24324,	0.24424,
0.24525,	0.24625,	0.24725,	0.24825,	0.24925,	0.25025,
0.25125,	0.25225,	0.25325,	0.25425,	0.25526,	0.25626,
0.25726,	0.25826,	0.25926,	0.26026,	0.26126,	0.26226,
0.26326,	0.26426,	0.26527,	0.26627,	0.26727,	0.26827,
0.26927,	0.27027,	0.27127,	0.27227,	0.27327,	0.27427,
0.27528,	0.27628,	0.27728,	0.27828,	0.27928,	0.28028,
0.28128,	0.28228,	0.28328,	0.28428,	0.28529,	0.28629,
0.28729,	0.28829,	0.28929,	0.29029,	0.29129,	0.29229,
0.29329,	0.29429,	0.2953,	0.2963,	0.2973,	0.2983,
0.2993,	0.3003,	0.3013,	0.3023,	0.3033,	0.3043,
0.30531,	0.30631,	0.30731,	0.30831,	0.30931,	0.31031,
0.31131,	0.31231,	0.31331,	0.31431,	0.31532,	0.31632,

0.31732,	0.31832,	0.31932,	0.32032,	0.32132,	0.32232,
0.32332,	0.32432,	0.32533,	0.32633,	0.32733,	0.32833,
0.32933,	0.33033,	0.33133,	0.33233,	0.33333,	0.33433,
0.33534,					
	0.33634,	0.33734,	0.33834,	0.33934,	0.34034,
0.34134,	0.34234,	0.34334,	0.34434,	0.34535,	0.34635,
0.34735,	0.34835,	0.34935,	0.35035,	0.35135,	0.35235,
0.35335,	0.35435,	0.35536,	0.35636,	0.35736,	0.35836,
0.35936,					
	0.36036,	0.36136,	0.36236,	0.36336,	0.36436,
0.36537,	0.36637,	0.36737,	0.36837,	0.36937,	0.37037,
0.37137,	0.37237,	0.37337,	0.37437,	0.37538,	0.37638,
0.37738,	0.37838,	0.37938,	0.38038,	0.38138,	0.38238,
0.38338,					
	0.38438,	0.38539,	0.38639,	0.38739,	0.38839,
0.38939,	0.39039,	0.39139,	0.39239,	0.39339,	0.39439,
0.3954,	0.3964,	0.3974,	0.3984,	0.3994,	0.4004,
0.4014,	0.4024,	0.4034,	0.4044,	0.40541,	0.40641,
0.40741,					
	0.40841,	0.40941,	0.41041,	0.41141,	0.41241,
0.41341,	0.41441,	0.41542,	0.41642,	0.41742,	0.41842,
0.41942,	0.42042,	0.42142,	0.42242,	0.42342,	0.42442,
0.42543,	0.42643,	0.42743,	0.42843,	0.42943,	0.43043,
0.43143,					
	0.43243,	0.43343,	0.43443,	0.43544,	0.43644,
0.43744,	0.43844,	0.43944,	0.44044,	0.44144,	0.44244,
0.44344,	0.44444,	0.44545,	0.44645,	0.44745,	0.44845,
0.44945,	0.45045,	0.45145,	0.45245,	0.45345,	0.45445,
0.45546,					
	0.45646,	0.45746,	0.45846,	0.45946,	0.46046,
0.46146,	0.46246,	0.46346,	0.46446,	0.46547,	0.46647,
0.46747,	0.46847,	0.46947,	0.47047,	0.47147,	0.47247,
0.47347,	0.47447,	0.47548,	0.47648,	0.47748,	0.47848,
0.47948,					
	0.48048,	0.48148,	0.48248,	0.48348,	0.48448,
0.48549,	0.48649,	0.48749,	0.48849,	0.48949,	0.49049,
0.49149,	0.49249,	0.49349,	0.49449,	0.4955,	0.4965,
0.4975,	0.4985,	0.4995,	0.5005,	0.5015,	0.5025,
0.5035,					
	0.5045,	0.50551,	0.50651,	0.50751,	0.50851,
0.50951,	0.51051,	0.51151,	0.51251,	0.51351,	0.51451,
0.51552,	0.51652,	0.51752,	0.51852,	0.51952,	0.52052,
0.52152,	0.52252,	0.52352,	0.52452,	0.52553,	0.52653,
0.52753,					
	0.52853,	0.52953,	0.53053,	0.53153,	0.53253,
0.53353,	0.53453,	0.53554,	0.53654,	0.53754,	0.53854,
0.53954,	0.54054,	0.54154,	0.54254,	0.54354,	0.54454,
0.54555,	0.54655,	0.54755,	0.54855,	0.54955,	0.55055,

0.55155,	0.55255,	0.55355,	0.55455,	0.55556,	0.55656,
0.55756,	0.55856,	0.55956,	0.56056,	0.56156,	0.56256,
0.56356,	0.56456,	0.56557,	0.56657,	0.56757,	0.56857,
0.56957,	0.57057,	0.57157,	0.57257,	0.57357,	0.57457,
0.57558,	0.57658,	0.57758,	0.57858,	0.57958,	0.58058,
0.58158,	0.58258,	0.58358,	0.58458,	0.58559,	0.58659,
0.58759,	0.58859,	0.58959,	0.59059,	0.59159,	0.59259,
0.59359,	0.59459,	0.5956,	0.5966,	0.5976,	0.5986,
0.5996,	0.6006,	0.6016,	0.6026,	0.6036,	0.6046,
0.60561,	0.60661,	0.60761,	0.60861,	0.60961,	0.61061,
0.61161,	0.61261,	0.61361,	0.61461,	0.61562,	0.61662,
0.61762,	0.61862,	0.61962,	0.62062,	0.62162,	0.62262,
0.62362,	0.62462,	0.62563,	0.62663,	0.62763,	0.62863,
0.62963,	0.63063,	0.63163,	0.63263,	0.63363,	0.63463,
0.63564,	0.63664,	0.63764,	0.63864,	0.63964,	0.64064,
0.64164,	0.64264,	0.64364,	0.64464,	0.64565,	0.64665,
0.64765,	0.64865,	0.64965,	0.65065,	0.65165,	0.65265,
0.65365,	0.65465,	0.65566,	0.65666,	0.65766,	0.65866,
0.65966,	0.66066,	0.66166,	0.66266,	0.66366,	0.66466,
0.66567,	0.66667,	0.66767,	0.66867,	0.66967,	0.67067,
0.67167,	0.67267,	0.67367,	0.67467,	0.67568,	0.67668,
0.67768,	0.67868,	0.67968,	0.68068,	0.68168,	0.68268,
0.68368,	0.68468,	0.68569,	0.68669,	0.68769,	0.68869,
0.68969,	0.69069,	0.69169,	0.69269,	0.69369,	0.69469,
0.6957,	0.6967,	0.6977,	0.6987,	0.6997,	0.7007,
0.7017,	0.7027,	0.7037,	0.7047,	0.70571,	0.70671,
0.70771,	0.70871,	0.70971,	0.71071,	0.71171,	0.71271,
0.71371,	0.71471,	0.71572,	0.71672,	0.71772,	0.71872,
0.71972,	0.72072,	0.72172,	0.72272,	0.72372,	0.72472,
0.72573,	0.72673,	0.72773,	0.72873,	0.72973,	0.73073,
0.73173,	0.73273,	0.73373,	0.73473,	0.73574,	0.73674,
0.73774,	0.73874,	0.73974,	0.74074,	0.74174,	0.74274,
0.74374,	0.74474,	0.74575,	0.74675,	0.74775,	0.74875,
0.74975,	0.75075,	0.75175,	0.75275,	0.75375,	0.75475,
0.75576,	0.75676,	0.75776,	0.75876,	0.75976,	0.76076,
0.76176,	0.76276,	0.76376,	0.76476,	0.76577,	0.76677,
0.76777,	0.76877,	0.76977,	0.77077,	0.77177,	0.77277,
0.77377,	0.77477,	0.77578,	0.77678,	0.77778,	0.77878,

0.77978,	0.78078,	0.78178,	0.78278,	0.78378,	0.78478,
0.78579,	0.78679,	0.78779,	0.78879,	0.78979,	0.79079,
0.79179,					
	0.79279,	0.79379,	0.79479,	0.7958,	0.7968,
0.7978,	0.7988,	0.7998,	0.8008,	0.8018,	0.8028,
0.8038,	0.8048,	0.80581,	0.80681,	0.80781,	0.80881,
0.80981,	0.81081,	0.81181,	0.81281,	0.81381,	0.81481,
0.81582,					
	0.81682,	0.81782,	0.81882,	0.81982,	0.82082,
0.82182,	0.82282,	0.82382,	0.82482,	0.82583,	0.82683,
0.82783,	0.82883,	0.82983,	0.83083,	0.83183,	0.83283,
0.83383,	0.83483,	0.83584,	0.83684,	0.83784,	0.83884,
0.83984,					
	0.84084,	0.84184,	0.84284,	0.84384,	0.84484,
0.84585,	0.84685,	0.84785,	0.84885,	0.84985,	0.85085,
0.85185,	0.85285,	0.85385,	0.85485,	0.85586,	0.85686,
0.85786,	0.85886,	0.85986,	0.86086,	0.86186,	0.86286,
0.86386,					
	0.86486,	0.86587,	0.86687,	0.86787,	0.86887,
0.86987,	0.87087,	0.87187,	0.87287,	0.87387,	0.87487,
0.87588,	0.87688,	0.87788,	0.87888,	0.87988,	0.88088,
0.88188,	0.88288,	0.88388,	0.88488,	0.88589,	0.88689,
0.88789,					
	0.88889,	0.88989,	0.89089,	0.89189,	0.89289,
0.89389,	0.89489,	0.8959,	0.8969,	0.8979,	0.8989,
0.8999,	0.9009,	0.9019,	0.9029,	0.9039,	0.9049,
0.90591,	0.90691,	0.90791,	0.90891,	0.90991,	0.91091,
0.91191,					
	0.91291,	0.91391,	0.91491,	0.91592,	0.91692,
0.91792,	0.91892,	0.91992,	0.92092,	0.92192,	0.92292,
0.92392,	0.92492,	0.92593,	0.92693,	0.92793,	0.92893,
0.92993,	0.93093,	0.93193,	0.93293,	0.93393,	0.93493,
0.93594,					
	0.93694,	0.93794,	0.93894,	0.93994,	0.94094,
0.94194,	0.94294,	0.94394,	0.94494,	0.94595,	0.94695,
0.94795,	0.94895,	0.94995,	0.95095,	0.95195,	0.95295,
0.95395,	0.95495,	0.95596,	0.95696,	0.95796,	0.95896,
0.95996,					
	0.96096,	0.96196,	0.96296,	0.96396,	0.96496,
0.96597,	0.96697,	0.96797,	0.96897,	0.96997,	0.97097,
0.97197,	0.97297,	0.97397,	0.97497,	0.97598,	0.97698,
0.97798,	0.97898,	0.97998,	0.98098,	0.98198,	0.98298,
0.98398,					
	0.98498,	0.98599,	0.98699,	0.98799,	0.98899,
0.98999,	0.99099,	0.99199,	0.99299,	0.99399,	0.99499,
0.996,	0.997,	0.998,	0.999,	1]), array([[
0.026534,	0.026546,	0.042002,	...,	0,	0,
[0.027628,	0.027649,	0.051998,	...,	0,
				0,	0,

```

0],
[  0.025896,   0.025916,   0.04415, ...,          0,          0,
0]], 'Confidence', 'F1'], [array([
0.003003,   0.004004,   0.005005,   0.006006,   0.007007,   0.008008,
0.009009,   0.01001,    0.011011,   0.012012,   0.013013,   0.014014,
0.015015,   0.016016,   0.017017,   0.018018,   0.019019,   0.02002,
0.021021,   0.022022,   0.023023,
0.024024,   0.025025,   0.026026,   0.027027,   0.028028,
0.029029,   0.03003,    0.031031,   0.032032,   0.033033,   0.034034,
0.035035,   0.036036,   0.037037,   0.038038,   0.039039,   0.04004,
0.041041,   0.042042,   0.043043,   0.044044,   0.045045,   0.046046,
0.047047,
0.048048,   0.049049,   0.05005,    0.051051,   0.052052,
0.053053,   0.054054,   0.055055,   0.056056,   0.057057,   0.058058,
0.059059,   0.06006,    0.061061,   0.062062,   0.063063,   0.064064,
0.065065,   0.066066,   0.067067,   0.068068,   0.069069,   0.07007,
0.071071,
0.072072,   0.073073,   0.074074,   0.075075,   0.076076,
0.077077,   0.078078,   0.079079,   0.08008,    0.081081,   0.082082,
0.083083,   0.084084,   0.085085,   0.086086,   0.087087,   0.088088,
0.089089,   0.09009,    0.091091,   0.092092,   0.093093,   0.094094,
0.095095,
0.096096,   0.097097,   0.098098,   0.099099,   0.1001,
0.1011,     0.1021,    0.1031,    0.1041,    0.10511,   0.10611,
0.10711,    0.10811,   0.10911,   0.11011,   0.11111,   0.11211,
0.11311,    0.11411,   0.11512,   0.11612,   0.11712,   0.11812,
0.11912,
0.12012,    0.12112,   0.12212,   0.12312,   0.12412,
0.12513,    0.12613,   0.12713,   0.12813,   0.12913,   0.13013,
0.13113,    0.13213,   0.13313,   0.13413,   0.13514,   0.13614,
0.13714,    0.13814,   0.13914,   0.14014,   0.14114,   0.14214,
0.14314,
0.14414,    0.14515,   0.14615,   0.14715,   0.14815,
0.14915,    0.15015,   0.15115,   0.15215,   0.15315,   0.15415,
0.15516,    0.15616,   0.15716,   0.15816,   0.15916,   0.16016,
0.16116,    0.16216,   0.16316,   0.16416,   0.16517,   0.16617,
0.16717,
0.16817,    0.16917,   0.17017,   0.17117,   0.17217,
0.17317,    0.17417,   0.17518,   0.17618,   0.17718,   0.17818,
0.17918,    0.18018,   0.18118,   0.18218,   0.18318,   0.18418,
0.18519,    0.18619,   0.18719,   0.18819,   0.18919,   0.19019,
0.19119,
0.19219,    0.19319,   0.19419,   0.1952,    0.1962,
0.1972,     0.1982,    0.1992,    0.2002,    0.2012,    0.2022,
0.2032,     0.2042,    0.20521,   0.20621,   0.20721,   0.20821,
0.20921,    0.21021,   0.21121,   0.21221,   0.21321,   0.21421,
0.21522,
0.21622,    0.21722,   0.21822,   0.21922,   0.22022,

```

0.22122,	0.22222,	0.22322,	0.22422,	0.22523,	0.22623,
0.22723,	0.22823,	0.22923,	0.23023,	0.23123,	0.23223,
0.23323,	0.23423,	0.23524,	0.23624,	0.23724,	0.23824,
0.23924,	0.24024,	0.24124,	0.24224,	0.24324,	0.24424,
0.24525,	0.24625,	0.24725,	0.24825,	0.24925,	0.25025,
0.25125,	0.25225,	0.25325,	0.25425,	0.25526,	0.25626,
0.25726,	0.25826,	0.25926,	0.26026,	0.26126,	0.26226,
0.26326,	0.26426,	0.26527,	0.26627,	0.26727,	0.26827,
0.26927,	0.27027,	0.27127,	0.27227,	0.27327,	0.27427,
0.27528,	0.27628,	0.27728,	0.27828,	0.27928,	0.28028,
0.28128,	0.28228,	0.28328,	0.28428,	0.28529,	0.28629,
0.28729,	0.28829,	0.28929,	0.29029,	0.29129,	0.29229,
0.29329,	0.29429,	0.2953,	0.2963,	0.2973,	0.2983,
0.2993,	0.3003,	0.3013,	0.3023,	0.3033,	0.3043,
0.30531,	0.30631,	0.30731,	0.30831,	0.30931,	0.31031,
0.31131,	0.31231,	0.31331,	0.31431,	0.31532,	0.31632,
0.31732,	0.31832,	0.31932,	0.32032,	0.32132,	0.32232,
0.32332,	0.32432,	0.32533,	0.32633,	0.32733,	0.32833,
0.32933,	0.33033,	0.33133,	0.33233,	0.33333,	0.33433,
0.33534,	0.33634,	0.33734,	0.33834,	0.33934,	0.34034,
0.34134,	0.34234,	0.34334,	0.34434,	0.34535,	0.34635,
0.34735,	0.34835,	0.34935,	0.35035,	0.35135,	0.35235,
0.35335,	0.35435,	0.35536,	0.35636,	0.35736,	0.35836,
0.35936,	0.36036,	0.36136,	0.36236,	0.36336,	0.36436,
0.36537,	0.36637,	0.36737,	0.36837,	0.36937,	0.37037,
0.37137,	0.37237,	0.37337,	0.37437,	0.37538,	0.37638,
0.37738,	0.37838,	0.37938,	0.38038,	0.38138,	0.38238,
0.38338,	0.38438,	0.38539,	0.38639,	0.38739,	0.38839,
0.38939,	0.39039,	0.39139,	0.39239,	0.39339,	0.39439,
0.3954,	0.3964,	0.3974,	0.3984,	0.3994,	0.4004,
0.4014,	0.4024,	0.4034,	0.4044,	0.40541,	0.40641,
0.40741,	0.40841,	0.40941,	0.41041,	0.41141,	0.41241,
0.41341,	0.41441,	0.41542,	0.41642,	0.41742,	0.41842,
0.41942,	0.42042,	0.42142,	0.42242,	0.42342,	0.42442,
0.42543,	0.42643,	0.42743,	0.42843,	0.42943,	0.43043,
0.43143,	0.43243,	0.43343,	0.43443,	0.43544,	0.43644,
0.43744,	0.43844,	0.43944,	0.44044,	0.44144,	0.44244,
0.44344,	0.44444,	0.44545,	0.44645,	0.44745,	0.44845,
0.44945,	0.45045,	0.45145,	0.45245,	0.45345,	0.45445,

0.45546,	0.45646,	0.45746,	0.45846,	0.45946,	0.46046,
0.46146,	0.46246,	0.46346,	0.46446,	0.46547,	0.46647,
0.46747,	0.46847,	0.46947,	0.47047,	0.47147,	0.47247,
0.47347,	0.47447,	0.47548,	0.47648,	0.47748,	0.47848,
0.47948,	0.48048,	0.48148,	0.48248,	0.48348,	0.48448,
0.48549,	0.48649,	0.48749,	0.48849,	0.48949,	0.49049,
0.49149,	0.49249,	0.49349,	0.49449,	0.4955,	0.4965,
0.4975,	0.4985,	0.4995,	0.5005,	0.5015,	0.5025,
0.5035,	0.5045,	0.50551,	0.50651,	0.50751,	0.50851,
0.50951,	0.51051,	0.51151,	0.51251,	0.51351,	0.51451,
0.51552,	0.51652,	0.51752,	0.51852,	0.51952,	0.52052,
0.52152,	0.52252,	0.52352,	0.52452,	0.52553,	0.52653,
0.52753,	0.52853,	0.52953,	0.53053,	0.53153,	0.53253,
0.53353,	0.53453,	0.53554,	0.53654,	0.53754,	0.53854,
0.53954,	0.54054,	0.54154,	0.54254,	0.54354,	0.54454,
0.54555,	0.54655,	0.54755,	0.54855,	0.54955,	0.55055,
0.55155,	0.55255,	0.55355,	0.55455,	0.55556,	0.55656,
0.55756,	0.55856,	0.55956,	0.56056,	0.56156,	0.56256,
0.56356,	0.56456,	0.56557,	0.56657,	0.56757,	0.56857,
0.56957,	0.57057,	0.57157,	0.57257,	0.57357,	0.57457,
0.57558,	0.57658,	0.57758,	0.57858,	0.57958,	0.58058,
0.58158,	0.58258,	0.58358,	0.58458,	0.58559,	0.58659,
0.58759,	0.58859,	0.58959,	0.59059,	0.59159,	0.59259,
0.59359,	0.59459,	0.5956,	0.5966,	0.5976,	0.5986,
0.5996,	0.6006,	0.6016,	0.6026,	0.6036,	0.6046,
0.60561,	0.60661,	0.60761,	0.60861,	0.60961,	0.61061,
0.61161,	0.61261,	0.61361,	0.61461,	0.61562,	0.61662,
0.61762,	0.61862,	0.61962,	0.62062,	0.62162,	0.62262,
0.62362,	0.62462,	0.62563,	0.62663,	0.62763,	0.62863,
0.62963,	0.63063,	0.63163,	0.63263,	0.63363,	0.63463,
0.63564,	0.63664,	0.63764,	0.63864,	0.63964,	0.64064,
0.64164,	0.64264,	0.64364,	0.64464,	0.64565,	0.64665,
0.64765,	0.64865,	0.64965,	0.65065,	0.65165,	0.65265,
0.65365,	0.65465,	0.65566,	0.65666,	0.65766,	0.65866,
0.65966,	0.66066,	0.66166,	0.66266,	0.66366,	0.66466,
0.66567,	0.66667,	0.66767,	0.66867,	0.66967,	0.67067,
0.67167,	0.67267,	0.67367,	0.67467,	0.67568,	0.67668,
0.67768,	0.67868,	0.67968,	0.68068,	0.68168,	0.68268,

0.68368,	0.68468,	0.68569,	0.68669,	0.68769,	0.68869,
0.68969,	0.69069,	0.69169,	0.69269,	0.69369,	0.69469,
0.6957,					
	0.6967,	0.6977,	0.6987,	0.6997,	0.7007,
0.7017,	0.7027,	0.7037,	0.7047,	0.70571,	0.70671,
0.70771,	0.70871,	0.70971,	0.71071,	0.71171,	0.71271,
0.71371,	0.71471,	0.71572,	0.71672,	0.71772,	0.71872,
0.71972,					
	0.72072,	0.72172,	0.72272,	0.72372,	0.72472,
0.72573,	0.72673,	0.72773,	0.72873,	0.72973,	0.73073,
0.73173,	0.73273,	0.73373,	0.73473,	0.73574,	0.73674,
0.73774,	0.73874,	0.73974,	0.74074,	0.74174,	0.74274,
0.74374,					
	0.74474,	0.74575,	0.74675,	0.74775,	0.74875,
0.74975,	0.75075,	0.75175,	0.75275,	0.75375,	0.75475,
0.75576,	0.75676,	0.75776,	0.75876,	0.75976,	0.76076,
0.76176,	0.76276,	0.76376,	0.76476,	0.76577,	0.76677,
0.76777,					
	0.76877,	0.76977,	0.77077,	0.77177,	0.77277,
0.77377,	0.77477,	0.77578,	0.77678,	0.77778,	0.77878,
0.77978,	0.78078,	0.78178,	0.78278,	0.78378,	0.78478,
0.78579,	0.78679,	0.78779,	0.78879,	0.78979,	0.79079,
0.79179,					
	0.79279,	0.79379,	0.79479,	0.7958,	0.7968,
0.7978,	0.7988,	0.7998,	0.8008,	0.8018,	0.8028,
0.8038,	0.8048,	0.80581,	0.80681,	0.80781,	0.80881,
0.80981,	0.81081,	0.81181,	0.81281,	0.81381,	0.81481,
0.81582,					
	0.81682,	0.81782,	0.81882,	0.81982,	0.82082,
0.82182,	0.82282,	0.82382,	0.82482,	0.82583,	0.82683,
0.82783,	0.82883,	0.82983,	0.83083,	0.83183,	0.83283,
0.83383,	0.83483,	0.83584,	0.83684,	0.83784,	0.83884,
0.83984,					
	0.84084,	0.84184,	0.84284,	0.84384,	0.84484,
0.84585,	0.84685,	0.84785,	0.84885,	0.84985,	0.85085,
0.85185,	0.85285,	0.85385,	0.85485,	0.85586,	0.85686,
0.85786,	0.85886,	0.85986,	0.86086,	0.86186,	0.86286,
0.86386,					
	0.86486,	0.86587,	0.86687,	0.86787,	0.86887,
0.86987,	0.87087,	0.87187,	0.87287,	0.87387,	0.87487,
0.87588,	0.87688,	0.87788,	0.87888,	0.87988,	0.88088,
0.88188,	0.88288,	0.88388,	0.88488,	0.88589,	0.88689,
0.88789,					
	0.88889,	0.88989,	0.89089,	0.89189,	0.89289,
0.89389,	0.89489,	0.8959,	0.8969,	0.8979,	0.8989,
0.8999,	0.9009,	0.9019,	0.9029,	0.9039,	0.9049,
0.90591,	0.90691,	0.90791,	0.90891,	0.90991,	0.91091,
0.91191,					

```

    0.91291,      0.91391,      0.91491,      0.91592,      0.91692,
0.91792,      0.91892,      0.91992,      0.92092,      0.92192,      0.92292,
0.92392,      0.92492,      0.92593,      0.92693,      0.92793,      0.92893,
0.92993,      0.93093,      0.93193,      0.93293,      0.93393,      0.93493,
0.93594,      0.93694,      0.93794,      0.93894,      0.93994,      0.94094,
0.94194,      0.94294,      0.94394,      0.94494,      0.94595,      0.94695,
0.94795,      0.94895,      0.94995,      0.95095,      0.95195,      0.95295,
0.95395,      0.95495,      0.95596,      0.95696,      0.95796,      0.95896,
0.95996,      0.96096,      0.96196,      0.96296,      0.96396,      0.96496,
0.96597,      0.96697,      0.96797,      0.96897,      0.96997,      0.97097,
0.97197,      0.97297,      0.97397,      0.97497,      0.97598,      0.97698,
0.97798,      0.97898,      0.97998,      0.98098,      0.98198,      0.98298,
0.98398,      0.98498,      0.98599,      0.98699,      0.98799,      0.98899,
0.98999,      0.99099,      0.99199,      0.99299,      0.99399,      0.99499,
0.996,        0.997,        0.998,        0.999,        1], array([[[
0.013483,    0.013489,    0.021562, ..., 1, 1, 1,
[ 0.014029,   0.01404,   0.026795, ..., 1, 1,
1,
[ 0.013152,   0.013162,   0.022694, ..., 1, 1,
1]], 'Confidence', 'Precision'], [array([
0, 0.001001, 0.002002,
0.003003, 0.004004, 0.005005, 0.006006, 0.007007, 0.008008,
0.009009, 0.01001, 0.011011, 0.012012, 0.013013, 0.014014,
0.015015, 0.016016, 0.017017, 0.018018, 0.019019, 0.02002,
0.021021, 0.022022, 0.023023,
0.024024, 0.025025, 0.026026, 0.027027, 0.028028,
0.029029, 0.03003, 0.031031, 0.032032, 0.033033, 0.034034,
0.035035, 0.036036, 0.037037, 0.038038, 0.039039, 0.04004,
0.041041, 0.042042, 0.043043, 0.044044, 0.045045, 0.046046,
0.047047,
0.048048, 0.049049, 0.05005, 0.051051, 0.052052,
0.053053, 0.054054, 0.055055, 0.056056, 0.057057, 0.058058,
0.059059, 0.06006, 0.061061, 0.062062, 0.063063, 0.064064,
0.065065, 0.066066, 0.067067, 0.068068, 0.069069, 0.07007,
0.071071,
0.072072, 0.073073, 0.074074, 0.075075, 0.076076,
0.077077, 0.078078, 0.079079, 0.08008, 0.081081, 0.082082,
0.083083, 0.084084, 0.085085, 0.086086, 0.087087, 0.088088,
0.089089, 0.09009, 0.091091, 0.092092, 0.093093, 0.094094,
0.095095,
0.096096, 0.097097, 0.098098, 0.099099, 0.1001,
0.1011, 0.1021, 0.1031, 0.1041, 0.10511, 0.10611,
0.10711, 0.10811, 0.10911, 0.11011, 0.11111, 0.11211,
0.11311, 0.11411, 0.11512, 0.11612, 0.11712, 0.11812,
0.11912,
0.12012, 0.12112, 0.12212, 0.12312, 0.12412,

```

0.12513,	0.12613,	0.12713,	0.12813,	0.12913,	0.13013,
0.13113,	0.13213,	0.13313,	0.13413,	0.13514,	0.13614,
0.13714,	0.13814,	0.13914,	0.14014,	0.14114,	0.14214,
0.14314,	0.14414,	0.14515,	0.14615,	0.14715,	0.14815,
0.14915,	0.15015,	0.15115,	0.15215,	0.15315,	0.15415,
0.15516,	0.15616,	0.15716,	0.15816,	0.15916,	0.16016,
0.16116,	0.16216,	0.16316,	0.16416,	0.16517,	0.16617,
0.16717,	0.16817,	0.16917,	0.17017,	0.17117,	0.17217,
0.17317,	0.17417,	0.17518,	0.17618,	0.17718,	0.17818,
0.17918,	0.18018,	0.18118,	0.18218,	0.18318,	0.18418,
0.18519,	0.18619,	0.18719,	0.18819,	0.18919,	0.19019,
0.19119,	0.19219,	0.19319,	0.19419,	0.1952,	0.1962,
0.1972,	0.1982,	0.1992,	0.2002,	0.2012,	0.2022,
0.2032,	0.2042,	0.20521,	0.20621,	0.20721,	0.20821,
0.20921,	0.21021,	0.21121,	0.21221,	0.21321,	0.21421,
0.21522,	0.21622,	0.21722,	0.21822,	0.21922,	0.22022,
0.22122,	0.22222,	0.22322,	0.22422,	0.22523,	0.22623,
0.22723,	0.22823,	0.22923,	0.23023,	0.23123,	0.23223,
0.23323,	0.23423,	0.23524,	0.23624,	0.23724,	0.23824,
0.23924,	0.24024,	0.24124,	0.24224,	0.24324,	0.24424,
0.24525,	0.24625,	0.24725,	0.24825,	0.24925,	0.25025,
0.25125,	0.25225,	0.25325,	0.25425,	0.25526,	0.25626,
0.25726,	0.25826,	0.25926,	0.26026,	0.26126,	0.26226,
0.26326,	0.26426,	0.26527,	0.26627,	0.26727,	0.26827,
0.26927,	0.27027,	0.27127,	0.27227,	0.27327,	0.27427,
0.27528,	0.27628,	0.27728,	0.27828,	0.27928,	0.28028,
0.28128,	0.28228,	0.28328,	0.28428,	0.28529,	0.28629,
0.28729,	0.28829,	0.28929,	0.29029,	0.29129,	0.29229,
0.29329,	0.29429,	0.2953,	0.2963,	0.2973,	0.2983,
0.2993,	0.3003,	0.3013,	0.3023,	0.3033,	0.3043,
0.30531,	0.30631,	0.30731,	0.30831,	0.30931,	0.31031,
0.31131,	0.31231,	0.31331,	0.31431,	0.31532,	0.31632,
0.31732,	0.31832,	0.31932,	0.32032,	0.32132,	0.32232,
0.32332,	0.32432,	0.32533,	0.32633,	0.32733,	0.32833,
0.32933,	0.33033,	0.33133,	0.33233,	0.33333,	0.33433,
0.33534,	0.33634,	0.33734,	0.33834,	0.33934,	0.34034,
0.34134,	0.34234,	0.34334,	0.34434,	0.34535,	0.34635,
0.34735,	0.34835,	0.34935,	0.35035,	0.35135,	0.35235,
0.35335,	0.35435,	0.35536,	0.35636,	0.35736,	0.35836,

0.35936,					
	0.36036,	0.36136,	0.36236,	0.36336,	0.36436,
0.36537,	0.36637,	0.36737,	0.36837,	0.36937,	0.37037,
0.37137,	0.37237,	0.37337,	0.37437,	0.37538,	0.37638,
0.37738,	0.37838,	0.37938,	0.38038,	0.38138,	0.38238,
0.38338,					
	0.38438,	0.38539,	0.38639,	0.38739,	0.38839,
0.38939,	0.39039,	0.39139,	0.39239,	0.39339,	0.39439,
0.3954,	0.3964,	0.3974,	0.3984,	0.3994,	0.4004,
0.4014,	0.4024,	0.4034,	0.4044,	0.40541,	0.40641,
0.40741,					
	0.40841,	0.40941,	0.41041,	0.41141,	0.41241,
0.41341,	0.41441,	0.41542,	0.41642,	0.41742,	0.41842,
0.41942,	0.42042,	0.42142,	0.42242,	0.42342,	0.42442,
0.42543,	0.42643,	0.42743,	0.42843,	0.42943,	0.43043,
0.43143,					
	0.43243,	0.43343,	0.43443,	0.43544,	0.43644,
0.43744,	0.43844,	0.43944,	0.44044,	0.44144,	0.44244,
0.44344,	0.44444,	0.44545,	0.44645,	0.44745,	0.44845,
0.44945,	0.45045,	0.45145,	0.45245,	0.45345,	0.45445,
0.45546,					
	0.45646,	0.45746,	0.45846,	0.45946,	0.46046,
0.46146,	0.46246,	0.46346,	0.46446,	0.46547,	0.46647,
0.46747,	0.46847,	0.46947,	0.47047,	0.47147,	0.47247,
0.47347,	0.47447,	0.47548,	0.47648,	0.47748,	0.47848,
0.47948,					
	0.48048,	0.48148,	0.48248,	0.48348,	0.48448,
0.48549,	0.48649,	0.48749,	0.48849,	0.48949,	0.49049,
0.49149,	0.49249,	0.49349,	0.49449,	0.4955,	0.4965,
0.4975,	0.4985,	0.4995,	0.5005,	0.5015,	0.5025,
0.5035,					
	0.5045,	0.50551,	0.50651,	0.50751,	0.50851,
0.50951,	0.51051,	0.51151,	0.51251,	0.51351,	0.51451,
0.51552,	0.51652,	0.51752,	0.51852,	0.51952,	0.52052,
0.52152,	0.52252,	0.52352,	0.52452,	0.52553,	0.52653,
0.52753,					
	0.52853,	0.52953,	0.53053,	0.53153,	0.53253,
0.53353,	0.53453,	0.53554,	0.53654,	0.53754,	0.53854,
0.53954,	0.54054,	0.54154,	0.54254,	0.54354,	0.54454,
0.54555,	0.54655,	0.54755,	0.54855,	0.54955,	0.55055,
0.55155,					
	0.55255,	0.55355,	0.55455,	0.55556,	0.55656,
0.55756,	0.55856,	0.55956,	0.56056,	0.56156,	0.56256,
0.56356,	0.56456,	0.56557,	0.56657,	0.56757,	0.56857,
0.56957,	0.57057,	0.57157,	0.57257,	0.57357,	0.57457,
0.57558,					
	0.57658,	0.57758,	0.57858,	0.57958,	0.58058,
0.58158,	0.58258,	0.58358,	0.58458,	0.58559,	0.58659,

0.58759,	0.58859,	0.58959,	0.59059,	0.59159,	0.59259,
0.59359,	0.59459,	0.5956,	0.5966,	0.5976,	0.5986,
0.5996,					
	0.6006,	0.6016,	0.6026,	0.6036,	0.6046,
0.60561,	0.60661,	0.60761,	0.60861,	0.60961,	0.61061,
0.61161,	0.61261,	0.61361,	0.61461,	0.61562,	0.61662,
0.61762,	0.61862,	0.61962,	0.62062,	0.62162,	0.62262,
0.62362,					
	0.62462,	0.62563,	0.62663,	0.62763,	0.62863,
0.62963,	0.63063,	0.63163,	0.63263,	0.63363,	0.63463,
0.63564,	0.63664,	0.63764,	0.63864,	0.63964,	0.64064,
0.64164,	0.64264,	0.64364,	0.64464,	0.64565,	0.64665,
0.64765,					
	0.64865,	0.64965,	0.65065,	0.65165,	0.65265,
0.65365,	0.65465,	0.65566,	0.65666,	0.65766,	0.65866,
0.65966,	0.66066,	0.66166,	0.66266,	0.66366,	0.66466,
0.66567,	0.66667,	0.66767,	0.66867,	0.66967,	0.67067,
0.67167,					
	0.67267,	0.67367,	0.67467,	0.67568,	0.67668,
0.67768,	0.67868,	0.67968,	0.68068,	0.68168,	0.68268,
0.68368,	0.68468,	0.68569,	0.68669,	0.68769,	0.68869,
0.68969,	0.69069,	0.69169,	0.69269,	0.69369,	0.69469,
0.6957,					
	0.6967,	0.6977,	0.6987,	0.6997,	0.7007,
0.7017,	0.7027,	0.7037,	0.7047,	0.70571,	0.70671,
0.70771,	0.70871,	0.70971,	0.71071,	0.71171,	0.71271,
0.71371,	0.71471,	0.71572,	0.71672,	0.71772,	0.71872,
0.71972,					
	0.72072,	0.72172,	0.72272,	0.72372,	0.72472,
0.72573,	0.72673,	0.72773,	0.72873,	0.72973,	0.73073,
0.73173,	0.73273,	0.73373,	0.73473,	0.73574,	0.73674,
0.73774,	0.73874,	0.73974,	0.74074,	0.74174,	0.74274,
0.74374,					
	0.74474,	0.74575,	0.74675,	0.74775,	0.74875,
0.74975,	0.75075,	0.75175,	0.75275,	0.75375,	0.75475,
0.75576,	0.75676,	0.75776,	0.75876,	0.75976,	0.76076,
0.76176,	0.76276,	0.76376,	0.76476,	0.76577,	0.76677,
0.76777,					
	0.76877,	0.76977,	0.77077,	0.77177,	0.77277,
0.77377,	0.77477,	0.77578,	0.77678,	0.77778,	0.77878,
0.77978,	0.78078,	0.78178,	0.78278,	0.78378,	0.78478,
0.78579,	0.78679,	0.78779,	0.78879,	0.78979,	0.79079,
0.79179,					
	0.79279,	0.79379,	0.79479,	0.7958,	0.7968,
0.7978,	0.7988,	0.7998,	0.8008,	0.8018,	0.8028,
0.8038,	0.8048,	0.80581,	0.80681,	0.80781,	0.80881,
0.80981,	0.81081,	0.81181,	0.81281,	0.81381,	0.81481,
0.81582,					

```

0.81682,      0.81782,      0.81882,      0.81982,      0.82082,
0.82182,      0.82282,      0.82382,      0.82482,      0.82583,      0.82683,
0.82783,      0.82883,      0.82983,      0.83083,      0.83183,      0.83283,
0.83383,      0.83483,      0.83584,      0.83684,      0.83784,      0.83884,
0.83984,
0.84084,      0.84184,      0.84284,      0.84384,      0.84484,
0.84585,      0.84685,      0.84785,      0.84885,      0.84985,      0.85085,
0.85185,      0.85285,      0.85385,      0.85485,      0.85586,      0.85686,
0.85786,      0.85886,      0.85986,      0.86086,      0.86186,      0.86286,
0.86386,
0.86486,      0.86587,      0.86687,      0.86787,      0.86887,
0.86987,      0.87087,      0.87187,      0.87287,      0.87387,      0.87487,
0.87588,      0.87688,      0.87788,      0.87888,      0.87988,      0.88088,
0.88188,      0.88288,      0.88388,      0.88488,      0.88589,      0.88689,
0.88789,
0.88889,      0.88989,      0.89089,      0.89189,      0.89289,
0.89389,      0.89489,      0.8959,       0.8969,       0.8979,       0.8989,
0.8999,       0.9009,       0.9019,       0.9029,       0.9039,       0.9049,
0.90591,      0.90691,      0.90791,      0.90891,      0.90991,      0.91091,
0.91191,
0.91291,      0.91391,      0.91491,      0.91592,      0.91692,
0.91792,      0.91892,      0.91992,      0.92092,      0.92192,      0.92292,
0.92392,      0.92492,      0.92593,      0.92693,      0.92793,      0.92893,
0.92993,      0.93093,      0.93193,      0.93293,      0.93393,      0.93493,
0.93594,
0.93694,      0.93794,      0.93894,      0.93994,      0.94094,
0.94194,      0.94294,      0.94394,      0.94494,      0.94595,      0.94695,
0.94795,      0.94895,      0.94995,      0.95095,      0.95195,      0.95295,
0.95395,      0.95495,      0.95596,      0.95696,      0.95796,      0.95896,
0.95996,
0.96096,      0.96196,      0.96296,      0.96396,      0.96496,
0.96597,      0.96697,      0.96797,      0.96897,      0.96997,      0.97097,
0.97197,      0.97297,      0.97397,      0.97497,      0.97598,      0.97698,
0.97798,      0.97898,      0.97998,      0.98098,      0.98198,      0.98298,
0.98398,
0.98498,      0.98599,      0.98699,      0.98799,      0.98899,
0.98999,      0.99099,      0.99199,      0.99299,      0.99399,      0.99499,
0.996,        0.997,        0.998,        0.999,        1]), array([[[
0.82825,      0.82825,      0.80738,      ...,          0,          0,          0],
[ 0.90072,      0.90072,      0.875,        ...,          0,          0,          0],
[ 0.83445,      0.83445,      0.81008,      ...,          0,          0,          0],
[ 0.83445,      0.83445,      0.81008,      ...,          0,          0,          0]]),
'Confidence', 'Recall']]
```

fitness: 0.2889355431761369

keys: ['metrics/precision(B)', 'metrics/recall(B)', 'metrics/mAP50(B)', 'metrics/mAP50-95(B)']

maps: array([0.16198, 0.40193, 0.22017])

names: {0: 'label0', 1: 'label1', 2: 'label2'}

```

plot: True
results_dict: {'metrics/precision(B)': 0.6685934100107295, 'metrics/recall(B)': 0.5080120793867091, 'metrics/mAP50(B)': 0.5371115183986149, 'metrics/mAP50-95(B)': 0.26136043481808374, 'fitness': 0.2889355431761369}
save_dir: PosixPath('runs/detect/train122')
speed: {'preprocess': 0.17830085272740834, 'inference': 1.4606337354640768, 'loss': 0.002456674672136403, 'postprocess': 0.6857709451155228}
task: 'detect'

```

Running Testing on Test Dataset...

Ultralytics 8.3.38 Python-3.10.15 torch-2.5.1+cu118 CUDA:0 (NVIDIA A100-SXM4-40GB, 40514MiB)

```

val: Scanning /root/notebooks/xray/data/BrainTumorDetectionYolov9/B
rainTumorDetectionYolov9/test/labels... 990 images, 17 backgrounds, 0 corrupt:
100%| 990/990 [00:02<00:00, 346.19it/s]

```

```

val: New cache created: /root/notebooks/xray/data/BrainTumorDetecti
onYolov9/BrainTumorDetectionYolov9/test/labels.cache

```

	Class	Images	Instances	Box(P)	R	mAP50
mAP50-95): 100%		248/248	[00:05<00:00, 43.67it/s]			
0.276	all	990	2104	0.646	0.516	0.553
0.188	label0	588	588	0.606	0.4	0.443
0.407	label1	962	962	0.733	0.674	0.707
0.232	label2	554	554	0.599	0.475	0.511

Speed: 0.2ms preprocess, 1.8ms inference, 0.0ms loss, 0.8ms postprocess per image

Results saved to runs/detect/train123

Test Results:

ultralytics.utils.metrics.DetMetrics object with attributes:

```

ap_class_index: array([0, 1, 2])
box: ultralytics.utils.metrics.Metric object
confusion_matrix: <ultralytics.utils.metrics.ConfusionMatrix object at
0x7f0e7062b5e0>
curves: ['Precision-Recall(B)', 'F1-Confidence(B)', 'Precision-Confidence(B)', 'Recall-Confidence(B)']
curves_results: [[array([
0,      0.001001,     0.002002,     0.003003,
0.004004,    0.005005,     0.006006,     0.007007,     0.008008,     0.009009,
0.01001,    0.011011,     0.012012,     0.013013,     0.014014,     0.015015,
0.016016,    0.017017,     0.018018,     0.019019,     0.02002,      0.021021,
0.022022,    0.023023,
0.024024,    0.025025,     0.026026,     0.027027,     0.028028,

```

0.029029,	0.03003,	0.031031,	0.032032,	0.033033,	0.034034,
0.035035,	0.036036,	0.037037,	0.038038,	0.039039,	0.04004,
0.041041,	0.042042,	0.043043,	0.044044,	0.045045,	0.046046,
0.047047,					
	0.048048,	0.049049,	0.05005,	0.051051,	0.052052,
0.053053,	0.054054,	0.055055,	0.056056,	0.057057,	0.058058,
0.059059,	0.06006,	0.061061,	0.062062,	0.063063,	0.064064,
0.065065,	0.066066,	0.067067,	0.068068,	0.069069,	0.07007,
0.071071,					
	0.072072,	0.073073,	0.074074,	0.075075,	0.076076,
0.077077,	0.078078,	0.079079,	0.08008,	0.081081,	0.082082,
0.083083,	0.084084,	0.085085,	0.086086,	0.087087,	0.088088,
0.089089,	0.09009,	0.091091,	0.092092,	0.093093,	0.094094,
0.095095,					
	0.096096,	0.097097,	0.098098,	0.099099,	0.1001,
0.1011,	0.1021,	0.1031,	0.1041,	0.10511,	0.10611,
0.10711,	0.10811,	0.10911,	0.11011,	0.11111,	0.11211,
0.11311,	0.11411,	0.11512,	0.11612,	0.11712,	0.11812,
0.11912,					
	0.12012,	0.12112,	0.12212,	0.12312,	0.12412,
0.12513,	0.12613,	0.12713,	0.12813,	0.12913,	0.13013,
0.13113,	0.13213,	0.13313,	0.13413,	0.13514,	0.13614,
0.13714,	0.13814,	0.13914,	0.14014,	0.14114,	0.14214,
0.14314,					
	0.14414,	0.14515,	0.14615,	0.14715,	0.14815,
0.14915,	0.15015,	0.15115,	0.15215,	0.15315,	0.15415,
0.15516,	0.15616,	0.15716,	0.15816,	0.15916,	0.16016,
0.16116,	0.16216,	0.16316,	0.16416,	0.16517,	0.16617,
0.16717,					
	0.16817,	0.16917,	0.17017,	0.17117,	0.17217,
0.17317,	0.17417,	0.17518,	0.17618,	0.17718,	0.17818,
0.17918,	0.18018,	0.18118,	0.18218,	0.18318,	0.18418,
0.18519,	0.18619,	0.18719,	0.18819,	0.18919,	0.19019,
0.19119,					
	0.19219,	0.19319,	0.19419,	0.1952,	0.1962,
0.1972,	0.1982,	0.1992,	0.2002,	0.2012,	0.2022,
0.2032,	0.2042,	0.20521,	0.20621,	0.20721,	0.20821,
0.20921,	0.21021,	0.21121,	0.21221,	0.21321,	0.21421,
0.21522,					
	0.21622,	0.21722,	0.21822,	0.21922,	0.22022,
0.22122,	0.22222,	0.22322,	0.22422,	0.22523,	0.22623,
0.22723,	0.22823,	0.22923,	0.23023,	0.23123,	0.23223,
0.23323,	0.23423,	0.23524,	0.23624,	0.23724,	0.23824,
0.23924,					
	0.24024,	0.24124,	0.24224,	0.24324,	0.24424,
0.24525,	0.24625,	0.24725,	0.24825,	0.24925,	0.25025,
0.25125,	0.25225,	0.25325,	0.25425,	0.25526,	0.25626,
0.25726,	0.25826,	0.25926,	0.26026,	0.26126,	0.26226,

0.26326,	0.26426,	0.26527,	0.26627,	0.26727,	0.26827,
0.26927,	0.27027,	0.27127,	0.27227,	0.27327,	0.27427,
0.27528,	0.27628,	0.27728,	0.27828,	0.27928,	0.28028,
0.28128,	0.28228,	0.28328,	0.28428,	0.28529,	0.28629,
0.28729,	0.28829,	0.28929,	0.29029,	0.29129,	0.29229,
0.29329,	0.29429,	0.2953,	0.2963,	0.2973,	0.2983,
0.2993,	0.3003,	0.3013,	0.3023,	0.3033,	0.3043,
0.30531,	0.30631,	0.30731,	0.30831,	0.30931,	0.31031,
0.31131,	0.31231,	0.31331,	0.31431,	0.31532,	0.31632,
0.31732,	0.31832,	0.31932,	0.32032,	0.32132,	0.32232,
0.32332,	0.32432,	0.32533,	0.32633,	0.32733,	0.32833,
0.32933,	0.33033,	0.33133,	0.33233,	0.33333,	0.33433,
0.33534,	0.33634,	0.33734,	0.33834,	0.33934,	0.34034,
0.34134,	0.34234,	0.34334,	0.34434,	0.34535,	0.34635,
0.34735,	0.34835,	0.34935,	0.35035,	0.35135,	0.35235,
0.35335,	0.35435,	0.35536,	0.35636,	0.35736,	0.35836,
0.35936,	0.36036,	0.36136,	0.36236,	0.36336,	0.36436,
0.36537,	0.36637,	0.36737,	0.36837,	0.36937,	0.37037,
0.37137,	0.37237,	0.37337,	0.37437,	0.37538,	0.37638,
0.37738,	0.37838,	0.37938,	0.38038,	0.38138,	0.38238,
0.38338,	0.38438,	0.38539,	0.38639,	0.38739,	0.38839,
0.38939,	0.39039,	0.39139,	0.39239,	0.39339,	0.39439,
0.3954,	0.3964,	0.3974,	0.3984,	0.3994,	0.4004,
0.4014,	0.4024,	0.4034,	0.4044,	0.40541,	0.40641,
0.40741,	0.40841,	0.40941,	0.41041,	0.41141,	0.41241,
0.41341,	0.41441,	0.41542,	0.41642,	0.41742,	0.41842,
0.41942,	0.42042,	0.42142,	0.42242,	0.42342,	0.42442,
0.42543,	0.42643,	0.42743,	0.42843,	0.42943,	0.43043,
0.43143,	0.43243,	0.43343,	0.43443,	0.43544,	0.43644,
0.43744,	0.43844,	0.43944,	0.44044,	0.44144,	0.44244,
0.44344,	0.44444,	0.44545,	0.44645,	0.44745,	0.44845,
0.44945,	0.45045,	0.45145,	0.45245,	0.45345,	0.45445,
0.45546,	0.45646,	0.45746,	0.45846,	0.45946,	0.46046,
0.46146,	0.46246,	0.46346,	0.46446,	0.46547,	0.46647,
0.46747,	0.46847,	0.46947,	0.47047,	0.47147,	0.47247,
0.47347,	0.47447,	0.47548,	0.47648,	0.47748,	0.47848,
0.47948,	0.48048,	0.48148,	0.48248,	0.48348,	0.48448,
0.48549,	0.48649,	0.48749,	0.48849,	0.48949,	0.49049,

0.49149,	0.49249,	0.49349,	0.49449,	0.4955,	0.4965,
0.4975,	0.4985,	0.4995,	0.5005,	0.5015,	0.5025,
0.5035,					
	0.5045,	0.50551,	0.50651,	0.50751,	0.50851,
0.50951,	0.51051,	0.51151,	0.51251,	0.51351,	0.51451,
0.51552,	0.51652,	0.51752,	0.51852,	0.51952,	0.52052,
0.52152,	0.52252,	0.52352,	0.52452,	0.52553,	0.52653,
0.52753,					
	0.52853,	0.52953,	0.53053,	0.53153,	0.53253,
0.53353,	0.53453,	0.53554,	0.53654,	0.53754,	0.53854,
0.53954,	0.54054,	0.54154,	0.54254,	0.54354,	0.54454,
0.54555,	0.54655,	0.54755,	0.54855,	0.54955,	0.55055,
0.55155,					
	0.55255,	0.55355,	0.55455,	0.55556,	0.55656,
0.55756,	0.55856,	0.55956,	0.56056,	0.56156,	0.56256,
0.56356,	0.56456,	0.56557,	0.56657,	0.56757,	0.56857,
0.56957,	0.57057,	0.57157,	0.57257,	0.57357,	0.57457,
0.57558,					
	0.57658,	0.57758,	0.57858,	0.57958,	0.58058,
0.58158,	0.58258,	0.58358,	0.58458,	0.58559,	0.58659,
0.58759,	0.58859,	0.58959,	0.59059,	0.59159,	0.59259,
0.59359,	0.59459,	0.5956,	0.5966,	0.5976,	0.5986,
0.5996,					
	0.6006,	0.6016,	0.6026,	0.6036,	0.6046,
0.60561,	0.60661,	0.60761,	0.60861,	0.60961,	0.61061,
0.61161,	0.61261,	0.61361,	0.61461,	0.61562,	0.61662,
0.61762,	0.61862,	0.61962,	0.62062,	0.62162,	0.62262,
0.62362,					
	0.62462,	0.62563,	0.62663,	0.62763,	0.62863,
0.62963,	0.63063,	0.63163,	0.63263,	0.63363,	0.63463,
0.63564,	0.63664,	0.63764,	0.63864,	0.63964,	0.64064,
0.64164,	0.64264,	0.64364,	0.64464,	0.64565,	0.64665,
0.64765,					
	0.64865,	0.64965,	0.65065,	0.65165,	0.65265,
0.65365,	0.65465,	0.65566,	0.65666,	0.65766,	0.65866,
0.65966,	0.66066,	0.66166,	0.66266,	0.66366,	0.66466,
0.66567,	0.66667,	0.66767,	0.66867,	0.66967,	0.67067,
0.67167,					
	0.67267,	0.67367,	0.67467,	0.67568,	0.67668,
0.67768,	0.67868,	0.67968,	0.68068,	0.68168,	0.68268,
0.68368,	0.68468,	0.68569,	0.68669,	0.68769,	0.68869,
0.68969,	0.69069,	0.69169,	0.69269,	0.69369,	0.69469,
0.6957,					
	0.6967,	0.6977,	0.6987,	0.6997,	0.7007,
0.7017,	0.7027,	0.7037,	0.7047,	0.70571,	0.70671,
0.70771,	0.70871,	0.70971,	0.71071,	0.71171,	0.71271,
0.71371,	0.71471,	0.71572,	0.71672,	0.71772,	0.71872,
0.71972,					

	0.72072,	0.72172,	0.72272,	0.72372,	0.72472,
0.72573,	0.72673,	0.72773,	0.72873,	0.72973,	0.73073,
0.73173,	0.73273,	0.73373,	0.73473,	0.73574,	0.73674,
0.73774,	0.73874,	0.73974,	0.74074,	0.74174,	0.74274,
0.74374,					
	0.74474,	0.74575,	0.74675,	0.74775,	0.74875,
0.74975,	0.75075,	0.75175,	0.75275,	0.75375,	0.75475,
0.75576,	0.75676,	0.75776,	0.75876,	0.75976,	0.76076,
0.76176,	0.76276,	0.76376,	0.76476,	0.76577,	0.76677,
0.76777,					
	0.76877,	0.76977,	0.77077,	0.77177,	0.77277,
0.77377,	0.77477,	0.77578,	0.77678,	0.77778,	0.77878,
0.77978,	0.78078,	0.78178,	0.78278,	0.78378,	0.78478,
0.78579,	0.78679,	0.78779,	0.78879,	0.78979,	0.79079,
0.79179,					
	0.79279,	0.79379,	0.79479,	0.7958,	0.7968,
0.7978,	0.7988,	0.7998,	0.8008,	0.8018,	0.8028,
0.8038,	0.8048,	0.80581,	0.80681,	0.80781,	0.80881,
0.80981,	0.81081,	0.81181,	0.81281,	0.81381,	0.81481,
0.81582,					
	0.81682,	0.81782,	0.81882,	0.81982,	0.82082,
0.82182,	0.82282,	0.82382,	0.82482,	0.82583,	0.82683,
0.82783,	0.82883,	0.82983,	0.83083,	0.83183,	0.83283,
0.83383,	0.83483,	0.83584,	0.83684,	0.83784,	0.83884,
0.83984,					
	0.84084,	0.84184,	0.84284,	0.84384,	0.84484,
0.84585,	0.84685,	0.84785,	0.84885,	0.84985,	0.85085,
0.85185,	0.85285,	0.85385,	0.85485,	0.85586,	0.85686,
0.85786,	0.85886,	0.85986,	0.86086,	0.86186,	0.86286,
0.86386,					
	0.86486,	0.86587,	0.86687,	0.86787,	0.86887,
0.86987,	0.87087,	0.87187,	0.87287,	0.87387,	0.87487,
0.87588,	0.87688,	0.87788,	0.87888,	0.87988,	0.88088,
0.88188,	0.88288,	0.88388,	0.88488,	0.88589,	0.88689,
0.88789,					
	0.88889,	0.88989,	0.89089,	0.89189,	0.89289,
0.89389,	0.89489,	0.8959,	0.8969,	0.8979,	0.8989,
0.8999,	0.9009,	0.9019,	0.9029,	0.9039,	0.9049,
0.90591,	0.90691,	0.90791,	0.90891,	0.90991,	0.91091,
0.91191,					
	0.91291,	0.91391,	0.91491,	0.91592,	0.91692,
0.91792,	0.91892,	0.91992,	0.92092,	0.92192,	0.92292,
0.92392,	0.92492,	0.92593,	0.92693,	0.92793,	0.92893,
0.92993,	0.93093,	0.93193,	0.93293,	0.93393,	0.93493,
0.93594,					
	0.93694,	0.93794,	0.93894,	0.93994,	0.94094,
0.94194,	0.94294,	0.94394,	0.94494,	0.94595,	0.94695,
0.94795,	0.94895,	0.94995,	0.95095,	0.95195,	0.95295,

0.95395,	0.95495,	0.95596,	0.95696,	0.95796,	0.95896,
0.95996,	0.96096,	0.96196,	0.96296,	0.96396,	0.96496,
0.96597,	0.96697,	0.96797,	0.96897,	0.96997,	0.97097,
0.97197,	0.97297,	0.97397,	0.97497,	0.97598,	0.97698,
0.97798,	0.97898,	0.97998,	0.98098,	0.98198,	0.98298,
0.98398,	0.98498,	0.98599,	0.98699,	0.98799,	0.98899,
0.98999,	0.99099,	0.99199,	0.99299,	0.99399,	0.99499,
0.996,	0.997,	0.998,	0.999,	1]), array([[
1,	1,	1, ... , 0.00014273,	7.1367e-05,	0],	
[1,	1,	1, ... , 0.00029556,	0.00014778,	
0],	[1,	1,	1, ... , 0.00019251,	9.6254e-05,
0]], 'Recall', 'Precision'], [array([0,	0.001001,	0.002002,		
0.003003,	0.004004,	0.005005,	0.006006,	0.007007,	0.008008,
0.009009,	0.01001,	0.011011,	0.012012,	0.013013,	0.014014,
0.015015,	0.016016,	0.017017,	0.018018,	0.019019,	0.02002,
0.021021,	0.022022,	0.023023,	0.024024,	0.025025,	0.026026,
			0.027027,	0.028028,	
0.029029,	0.03003,	0.031031,	0.032032,	0.033033,	0.034034,
0.035035,	0.036036,	0.037037,	0.038038,	0.039039,	0.04004,
0.041041,	0.042042,	0.043043,	0.044044,	0.045045,	0.046046,
0.047047,	0.048048,	0.049049,	0.05005,	0.051051,	0.052052,
0.053053,	0.054054,	0.055055,	0.056056,	0.057057,	0.058058,
0.059059,	0.06006,	0.061061,	0.062062,	0.063063,	0.064064,
0.065065,	0.066066,	0.067067,	0.068068,	0.069069,	0.07007,
0.071071,	0.072072,	0.073073,	0.074074,	0.075075,	0.076076,
0.077077,	0.078078,	0.079079,	0.08008,	0.081081,	0.082082,
0.083083,	0.084084,	0.085085,	0.086086,	0.087087,	0.088088,
0.089089,	0.09009,	0.091091,	0.092092,	0.093093,	0.094094,
0.095095,	0.096096,	0.097097,	0.098098,	0.099099,	0.1001,
0.1011,	0.1021,	0.1031,	0.1041,	0.10511,	0.10611,
0.10711,	0.10811,	0.10911,	0.11011,	0.11111,	0.11211,
0.11311,	0.11411,	0.11512,	0.11612,	0.11712,	0.11812,
0.11912,	0.12012,	0.12112,	0.12212,	0.12312,	0.12412,
0.12513,	0.12613,	0.12713,	0.12813,	0.12913,	0.13013,
0.13113,	0.13213,	0.13313,	0.13413,	0.13514,	0.13614,
0.13714,	0.13814,	0.13914,	0.14014,	0.14114,	0.14214,
0.14314,	0.14414,	0.14515,	0.14615,	0.14715,	0.14815,
0.14915,	0.15015,	0.15115,	0.15215,	0.15315,	0.15415,
0.15516,	0.15616,	0.15716,	0.15816,	0.15916,	0.16016,
0.16116,	0.16216,	0.16316,	0.16416,	0.16517,	0.16617,

0.16717,	0.16817,	0.16917,	0.17017,	0.17117,	0.17217,
0.17317,	0.17417,	0.17518,	0.17618,	0.17718,	0.17818,
0.17918,	0.18018,	0.18118,	0.18218,	0.18318,	0.18418,
0.18519,	0.18619,	0.18719,	0.18819,	0.18919,	0.19019,
0.19119,	0.19219,	0.19319,	0.19419,	0.1952,	0.1962,
0.1972,	0.1982,	0.1992,	0.2002,	0.2012,	0.2022,
0.2032,	0.2042,	0.20521,	0.20621,	0.20721,	0.20821,
0.20921,	0.21021,	0.21121,	0.21221,	0.21321,	0.21421,
0.21522,	0.21622,	0.21722,	0.21822,	0.21922,	0.22022,
0.22122,	0.22222,	0.22322,	0.22422,	0.22523,	0.22623,
0.22723,	0.22823,	0.22923,	0.23023,	0.23123,	0.23223,
0.23323,	0.23423,	0.23524,	0.23624,	0.23724,	0.23824,
0.23924,	0.24024,	0.24124,	0.24224,	0.24324,	0.24424,
0.24525,	0.24625,	0.24725,	0.24825,	0.24925,	0.25025,
0.25125,	0.25225,	0.25325,	0.25425,	0.25526,	0.25626,
0.25726,	0.25826,	0.25926,	0.26026,	0.26126,	0.26226,
0.26326,	0.26426,	0.26527,	0.26627,	0.26727,	0.26827,
0.26927,	0.27027,	0.27127,	0.27227,	0.27327,	0.27427,
0.27528,	0.27628,	0.27728,	0.27828,	0.27928,	0.28028,
0.28128,	0.28228,	0.28328,	0.28428,	0.28529,	0.28629,
0.28729,	0.28829,	0.28929,	0.29029,	0.29129,	0.29229,
0.29329,	0.29429,	0.2953,	0.2963,	0.2973,	0.2983,
0.2993,	0.3003,	0.3013,	0.3023,	0.3033,	0.3043,
0.30531,	0.30631,	0.30731,	0.30831,	0.30931,	0.31031,
0.31131,	0.31231,	0.31331,	0.31431,	0.31532,	0.31632,
0.31732,	0.31832,	0.31932,	0.32032,	0.32132,	0.32232,
0.32332,	0.32432,	0.32533,	0.32633,	0.32733,	0.32833,
0.32933,	0.33033,	0.33133,	0.33233,	0.33333,	0.33433,
0.33534,	0.33634,	0.33734,	0.33834,	0.33934,	0.34034,
0.34134,	0.34234,	0.34334,	0.34434,	0.34535,	0.34635,
0.34735,	0.34835,	0.34935,	0.35035,	0.35135,	0.35235,
0.35335,	0.35435,	0.35536,	0.35636,	0.35736,	0.35836,
0.35936,	0.36036,	0.36136,	0.36236,	0.36336,	0.36436,
0.36537,	0.36637,	0.36737,	0.36837,	0.36937,	0.37037,
0.37137,	0.37237,	0.37337,	0.37437,	0.37538,	0.37638,
0.37738,	0.37838,	0.37938,	0.38038,	0.38138,	0.38238,
0.38338,	0.38438,	0.38539,	0.38639,	0.38739,	0.38839,
0.38939,	0.39039,	0.39139,	0.39239,	0.39339,	0.39439,

0.3954,	0.3964,	0.3974,	0.3984,	0.3994,	0.4004,
0.4014,	0.4024,	0.4034,	0.4044,	0.40541,	0.40641,
0.40741,					
	0.40841,	0.40941,	0.41041,	0.41141,	0.41241,
0.41341,	0.41441,	0.41542,	0.41642,	0.41742,	0.41842,
0.41942,	0.42042,	0.42142,	0.42242,	0.42342,	0.42442,
0.42543,	0.42643,	0.42743,	0.42843,	0.42943,	0.43043,
0.43143,					
	0.43243,	0.43343,	0.43443,	0.43544,	0.43644,
0.43744,	0.43844,	0.43944,	0.44044,	0.44144,	0.44244,
0.44344,	0.44444,	0.44545,	0.44645,	0.44745,	0.44845,
0.44945,	0.45045,	0.45145,	0.45245,	0.45345,	0.45445,
0.45546,					
	0.45646,	0.45746,	0.45846,	0.45946,	0.46046,
0.46146,	0.46246,	0.46346,	0.46446,	0.46547,	0.46647,
0.46747,	0.46847,	0.46947,	0.47047,	0.47147,	0.47247,
0.47347,	0.47447,	0.47548,	0.47648,	0.47748,	0.47848,
0.47948,					
	0.48048,	0.48148,	0.48248,	0.48348,	0.48448,
0.48549,	0.48649,	0.48749,	0.48849,	0.48949,	0.49049,
0.49149,	0.49249,	0.49349,	0.49449,	0.4955,	0.4965,
0.4975,	0.4985,	0.4995,	0.5005,	0.5015,	0.5025,
0.5035,					
	0.5045,	0.50551,	0.50651,	0.50751,	0.50851,
0.50951,	0.51051,	0.51151,	0.51251,	0.51351,	0.51451,
0.51552,	0.51652,	0.51752,	0.51852,	0.51952,	0.52052,
0.52152,	0.52252,	0.52352,	0.52452,	0.52553,	0.52653,
0.52753,					
	0.52853,	0.52953,	0.53053,	0.53153,	0.53253,
0.53353,	0.53453,	0.53554,	0.53654,	0.53754,	0.53854,
0.53954,	0.54054,	0.54154,	0.54254,	0.54354,	0.54454,
0.54555,	0.54655,	0.54755,	0.54855,	0.54955,	0.55055,
0.55155,					
	0.55255,	0.55355,	0.55455,	0.55556,	0.55656,
0.55756,	0.55856,	0.55956,	0.56056,	0.56156,	0.56256,
0.56356,	0.56456,	0.56557,	0.56657,	0.56757,	0.56857,
0.56957,	0.57057,	0.57157,	0.57257,	0.57357,	0.57457,
0.57558,					
	0.57658,	0.57758,	0.57858,	0.57958,	0.58058,
0.58158,	0.58258,	0.58358,	0.58458,	0.58559,	0.58659,
0.58759,	0.58859,	0.58959,	0.59059,	0.59159,	0.59259,
0.59359,	0.59459,	0.5956,	0.5966,	0.5976,	0.5986,
0.5996,					
	0.6006,	0.6016,	0.6026,	0.6036,	0.6046,
0.60561,	0.60661,	0.60761,	0.60861,	0.60961,	0.61061,
0.61161,	0.61261,	0.61361,	0.61461,	0.61562,	0.61662,
0.61762,	0.61862,	0.61962,	0.62062,	0.62162,	0.62262,
0.62362,					

	0.62462,	0.62563,	0.62663,	0.62763,	0.62863,
0.62963,	0.63063,	0.63163,	0.63263,	0.63363,	0.63463,
0.63564,	0.63664,	0.63764,	0.63864,	0.63964,	0.64064,
0.64164,	0.64264,	0.64364,	0.64464,	0.64565,	0.64665,
0.64765,					
	0.64865,	0.64965,	0.65065,	0.65165,	0.65265,
0.65365,	0.65465,	0.65566,	0.65666,	0.65766,	0.65866,
0.65966,	0.66066,	0.66166,	0.66266,	0.66366,	0.66466,
0.66567,	0.66667,	0.66767,	0.66867,	0.66967,	0.67067,
0.67167,					
	0.67267,	0.67367,	0.67467,	0.67568,	0.67668,
0.67768,	0.67868,	0.67968,	0.68068,	0.68168,	0.68268,
0.68368,	0.68468,	0.68569,	0.68669,	0.68769,	0.68869,
0.68969,	0.69069,	0.69169,	0.69269,	0.69369,	0.69469,
0.6957,					
	0.6967,	0.6977,	0.6987,	0.6997,	0.7007,
0.7017,	0.7027,	0.7037,	0.7047,	0.70571,	0.70671,
0.70771,	0.70871,	0.70971,	0.71071,	0.71171,	0.71271,
0.71371,	0.71471,	0.71572,	0.71672,	0.71772,	0.71872,
0.71972,					
	0.72072,	0.72172,	0.72272,	0.72372,	0.72472,
0.72573,	0.72673,	0.72773,	0.72873,	0.72973,	0.73073,
0.73173,	0.73273,	0.73373,	0.73473,	0.73574,	0.73674,
0.73774,	0.73874,	0.73974,	0.74074,	0.74174,	0.74274,
0.74374,					
	0.74474,	0.74575,	0.74675,	0.74775,	0.74875,
0.74975,	0.75075,	0.75175,	0.75275,	0.75375,	0.75475,
0.75576,	0.75676,	0.75776,	0.75876,	0.75976,	0.76076,
0.76176,	0.76276,	0.76376,	0.76476,	0.76577,	0.76677,
0.76777,					
	0.76877,	0.76977,	0.77077,	0.77177,	0.77277,
0.77377,	0.77477,	0.77578,	0.77678,	0.77778,	0.77878,
0.77978,	0.78078,	0.78178,	0.78278,	0.78378,	0.78478,
0.78579,	0.78679,	0.78779,	0.78879,	0.78979,	0.79079,
0.79179,					
	0.79279,	0.79379,	0.79479,	0.7958,	0.7968,
0.7978,	0.7988,	0.7998,	0.8008,	0.8018,	0.8028,
0.8038,	0.8048,	0.80581,	0.80681,	0.80781,	0.80881,
0.80981,	0.81081,	0.81181,	0.81281,	0.81381,	0.81481,
0.81582,					
	0.81682,	0.81782,	0.81882,	0.81982,	0.82082,
0.82182,	0.82282,	0.82382,	0.82482,	0.82583,	0.82683,
0.82783,	0.82883,	0.82983,	0.83083,	0.83183,	0.83283,
0.83383,	0.83483,	0.83584,	0.83684,	0.83784,	0.83884,
0.83984,					
	0.84084,	0.84184,	0.84284,	0.84384,	0.84484,
0.84585,	0.84685,	0.84785,	0.84885,	0.84985,	0.85085,
0.85185,	0.85285,	0.85385,	0.85485,	0.85586,	0.85686,

0.85786,	0.85886,	0.85986,	0.86086,	0.86186,	0.86286,
0.86386,	0.86486,	0.86587,	0.86687,	0.86787,	0.86887,
0.86987,	0.87087,	0.87187,	0.87287,	0.87387,	0.87487,
0.87588,	0.87688,	0.87788,	0.87888,	0.87988,	0.88088,
0.88188,	0.88288,	0.88388,	0.88488,	0.88589,	0.88689,
0.88789,	0.88889,	0.88989,	0.89089,	0.89189,	0.89289,
0.89389,	0.89489,	0.8959,	0.8969,	0.8979,	0.8989,
0.8999,	0.9009,	0.9019,	0.9029,	0.9039,	0.9049,
0.90591,	0.90691,	0.90791,	0.90891,	0.90991,	0.91091,
0.91191,	0.91291,	0.91391,	0.91491,	0.91592,	0.91692,
0.91792,	0.91892,	0.91992,	0.92092,	0.92192,	0.92292,
0.92392,	0.92492,	0.92593,	0.92693,	0.92793,	0.92893,
0.92993,	0.93093,	0.93193,	0.93293,	0.93393,	0.93493,
0.93594,	0.93694,	0.93794,	0.93894,	0.93994,	0.94094,
0.94194,	0.94294,	0.94394,	0.94494,	0.94595,	0.94695,
0.94795,	0.94895,	0.94995,	0.95095,	0.95195,	0.95295,
0.95395,	0.95495,	0.95596,	0.95696,	0.95796,	0.95896,
0.95996,	0.96096,	0.96196,	0.96296,	0.96396,	0.96496,
0.96597,	0.96697,	0.96797,	0.96897,	0.96997,	0.97097,
0.97197,	0.97297,	0.97397,	0.97497,	0.97598,	0.97698,
0.97798,	0.97898,	0.97998,	0.98098,	0.98198,	0.98298,
0.98398,	0.98498,	0.98599,	0.98699,	0.98799,	0.98899,
0.98999,	0.99099,	0.99199,	0.99299,	0.99399,	0.99499,
0.996,	0.997,	0.998,	0.999,	1]), array([[
0.02484,	0.024857,	0.039516,	...,	0,	0,
[0.027209,	0.027236,	0.051099,	...,	0,
0],					0,
	[0.024977,	0.024996,	0.042863,	...,	0,
0]],	'Confidence',	'F1'],	array([0,	0.001001,
0.003003,	0.004004,	0.005005,	0.006006,	0.007007,	0.008008,
0.009009,	0.01001,	0.011011,	0.012012,	0.013013,	0.014014,
0.015015,	0.016016,	0.017017,	0.018018,	0.019019,	0.02002,
0.021021,	0.022022,	0.023023,			
	0.024024,	0.025025,	0.026026,	0.027027,	0.028028,
0.029029,	0.03003,	0.031031,	0.032032,	0.033033,	0.034034,
0.035035,	0.036036,	0.037037,	0.038038,	0.039039,	0.04004,
0.041041,	0.042042,	0.043043,	0.044044,	0.045045,	0.046046,
0.047047,	0.048048,	0.049049,	0.05005,	0.051051,	0.052052,
0.053053,	0.054054,	0.055055,	0.056056,	0.057057,	0.058058,
0.059059,	0.06006,	0.061061,	0.062062,	0.063063,	0.064064,
0.065065,	0.066066,	0.067067,	0.068068,	0.069069,	0.07007,

0.071071,					
	0.072072,	0.073073,	0.074074,	0.075075,	0.076076,
0.077077,	0.078078,	0.079079,	0.08008,	0.081081,	0.082082,
0.083083,	0.084084,	0.085085,	0.086086,	0.087087,	0.088088,
0.089089,	0.09009,	0.091091,	0.092092,	0.093093,	0.094094,
0.095095,					
	0.096096,	0.097097,	0.098098,	0.099099,	0.1001,
0.1011,	0.1021,	0.1031,	0.1041,	0.10511,	0.10611,
0.10711,	0.10811,	0.10911,	0.11011,	0.11111,	0.11211,
0.11311,	0.11411,	0.11512,	0.11612,	0.11712,	0.11812,
0.11912,					
	0.12012,	0.12112,	0.12212,	0.12312,	0.12412,
0.12513,	0.12613,	0.12713,	0.12813,	0.12913,	0.13013,
0.13113,	0.13213,	0.13313,	0.13413,	0.13514,	0.13614,
0.13714,	0.13814,	0.13914,	0.14014,	0.14114,	0.14214,
0.14314,					
	0.14414,	0.14515,	0.14615,	0.14715,	0.14815,
0.14915,	0.15015,	0.15115,	0.15215,	0.15315,	0.15415,
0.15516,	0.15616,	0.15716,	0.15816,	0.15916,	0.16016,
0.16116,	0.16216,	0.16316,	0.16416,	0.16517,	0.16617,
0.16717,					
	0.16817,	0.16917,	0.17017,	0.17117,	0.17217,
0.17317,	0.17417,	0.17518,	0.17618,	0.17718,	0.17818,
0.17918,	0.18018,	0.18118,	0.18218,	0.18318,	0.18418,
0.18519,	0.18619,	0.18719,	0.18819,	0.18919,	0.19019,
0.19119,					
	0.19219,	0.19319,	0.19419,	0.1952,	0.1962,
0.1972,	0.1982,	0.1992,	0.2002,	0.2012,	0.2022,
0.2032,	0.2042,	0.20521,	0.20621,	0.20721,	0.20821,
0.20921,	0.21021,	0.21121,	0.21221,	0.21321,	0.21421,
0.21522,					
	0.21622,	0.21722,	0.21822,	0.21922,	0.22022,
0.22122,	0.22222,	0.22322,	0.22422,	0.22523,	0.22623,
0.22723,	0.22823,	0.22923,	0.23023,	0.23123,	0.23223,
0.23323,	0.23423,	0.23524,	0.23624,	0.23724,	0.23824,
0.23924,					
	0.24024,	0.24124,	0.24224,	0.24324,	0.24424,
0.24525,	0.24625,	0.24725,	0.24825,	0.24925,	0.25025,
0.25125,	0.25225,	0.25325,	0.25425,	0.25526,	0.25626,
0.25726,	0.25826,	0.25926,	0.26026,	0.26126,	0.26226,
0.26326,					
	0.26426,	0.26527,	0.26627,	0.26727,	0.26827,
0.26927,	0.27027,	0.27127,	0.27227,	0.27327,	0.27427,
0.27528,	0.27628,	0.27728,	0.27828,	0.27928,	0.28028,
0.28128,	0.28228,	0.28328,	0.28428,	0.28529,	0.28629,
0.28729,					
	0.28829,	0.28929,	0.29029,	0.29129,	0.29229,
0.29329,	0.29429,	0.2953,	0.2963,	0.2973,	0.2983,

0.2993,	0.3003,	0.3013,	0.3023,	0.3033,	0.3043,
0.30531,	0.30631,	0.30731,	0.30831,	0.30931,	0.31031,
0.31131,	0.31231,	0.31331,	0.31431,	0.31532,	0.31632,
0.31732,	0.31832,	0.31932,	0.32032,	0.32132,	0.32232,
0.32332,	0.32432,	0.32533,	0.32633,	0.32733,	0.32833,
0.32933,	0.33033,	0.33133,	0.33233,	0.33333,	0.33433,
0.33534,	0.33634,	0.33734,	0.33834,	0.33934,	0.34034,
0.34134,	0.34234,	0.34334,	0.34434,	0.34535,	0.34635,
0.34735,	0.34835,	0.34935,	0.35035,	0.35135,	0.35235,
0.35335,	0.35435,	0.35536,	0.35636,	0.35736,	0.35836,
0.35936,	0.36036,	0.36136,	0.36236,	0.36336,	0.36436,
0.36537,	0.36637,	0.36737,	0.36837,	0.36937,	0.37037,
0.37137,	0.37237,	0.37337,	0.37437,	0.37538,	0.37638,
0.37738,	0.37838,	0.37938,	0.38038,	0.38138,	0.38238,
0.38338,	0.38438,	0.38539,	0.38639,	0.38739,	0.38839,
0.38939,	0.39039,	0.39139,	0.39239,	0.39339,	0.39439,
0.3954,	0.3964,	0.3974,	0.3984,	0.3994,	0.4004,
0.4014,	0.4024,	0.4034,	0.4044,	0.40541,	0.40641,
0.40741,	0.40841,	0.40941,	0.41041,	0.41141,	0.41241,
0.41341,	0.41441,	0.41542,	0.41642,	0.41742,	0.41842,
0.41942,	0.42042,	0.42142,	0.42242,	0.42342,	0.42442,
0.42543,	0.42643,	0.42743,	0.42843,	0.42943,	0.43043,
0.43143,	0.43243,	0.43343,	0.43443,	0.43544,	0.43644,
0.43744,	0.43844,	0.43944,	0.44044,	0.44144,	0.44244,
0.44344,	0.44444,	0.44545,	0.44645,	0.44745,	0.44845,
0.44945,	0.45045,	0.45145,	0.45245,	0.45345,	0.45445,
0.45546,	0.45646,	0.45746,	0.45846,	0.45946,	0.46046,
0.46146,	0.46246,	0.46346,	0.46446,	0.46547,	0.46647,
0.46747,	0.46847,	0.46947,	0.47047,	0.47147,	0.47247,
0.47347,	0.47447,	0.47548,	0.47648,	0.47748,	0.47848,
0.47948,	0.48048,	0.48148,	0.48248,	0.48348,	0.48448,
0.48549,	0.48649,	0.48749,	0.48849,	0.48949,	0.49049,
0.49149,	0.49249,	0.49349,	0.49449,	0.4955,	0.4965,
0.4975,	0.4985,	0.4995,	0.5005,	0.5015,	0.5025,
0.5035,	0.5045,	0.50551,	0.50651,	0.50751,	0.50851,
0.50951,	0.51051,	0.51151,	0.51251,	0.51351,	0.51451,
0.51552,	0.51652,	0.51752,	0.51852,	0.51952,	0.52052,
0.52152,	0.52252,	0.52352,	0.52452,	0.52553,	0.52653,
0.52753,					

	0.52853,	0.52953,	0.53053,	0.53153,	0.53253,
0.53353,	0.53453,	0.53554,	0.53654,	0.53754,	0.53854,
0.53954,	0.54054,	0.54154,	0.54254,	0.54354,	0.54454,
0.54555,	0.54655,	0.54755,	0.54855,	0.54955,	0.55055,
0.55155,					
	0.55255,	0.55355,	0.55455,	0.55556,	0.55656,
0.55756,	0.55856,	0.55956,	0.56056,	0.56156,	0.56256,
0.56356,	0.56456,	0.56557,	0.56657,	0.56757,	0.56857,
0.56957,	0.57057,	0.57157,	0.57257,	0.57357,	0.57457,
0.57558,					
	0.57658,	0.57758,	0.57858,	0.57958,	0.58058,
0.58158,	0.58258,	0.58358,	0.58458,	0.58559,	0.58659,
0.58759,	0.58859,	0.58959,	0.59059,	0.59159,	0.59259,
0.59359,	0.59459,	0.5956,	0.5966,	0.5976,	0.5986,
0.5996,					
	0.6006,	0.6016,	0.6026,	0.6036,	0.6046,
0.60561,	0.60661,	0.60761,	0.60861,	0.60961,	0.61061,
0.61161,	0.61261,	0.61361,	0.61461,	0.61562,	0.61662,
0.61762,	0.61862,	0.61962,	0.62062,	0.62162,	0.62262,
0.62362,					
	0.62462,	0.62563,	0.62663,	0.62763,	0.62863,
0.62963,	0.63063,	0.63163,	0.63263,	0.63363,	0.63463,
0.63564,	0.63664,	0.63764,	0.63864,	0.63964,	0.64064,
0.64164,	0.64264,	0.64364,	0.64464,	0.64565,	0.64665,
0.64765,					
	0.64865,	0.64965,	0.65065,	0.65165,	0.65265,
0.65365,	0.65465,	0.65566,	0.65666,	0.65766,	0.65866,
0.65966,	0.66066,	0.66166,	0.66266,	0.66366,	0.66466,
0.66567,	0.66667,	0.66767,	0.66867,	0.66967,	0.67067,
0.67167,					
	0.67267,	0.67367,	0.67467,	0.67568,	0.67668,
0.67768,	0.67868,	0.67968,	0.68068,	0.68168,	0.68268,
0.68368,	0.68468,	0.68569,	0.68669,	0.68769,	0.68869,
0.68969,	0.69069,	0.69169,	0.69269,	0.69369,	0.69469,
0.6957,					
	0.6967,	0.6977,	0.6987,	0.6997,	0.7007,
0.7017,	0.7027,	0.7037,	0.7047,	0.70571,	0.70671,
0.70771,	0.70871,	0.70971,	0.71071,	0.71171,	0.71271,
0.71371,	0.71471,	0.71572,	0.71672,	0.71772,	0.71872,
0.71972,					
	0.72072,	0.72172,	0.72272,	0.72372,	0.72472,
0.72573,	0.72673,	0.72773,	0.72873,	0.72973,	0.73073,
0.73173,	0.73273,	0.73373,	0.73473,	0.73574,	0.73674,
0.73774,	0.73874,	0.73974,	0.74074,	0.74174,	0.74274,
0.74374,					
	0.74474,	0.74575,	0.74675,	0.74775,	0.74875,
0.74975,	0.75075,	0.75175,	0.75275,	0.75375,	0.75475,
0.75576,	0.75676,	0.75776,	0.75876,	0.75976,	0.76076,

0.76176,	0.76276,	0.76376,	0.76476,	0.76577,	0.76677,
0.76777,	0.76877,	0.76977,	0.77077,	0.77177,	0.77277,
0.77377,	0.77477,	0.77578,	0.77678,	0.77778,	0.77878,
0.77978,	0.78078,	0.78178,	0.78278,	0.78378,	0.78478,
0.78579,	0.78679,	0.78779,	0.78879,	0.78979,	0.79079,
0.79179,	0.79279,	0.79379,	0.79479,	0.7958,	0.7968,
0.7978,	0.7988,	0.7998,	0.8008,	0.8018,	0.8028,
0.8038,	0.8048,	0.80581,	0.80681,	0.80781,	0.80881,
0.80981,	0.81081,	0.81181,	0.81281,	0.81381,	0.81481,
0.81582,	0.81682,	0.81782,	0.81882,	0.81982,	0.82082,
0.82182,	0.82282,	0.82382,	0.82482,	0.82583,	0.82683,
0.82783,	0.82883,	0.82983,	0.83083,	0.83183,	0.83283,
0.83383,	0.83483,	0.83584,	0.83684,	0.83784,	0.83884,
0.83984,	0.84084,	0.84184,	0.84284,	0.84384,	0.84484,
0.84585,	0.84685,	0.84785,	0.84885,	0.84985,	0.85085,
0.85185,	0.85285,	0.85385,	0.85485,	0.85586,	0.85686,
0.85786,	0.85886,	0.85986,	0.86086,	0.86186,	0.86286,
0.86386,	0.86486,	0.86587,	0.86687,	0.86787,	0.86887,
0.86987,	0.87087,	0.87187,	0.87287,	0.87387,	0.87487,
0.87588,	0.87688,	0.87788,	0.87888,	0.87988,	0.88088,
0.88188,	0.88288,	0.88388,	0.88488,	0.88589,	0.88689,
0.88789,	0.88889,	0.88989,	0.89089,	0.89189,	0.89289,
0.89389,	0.89489,	0.8959,	0.8969,	0.8979,	0.8989,
0.8999,	0.9009,	0.9019,	0.9029,	0.9039,	0.9049,
0.90591,	0.90691,	0.90791,	0.90891,	0.90991,	0.91091,
0.91191,	0.91291,	0.91391,	0.91491,	0.91592,	0.91692,
0.91792,	0.91892,	0.91992,	0.92092,	0.92192,	0.92292,
0.92392,	0.92492,	0.92593,	0.92693,	0.92793,	0.92893,
0.92993,	0.93093,	0.93193,	0.93293,	0.93393,	0.93493,
0.93594,	0.93694,	0.93794,	0.93894,	0.93994,	0.94094,
0.94194,	0.94294,	0.94394,	0.94494,	0.94595,	0.94695,
0.94795,	0.94895,	0.94995,	0.95095,	0.95195,	0.95295,
0.95395,	0.95495,	0.95596,	0.95696,	0.95796,	0.95896,
0.95996,	0.96096,	0.96196,	0.96296,	0.96396,	0.96496,
0.96597,	0.96697,	0.96797,	0.96897,	0.96997,	0.97097,
0.97197,	0.97297,	0.97397,	0.97497,	0.97598,	0.97698,
0.97798,	0.97898,	0.97998,	0.98098,	0.98198,	0.98298,
0.98398,	0.98498,	0.98599,	0.98699,	0.98799,	0.98899,

```

0.98999,      0.99099,      0.99199,      0.99299,      0.99399,      0.99499,
0.996,        0.997,        0.998,        0.999,        1],       array([[ 
0.01261,      0.012619,     0.020252, ...,      1,          1,          1],
   [ 0.013812,     0.013826,     0.026305, ...,      1,          1,          1],
1],
   [ 0.012671,     0.012681,     0.021993, ...,      1,          1,          1,
1]], 'Confidence', 'Precision'], [array([
0,          0.001001,     0.002002,
0.003003,     0.004004,     0.005005,     0.006006,     0.007007,     0.008008,
0.009009,     0.01001,      0.011011,     0.012012,     0.013013,     0.014014,
0.015015,     0.016016,     0.017017,     0.018018,     0.019019,     0.02002,
0.021021,     0.022022,     0.023023,
   0.024024,     0.025025,     0.026026,     0.027027,     0.028028,
0.029029,     0.03003,      0.031031,     0.032032,     0.033033,     0.034034,
0.035035,     0.036036,     0.037037,     0.038038,     0.039039,     0.04004,
0.041041,     0.042042,     0.043043,     0.044044,     0.045045,     0.046046,
0.047047,
   0.048048,     0.049049,     0.05005,      0.051051,     0.052052,
0.053053,     0.054054,     0.055055,     0.056056,     0.057057,     0.058058,
0.059059,     0.06006,      0.061061,     0.062062,     0.063063,     0.064064,
0.065065,     0.066066,     0.067067,     0.068068,     0.069069,     0.07007,
0.071071,
   0.072072,     0.073073,     0.074074,     0.075075,     0.076076,
0.077077,     0.078078,     0.079079,     0.08008,      0.081081,     0.082082,
0.083083,     0.084084,     0.085085,     0.086086,     0.087087,     0.088088,
0.089089,     0.09009,      0.091091,     0.092092,     0.093093,     0.094094,
0.095095,
   0.096096,     0.097097,     0.098098,     0.099099,     0.1001,
0.1011,        0.1021,      0.1031,      0.1041,      0.10511,     0.10611,
0.10711,       0.10811,     0.10911,     0.11011,     0.11111,     0.11211,
0.11311,       0.11411,     0.11512,     0.11612,     0.11712,     0.11812,
0.11912,
   0.12012,       0.12112,     0.12212,     0.12312,     0.12412,
0.12513,       0.12613,     0.12713,     0.12813,     0.12913,     0.13013,
0.13113,       0.13213,     0.13313,     0.13413,     0.13514,     0.13614,
0.13714,       0.13814,     0.13914,     0.14014,     0.14114,     0.14214,
0.14314,
   0.14414,       0.14515,     0.14615,     0.14715,     0.14815,
0.14915,       0.15015,     0.15115,     0.15215,     0.15315,     0.15415,
0.15516,       0.15616,     0.15716,     0.15816,     0.15916,     0.16016,
0.16116,       0.16216,     0.16316,     0.16416,     0.16517,     0.16617,
0.16717,
   0.16817,       0.16917,     0.17017,     0.17117,     0.17217,
0.17317,       0.17417,     0.17518,     0.17618,     0.17718,     0.17818,
0.17918,       0.18018,     0.18118,     0.18218,     0.18318,     0.18418,
0.18519,       0.18619,     0.18719,     0.18819,     0.18919,     0.19019,
0.19119,
   0.19219,       0.19319,     0.19419,     0.1952,      0.1962,
0.1972,        0.1982,      0.1992,      0.2002,      0.2012,      0.2022,

```

0.2032,	0.2042,	0.20521,	0.20621,	0.20721,	0.20821,
0.20921,	0.21021,	0.21121,	0.21221,	0.21321,	0.21421,
0.21522,					
	0.21622,	0.21722,	0.21822,	0.21922,	0.22022,
0.22122,	0.22222,	0.22322,	0.22422,	0.22523,	0.22623,
0.22723,	0.22823,	0.22923,	0.23023,	0.23123,	0.23223,
0.23323,	0.23423,	0.23524,	0.23624,	0.23724,	0.23824,
0.23924,					
	0.24024,	0.24124,	0.24224,	0.24324,	0.24424,
0.24525,	0.24625,	0.24725,	0.24825,	0.24925,	0.25025,
0.25125,	0.25225,	0.25325,	0.25425,	0.25526,	0.25626,
0.25726,	0.25826,	0.25926,	0.26026,	0.26126,	0.26226,
0.26326,					
	0.26426,	0.26527,	0.26627,	0.26727,	0.26827,
0.26927,	0.27027,	0.27127,	0.27227,	0.27327,	0.27427,
0.27528,	0.27628,	0.27728,	0.27828,	0.27928,	0.28028,
0.28128,	0.28228,	0.28328,	0.28428,	0.28529,	0.28629,
0.28729,					
	0.28829,	0.28929,	0.29029,	0.29129,	0.29229,
0.29329,	0.29429,	0.2953,	0.2963,	0.2973,	0.2983,
0.2993,	0.3003,	0.3013,	0.3023,	0.3033,	0.3043,
0.30531,	0.30631,	0.30731,	0.30831,	0.30931,	0.31031,
0.31131,					
	0.31231,	0.31331,	0.31431,	0.31532,	0.31632,
0.31732,	0.31832,	0.31932,	0.32032,	0.32132,	0.32232,
0.32332,	0.32432,	0.32533,	0.32633,	0.32733,	0.32833,
0.32933,	0.33033,	0.33133,	0.33233,	0.33333,	0.33433,
0.33534,					
	0.33634,	0.33734,	0.33834,	0.33934,	0.34034,
0.34134,	0.34234,	0.34334,	0.34434,	0.34535,	0.34635,
0.34735,	0.34835,	0.34935,	0.35035,	0.35135,	0.35235,
0.35335,	0.35435,	0.35536,	0.35636,	0.35736,	0.35836,
0.35936,					
	0.36036,	0.36136,	0.36236,	0.36336,	0.36436,
0.36537,	0.36637,	0.36737,	0.36837,	0.36937,	0.37037,
0.37137,	0.37237,	0.37337,	0.37437,	0.37538,	0.37638,
0.37738,	0.37838,	0.37938,	0.38038,	0.38138,	0.38238,
0.38338,					
	0.38438,	0.38539,	0.38639,	0.38739,	0.38839,
0.38939,	0.39039,	0.39139,	0.39239,	0.39339,	0.39439,
0.3954,	0.3964,	0.3974,	0.3984,	0.3994,	0.4004,
0.4014,	0.4024,	0.4034,	0.4044,	0.40541,	0.40641,
0.40741,					
	0.40841,	0.40941,	0.41041,	0.41141,	0.41241,
0.41341,	0.41441,	0.41542,	0.41642,	0.41742,	0.41842,
0.41942,	0.42042,	0.42142,	0.42242,	0.42342,	0.42442,
0.42543,	0.42643,	0.42743,	0.42843,	0.42943,	0.43043,
0.43143,					

	0.43243,	0.43343,	0.43443,	0.43544,	0.43644,
0.43744,	0.43844,	0.43944,	0.44044,	0.44144,	0.44244,
0.44344,	0.44444,	0.44545,	0.44645,	0.44745,	0.44845,
0.44945,	0.45045,	0.45145,	0.45245,	0.45345,	0.45445,
0.45546,					
	0.45646,	0.45746,	0.45846,	0.45946,	0.46046,
0.46146,	0.46246,	0.46346,	0.46446,	0.46547,	0.46647,
0.46747,	0.46847,	0.46947,	0.47047,	0.47147,	0.47247,
0.47347,	0.47447,	0.47548,	0.47648,	0.47748,	0.47848,
0.47948,					
	0.48048,	0.48148,	0.48248,	0.48348,	0.48448,
0.48549,	0.48649,	0.48749,	0.48849,	0.48949,	0.49049,
0.49149,	0.49249,	0.49349,	0.49449,	0.4955,	0.4965,
0.4975,	0.4985,	0.4995,	0.5005,	0.5015,	0.5025,
0.5035,					
	0.5045,	0.50551,	0.50651,	0.50751,	0.50851,
0.50951,	0.51051,	0.51151,	0.51251,	0.51351,	0.51451,
0.51552,	0.51652,	0.51752,	0.51852,	0.51952,	0.52052,
0.52152,	0.52252,	0.52352,	0.52452,	0.52553,	0.52653,
0.52753,					
	0.52853,	0.52953,	0.53053,	0.53153,	0.53253,
0.53353,	0.53453,	0.53554,	0.53654,	0.53754,	0.53854,
0.53954,	0.54054,	0.54154,	0.54254,	0.54354,	0.54454,
0.54555,	0.54655,	0.54755,	0.54855,	0.54955,	0.55055,
0.55155,					
	0.55255,	0.55355,	0.55455,	0.55556,	0.55656,
0.55756,	0.55856,	0.55956,	0.56056,	0.56156,	0.56256,
0.56356,	0.56456,	0.56557,	0.56657,	0.56757,	0.56857,
0.56957,	0.57057,	0.57157,	0.57257,	0.57357,	0.57457,
0.57558,					
	0.57658,	0.57758,	0.57858,	0.57958,	0.58058,
0.58158,	0.58258,	0.58358,	0.58458,	0.58559,	0.58659,
0.58759,	0.58859,	0.58959,	0.59059,	0.59159,	0.59259,
0.59359,	0.59459,	0.5956,	0.5966,	0.5976,	0.5986,
0.5996,					
	0.6006,	0.6016,	0.6026,	0.6036,	0.6046,
0.60561,	0.60661,	0.60761,	0.60861,	0.60961,	0.61061,
0.61161,	0.61261,	0.61361,	0.61461,	0.61562,	0.61662,
0.61762,	0.61862,	0.61962,	0.62062,	0.62162,	0.62262,
0.62362,					
	0.62462,	0.62563,	0.62663,	0.62763,	0.62863,
0.62963,	0.63063,	0.63163,	0.63263,	0.63363,	0.63463,
0.63564,	0.63664,	0.63764,	0.63864,	0.63964,	0.64064,
0.64164,	0.64264,	0.64364,	0.64464,	0.64565,	0.64665,
0.64765,					
	0.64865,	0.64965,	0.65065,	0.65165,	0.65265,
0.65365,	0.65465,	0.65566,	0.65666,	0.65766,	0.65866,
0.65966,	0.66066,	0.66166,	0.66266,	0.66366,	0.66466,

0.66567,	0.66667,	0.66767,	0.66867,	0.66967,	0.67067,
0.67167,	0.67267,	0.67367,	0.67467,	0.67568,	0.67668,
0.67768,	0.67868,	0.67968,	0.68068,	0.68168,	0.68268,
0.68368,	0.68468,	0.68569,	0.68669,	0.68769,	0.68869,
0.68969,	0.69069,	0.69169,	0.69269,	0.69369,	0.69469,
0.6957,	0.6967,	0.6977,	0.6987,	0.6997,	0.7007,
0.7017,	0.7027,	0.7037,	0.7047,	0.70571,	0.70671,
0.70771,	0.70871,	0.70971,	0.71071,	0.71171,	0.71271,
0.71371,	0.71471,	0.71572,	0.71672,	0.71772,	0.71872,
0.71972,	0.72072,	0.72172,	0.72272,	0.72372,	0.72472,
0.72573,	0.72673,	0.72773,	0.72873,	0.72973,	0.73073,
0.73173,	0.73273,	0.73373,	0.73473,	0.73574,	0.73674,
0.73774,	0.73874,	0.73974,	0.74074,	0.74174,	0.74274,
0.74374,	0.74474,	0.74575,	0.74675,	0.74775,	0.74875,
0.74975,	0.75075,	0.75175,	0.75275,	0.75375,	0.75475,
0.75576,	0.75676,	0.75776,	0.75876,	0.75976,	0.76076,
0.76176,	0.76276,	0.76376,	0.76476,	0.76577,	0.76677,
0.76777,	0.76877,	0.76977,	0.77077,	0.77177,	0.77277,
0.77377,	0.77477,	0.77578,	0.77678,	0.77778,	0.77878,
0.77978,	0.78078,	0.78178,	0.78278,	0.78378,	0.78478,
0.78579,	0.78679,	0.78779,	0.78879,	0.78979,	0.79079,
0.79179,	0.79279,	0.79379,	0.79479,	0.7958,	0.7968,
0.7978,	0.7988,	0.7998,	0.8008,	0.8018,	0.8028,
0.8038,	0.8048,	0.80581,	0.80681,	0.80781,	0.80881,
0.80981,	0.81081,	0.81181,	0.81281,	0.81381,	0.81481,
0.81582,	0.81682,	0.81782,	0.81882,	0.81982,	0.82082,
0.82182,	0.82282,	0.82382,	0.82482,	0.82583,	0.82683,
0.82783,	0.82883,	0.82983,	0.83083,	0.83183,	0.83283,
0.83383,	0.83483,	0.83584,	0.83684,	0.83784,	0.83884,
0.83984,	0.84084,	0.84184,	0.84284,	0.84384,	0.84484,
0.84585,	0.84685,	0.84785,	0.84885,	0.84985,	0.85085,
0.85185,	0.85285,	0.85385,	0.85485,	0.85586,	0.85686,
0.85786,	0.85886,	0.85986,	0.86086,	0.86186,	0.86286,
0.86386,	0.86486,	0.86587,	0.86687,	0.86787,	0.86887,
0.86987,	0.87087,	0.87187,	0.87287,	0.87387,	0.87487,
0.87588,	0.87688,	0.87788,	0.87888,	0.87988,	0.88088,
0.88188,	0.88288,	0.88388,	0.88488,	0.88589,	0.88689,
0.88789,	0.88889,	0.88989,	0.89089,	0.89189,	0.89289,

```

0.89389,      0.89489,      0.8959,       0.8969,       0.8979,       0.8989,
0.8999,       0.9009,       0.9019,       0.9029,       0.9039,       0.9049,
0.90591,      0.90691,      0.90791,      0.90891,      0.90991,      0.91091,
0.91191,      0.91291,      0.91391,      0.91491,      0.91592,      0.91692,
0.91792,      0.91892,      0.91992,      0.92092,      0.92192,      0.92292,
0.92392,      0.92492,      0.92593,      0.92693,      0.92793,      0.92893,
0.92993,      0.93093,      0.93193,      0.93293,      0.93393,      0.93493,
0.93594,      0.93694,      0.93794,      0.93894,      0.93994,      0.94094,
0.94194,      0.94294,      0.94394,      0.94494,      0.94595,      0.94695,
0.94795,      0.94895,      0.94995,      0.95095,      0.95195,      0.95295,
0.95395,      0.95495,      0.95596,      0.95696,      0.95796,      0.95896,
0.95996,      0.96096,      0.96196,      0.96296,      0.96396,      0.96496,
0.96597,      0.96697,      0.96797,      0.96897,      0.96997,      0.97097,
0.97197,      0.97297,      0.97397,      0.97497,      0.97598,      0.97698,
0.97798,      0.97898,      0.97998,      0.98098,      0.98198,      0.98298,
0.98398,      0.98498,      0.98599,      0.98699,      0.98799,      0.98899,
0.98999,      0.99099,      0.99199,      0.99299,      0.99399,      0.99499,
0.996,        0.997,        0.998,        0.999,        1], array([
0.82313,      0.82313,      0.80952, ..., 0,          0,          0],
[  0.90644,    0.90644,    0.88981, ..., 0,          0,          0],
[],           [  0.86823,    0.86823,    0.83935, ..., 0,          0,          0,
0]), 'Confidence', 'Recall']]]

fitness: 0.3033979353497532
keys: ['metrics/precision(B)', 'metrics/recall(B)', 'metrics/mAP50(B)',
'metrics/mAP50-95(B)']
maps: array([  0.18793,    0.40668,    0.23222])
names: {0: 'label0', 1: 'label1', 2: 'label2'}
plot: True
results_dict: {'metrics/precision(B)': 0.6460222341674677, 'metrics/recall(B)': 0.5159531334880804, 'metrics/mAP50(B)': 0.5534691968580986, 'metrics/mAP50-95(B)': 0.2756122396266037, 'fitness': 0.3033979353497532}
save_dir: PosixPath('runs/detect/train123')
speed: {'preprocess': 0.18286295611448963, 'inference': 1.8113593862514303, 'loss': 0.0030645216354215988, 'postprocess': 0.7888801170117927}
task: 'detect'

```

1.4.3 YOLOv11 - Main Run

Our initial goal was to test the highest model available within the YOLO framework, as it would leverage a larger amount of pre-trained data. While we initially chose YOLOv11 for this reason, further research revealed that it also outperformed YOLOv8 for several reasons, which are highlighted later in this notebook.

```
[22]: !wget https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo1n.pt

--2024-11-28 10:05:24--
https://github.com/ultralytics/assets/releases/download/v8.3.0/yolo1n.pt
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-
asset-2e65be/521807533/34b70ade-b6eb-4179-a60f-d6494307226b?X-Amz-
Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=releaseassetproduction%2F20241128%2Fus-
east-1%2Fs3%2Faws4_request&X-Amz-Date=20241128T100524Z&X-Amz-Expires=300&X-Amz-S
ignature=ac3497975c7adcb044a993d3ee0c9f80b6c83cba6870c223209bb0bfe5875373&X-Amz-
SignedHeaders=host&response-content-
disposition=attachment%3B%20filename%3Dyolo1n.pt&response-content-
type=application%2Foctet-stream [following]
--2024-11-28 10:05:24-- https://objects.githubusercontent.com/github-
production-release-
asset-2e65be/521807533/34b70ade-b6eb-4179-a60f-d6494307226b?X-Amz-
Algorithm=AWS4-HMAC-SHA256&X-Amz-
Credential=releaseassetproduction%2F20241128%2Fus-
east-1%2Fs3%2Faws4_request&X-Amz-Date=20241128T100524Z&X-Amz-Expires=300&X-Amz-S
ignature=ac3497975c7adcb044a993d3ee0c9f80b6c83cba6870c223209bb0bfe5875373&X-Amz-
SignedHeaders=host&response-content-
disposition=attachment%3B%20filename%3Dyolo1n.pt&response-content-
type=application%2Foctet-stream
Resolving objects.githubusercontent.com (objects.githubusercontent.com)...
185.199.109.133, 185.199.110.133, 185.199.111.133, ...
Connecting to objects.githubusercontent.com
(objects.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5613764 (5.4M) [application/octet-stream]
Saving to: 'yolo1n.pt.1'

yolo1n.pt.1      100%[=====] 5.35M --.-KB/s    in 0.02s

2024-11-28 10:05:24 (345 MB/s) - 'yolo1n.pt.1' saved [5613764/5613764]
```

```
[27]: os.path.exists('yolo1n.pt')
```

```
[27]: True
```

```
[ ]: # Load the YOLO model
model = YOLO('yolo1n.pt')
```

Training

```
[ ]: training_results = model.train(data=data_yaml_path, epochs=3, imgsz=640,
    ↪batch=4, verbose=False)

Ultralytics 8.3.38 Python-3.10.15 torch-2.5.1+cu118 CUDA:0 (NVIDIA
A100-SXM4-40GB, 40514MiB)
engine/trainer: task=detect, mode=train, model=yolo11n.pt, data=/ro
ot/notebooks/xray/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/data.
yaml, epochs=3, time=None, patience=100, batch=4, imgsz=640, save=True,
save_period=-1, cache=False, device=None, workers=8, project=None, name=train13,
exist_ok=False, pretrained=True, optimizer=auto, verbose=False, seed=0,
deterministic=True, single_cls=False, rect=False, cos_lr=False, close_mosaic=10,
resume=False, amp=True, fraction=1.0, profile=False, freeze=None,
multi_scale=False, overlap_mask=True, mask_ratio=4, dropout=0.0, val=True,
split=val, save_json=False, save_hybrid=False, conf=None, iou=0.7, max_det=300,
half=False, dnn=False, plots=True, source=None, vid_stride=1,
stream_buffer=False, visualize=False, augment=False, agnostic_nms=False,
classes=None, retina_masks=False, embed=None, show=False, save_frames=False,
save_txt=False, save_conf=False, save_crop=False, show_labels=True,
show_conf=True, show_boxes=True, line_width=None, format=torchscript,
keras=False, optimize=False, int8=False, dynamic=False, simplify=True,
opset=None, workspace=None, nms=False, lr0=0.01, lrf=0.01, momentum=0.937,
weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1,
box=7.5, cls=0.5, df1=1.5, pose=12.0, kobj=1.0, nbs=64, hsv_h=0.015, hsv_s=0.7,
hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0,
flipud=0.0, fliplr=0.5, bgr=0.0, mosaic=1.0, mixup=0.0, copy_paste=0.0,
copy_paste_mode=flip, auto_augment=randaugment, erasing=0.4, crop_fraction=1.0,
cfg=None, tracker=botsort.yaml, save_dir=runs/detect/train13
Overriding model.yaml nc=80 with nc=3

          from   n     params   module
arguments
  0           -1   1       464 ultralytics.nn.modules.conv.Conv
[3, 16, 3, 2]
  1           -1   1      4672 ultralytics.nn.modules.conv.Conv
[16, 32, 3, 2]
  2           -1   1      6640 ultralytics.nn.modules.block.C3k2
[32, 64, 1, False, 0.25]
  3           -1   1     36992 ultralytics.nn.modules.conv.Conv
[64, 64, 3, 2]
  4           -1   1     26080 ultralytics.nn.modules.block.C3k2
[64, 128, 1, False, 0.25]
  5           -1   1    147712 ultralytics.nn.modules.conv.Conv
[128, 128, 3, 2]
  6           -1   1     87040 ultralytics.nn.modules.block.C3k2
[128, 128, 1, True]
  7           -1   1    295424 ultralytics.nn.modules.conv.Conv
[128, 256, 3, 2]
  8           -1   1    346112 ultralytics.nn.modules.block.C3k2
```

```

[256, 256, 1, True]
 9           -1 1    164608 ultralytics.nn.modules.block.SPPF
[256, 256, 5]
 10          -1 1    249728 ultralytics.nn.modules.block.C2PSA
[256, 256, 1]
 11          -1 1      0 torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
 12         [-1, 6] 1      0 ultralytics.nn.modules.conv.Concat
[1]
 13          -1 1    111296 ultralytics.nn.modules.block.C3k2
[384, 128, 1, False]
 14          -1 1      0 torch.nn.modules.upsampling.Upsample
[None, 2, 'nearest']
 15         [-1, 4] 1      0 ultralytics.nn.modules.conv.Concat
[1]
 16          -1 1    32096 ultralytics.nn.modules.block.C3k2
[256, 64, 1, False]
 17          -1 1    36992 ultralytics.nn.modules.conv.Conv
[64, 64, 3, 2]
 18         [-1, 13] 1      0 ultralytics.nn.modules.conv.Concat
[1]
 19          -1 1    86720 ultralytics.nn.modules.block.C3k2
[192, 128, 1, False]
 20          -1 1   147712 ultralytics.nn.modules.conv.Conv
[128, 128, 3, 2]
 21         [-1, 10] 1      0 ultralytics.nn.modules.conv.Concat
[1]
 22          -1 1   378880 ultralytics.nn.modules.block.C3k2
[384, 256, 1, True]
 23        [16, 19, 22] 1   431257 ultralytics.nn.modules.head.Detect
[3, [64, 128, 256]]
YOLO11n summary: 319 layers, 2,590,425 parameters, 2,590,409 gradients, 6.4
GFLOPs

```

Transferred 448/499 items from pretrained weights
 WARNING Comet installed but not initialized correctly, not logging this run.
 Comet.ml requires an API key. Please provide as the first argument to
 Experiment(api_key) or as an environment variable named COMET_API_KEY
TensorBoard: Start with 'tensorboard --logdir runs/detect/train13',
 view at <http://localhost:6006/>

Freezing layer 'model.23.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks...
AMP: checks passed

train: Scanning /root/notebooks/xray/data/BrainTumorDetectionYolov9
 /BrainTumorDetectionYolov9/train/labels.cache... 6930 images, 79 backgrounds, 0
 corrupt: 100% | 6930/6930 [00:00<?, ?it/s]
val: Scanning /root/notebooks/xray/data/BrainTumorDetectionYolov9/B

```
rainTumorDetectionYolov9/valid/labels.cache... 1980 images, 17 backgrounds, 0
corrupt: 100% | 1980/1980 [00:00<?, ?it/s]
```

```
Plotting labels to runs/detect/train13/labels.jpg...
```

```
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and
'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum'
automatically...
```

```
optimizer: AdamW(lr=0.001429, momentum=0.9) with parameter groups
81 weight(decay=0.0), 88 weight(decay=0.0005), 87 bias(decay=0.0)
```

```
TensorBoard: model graph visualization added
```

```
Image sizes 640 train, 640 val
```

```
Using 8 dataloader workers
```

```
Logging results to runs/detect/train13
```

```
Starting training for 3 epochs...
```

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
1/3	0.782G	1.883	2.955	1.97	14	640:
100%	1733/1733 [02:16<00:00, 12.67it/s]					
	Class	Images	Instances	Box(P	R	mAP50
mAP50-95): 100%	248/248 [00:12<00:00, 19.84it/s]					
	all	1980	4380	0.462	0.382	0.364
0.157						

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
2/3	0.782G	1.807	2.291	1.902	4	640:
100%	1733/1733 [01:53<00:00, 15.28it/s]					
	Class	Images	Instances	Box(P	R	mAP50
mAP50-95): 100%	248/248 [00:10<00:00, 23.68it/s]					
	all	1980	4380	0.558	0.405	0.439
0.2						

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
3/3	0.776G	1.708	2.022	1.83	1	640:
100%	1733/1733 [01:51<00:00, 15.49it/s]					
	Class	Images	Instances	Box(P	R	mAP50
mAP50-95): 100%	248/248 [00:10<00:00, 23.09it/s]					
	all	1980	4380	0.636	0.478	0.515
0.256						

```
3 epochs completed in 0.123 hours.
```

```
Optimizer stripped from runs/detect/train13/weights/last.pt, 5.5MB
```

```
Optimizer stripped from runs/detect/train13/weights/best.pt, 5.5MB
```

```
Validating runs/detect/train13/weights/best.pt...
```

```
Ultralytics 8.3.38 Python-3.10.15 torch-2.5.1+cu118 CUDA:0 (NVIDIA
```

```

A100-SXM4-40GB, 40514MiB)
YOLO11n summary (fused): 238 layers, 2,582,737 parameters, 0 gradients, 6.3
GFLOPs

      Class     Images Instances     Box(P       R       mAP50
mAP50-95): 100% | 248/248 [00:08<00:00, 27.97it/s]
          all      1980      4380      0.638      0.477      0.516
0.256
Speed: 0.2ms preprocess, 1.2ms inference, 0.0ms loss, 0.7ms postprocess per
image
Results saved to runs/detect/train13

```

```

[ ]: # Extract metrics into a structured dictionary
def extract_metrics(results):
    return {
        "Precision (P)": results.results_dict['metrics/precision(B)'],
        "Recall (R)": results.results_dict['metrics/recall(B)'],
        "mAP@50": results.results_dict['metrics/mAP50(B)'],
        "mAP@50-95": results.results_dict['metrics/mAP50-95(B)'],
        "Label 0": results.maps[0],
        "Label 1": results.maps[1],
        "Label 2": results.maps[2]
    }

# Function to plot metrics
def plot_metrics(metrics, title):
    overall_metrics = {k: metrics[k] for k in ['Precision (P)', 'Recall (R)', 'mAP@50', 'mAP@50-95']}
    labels_metrics = {k: metrics[k] for k in ['Label 0', 'Label 1', 'Label 2']}
    categories_overall = list(overall_metrics.keys())
    categories_labels = list(labels_metrics.keys())

    x1 = np.arange(len(categories_overall))
    x2 = np.arange(len(categories_labels))

    width = 0.4
    plt.figure(figsize=(12, 6))

    # Plotting overall metrics
    plt.bar(x1, overall_metrics.values(), width=width, label='Overall Metrics')

    # Plotting label-specific metrics
    plt.bar(x2 + len(x1) + 1, labels_metrics.values(), width=width, label='Label Metrics')

    x_ticks = list(x1) + list(x2 + len(x1) + 1)
    x_labels = categories_overall + categories_labels

```

```

plt.xticks(x_ticks, x_labels, rotation=45)

plt.ylabel('Score')
plt.title(title)
plt.legend()
plt.tight_layout()
plt.show()

```

```
[ ]: train_metrics = extract_metrics(training_results)
validation_results = model.val()
val_metrics = extract_metrics(validation_results)
test_results = model.val(split='test')
test_metrics = extract_metrics(test_results)
```

Ultralytics 8.3.38 Python-3.10.15 torch-2.5.1+cu118 CUDA:0 (NVIDIA A100-SXM4-40GB, 40514MiB)

```
val: Scanning /root/notebooks/xray/data/BrainTumorDetectionYolov9/B
rainTumorDetectionYolov9/valid/labels.cache... 1980 images, 17 backgrounds, 0
corrupt: 100%|      | 1980/1980 [00:00<?, ?it/s]
          Class     Images   Instances    Box(P)      R      mAP50
mAP50-95): 100%|      | 495/495 [00:10<00:00, 45.15it/s]
```

	all	1980	4380	0.637	0.478	0.516
--	-----	------	------	-------	-------	-------

0.256

Speed: 0.2ms preprocess, 2.2ms inference, 0.0ms loss, 0.7ms postprocess per image

Results saved to runs/detect/train134

Ultralytics 8.3.38 Python-3.10.15 torch-2.5.1+cu118 CUDA:0 (NVIDIA A100-SXM4-40GB, 40514MiB)

```
val: Scanning /root/notebooks/xray/data/BrainTumorDetectionYolov9/B
rainTumorDetectionYolov9/test/labels.cache... 990 images, 17 backgrounds, 0
corrupt: 100%|      | 990/990 [00:00<?, ?it/s]
          Class     Images   Instances    Box(P)      R      mAP50
mAP50-95): 100%|      | 248/248 [00:05<00:00, 43.74it/s]
```

	all	990	2104	0.616	0.497	0.525
--	-----	-----	------	-------	-------	-------

0.262

Speed: 0.2ms preprocess, 2.4ms inference, 0.0ms loss, 0.7ms postprocess per image

Results saved to runs/detect/train135

```
[40]: metrics_dict = {
    "train_metrics": train_metrics,
    "val_metrics": val_metrics,
    "test_metrics": test_metrics
}
```

```
[22]: # Display tables
print("\nTraining Metrics:")
display(train_metrics)
print("\nValidation Metrics:")
display(val_metrics)
print("\nTest Metrics Table:")
display(test_metrics)
```

Training Metrics:

```
{'Precision (P)': 0.638246729008168,
'Recall (R)': 0.47726948781084544,
'mAP@50': 0.5157992769960671,
'mAP@50-95': 0.2556449039679959,
'Label 0': 0.15332801147123037,
'Label 1': 0.41374663104418474,
'Label 2': 0.19986006938857254}
```

Validation Metrics:

```
{'Precision (P)': 0.6373833019954881,
'Recall (R)': 0.47828115499843943,
'mAP@50': 0.5158711008652043,
'mAP@50-95': 0.255738592008355,
'Label 0': 0.15354613304372242,
'Label 1': 0.4133639892072063,
'Label 2': 0.20030565377413626}
```

Test Metrics Table:

```
{'Precision (P)': 0.6161164204133791,
'Recall (R)': 0.4973159405259324,
'mAP@50': 0.5246112984317924,
'mAP@50-95': 0.26180172893995085,
'Label 0': 0.16815688415002067,
'Label 1': 0.42024402939784045,
'Label 2': 0.19700427327199166}
```

Breakpoint - YOLO v1 Model & Metrics This breakpoint saves and loads the trained YOLOv1 model and metrics, allowing direct interaction without rerunning the full notebook.

```
[ ]: # Run this cell to define the paths
save_dir = os.path.join("results", "yolov1n_brain_tumor_detection_v1")
os.makedirs(save_dir, exist_ok=True)
metrics_path = os.path.join(save_dir, "metrics.pkl")
fine_tuned_model_path = os.path.join(save_dir, "yolov1n_brain_tumor_detection_v1.pt")
```

```
[ ]: # Saving in case if have to pull info. Run this cell to save
with open(metrics_path, "wb") as f:
    pickle.dump(metrics_dict, f)
print(f"Metrics saved to: {metrics_path}")

# Save the model
model.save(fine_tuned_model_path)
print(f"Model saved to: {fine_tuned_model_path}")
```

Metrics saved to: results/yolov11n_brain_tumor_detection_v1/metrics.pkl
 Model saved to:
 results/yolov11n_brain_tumor_detection_v1/yolov11n_brain_tumor_detection_v1.pt

```
[ ]: # Run this cell to read the data
with open(metrics_path, "rb") as f:
    metrics_dict = pickle.load(f)
print(f"Metrics loaded successfully from: {metrics_path}")
model = YOLO(fine_tuned_model_path)
print(f"Model loaded successfully from: {fine_tuned_model_path}")
```

Metrics loaded successfully from:
 results/yolov11n_brain_tumor_detection_v1/metrics.pkl
 Model loaded successfully from:
 results/yolov11n_brain_tumor_detection_v1/yolov11n_brain_tumor_detection_v1.pt

Model Metrics Analysis The following section presents metrics for different data groups and highlights label-class-specific metrics.

```
[ ]: def compare_all_metrics_bar_chart(train_metrics=None, val_metrics=None, □
    ↪test_metrics=None, title="Comparison of 'All' Metrics Across Datasets"):
    categories = ["Precision (P)", "Recall (R)", "mAP@50", "mAP@50-95"]
    x = np.arange(len(categories))

    bar_width = 0.25
    plt.figure(figsize=(10, 6))
    if train_metrics:
        train_scores = [train_metrics.get(k, 0) for k in categories]
        plt.bar(x - bar_width, train_scores, width=bar_width, label='Training', □
    ↪alpha=0.8, color='blue')
    if val_metrics:
        val_scores = [val_metrics.get(k, 0) for k in categories]
        plt.bar(x, val_scores, width=bar_width, label='Validation', alpha=0.8, □
    ↪color='orange')
    if test_metrics:
        test_scores = [test_metrics.get(k, 0) for k in categories]
        plt.bar(x + bar_width, test_scores, width=bar_width, label='Test', □
    ↪alpha=0.8, color='green') # Force test bars to green
```

```

plt.xticks(x, categories)
plt.ylabel("Score")
plt.title(title)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

```

Training Metrics

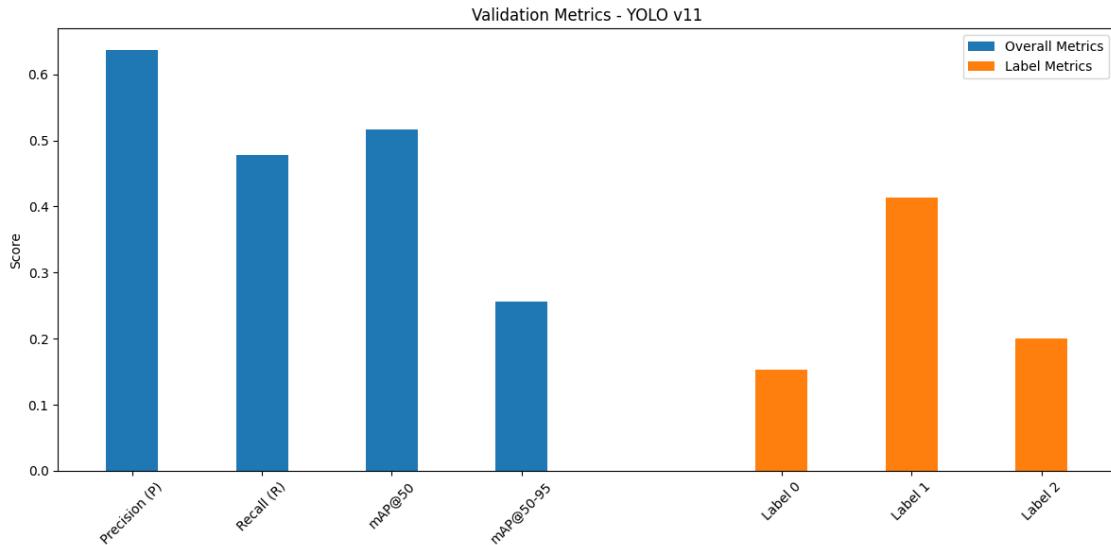
[41]: `plot_metrics(train_metrics, "Training Metrics - YOLO v11")`



On the training data, the model maintains a decent precision of around 63.8% and a recall just under 47.7%, suggesting it identifies over half of the tumor instances it sees but still misses a considerable portion. Its mAP@50 hovers near 51.6%, indicating that slightly more than half of its bounding boxes align well with ground truth, while the stricter mAP@50-95 dips to about 25.6%. Across classes, performance is uneven: for example, while one class (label 1) might reach around 41.4% in class-specific accuracy metrics, others struggle at about 15–20%, highlighting the need for more balanced data or refined training strategies.

Validation Metrics

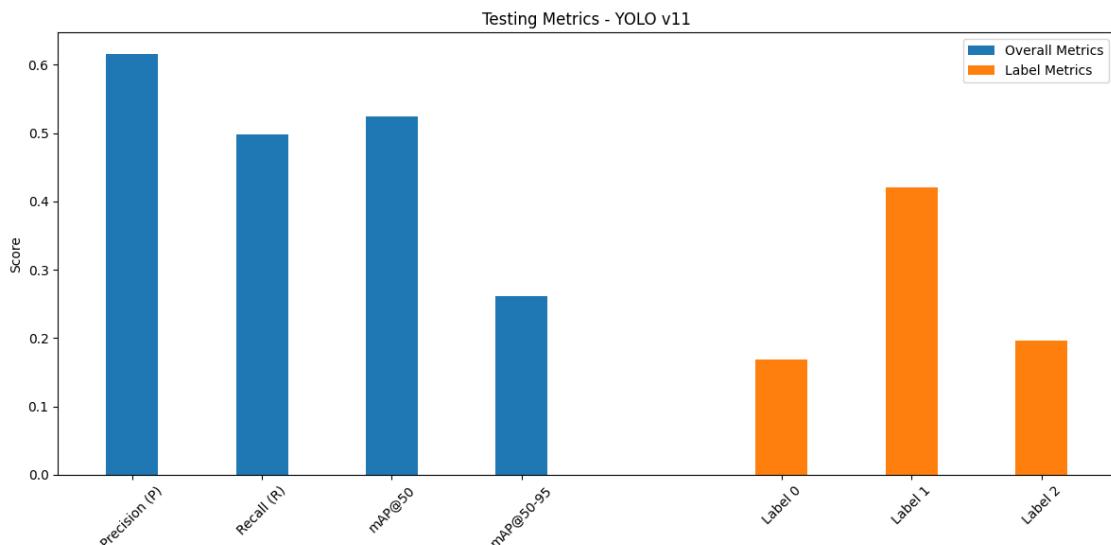
[40]: `plot_metrics(val_metrics, "Validation Metrics - YOLO v11")`



When applied to the validation data which is unseen data but from a similar distribution, the model's precision and recall remain consistent at roughly 63.7% and 47.8% respectively. This is similar to the training set performance. The mAP@50 also sits around 51.6%, and the mAP@50-95 is again about 25.6%, showing similar localization challenges. The class imbalance noted during training comes up here as well, with label 0 and label 2 classes till lagging behind.

Test Metrics

```
[39]: # Plot metrics
plot_metrics(test_metrics, "Testing Metrics - YOLO v11")
```

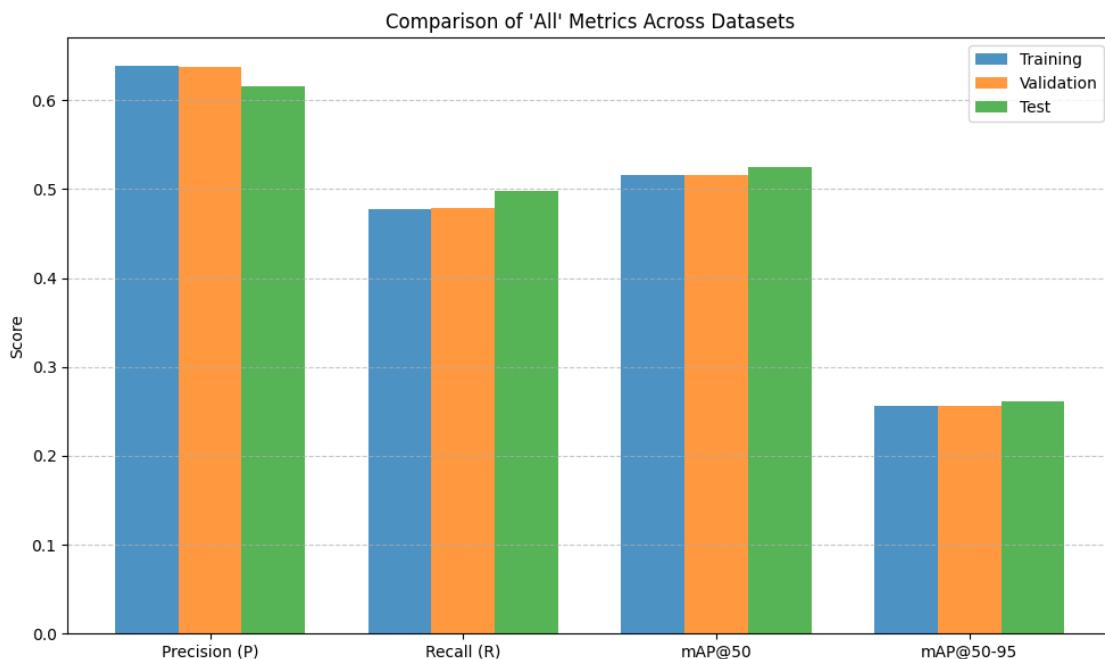


On the unseen test data, the model has a precision of 61.6% and a recall of 49.7%. Its mAP@50

is 52.5%, slightly better than the training and validation results, and its mAP@50-95 is 26.2%, showing some improvement with stricter localization. However, the performance across classes is uneven: Label 1 performs the best at 42.0%, while Labels 0 and 2 score much lower at 16.8% and 19.7%. This difference shows that the model works fairly well on new data and has a similar consistency to the trained data.

Overall Analysis and Implications

```
[ ]: compare_all_metrics_bar_chart(train_metrics, val_metrics, test_metrics)
```



The model's performance is stable across training, validation, and test sets. Precision is around 63% for training and validation and drops slightly to 62% on the test set. Recall is about 48% for training and validation and increases to 50% on the test set. The mAP@50 is steady at 52%, while the stricter mAP@50-95 remains in the mid-20% range. This shows the model has some difficulty with precise localization but performs consistently.

Label 1 performs much better than the others with around 41–42% accuracy across all sets. Labels 0 and 2 stay below 20%, showing the model struggles to detect them. This gap suggests that Label 1 tumors are easier to identify. This could also be because of the bigger size of the area while other tumors are smaller. We might need higher-quality data to be able to capture that. This could also be because label classes 0 and 2 have 37% and 39% fewer data instances than class 0, and their difference is 13%+ each from the full data.

Visualizing Model Performance

```
[ ]: def display_image(image_path, title = None):
    image = load_image(image_path)
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```

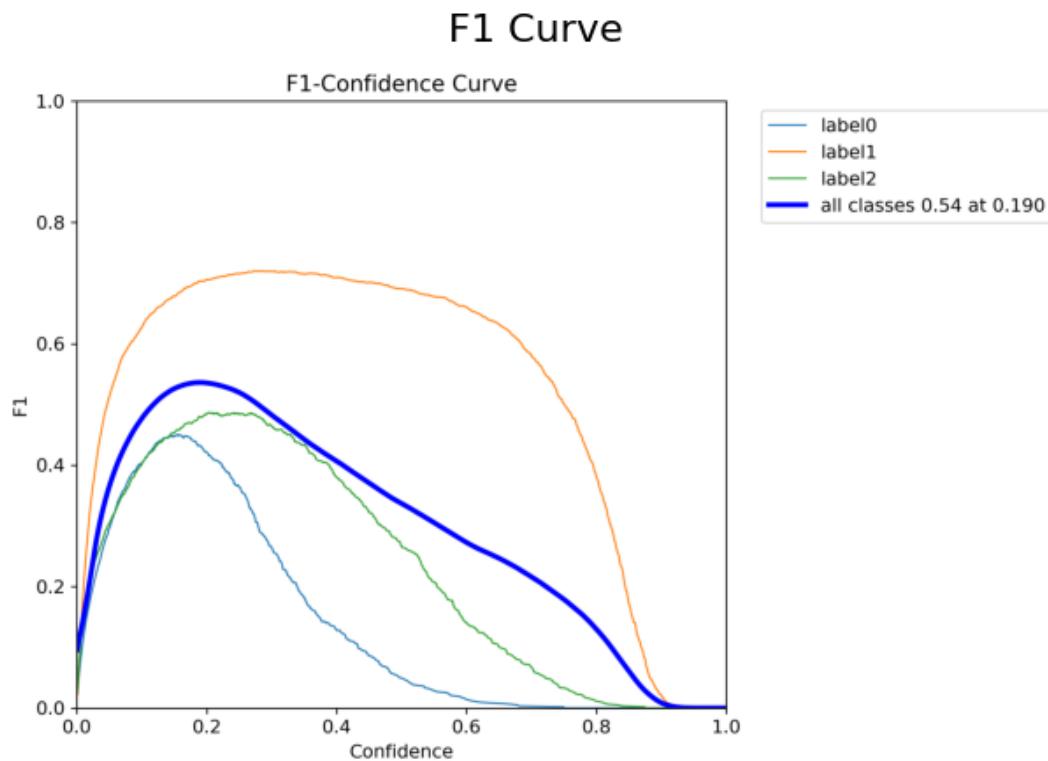
plt.figure(figsize=(8, 6))
plt.imshow(image_rgb)
plt.axis("off")
if title:
    plt.title(title, fontsize=16)
plt.show()

result_image_dir = os.path.join("results", "yolov11n_brain_tumor_detection_v1", "detect_yolo11")

```

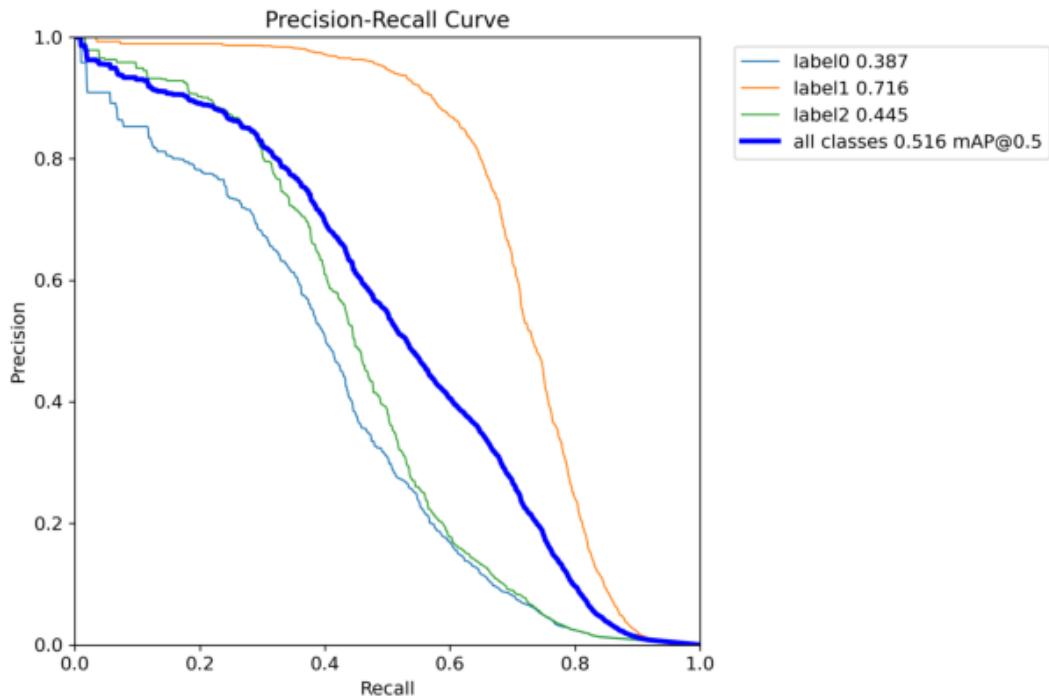
Performance Curves

[73]: `display_image(os.path.join(result_image_dir, "F1_curve.png"), title="F1 Curve")`



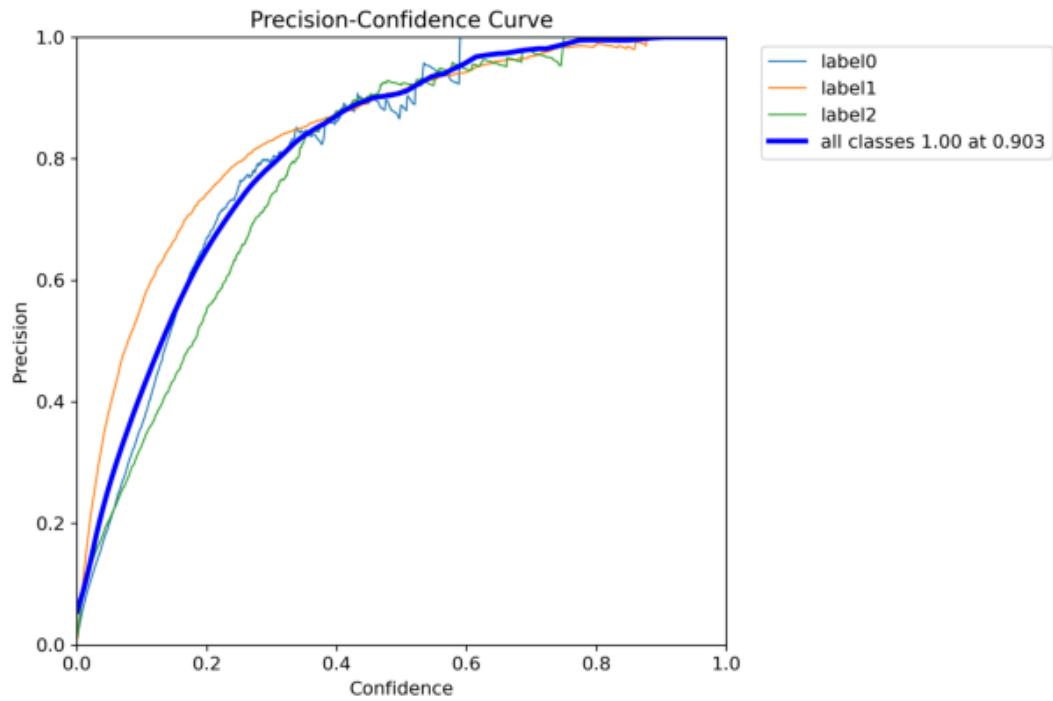
[74]: `display_image(os.path.join(result_image_dir, "PR_curve.png"), title="Precision-Recall Curve")`

Precision-Recall Curve



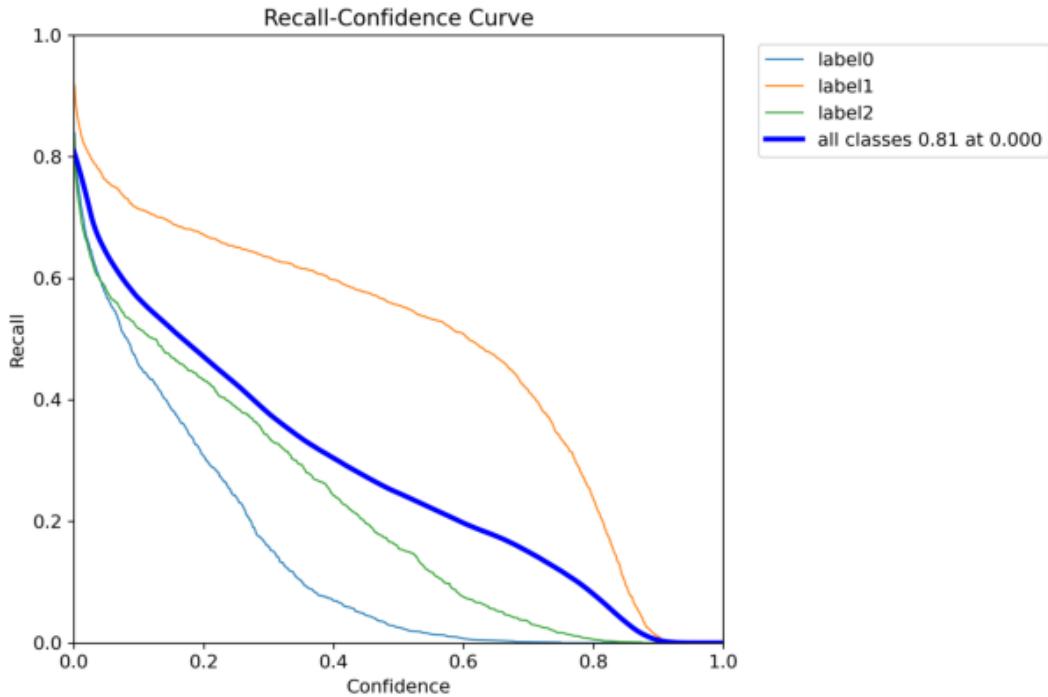
```
[76]: display_image(os.path.join(result_image_dir, "P_curve.png"), title="Precision  
Curve")
```

Precision Curve



```
[75]: display_image(os.path.join(result_image_dir, "R_curve.png"), title="Recall ↴ Curve")
```

Recall Curve



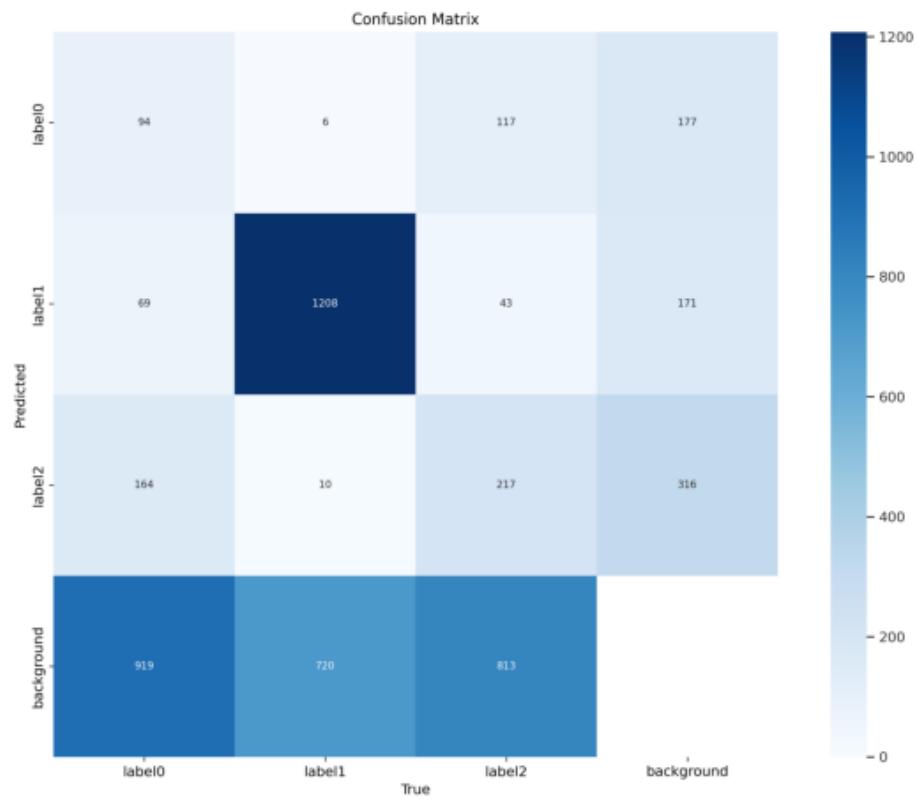
Analysis:

After examining all four curves (F1-Confidence, Precision-Recall, Precision, and Recall), we see a consistent pattern: Label 1 is detected more effectively than the other classes. The F1 curve shows its best score around 0.54 at a 0.19 confidence threshold, while the Precision-Recall curve confirms Label 1's advantage with a higher AUC (0.716) compared to others. As confidence increases, Precision reaches 1.00 for all classes by 0.903, but Label 1 achieves strong precision even at lower thresholds. Meanwhile, the Recall curve shows that all labels start high at low thresholds, though Label 1 maintains better recall as confidence increases. This is the same pattern we saw within "Model Metrics Analysis".

Confusion Matrices

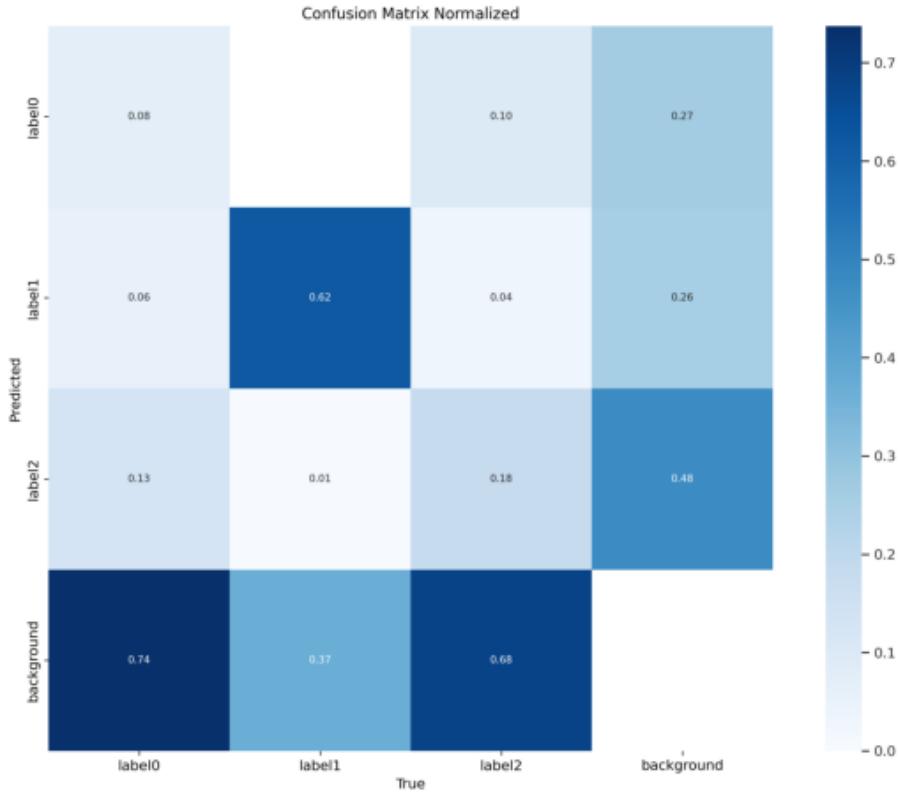
```
[77]: display_image(os.path.join(result_image_dir, "confusion_matrix.png"),  
    title="Confusion Matrix")
```

Confusion Matrix



```
[78]: display_image(os.path.join(result_image_dir, "confusion_matrix_normalized.  
png"), title="Normalized Confusion Matrix")
```

Normalized Confusion Matrix



Analysis

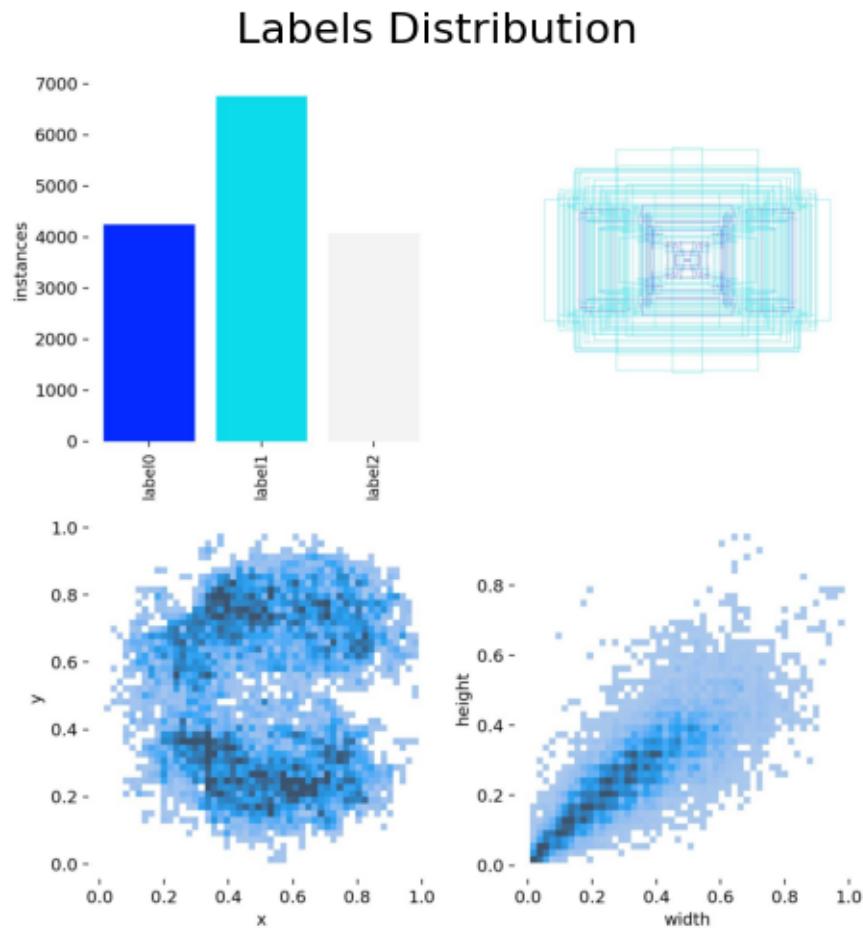
The confusion matrix and its normalized form show a strong imbalance in predictions. While the highest diagonal value (0.62 for `label1`) indicates the model best recognizes `label1`, it often misclassifies other labels. For instance, `label0` and `label2` examples are frequently predicted as `background`, and `label0` data is often mistaken for `label2` and vice versa.

The “background” category is also over-predicted, as seen in its high off-diagonal fractions, suggesting the model defaults to background when uncertain. This issue is common, as seen in Zain’s [AAI-521-Assignment-3](#). Several factors may explain why the model often predicts “background”:

- 1. Data Imbalance:** Background pixels far outnumber labeled objects, pushing the model to default to background. The paper *“Imbalance Problems in Object Detection”* by Oksuz et al. discusses how this affects accuracy.
- 2. Lack of Object Variety:** Without enough examples of each object, the model struggles to distinguish them, reverting to background. However, the dataset size (6930 training images) suggests this may not be the main issue.
- 3. Simple Model Architecture:** The chosen model (YOLOv11) may lack the complexity needed to handle subtle distinctions in cancer data.

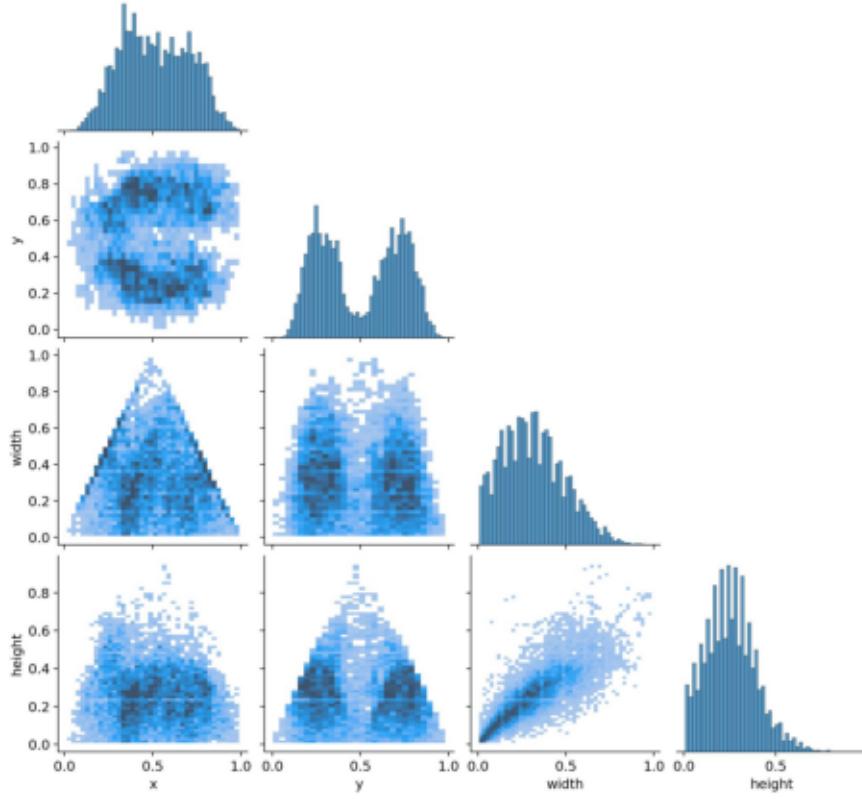
Label Visualizations

```
[79]: display_image(os.path.join(result_image_dir, "labels.jpg"), title="Labels Distribution")
```



```
[80]: display_image(os.path.join(result_image_dir, "labels_correlogram.jpg"), title="Labels Correlogram")
```

Labels Correlogram



From the labels distribution and correlogram visualizations, we see a continuation of the trends noted earlier: Label 1 is significantly more frequent, while Labels 0 and 2 are less represented and often co-occur. This imbalance reinforces the model's tendency to excel with Label 1 and struggle with the minority classes. The correlogram hints at underlying relationships between certain labels, which may affect how the model perceives multi-label contexts.

I wish we had more time to address these issues. To address these issues, I would try the following:

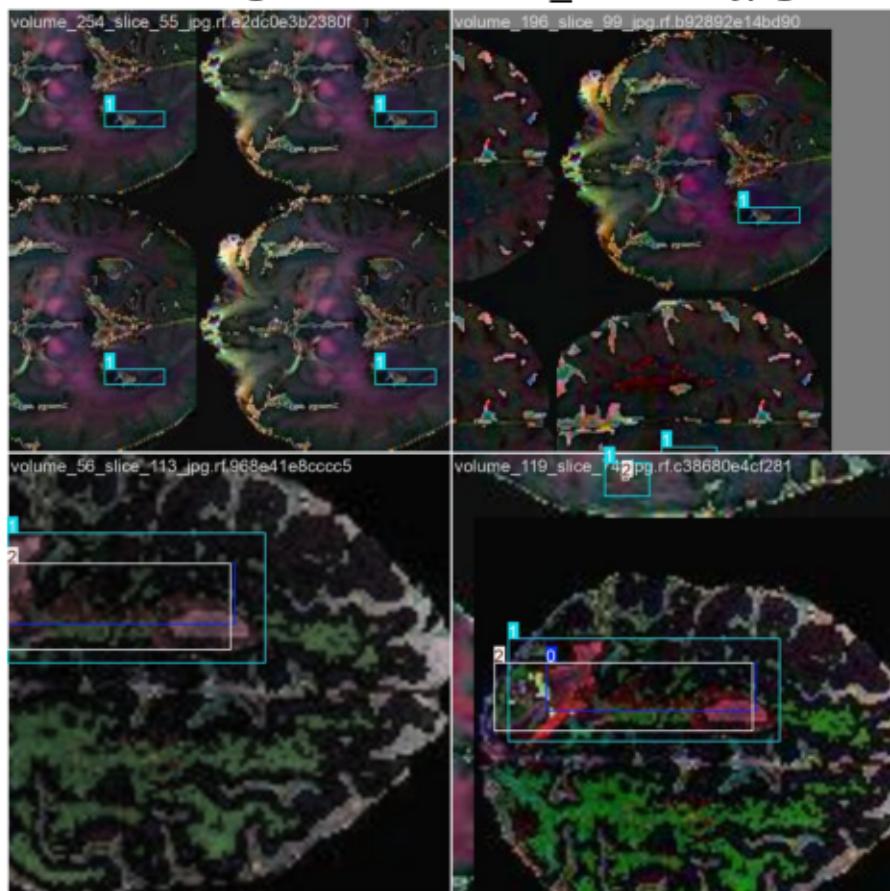
1. Data Augmentation: Generate more samples for minority classes using transformations (e.g., flipping, rotation).
2. Oversampling (SMOTE/GANs): Create synthetic samples for minority classes to balance the dataset.
3. Class Weighting: Assign higher weights to minority classes during training to focus on them more effectively.

These are inspired by some of steps highlighted within *8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset* by **Jason Brownlee**.

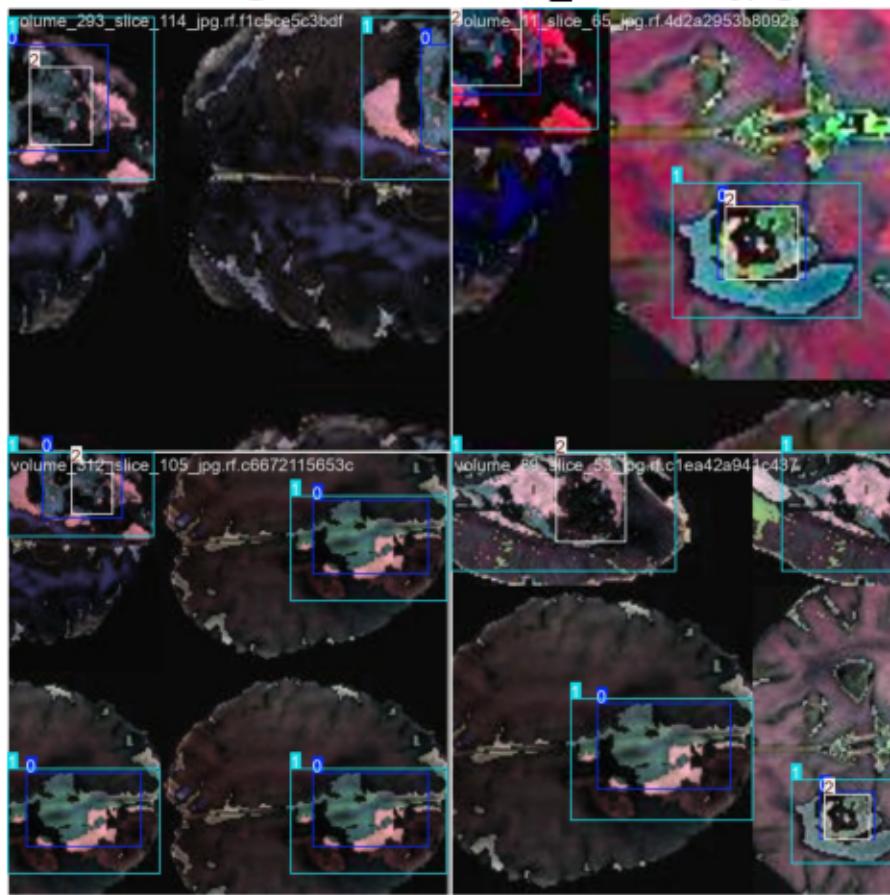
Training Batches

```
[81]: training_batches = [
    "train_batch0.jpg",
    "train_batch1.jpg",
    "train_batch2.jpg"
]
for batch in training_batches:
    display_image(os.path.join(result_image_dir, batch), title=f"Training Batch {batch}")
```

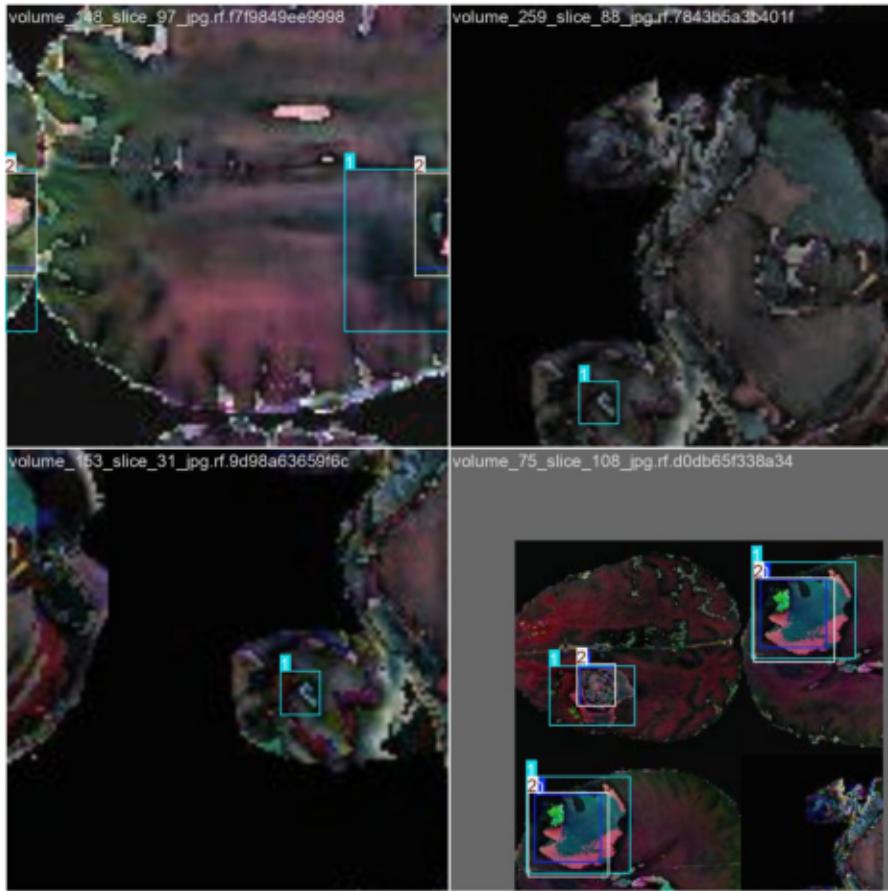
Training Batch - train_batch0.jpg



Training Batch - train_batch1.jpg



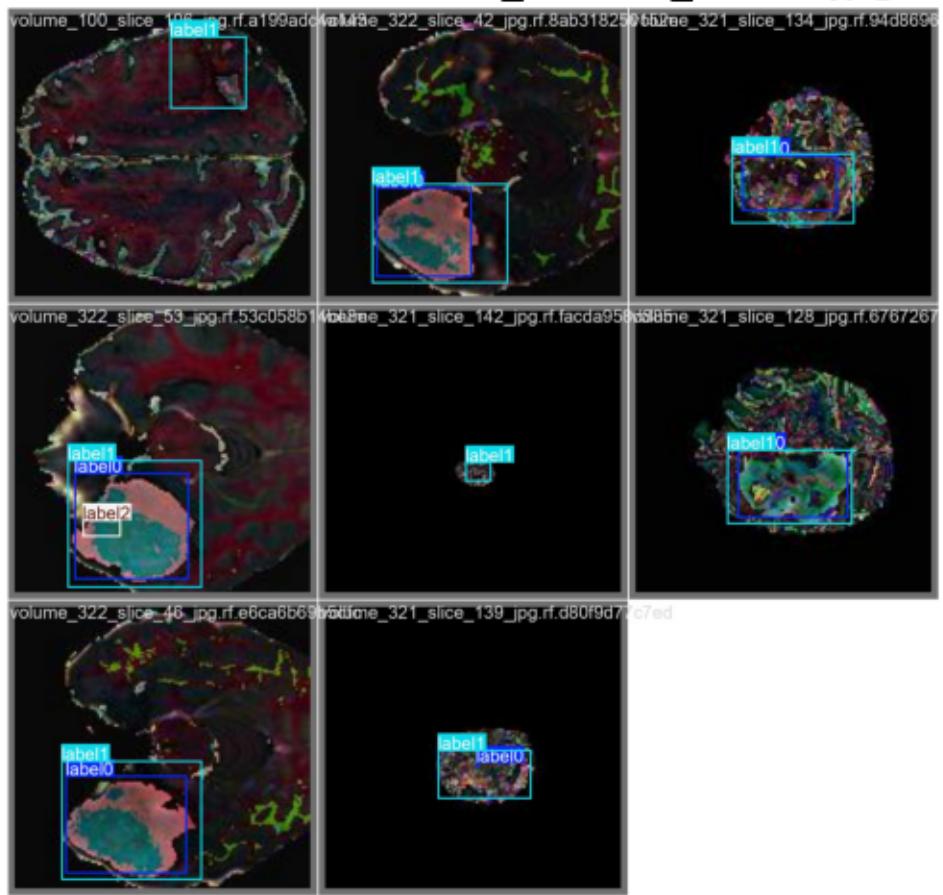
Training Batch - train_batch2.jpg



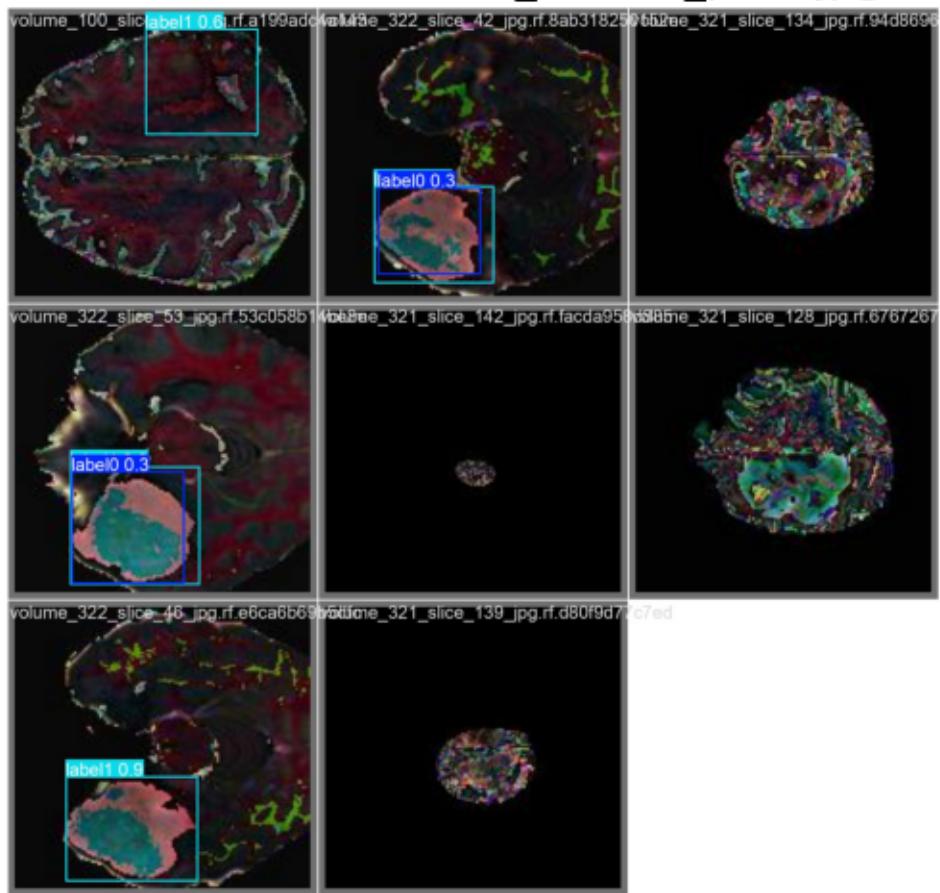
Validation Batches

```
[82]: validation_batches = [
    "val_batch0_labels.jpg",
    "val_batch0_pred.jpg",
    "val_batch1_labels.jpg",
    "val_batch1_pred.jpg",
    "val_batch2_labels.jpg",
    "val_batch2_pred.jpg"
]
for batch in validation_batches:
    display_image(os.path.join(result_image_dir, batch), title=f"Validation_{batch - {batch}}")
```

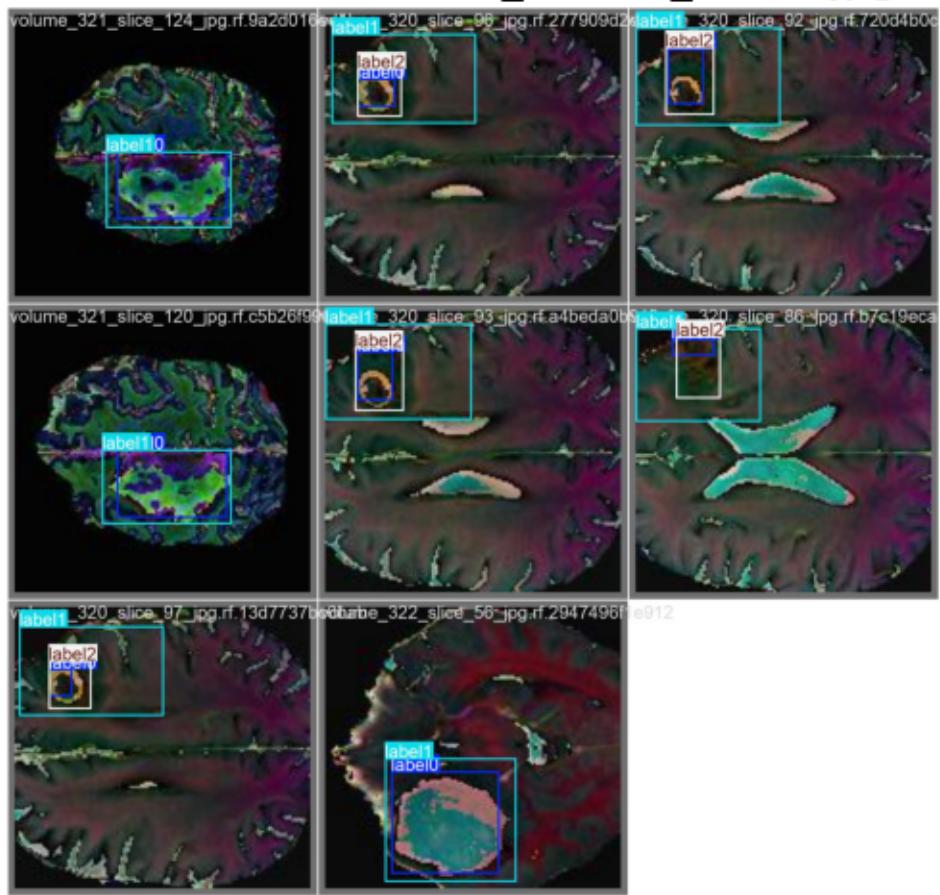
Validation Batch - val_batch0_labels.jpg



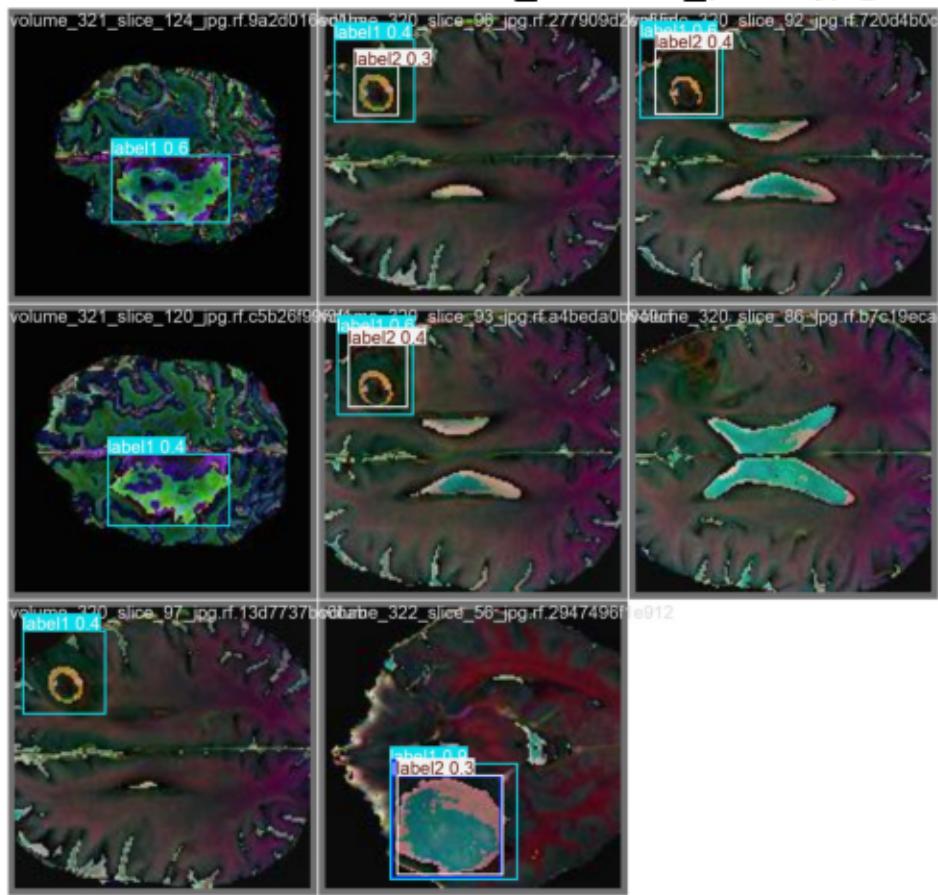
Validation Batch - val_batch0_pred.jpg



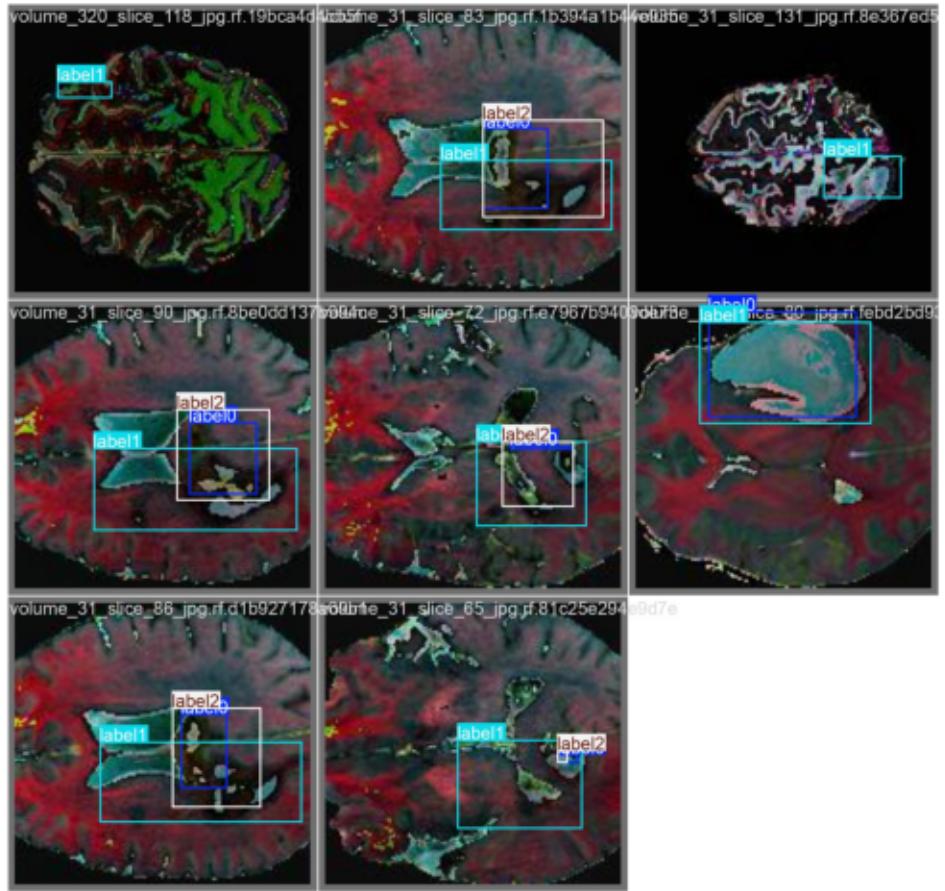
Validation Batch - val_batch1_labels.jpg



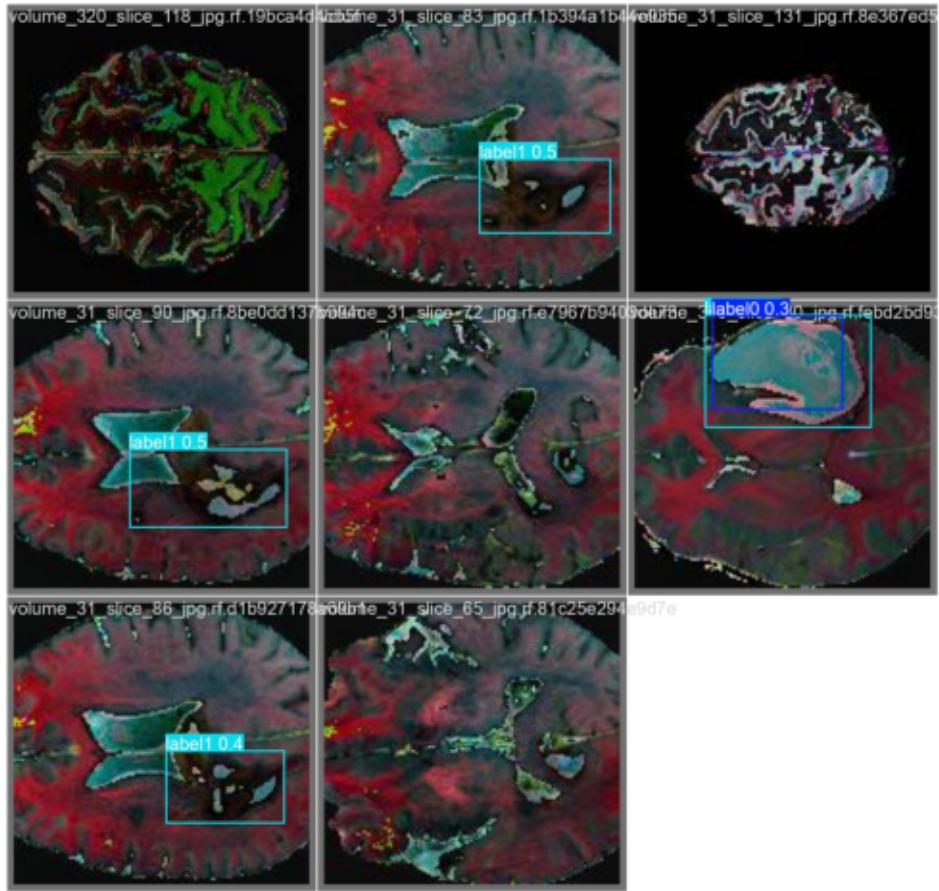
Validation Batch - val_batch1_pred.jpg



Validation Batch - val_batch2_labels.jpg



Validation Batch - val_batch2_pred.jpg



1.4.4 Model Comparison - YOLOv8 vs YOLOv11

Comparing models using the stored test data metrics.

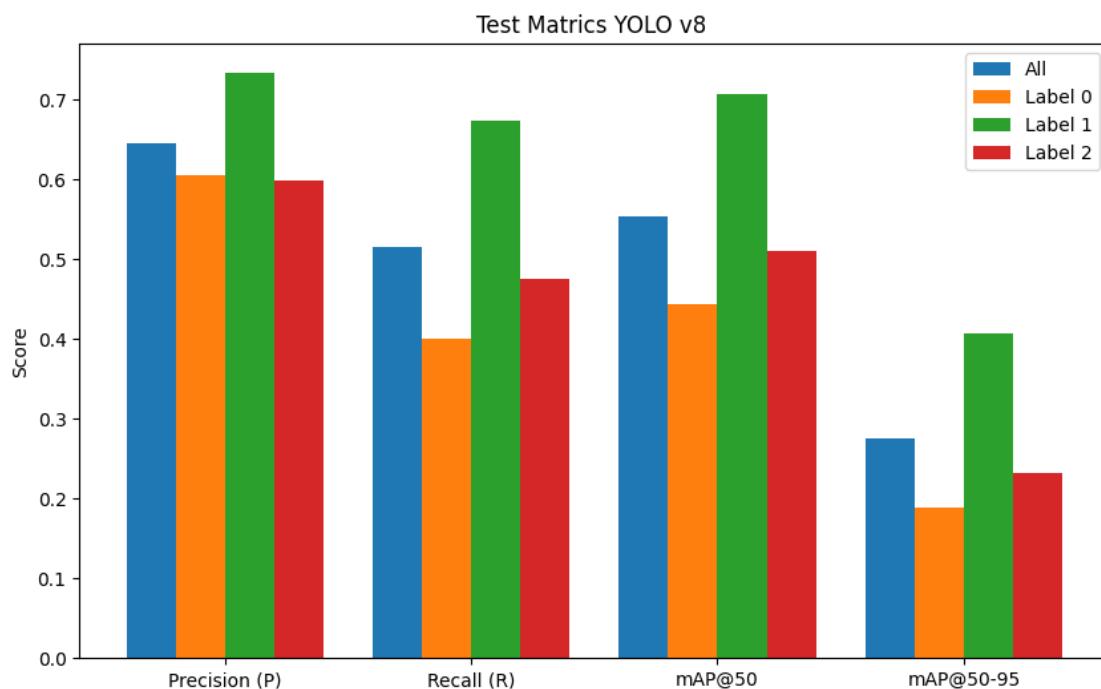
```
[ ]: # Handcoding these as we only saved the numbers rather than the pkl or the dictionary.  
yolo_v8_metrics = {  
    'All': {  
        "Precision (P)": 0.646,  
        "Recall (R)": 0.516,  
        "mAP@50": 0.553,  
        "mAP@50-95": 0.276,  
    },  
    'Label 0': {  
        "Precision (P)": 0.606,  
        "Recall (R)": 0.400,  
        "mAP@50": 0.443,  
    },  
}
```

```

        "mAP@50-95": 0.188,
    },
    'Label 1': {
        "Precision (P)": 0.733,
        "Recall (R)": 0.674,
        "mAP@50": 0.707,
        "mAP@50-95": 0.407,
    },
    'Label 2': {
        "Precision (P)": 0.599,
        "Recall (R)": 0.475,
        "mAP@50": 0.511,
        "mAP@50-95": 0.232,
    },
}

```

```
[ ]: plot_metrics(yolo_v8_metrics, "Test Metrics YOLO v8")
```



Metric Comparison

Metric	YOLOv8	YOLOv11	Difference	Significance
Precision (P)	64.6%	61.6%	-3.0%	Marginal, within normal variation.
Recall (R)	51.6%	49.7%	-1.9%	Minimal, not significant.

Metric	YOLOv8	YOLOv11	Difference	Significance
mAP@50	55.3%	52.5%	-2.8%	Slight drop but not critical.
mAP@50-95	27.6%	26.2%	-1.4%	Insignificant.

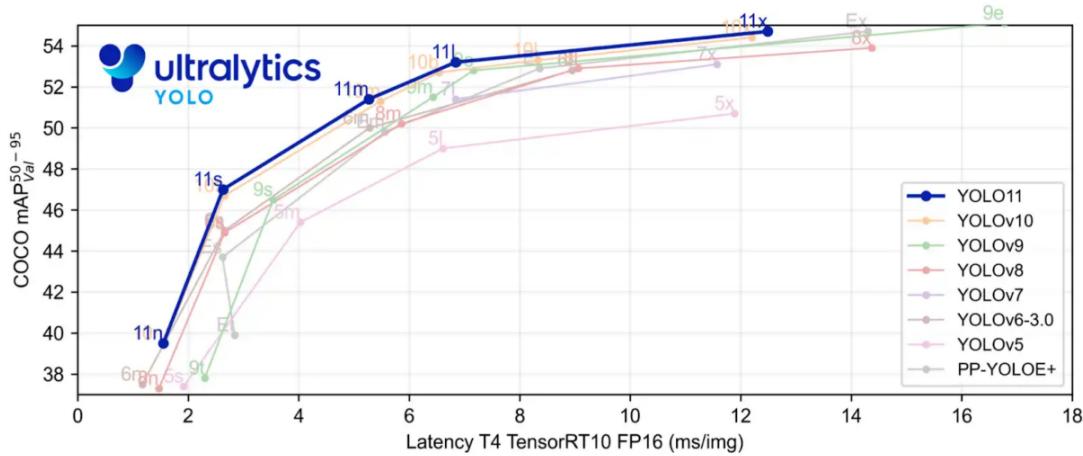
While YOLOv8 shows slightly higher scores across all metrics, the differences are minimal (1.4% to 3%), and none suggest a significant drop in performance. The results indicate YOLOv11 performs similarly to YOLOv8 in this context.

As per other metrics, the YOLOv8, just to run the test took us quite longer than YOLOv11. YOLOv11 was quite fast, about 30-40% within our runs. We did a deeper dive in the research and more metrics to help us in our decision.

```
[58]: # Showing image from references.
# The image comes from:
# https://datadance.ai/machine-learning/
→yolov11-the-next-leap-in-real-time-object-detection.

image = Image.open("external-research/yolo11-research.webp")
image_size = image.size
dpi = 100
fig_size = (image_size[0] / dpi, image_size[1] / dpi)

fig, ax = plt.subplots(figsize=fig_size, dpi=dpi)
ax.imshow(image)
ax.axis('off')
plt.show()
```



Key Features and Innovations in YOLOv11

Feature	Details	Impact
Transformer Back-bone	Captures long-range dependencies and enhances detection of small objects.	Better performance in detecting dense and overlapping objects.
Dynamic Head Design	Adapts to image complexity for optimized processing.	Improved efficiency and detection accuracy.
NMS-Free Training	Eliminates Non-Maximum Suppression, reducing inference time.	Faster and more streamlined detection pipeline.
Dual Label Assignment	Uses one-to-one and one-to-many label assignment.	Better detection in dense and overlapping object scenarios.
Large Kernel Convolutions	Enables more efficient feature extraction with fewer resources.	Boosts overall performance with lower computational cost.
Partial Self-Attention (PSA)	Selectively applies attention mechanisms to specific regions of the feature map.	Enhances global representation learning without significant computational overhead.

Performance Comparison Across YOLO Versions

Model	Speed (FPS)	Accuracy		Parameters	Use Case
		(mAP)			
YOLOv3	30 FPS	53.0%		62M	Balanced performance.
YOLOv4	40 FPS	55.4%		64M	Real-time detection.
YOLOv5	45 FPS	56.8%		44M	Lightweight for mobile/edge.
YOLOv10	50 FPS	58.2%		48M	Optimized for edge deployment.
YOLOv11	60 FPS	61.5%		40M	Faster and more accurate.

YOLOv11 brings important improvements in adaptability and efficiency (Datadance). This makes it a better choice for handling diverse data. The idea of our application is to be able to handle more data later on. While YOLOv8 performs slightly better in precision and recall, the differences are small. YOLOv11's ability to detect small and overlapping objects and its suitability for real-time applications make it a stronger option.

That being said, both of these models will be present for the doctors and physicians to use.

1.5 Dashboard UI API

The following code served two purposes:

- To randomly visually examine a few samples to see how the model is doing.
- To provide functions and API references for the UI to retrieve data and interact with the models.

1.5.1 Creating a models file

This structure is being created in case if we want to add more models later on.

```
[ ]: model_paths_file = "model_paths.txt"

[ ]: # Add more paths here if needed.
model_paths = {
    "YOLOv8n": "results/yolov8n_brain_tumor_detection_v1/
    ↪yolov8n_brain_tumor_detection_v1.pt",
    "YOLOv11n": "results/yolov11n_brain_tumor_detection_v1/
    ↪yolov11n_brain_tumor_detection_v1.pt"
}

[ ]: valid_model_paths = []
for model_name, path in model_paths.items():
    if os.path.exists(path):
        valid_model_paths[model_name] = path
    else:
        print(f"Warning: File not found for model '{model_name}' at path
    ↪'{path}'.")  
  
if valid_model_paths:
    with open(model_paths_file, "w") as f:
        for model_name, path in valid_model_paths.items():
            f.write(f"{model_name}:{path}\n")
    print(f"Valid model paths saved to {model_paths_file}")
else:
    print("No valid model paths to save.")
```

Valid model paths saved to model_paths.txt

```
[13]: def load_model_paths(file_path):
    model_paths_imported = {}
    try:
        with open(file_path, "r") as f:
            for line in f:
                if ":" in line:
                    model_name, model_path = line.strip().split(":")
                    model_paths_imported[model_name] = model_path
    except FileNotFoundError:
        print(f"The file '{file_path}' does not exist.")
        return {}
    if model_paths_imported:
        print("Available models:")
        for idx, (model_name, model_path) in enumerate(model_paths_imported.
            ↪items(), start=1):
            print(f"{idx}. {model_name}: {model_path}")
```

```

    else:
        print("No valid model paths found.")
    return model_paths_imported

model_paths = load_model_paths(model_paths_file)

```

Available models:

1. YOLOv8n:

results/yolov8n_brain_tumor_detection_v1/yolov8n_brain_tumor_detection_v1.pt

2. YOLOv11n:

results/yolov11n_brain_tumor_detection_v1/yolov11n_brain_tumor_detection_v1.pt

```
[14]: selected_model_name = "YOLOv8n"
model_path = model_paths.get(selected_model_name)
model_path
```

```
[14]: 'results/yolov8n_brain_tumor_detection_v1/yolov8n_brain_tumor_detection_v1.pt'
```

```
[17]: model = YOLO(model_path)
print(f"Model loaded successfully from: {model_path}")
```

Model loaded successfully from:

results/yolov8n_brain_tumor_detection_v1/yolov8n_brain_tumor_detection_v1.pt

```
[ ]: # Function to predict image and return predictions as a dictionary
def predict_image(image_path, model):
    results = model.predict(image_path)
    result = results[0]
    pred_boxes = result.boxes

    predictions = []
    for box in pred_boxes:
        x1, y1, x2, y2 = box.xyxy[0].tolist()
        conf = box.conf[0].item()
        class_id = int(box.cls[0].item())
        predictions.append({
            'class_id': class_id,
            'confidence': conf,
            'x1': x1,
            'y1': y1,
            'x2': x2,
            'y2': y2,
            'width': x2 - x1,
            'height': y2 - y1
        })
    return predictions
```

```
[ ]: # Function to draw predictions on an image
def draw_predictions_on_image(image, predictions, alpha=0.4):
    overlay = image.copy()
    for pred in predictions:
        x1 = int(pred['x1'])
        y1 = int(pred['y1'])
        x2 = int(pred['x2'])
        y2 = int(pred['y2'])
        class_id = pred['class_id']
        conf = pred['confidence']
        color = label_colors[class_id % len(label_colors)]
        # Draw filled rectangle with transparency
        cv2.rectangle(overlay, (x1, y1), (x2, y2), color, -1)
        # Add class label text
        cv2.putText(image, f"Pred {class_id}:{conf:.2f}", (x1, y1 - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)
    # Blend the overlay with the original image
    cv2.addWeighted(overlay, alpha, image, 1 - alpha, 0, image)
    return image

# Function to calculate IoU between two boxes
def calculate_iou(boxA, boxB):
    # Determine the coordinates of the intersection rectangle
    x_left = max(boxA['x1'], boxB['x1'])
    y_top = max(boxA['y1'], boxB['y1'])
    x_right = min(boxA['x2'], boxB['x2'])
    y_bottom = min(boxA['y2'], boxB['y2'])

    if x_right < x_left or y_bottom < y_top:
        return 0.0

    # Calculate the area of intersection rectangle
    inter_area = (x_right - x_left) * (y_bottom - y_top)

    # Calculate the areas of the prediction and ground truth rectangles
    boxA_area = (boxA['x2'] - boxA['x1']) * (boxA['y2'] - boxA['y1'])
    boxB_area = (boxB['x2'] - boxB['x1']) * (boxB['y2'] - boxB['y1'])

    # Calculate the IoU
    iou = inter_area / float(boxA_area + boxB_area - inter_area)
    return iou

# Function to calculate accuracy metrics
def calculate_accuracy(ground_truths, predictions, iou_threshold=0.5):
    matched = 0
    total_gt = len(ground_truths)
    total_pred = len(predictions)
```

```

gt_flags = [False] * total_gt # Flags to mark ground truths that have been
                                ↵matched

for pred in predictions:
    pred_box = {
        'x1': pred['x1'],
        'y1': pred['y1'],
        'x2': pred['x2'],
        'y2': pred['y2']
    }
    pred_class = pred['class_id']
    for idx, gt in enumerate(ground_truths):
        if gt_flags[idx]:
            continue # Skip already matched ground truths
        gt_box = {
            'x1': gt['x1'],
            'y1': gt['y1'],
            'x2': gt['x2'],
            'y2': gt['y2']
        }
        gt_class = gt['class_id']
        iou = calculate_iou(pred_box, gt_box)
        if iou >= iou_threshold and pred_class == gt_class:
            matched += 1
            gt_flags[idx] = True
            break # Move to next prediction

precision = matched / total_pred if total_pred > 0 else 0
recall = matched / total_gt if total_gt > 0 else 0

return {
    'matched': matched,
    'total_ground_truths': total_gt,
    'total_predictions': total_pred,
    'precision': precision,
    'recall': recall
}

```

```

[ ]: # Function to process an image
def process_image(image_path, model, label_path = None, scale_factor=1.5,
                                ↵mode='testing', detailed=True):
    image = cv2.imread(image_path)
    if image is None:
        print(f"Failed to load image at {image_path}")
        return
    orig_h, orig_w = image.shape[:2]

```

```

image = cv2.resize(image, None, fx=scale_factor, fy=scale_factor)
new_h, new_w = image.shape[:2]
scale_x = new_w / orig_w
scale_y = new_h / orig_h

# Initialize variables to be returned
results_dict = {}
final_image = None

if mode == 'testing':
    if label_path is None:
        print("Error: 'label_path' is required in 'testing' mode.")
        return
    # Load ground truth labels and convert labels to absolute coordinates
    labels = load_label(label_path)
    ground_truths = []
    for label in labels:
        class_id = label['class_id']
        x_center = label['x_center'] * orig_w
        y_center = label['y_center'] * orig_h
        width = label['width'] * orig_w
        height = label['height'] * orig_h
        x1 = (x_center - width / 2) * scale_x
        y1 = (y_center - height / 2) * scale_y
        x2 = (x_center + width / 2) * scale_x
        y2 = (y_center + height / 2) * scale_y
        ground_truths.append({
            'class_id': class_id,
            'x1': x1,
            'y1': y1,
            'x2': x2,
            'y2': y2,
            'width': width * scale_x,
            'height': height * scale_y
        })
    # Get predictions
    predictions = predict_image(image_path, model)
    for pred in predictions:
        pred['x1'] *= scale_x
        pred['y1'] *= scale_y
        pred['x2'] *= scale_x
        pred['y2'] *= scale_y
        pred['width'] *= scale_x
        pred['height'] *= scale_y

# Draw predictions on image

```

```

image_pred = image.copy()
image_pred = draw_predictions_on_image(image_pred, predictions)

if mode == 'testing':
    # Draw ground truth labels on image
    image_gt = image.copy()
    image_gt = draw_labels(image_gt, labels)

    # Display the images side by side
    plt.figure(figsize=(20, 10))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image_gt, cv2.COLOR_BGR2RGB))
    plt.title('Ground Truth')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(image_pred, cv2.COLOR_BGR2RGB))
    plt.title('Model Predictions')
    plt.axis('off')

    plt.show()

if detailed:
    # Convert ground truths and predictions to DataFrames
    gt_df = pd.DataFrame(ground_truths)
    pred_df = pd.DataFrame(predictions)

    print("\nGround Truth Bounding Boxes:")
    display(gt_df)
    print("\nPredicted Bounding Boxes:")
    display(pred_df)
    accuracy = calculate_accuracy(ground_truths, predictions)
    print("\nAccuracy Metrics:")
    print(f"Matched Boxes: {accuracy['matched']}")
    print(f"Total Ground Truth Boxes: {accuracy['total_ground_truths']}")
    print(f"Total Predicted Boxes: {accuracy['total_predictions']}")
    print(f"Precision: {accuracy['precision']:.2f}")
    print(f"Recall: {accuracy['recall']:.2f}")

    results_dict = {
        'ground_truths': ground_truths,
        'predictions': predictions,
        'accuracy': accuracy
    }

final_image = image_pred

```

```

    else:
        # In 'prod' mode, only display the prediction image. This is to remove clutter
        plt.figure(figsize=(10, 10))
        plt.imshow(cv2.cvtColor(image_pred, cv2.COLOR_BGR2RGB))
        plt.title('Model Predictions')
        plt.axis('off')
        plt.show()

    if detailed:
        results_dict = {
            'predictions': predictions
        }

    final_image = image_pred

    if detailed:
        return final_image, results_dict
    else:
        return final_image, None

```

```

[ ]: # This function is for checking how well the model did compared to the ground
def random_check(dataset_df, data_path, model, scale_factor=1.5, mode='testing', detailed=True):

    if detailed:
        print(f"Running in '{mode}' mode.")
    if dataset_df.empty:
        print("Dataset is empty. Cannot perform random check.")
        return
    sample_entry = dataset_df.sample(n=1).iloc[0]
    image_path = os.path.join(data_path, sample_entry['image_path'])
    if mode == 'testing':
        label_path = os.path.join(data_path, sample_entry['label_path'])
    else:
        label_path = None

    if detailed:
        print(f"Randomly selected image: {image_path}")
        if label_path:
            print(f"Corresponding label file: {label_path}")
    results = process_image(
        image_path=image_path,
        model=model,
        label_path=label_path,
        scale_factor=scale_factor,
        mode=mode,

```

```

    detailed=detailed
)
return results

```

Random visual check for YOLO 8

[]: random_check(dataset_df, data_path, model)

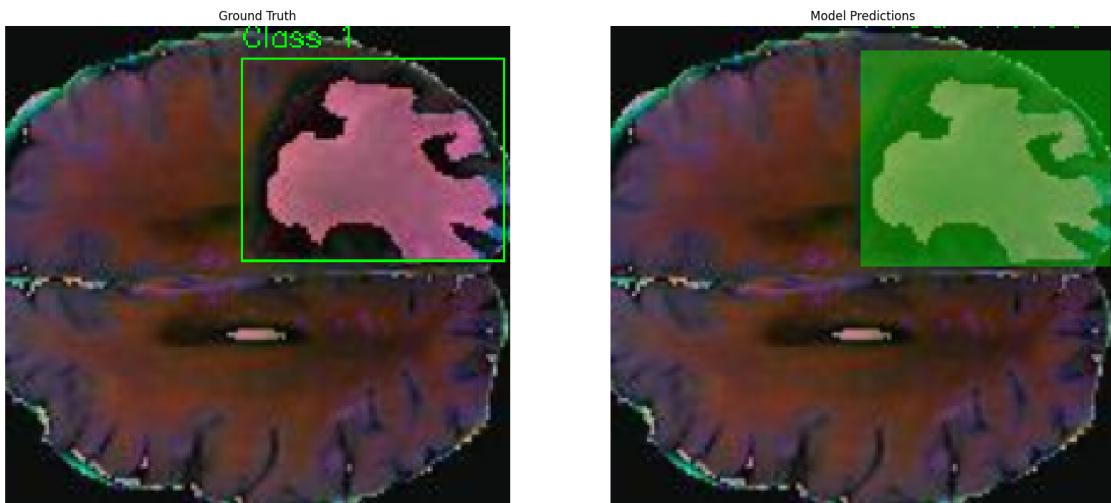
Running in 'testing' mode.

Randomly selected image: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/volume_132_slice_104.jpg.rf.983aab91f58847949a7f580dc0fffc0.jpg

Corresponding label file: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/labels/volume_132_slice_104.jpg.rf.983aab91f58847949a7f580dc0fffc0.txt

image 1/1 /Users/zaina/Desktop/Masters/AI-521-Final/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/volume_132_slice_104.jpg.rf.983aab91f58847949a7f580dc0fffc0.jpg: 608x640 1 label1, 48.7ms

Speed: 2.7ms preprocess, 48.7ms inference, 0.4ms postprocess per image at shape (1, 3, 608, 640)



Ground Truth Bounding Boxes:

	class_id	x1	y1	x2	y2	width	height
0	1	97.266187	13.5	205.007194	96.0	107.741007	82.5

Predicted Bounding Boxes:

	class_id	confidence	x1	y1	x2	y2	\
0	1	0.903426	103.899351	10.638992	205.176731	98.250961	

```
    width      height
0  101.277381  87.611969
```

Accuracy Metrics:

Matched Boxes: 1

Total Ground Truth Boxes: 1

Total Predicted Boxes: 1

Precision: 1.00

Recall: 1.00

```
[ ]: (array([[[13, 14, 12],
               [13, 14, 12],
               [13, 14, 12],
               ...,
               [13, 14, 12],
               [13, 14, 12],
               [13, 14, 12]],
              [[13, 14, 12],
               [13, 14, 12],
               [13, 14, 12],
               ...,
               [13, 14, 12],
               [13, 14, 12],
               [13, 14, 12]],
              [[13, 14, 12],
               [13, 14, 12],
               [13, 14, 12],
               ...,
               [13, 14, 12],
               [13, 14, 12],
               [13, 14, 12]],
              ...,
              [[13, 14, 12],
               [13, 14, 12],
               [13, 14, 12],
               ...,
               [13, 14, 12],
               [13, 14, 12],
               [13, 14, 12]]],
```

```

    ...,
    [13, 14, 12],
    [13, 14, 12],
    [13, 14, 12]],

    [[13, 14, 12],
     [13, 14, 12],
     [13, 14, 12],
     ...,
     [13, 14, 12],
     [13, 14, 12],
     [13, 14, 12]]], dtype=uint8),
{'ground_truths': {'class_id': 1,
  'x1': 97.26618705035972,
  'y1': 13.5,
  'x2': 205.0071942446043,
  'y2': 96.0,
  'width': 107.7410071942446,
  'height': 82.5}],
'predictions': {'class_id': 1,
  'confidence': 0.9034256339073181,
  'x1': 103.89935083183454,
  'y1': 10.638992071151733,
  'x2': 205.17673146638938,
  'y2': 98.25096130371094,
  'width': 101.27738063455486,
  'height': 87.6119692325592}],
'accuracy': {'matched': 1,
  'total_ground_truths': 1,
  'total_predictions': 1,
  'precision': 1.0,
  'recall': 1.0}})

```

[33]: random_check(dataset_df, data_path, model)

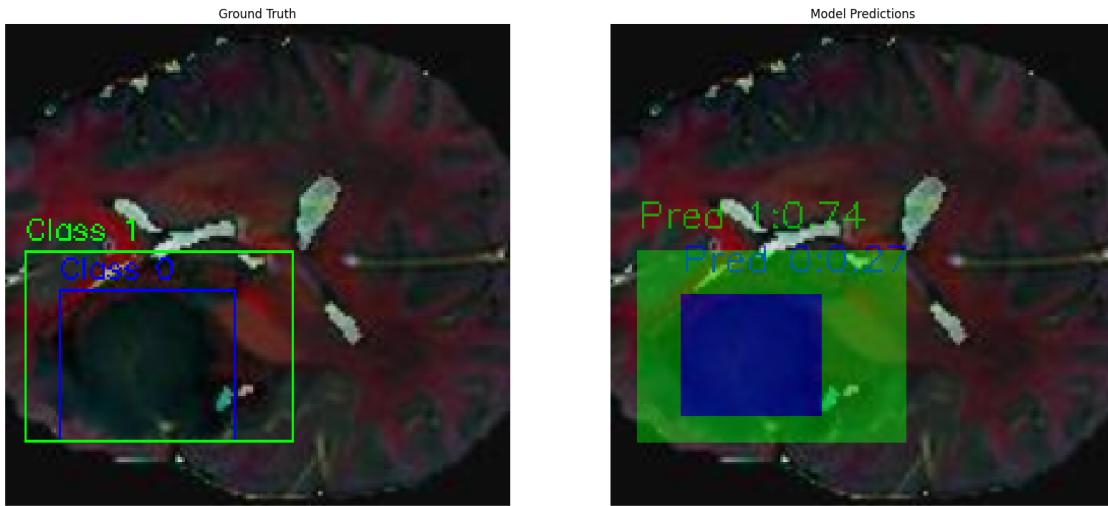
Running in 'testing' mode.

Randomly selected image: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/volume_335_slice_81.jpg.rf.ebc89d6b9d3abb97de8b222d439548f1.jpg

Corresponding label file: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/labels/volume_335_slice_81.jpg.rf.ebc89d6b9d3abb97de8b222d439548f1.txt

image 1/1 /Users/zaina/Desktop/Masters/AI-521-Final/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/volume_335_slice_81.jpg.rf.ebc89d6b9d3abb97de8b222d439548f1.jpg: 608x640 1 label0, 1 label1, 75.2ms

Speed: 3.6ms preprocess, 75.2ms inference, 0.7ms postprocess per image at shape (1, 3, 608, 640)



Ground Truth Bounding Boxes:

	class_id	x1	y1	x2	y2	width	height
0	0	22.446043	109.5	94.273381	171.0	71.827338	61.5
1	1	8.978417	93.0	118.215827	171.0	109.237410	78.0

Predicted Bounding Boxes:

	class_id	confidence	x1	y1	x2	y2	\
0	1	0.740184	11.929348	93.75753	121.734221	171.944515	
1	0	0.267801	29.311105	111.73455	86.462505	160.482113	
width height							
0	109.804873	78.186985					
1	57.151400	48.747562					

Accuracy Metrics:

Matched Boxes: 2

Total Ground Truth Boxes: 2

Total Predicted Boxes: 2

Precision: 1.00

Recall: 1.00

```
[33]: (array([[13, 13, 13],
       [13, 13, 13],
       [13, 13, 13],
       ...,
       [13, 13, 13],
       [13, 13, 13],
```

```

[13, 13, 13]],

[[13, 13, 13],
 [13, 13, 13],
 [13, 13, 13],
 ...,
 [13, 13, 13],
 [13, 13, 13],
 [13, 13, 13]],

[[13, 13, 13],
 [13, 13, 13],
 [13, 13, 13],
 ...,
 [13, 13, 13],
 [13, 13, 13],
 [13, 13, 13]],

...,

[[13, 13, 13],
 [13, 13, 13],
 [13, 13, 13],
 ...,
 [13, 13, 13],
 [13, 13, 13],
 [13, 13, 13]],

[[13, 13, 13],
 [13, 13, 13],
 [13, 13, 13],
 ...,
 [13, 13, 13],
 [13, 13, 13],
 [13, 13, 13]],

[[13, 13, 13],
 [13, 13, 13],
 [13, 13, 13],
 ...,
 [13, 13, 13],
 [13, 13, 13],
 [13, 13, 13]]], dtype=uint8),
{'ground_truths': [{ 'class_id': 0,
 'x1': 22.446043165467625,
 'y1': 109.5,
 'x2': 94.27338129496403,

```

```

'y2': 171.0,
'width': 71.82733812949641,
'height': 61.5},
{'class_id': 1,
'x1': 8.978417266187051,
'y1': 93.0,
'x2': 118.2158273381295,
'y2': 171.0,
'width': 109.23741007194245,
'height': 78.0}],
'predictions': [ {'class_id': 1,
'confidence': 0.7401844263076782,
'x1': 11.929348184050417,
'y1': 93.75753021240234,
'x2': 121.73422131435477,
'y2': 171.94451522827148,
'width': 109.80487313030436,
'height': 78.18698501586914},
{'class_id': 0,
'confidence': 0.26780110597610474,
'x1': 29.3111047538922,
'y1': 111.73455047607422,
'x2': 86.46250465447955,
'y2': 160.48211288452148,
'width': 57.15139990058734,
'height': 48.747562408447266}],
'accuracy': {'matched': 2,
'total_ground_truths': 2,
'total_predictions': 2,
'precision': 1.0,
'recall': 1.0}])

```

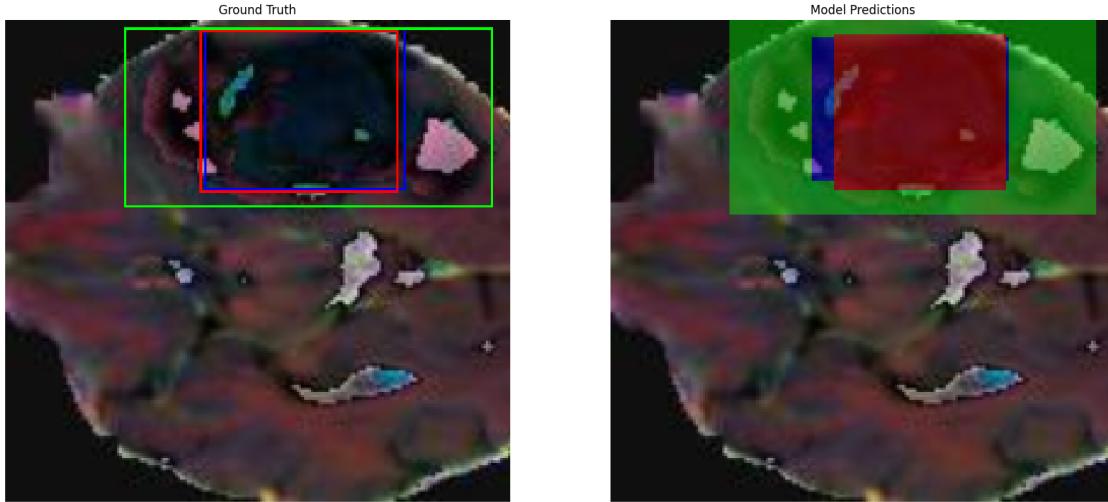
[38]: random_check(dataset_df, data_path, model)

```

Running in 'testing' mode.
Randomly selected image: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/test/images/volume_343_slice_59.jpg.rf.d83b460cbcd039d1b2b8e509199eb4a2.jpg
Corresponding label file: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/test/labels/volume_343_slice_59.jpg.rf.d83b460cbcd039d1b2b8e509199eb4a2.txt

image 1/1 /Users/zaina/Desktop/Masters/AI-521-Final/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/test/images/volume_343_slice_59.jpg.rf.d83b460cbcd039d1b2b8e509199eb4a2.jpg: 608x640 1 label0, 1 label1, 1 label2, 53.5ms
Speed: 2.2ms preprocess, 53.5ms inference, 0.3ms postprocess per image at shape (1, 3, 608, 640)

```



Ground Truth Bounding Boxes:

	class_id	x1	y1	x2	y2	width	height
0	0	82.302158	3.0	164.604317	69.0	82.302158	66.0
1	1	49.381295	3.0	200.517986	76.5	151.136691	73.5
2	2	80.805755	4.5	161.611511	70.5	80.805755	66.0

Predicted Bounding Boxes:

	class_id	confidence	x1	y1	x2	y2	\
0	1	0.743229	49.301110	0.978432	199.327321	79.022907	
1	0	0.415913	83.707513	7.574136	163.843260	65.148314	
2	2	0.385121	92.647434	6.996442	162.499304	69.150536	
							width height
0	150.026210	78.044475					
1	80.135747	57.574178					
2	69.851870	62.154094					

Accuracy Metrics:

Matched Boxes: 3

Total Ground Truth Boxes: 3

Total Predicted Boxes: 3

Precision: 1.00

Recall: 1.00

```
[38]: (array([[15, 15, 15],
       [15, 15, 15],
       [15, 15, 15],
```

```
...,
[[15, 15, 15],
[15, 15, 15],
[15, 15, 15]],

[[15, 15, 15],
[15, 15, 15],
[15, 15, 15],
...,
[15, 15, 15],
[15, 15, 15],
[15, 15, 15]],

[[15, 15, 15],
[15, 15, 15],
[15, 15, 15],
...,
[15, 15, 15],
[15, 15, 15],
[15, 15, 15]],

...,

[[15, 15, 15],
[15, 15, 15],
[15, 15, 15],
...,
[15, 15, 15],
[15, 15, 15],
[15, 15, 15]],

[[15, 15, 15],
[15, 15, 15],
[15, 15, 15],
...,
[15, 15, 15],
[15, 15, 15],
[15, 15, 15]],

[[15, 15, 15],
[15, 15, 15],
[15, 15, 15],
...,
[15, 15, 15],
[15, 15, 15],
[15, 15, 15]],

[['ground_truths': {'class_id': 0,
```

```

'x1': 82.3021582733813,
'y1': 3.0,
'x2': 164.6043165467626,
'y2': 69.0,
'width': 82.3021582733813,
'height': 66.0},
{'class_id': 1,
'x1': 49.381294964028775,
'y1': 3.0,
'x2': 200.5179856115108,
'y2': 76.5,
'width': 151.13669064748203,
'height': 73.5},
{'class_id': 2,
'x1': 80.80575539568345,
'y1': 4.5,
'x2': 161.6115107913669,
'y2': 70.5,
'width': 80.80575539568346,
'height': 66.0}],
'predictions': [ {'class_id': 1,
'confidence': 0.7432289123535156,
'x1': 49.3011101372808,
'y1': 0.9784323871135712,
'x2': 199.32732056542267,
'y2': 79.02290725708008,
'width': 150.02621042814187,
'height': 78.0444748699665},
{'class_id': 0,
'confidence': 0.4159131944179535,
'x1': 83.70751338382419,
'y1': 7.5741355419158936,
'x2': 163.84325996234261,
'y2': 65.14831352233887,
'width': 80.13574657851844,
'height': 57.574177980422974},
{'class_id': 2,
'confidence': 0.38512054085731506,
'x1': 92.6474337131857,
'y1': 6.996442079544067,
'x2': 162.49930358447617,
'y2': 69.1505355834961,
'width': 69.85186987129048,
'height': 62.154093503952026}],
'accuracy': {'matched': 3,
'total_ground_truths': 3,
'total_predictions': 3,

```

```
'precision': 1.0,  
'recall': 1.0}})
```

Random visual check for YOLO 11

```
[39]: model_path = model_paths.get("YOLOv11n")  
yolov11n_model = YOLO(model_path)
```

```
[41]: random_check(dataset_df, data_path, yolov11n_model)
```

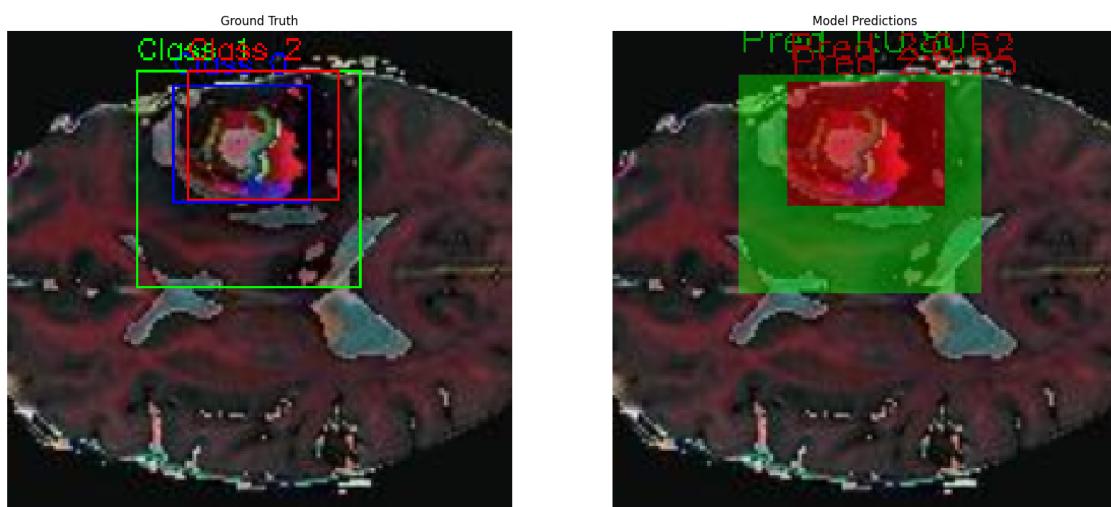
Running in 'testing' mode.

Randomly selected image: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/valid/images/volume_187_slice_80.jpg.rf.d230dfc4b95e4e31521f355c7b9e4c80.jpg

Corresponding label file: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/valid/labels/volume_187_slice_80.jpg.rf.d230dfc4b95e4e31521f355c7b9e4c80.txt

image 1/1 /Users/zaina/Desktop/Masters/AI-521-Final/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/valid/images/volume_187_slice_80.jpg.rf.d230dfc4b95e4e31521f355c7b9e4c80.jpg: 608x640 1 label1, 2 label2s, 59.4ms

Speed: 1.8ms preprocess, 59.4ms inference, 0.6ms postprocess per image at shape (1, 3, 608, 640)



Ground Truth Bounding Boxes:

	class_id	x1	y1	x2	y2	width	height
0	0	68.834532	22.5	124.201439	70.5	55.366906	48.0
1	1	53.870504	16.5	145.151079	105.0	91.280576	88.5
2	2	74.820144	16.5	136.172662	69.0	61.352518	52.5

Predicted Bounding Boxes:

	class_id	confidence	x1	y1	x2	y2	\
0	1	0.795485	52.945047	18.250212	151.439346	107.036282	
1	2	0.620239	72.524610	21.229875	136.042592	71.111240	
2	2	0.251206	73.088238	27.909963	122.791711	69.268204	

	width	height
0	98.494299	88.786069
1	63.517982	49.881366
2	49.703473	41.358241

Accuracy Metrics:

Matched Boxes: 2

Total Ground Truth Boxes: 3

Total Predicted Boxes: 3

Precision: 0.67

Recall: 0.67

```
[41]: (array([[12, 13, 11],  
           [12, 13, 11],  
           [12, 13, 11],  
           ...,  
           [12, 13, 11],  
           [12, 13, 11],  
           [12, 13, 11]],  
  
       [[12, 13, 11],  
        [12, 13, 11],  
        [12, 13, 11],  
        ...,  
        [12, 13, 11],  
        [12, 13, 11],  
        [12, 13, 11]],  
  
       [[12, 13, 11],  
        [12, 13, 11],  
        [12, 13, 11],  
        ...,  
        [12, 13, 11],  
        [12, 13, 11],  
        [12, 13, 11]],  
  
       ...,  
  
       [[12, 13, 11],  
        [12, 13, 11],  
        [12, 13, 11],  
        ...]),
```

```

[12, 13, 11],
[12, 13, 11],
[12, 13, 11]],

[[12, 13, 11],
[12, 13, 11],
[12, 13, 11],
...,
[12, 13, 11],
[12, 13, 11],
[12, 13, 11]],

[[12, 13, 11],
[12, 13, 11],
[12, 13, 11],
...,
[12, 13, 11],
[12, 13, 11],
[12, 13, 11]]], dtype=uint8),
{'ground_truths': [ {'class_id': 0,
  'x1': 68.83453237410072,
  'y1': 22.5,
  'x2': 124.20143884892086,
  'y2': 70.5,
  'width': 55.36690647482015,
  'height': 48.0},
  {'class_id': 1,
  'x1': 53.8705035971223,
  'y1': 16.5,
  'x2': 145.15107913669064,
  'y2': 105.0,
  'width': 91.28057553956835,
  'height': 88.5},
  {'class_id': 2,
  'x1': 74.82014388489209,
  'y1': 16.5,
  'x2': 136.1726618705036,
  'y2': 69.0,
  'width': 61.35251798561151,
  'height': 52.5}],
 'predictions': [ {'class_id': 1,
  'confidence': 0.7954846024513245,
  'x1': 52.94504728248651,
  'y1': 18.2502121925354,
  'x2': 151.43934598415018,
  'y2': 107.03628158569336,
  'width': 98.49429870166367,

```

```

'height': 88.78606939315796},
{'class_id': 2,
'confidence': 0.6202394366264343,
'x1': 72.52460990192222,
'y1': 21.22987461090088,
'x2': 136.04259200061827,
'y2': 71.11124038696289,
'width': 63.517982098696045,
'height': 49.88136577606201},
{'class_id': 2,
'confidence': 0.25120633840560913,
'x1': 73.08823839366008,
'y1': 27.90996265411377,
'x2': 122.79171116232014,
'y2': 69.26820373535156,
'width': 49.70347276866007,
'height': 41.35824108123779}],
'accuracy': {'matched': 2,
'total_ground_truths': 3,
'total_predictions': 3,
'precision': 0.6666666666666666,
'recall': 0.6666666666666666}})

```

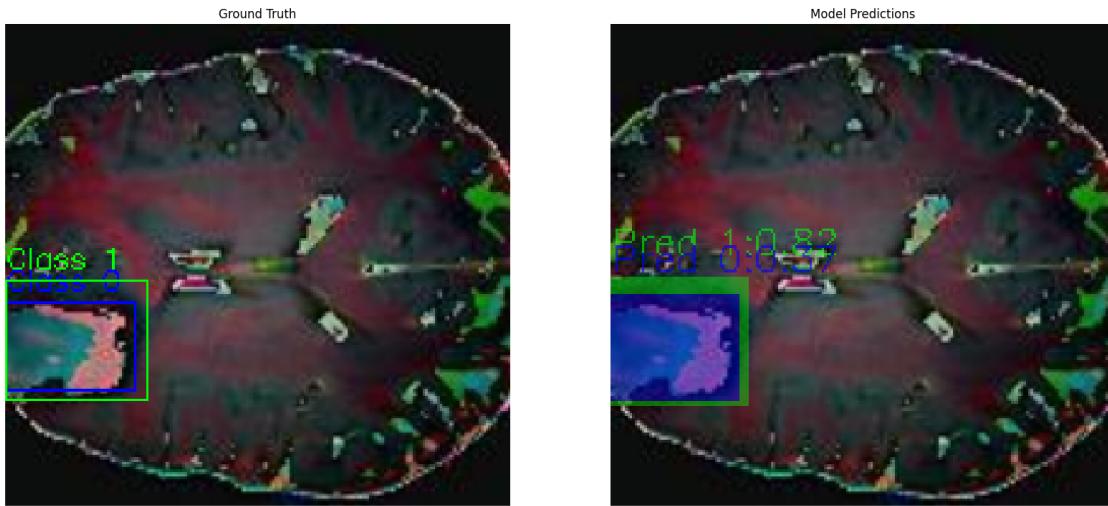
[47]: random_check(dataset_df, data_path, yolov11n_model)

```

Running in 'testing' mode.
Randomly selected image: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov
9/test/images/volume_269_slice_83.jpg.rf.f58ed2754d532a521da29ad0aeb96acf.jpg
Corresponding label file: data/BrainTumorDetectionYolov9/BrainTumorDetectionYolo
v9/test/labels/volume_269_slice_83.jpg.rf.f58ed2754d532a521da29ad0aeb96acf.txt

image 1/1 /Users/zaina/Desktop/Masters/AI-521-Final/data/BrainTumorDetectionYol
ov9/BrainTumorDetectionYolov9/test/images/volume_269_slice_83.jpg.rf.f58ed2754d5
32a521da29ad0aeb96acf.jpg: 608x640 1 label0, 1 label1, 50.1ms
Speed: 2.2ms preprocess, 50.1ms inference, 0.5ms postprocess per image at shape
(1, 3, 608, 640)

```



Ground Truth Bounding Boxes:

	class_id	x1	y1	x2	y2	width	height
0	0	0.0	114.0	53.870504	150.0	53.870504	36.0
1	1	0.0	105.0	58.359712	154.5	58.359712	49.5

Predicted Bounding Boxes:

	class_id	confidence	x1	y1	x2	y2	\
0	1	0.82101	0.676556	104.845207	56.927043	156.466084	
1	0	0.36670	0.160997	111.251690	52.619656	154.875629	
width height							
0	56.250487	51.620876					
1	52.458658	43.623940					

Accuracy Metrics:

Matched Boxes: 2

Total Ground Truth Boxes: 2

Total Predicted Boxes: 2

Precision: 1.00

Recall: 1.00

```
[47]: (array([[13, 14, 12],
       [13, 14, 12],
       [13, 14, 12],
       ...,
       [13, 14, 12],
       [13, 14, 12],
```

```
[13, 14, 12]],

[[13, 14, 12],
 [13, 14, 12],
 [13, 14, 12],
 ...,
 [13, 14, 12],
 [13, 14, 12],
 [13, 14, 12]],

[[13, 14, 12],
 [13, 14, 12],
 [13, 14, 12],
 ...,
 [13, 14, 12],
 [13, 14, 12],
 [13, 14, 12]],

...,

[[13, 14, 12],
 [13, 14, 12],
 [13, 14, 12],
 ...,
 [13, 14, 12],
 [13, 14, 12],
 [13, 14, 12]],

[[13, 14, 12],
 [13, 14, 12],
 [13, 14, 12],
 ...,
 [13, 14, 12],
 [13, 14, 12],
 [13, 14, 12]],

[[13, 14, 12],
 [13, 14, 12],
 [13, 14, 12],
 ...,
 [13, 14, 12],
 [13, 14, 12],
 [13, 14, 12]]], dtype=uint8),
{'ground_truths': [{ 'class_id': 0,
 'x1': 0.0,
 'y1': 114.0,
 'x2': 53.8705035971223,
```

```

'y2': 150.0,
'width': 53.8705035971223,
'height': 36.0},
{'class_id': 1,
'x1': 0.0,
'y1': 105.0,
'x2': 58.35971223021583,
'y2': 154.5,
'width': 58.35971223021583,
'height': 49.5}],
'predictions': [ {'class_id': 1,
'confidence': 0.8210102319717407,
'x1': 0.676555997176136,
'y1': 104.84520721435547,
'x2': 56.92704278959645,
'y2': 156.46608352661133,
'width': 56.25048679242031,
'height': 51.62087631225586},
{'class_id': 0,
'confidence': 0.3667004704475403,
'x1': 0.16099737959800006,
'y1': 111.25168991088867,
'x2': 52.619655691462455,
'y2': 154.87562942504883,
'width': 52.458658311864454,
'height': 43.623939514160156}],
'accuracy': {'matched': 2,
'total_ground_truths': 2,
'total_predictions': 2,
'precision': 1.0,
'recall': 1.0}])

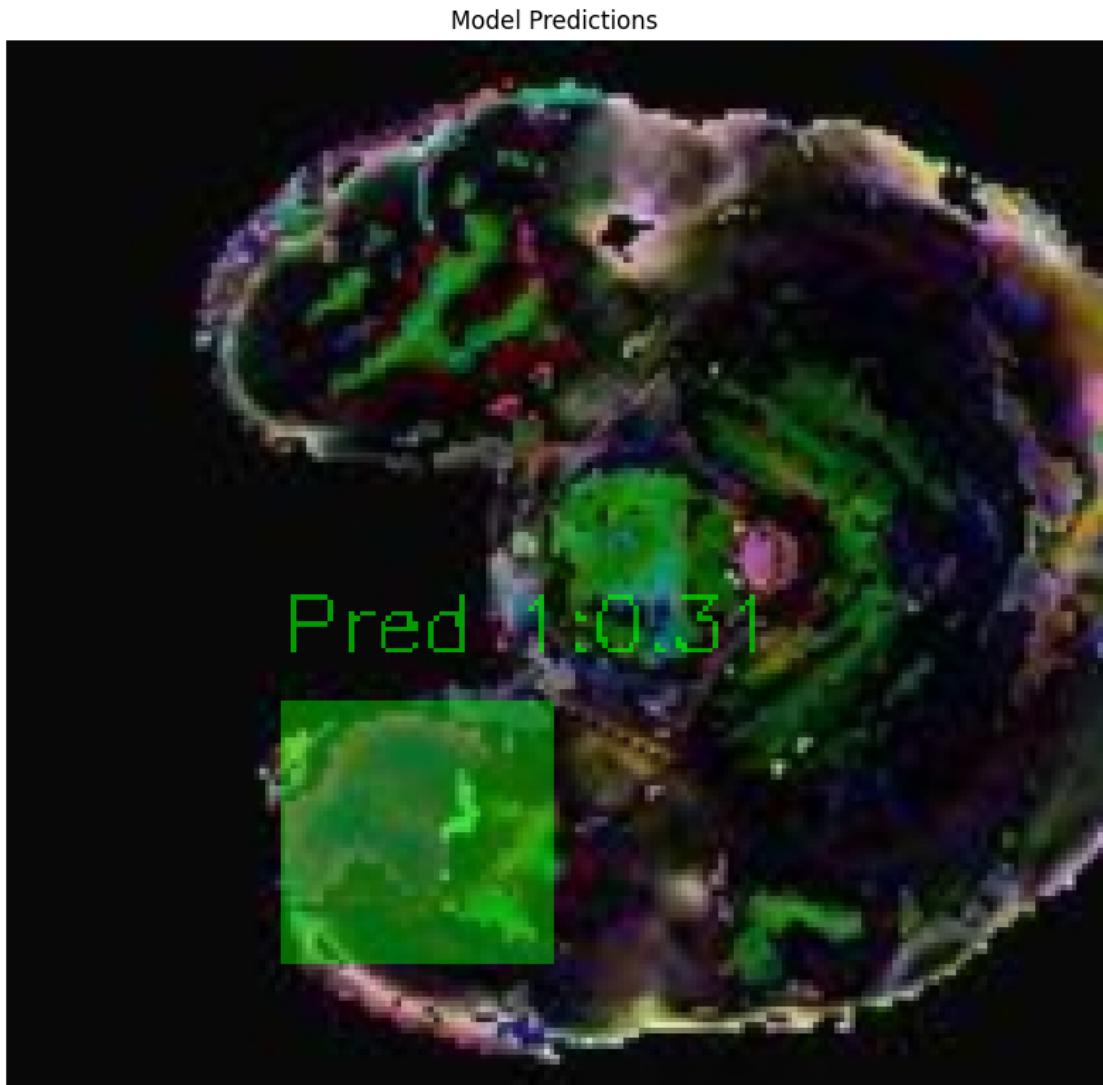
```

```
[58]: # Run random check in 'prod' mode
results = random_check(dataset_df, data_path, model, mode='prod', detailed=False)
```

```

image 1/1 /Users/zaina/Desktop/Masters/AI-521-Final/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/valid/images/volume_330_slice_44.jpg.rf.bb01ac724e37743f1bbb931c01ed06e3.jpg: 608x640 1 label1, 38.8ms
Speed: 1.4ms preprocess, 38.8ms inference, 0.3ms postprocess per image at shape (1, 3, 608, 640)

```



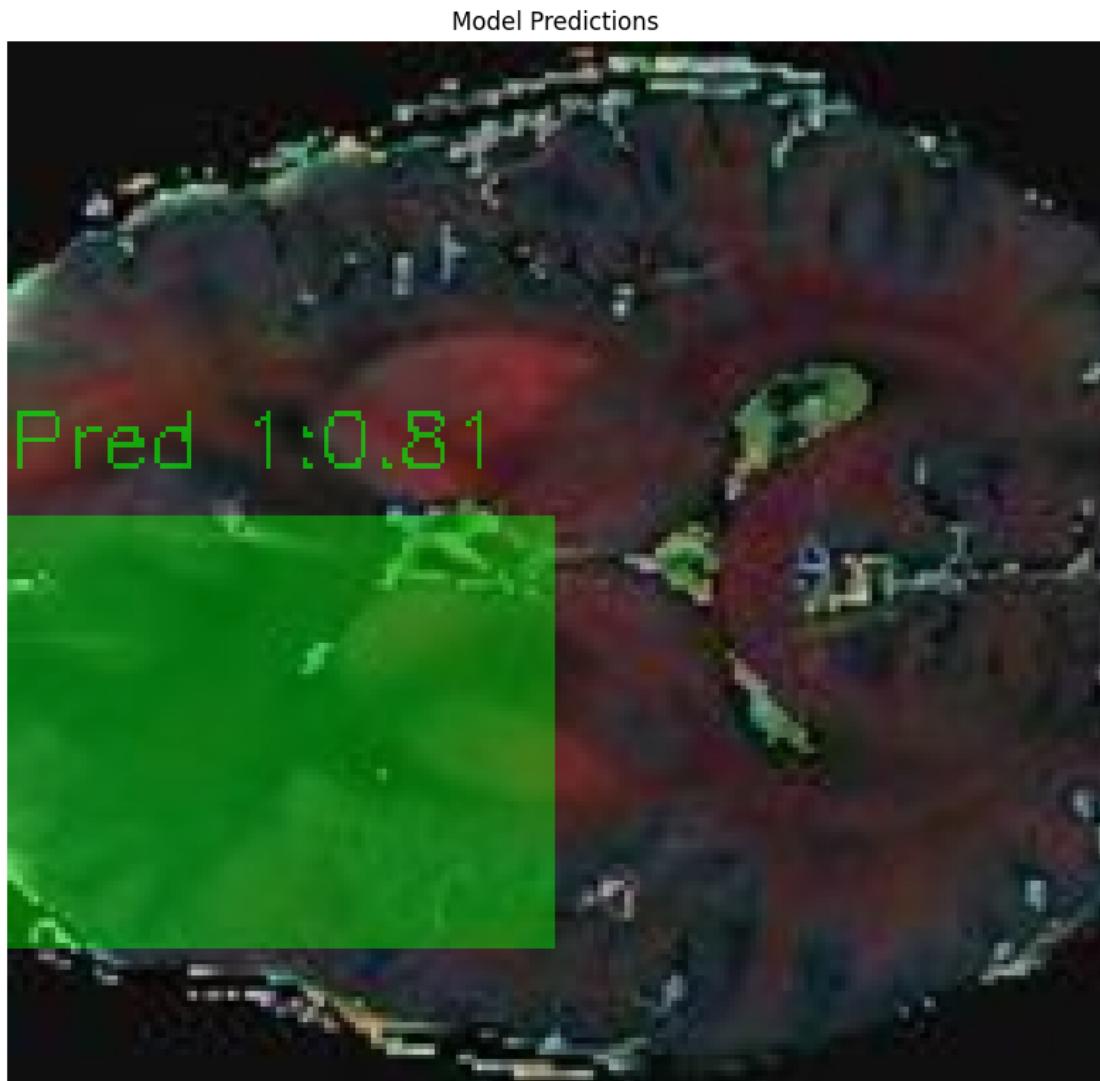
1.5.2 API information

Use the following function for API. This can be used to either expand the existing functionality to attach to the UI dashboard

Methology 1: Getting the image. Following is one of two ways to grab results from the our system:

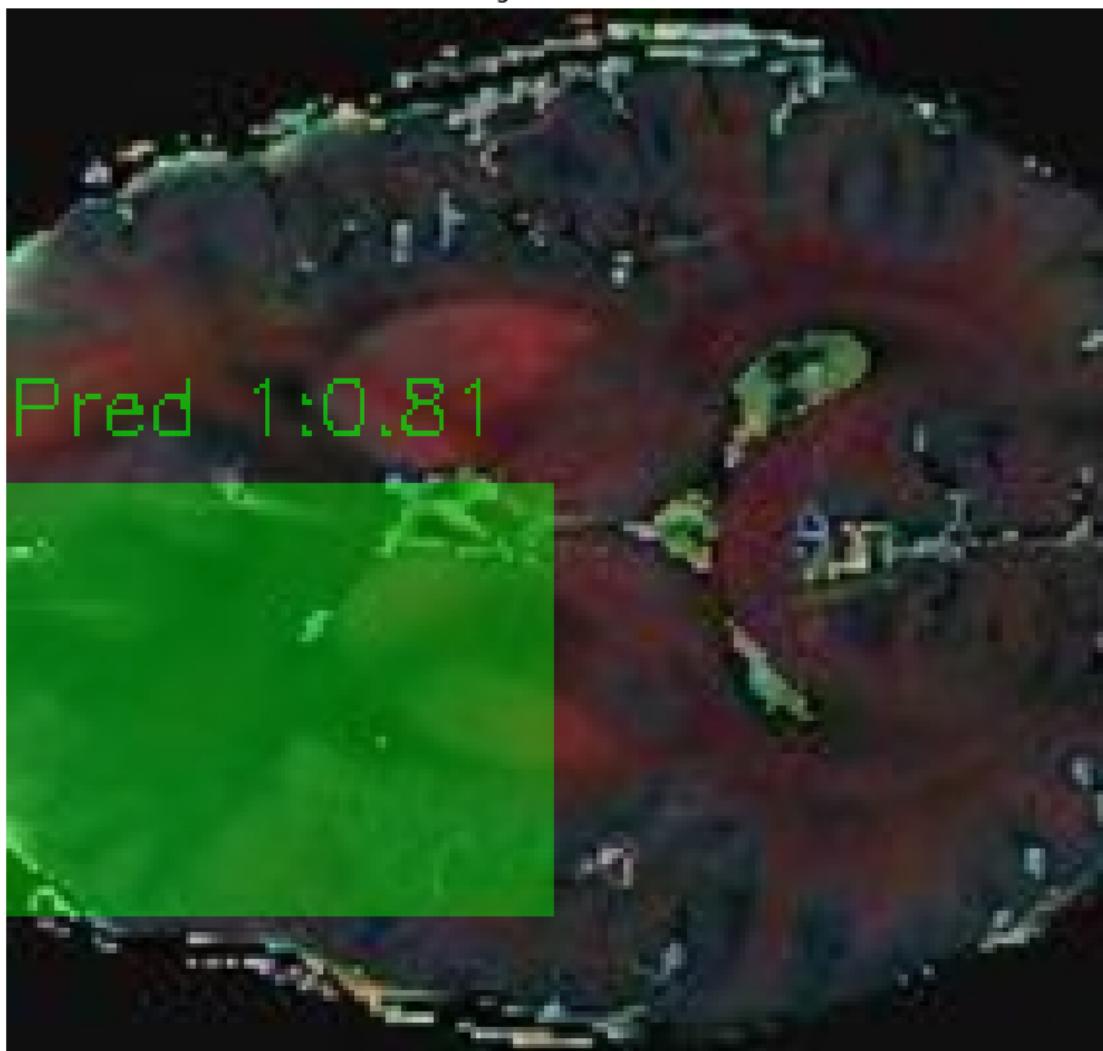
```
[59]: image_path = "data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/  
↪images/volume_301_slice_74.jpg.rf.2537f3cf0f4b5c51208f4c5e638b818a.jpg"  
final_image, _ = process_image(image_path, yolov11n_model, mode="prod",  
↪detailed=False)
```

```
image 1/1 /Users/zaina/Desktop/Masters/AAI-521-Final/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/volume_301_slice_74.jpg.rf.2537f3cf0f4b5c51208f4c5e638b818a.jpg: 608x640 1 label1, 50.5ms
Speed: 2.5ms preprocess, 50.5ms inference, 0.4ms postprocess per image at shape
(1, 3, 608, 640)
```



```
[60]: # The `final_image` can now be used within the dashboard.
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(final_image, cv2.COLOR_BGR2RGB))
plt.title("Final Image with Predictions")
plt.axis('off')
plt.show()
```

Final Image with Predictions



Methology 2: Getting the data Rather than getting an image, you can directly get the prediction results in a dictionary

```
[61]: predictions = predict_image(image_path, yolov11n_model)
```

```
image 1/1 /Users/zaina/Desktop/Masters/AI-521-Final/data/BrainTumorDetectionYolov9/BrainTumorDetectionYolov9/train/images/volume_301_slice_74.jpg.rf.2537f3cf0f4b5c51208f4c5e638b818a.jpg: 608x640 1 label1, 69.6ms
Speed: 2.8ms preprocess, 69.6ms inference, 0.5ms postprocess per image at shape (1, 3, 608, 640)
```

```
[62]: display(predictions)
```

```
[{'class_id': 1,
 'confidence': 0.8074972629547119,
 'x1': 0.1095481812953949,
 'y1': 60.11636734008789,
 'x2': 69.34325408935547,
 'y2': 114.26077270507812,
 'width': 69.23370590806007,
 'height': 54.144405364990234}]
```

1.6 Code for UI

The following code represents the standalone Python script for the user interface (UI). Rather than being directly integrated with the API, this UI is designed to function independently. This approach allows users to access the dashboard and its features without needing to run the full Jupyter notebook.

Note: The script, named `ScanInsight-UI.py`, is optimized for standalone execution and may not be compatible with Jupyter or Mac environments. Please refer to the `README` file for detailed instructions on how to run the script.

```
[ ]: import os
import cv2
import tkinter as tk
from tkinter import filedialog, Text, messagebox, ttk, Scrollbar
from PIL import Image, ImageTk
import numpy as np
import torch
import torchvision.transforms as transforms
from ultralytics import YOLO

class ImageAnalyzerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Tumor Detection Model")
        self.root.geometry("800x800")

        # Intro disclaimer in the main window
        self.intro_label = tk.Label(self.root, text="This is a demo version of\u
        ↵automated tumor classification, created as a computer vision project by Zain\u
        ↵Ali, Halladay Kinsey, and Liam Richardson. By clicking accept, you\u
        ↵acknowledge that this is for educational purposes only. Please consult the\u
        ↵performance report below to understand the current capabilities of the model.\u
        ↵", wraplength=700, justify="left")
        self.intro_label.pack(pady=20)

        # Metrics disclaimer image
        self.metrics_image_path = "results/yolov11n_brain_tumor_detection_v1/\u
        ↵detect_yolo11/metrics_disclaimer.jpg"
```

```

    if os.path.exists(self.metrics_image_path):
        image = Image.open(self.metrics_image_path)
        image.thumbnail((800, 600), Image.LANCZOS) # Resize the image
    ↵while maintaining aspect ratio
        self.metrics_image = ImageTk.PhotoImage(image)
        self.metrics_image_label = tk.Label(self.root, image=self.
    ↵metrics_image)
            self.metrics_image_label.pack(pady=10)
    else:
        messagebox.showwarning("Warning", "results/
    ↵yolov1n_brain_tumor_detection_v1/detect_yolo11/metrics_disclaimer.jpg not
    ↵found in the current directory.")
            # Accept and reject buttons
            self.accept_button = tk.Button(self.root, text="Accept", command=self.
    ↵initialize_main_window)
                self.accept_button.pack(side="left", padx=20, pady=10)

            self.reject_button = tk.Button(self.root, text="Reject", command=self.
    ↵root.quit)
                self.reject_button.pack(side="right", padx=20, pady=10)

            # Disable the main interface until accept is pressed
            self.interface_enabled = False

            # File path to save comments
            self.selected_file_path = None
            self.selected_volume_path = None

    def initialize_main_window(self):
        # Enable main interface components
        self.intro_label.pack_forget()
        self.accept_button.pack_forget()
        self.reject_button.pack_forget()
        self.metrics_image_label.pack_forget() # Ensure metrics image is also
    ↵hidden
        self.interface_enabled = True

        # Load YOLO model - set up model path selection
        self.load_model_selection()

        # Widgets for selecting file or volume
        self.selection_frame = tk.Frame(self.root)
        self.selection_frame.pack(pady=10)

            self.select_button = tk.Button(self.selection_frame, text="Option 1:
    ↵Select Image", command=self.select_image)

```

```

    self.select_button.pack(side="left", padx=10)

    self.select_volume_button = tk.Button(self.selection_frame, text="Option 2: Select Folder of Images", command=self.select_volume)
    self.select_volume_button.pack(side="left", padx=10)

    # Canvas to display image with a frame to hold it and the scrollbar
    self.image_frame = tk.Frame(self.root)
    self.image_frame.pack()
    self.image_canvas = tk.Label(self.image_frame)
    self.image_canvas.pack(side="left")

    # Scrollbar for scrolling through volume images (vertical)
    self.scrollbar = tk.Scale(self.image_frame, from_=0, to=0, orient=tk.VERTICAL, command=self.on_scroll)
    self.scrollbar.pack(side="right", fill="y", padx=10)
    self.scrollbar.pack_forget() # Hide initially

    # Result Label
    self.result_label = tk.Label(self.root, text="Result: None", font=("Helvetica", 16))
    self.result_label.pack(pady=10)

    # Comment Box
    self.comment_box = Text(self.root, height=10, width=70)
    self.comment_box.pack(pady=10)

    # Save Comment Button
    self.save_comment_button = tk.Button(self.root, text="Save Comment", command=self.save_comment)
    self.save_comment_button.pack(pady=5)

    # Save JSON Button
    self.save_json_button = tk.Button(self.root, text="Save All Comments to JSON", command=self.save_all_comments)
    self.save_json_button.pack(pady=5)

def load_model_selection(self):
    # Load model paths from model_paths.txt
    model_paths_file = os.path.join(os.getcwd(), 'model_paths.txt')
    if not os.path.exists(model_paths_file):
        messagebox.showerror("Error", "model_paths.txt file not found.")
        self.root.quit()
        return

    with open(model_paths_file, 'r') as file:
        model_paths = file.readlines()

```

```

# Parse model names and paths from file
self.models = {}
for line in model_paths:
    if ':' in line:
        model_name, model_path = line.split(':', 1)
        self.models[model_name.strip()] = model_path.strip()

# Add dropdown menu for selecting the model
self.model_selection_frame = tk.Frame(self.root)
self.model_selection_frame.pack(pady=10)

    self.model_selection_label = tk.Label(self.model_selection_frame, text="Select Model:")
    self.model_selection_label.pack(side="left", padx=5)

    self.model_var = tk.StringVar(self.root)
model_names = list(self.models.keys())
self.model_var.set(model_names[0]) # set the default value

    self.model_dropdown = ttk.Combobox(self.model_selection_frame, textvariable=self.model_var, values=model_names, state="readonly")
    self.model_dropdown.pack(side="left", padx=5)

# Load model button (moved to the right of the dropdown)
    self.load_model_button = tk.Button(self.model_selection_frame, text="Load Selected Model", command=self.load_model)
    self.load_model_button.pack(side="left", padx=5)

def load_model(self):
    # Get the selected model path from the dropdown
    selected_model_name = self.model_var.get()
    self.model_path = self.models.get(selected_model_name)

    if not os.path.exists(self.model_path):
        messagebox.showerror("Error", "Invalid model selected.")
        self.root.quit()
        return

    # Load YOLO model
    try:
        self.model = YOLO(self.model_path)
        messagebox.showinfo("Success", f"Model loaded from {self.model_path}")
    except Exception as e:
        messagebox.showerror("Error", f"Failed to load model: {str(e)}")

```

```

def select_image(self):
    # Ensure the interface is enabled
    if not self.interface_enabled:
        messagebox.showwarning("Warning", "Please accept the disclaimer\u2022")
    return

    # Open file dialog to select image
    file_path = filedialog.askopenfilename(initialdir=os.getcwd(),
                                            title="Select Image File",
                                            filetypes=(
                                                ("Image files", "*.jpg *.
                                                jpeg *.png *.bmp *.tiff"),
                                                ("All files", "*.*"))
                                            ))
    if file_path:
        # Load image using OpenCV
        image = cv2.imread(file_path)
        if image is not None:
            self.selected_file_path = file_path
            self.display_image(image)
            self.analyze_image(image)
        else:
            messagebox.showerror("Error", "Unable to read the selected\u2022
image.")

def select_volume(self):
    # Ensure the interface is enabled
    if not self.interface_enabled:
        messagebox.showwarning("Warning", "Please accept the disclaimer\u2022
first.")
    return

    # Open file dialog to select volume (folder of images)
    volume_path = filedialog.askdirectory(initialdir=os.getcwd(),\u2022
title="Select Volume Folder")
    if volume_path:
        self.selected_volume_path = volume_path
        self.volume_images = [os.path.join(volume_path, f) for f in os.
listdir(volume_path) if f.lower().endswith('.jpg', '.jpeg', '.png', '.bmp',\u2022
'.tiff')]]
        self.current_volume_index = 0

        if len(self.volume_images) > 0:
            self.scrollbar.config(to=len(self.volume_images) - 1)
            self.scrollbar.set(self.current_volume_index)

```

```

        self.scrollbar.pack(side="right", fill="y", padx=10) # Show
→scrollbar
        self.display_volume_image()
        self.analyze_volume_image()
    else:
        messagebox.showerror("Error", "No images found in the selected
→folder.")
    ↵

    def on_scroll(self, value):
        # Update current index from scrollbar
        self.current_volume_index = int(value)
        self.display_volume_image()
        self.analyze_volume_image()

    def display_volume_image(self):
        # Load current image from volume
        image_path = self.volume_images[self.current_volume_index]
        image = cv2.imread(image_path)
        if image is not None:
            self.display_image(image)

    def analyze_volume_image(self):
        # Load current image from volume
        image_path = self.volume_images[self.current_volume_index]
        image = cv2.imread(image_path)
        if image is not None:
            self.analyze_image(image)

    def display_image(self, image):
        # Convert image to RGB and resize for display (displaying at 400 pixels
→width/height)
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image_pil = Image.fromarray(image_rgb)
        image_pil = image_pil.resize((400, 400), Image.LANCZOS)
        image_tk = ImageTk.PhotoImage(image_pil)
        self.image_canvas.configure(image=image_tk)
        self.image_canvas.image = image_tk

    def analyze_image(self, image):
        # Ensure the model is loaded
        if not hasattr(self, 'model'):
            messagebox.showwarning("Warning", "Please load a model first.")
            return

        # Preprocess the image for YOLO model
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

```

```

    image_resized = cv2.resize(image_rgb, (640, 640)) # Resize for YOLO
    ↵input
    transform = transforms.ToTensor()
    image_tensor = transform(image_resized).unsqueeze(0)

    # Run inference on the image
    results = self.model.predict(image_tensor, imgsz=640)

    # Convert the results to an image with bounding boxes and labels drawn
    results_img = results[0].plot()

    # Draw a small box at the center of each bounding box to indicate the
    ↵center point
    for box in results[0].boxes.xywh.cpu().numpy():
        center_x, center_y, width, height = box[:4] # Extract center x, y,
    ↵width, and height from the box
        box_size = 6 # Size of the small box to draw at the center
        top_left = (int(center_x - box_size / 2), int(center_y - box_size /
    ↵2))
        bottom_right = (int(center_x + box_size / 2), int(center_y +
    ↵box_size / 2))
        cv2.rectangle(results_img, top_left, bottom_right, (0, 0, 255), -1)
    ↵ # Filled small box in green

    # Print detected labels and confidence scores
    labels = results[0].boxes.cls.cpu().numpy()
    confidences = results[0].boxes.conf.cpu().numpy()
    detected_info = "Detected Labels and Confidence Scores:\n"
    for label, confidence, box in zip(labels, confidences, results[0].boxes.
    ↵xywh.cpu().numpy()):
        center_x, center_y = box[:2] # Extract center x and y from the box
        detected_info += f"Label: {int(label)}, Confidence: {confidence:.2f}\n"

    if self.selected_file_path:
        self.result_label.config(text=detected_info, fg="black")
    else:
        self.result_label.config(text="", fg="blue")

    # Convert results image to display
    result_img_rgb = cv2.cvtColor(results_img, cv2.COLOR_BGR2RGB)
    result_pil = Image.fromarray(result_img_rgb)
    result_pil = result_pil.resize((400, 400), Image.LANCZOS)
    result_tk = ImageTk.PhotoImage(result_pil)
    self.image_canvas.configure(image=result_tk)
    self.image_canvas.image = result_tk

```

```

def save_comment(self):
    # Save the comment to a text file with the same name as the image
    if self.selected_file_path:
        comment = self.comment_box.get("1.0", tk.END).strip()
        if comment:
            comment_file_path = f"[os.path.splitext(self.
↪selected_file_path)[0]}_comments.json"
            import json
            comments_data = {}
            if os.path.exists(comment_file_path):
                with open(comment_file_path, 'r') as file:
                    comments_data = json.load(file)
                comments_data[os.path.basename(self.selected_file_path)] =_
↪comment
                with open(comment_file_path, 'w') as file:
                    json.dump(comments_data, file, indent=4)
                messagebox.showinfo("Success", f"Comment saved to_
↪{comment_file_path}")
            else:
                messagebox.showwarning("Warning", "Please enter a comment_
↪before saving.")
        else:
            messagebox.showwarning("Warning", "No image selected.")

def save_all_comments(self):
    # Save all comments to the JSON file associated with the image
    if self.selected_file_path:
        comment = self.comment_box.get("1.0", tk.END).strip()
        if comment:
            self.save_comment()
        else:
            messagebox.showwarning("Warning", "Please enter a comment_
↪before saving.")
    else:
        messagebox.showwarning("Warning", "No image selected.")

if __name__ == "__main__":
    root = tk.Tk()
    app = ImageAnalyzerApp(root)
    root.mainloop()

```

1.7 References

- Ali, Z. (2024). Object Detection and Segmentation - Assignment 3. In <https://sandiego.instructure.com>. Internal Canvas.
- Brownlee, J. (2020, August 15). 8 Tactics to combat imbalanced classes in your machine learning dataset. MachineLearningMastery.com. <https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/>
- Darabi, P. K. (2020). Medical image dataset for brain tumor detection [Data set]. Kaggle. <https://www.kaggle.com/datasets/pkdarabi/medical-image-dataset-brain-tumor-detection/data>
- Datadance. (2024, October 24). YOLOV11: The next leap in Real-Time Object Detection. <https://datadance.ai/machine-learning/yolov11-the-next-leap-in-real-time-object-detection>
- Kasai, Y., & Kinoshita, K. (2020). “Clinical Use of Color-coded MRI: MRI Color Imaging, Applications in Neurology and Psychiatry.” Magnetic Resonance Imaging Clinics of North America, 28(4), 563-574. Harvard University. (2024, September). New AI tool can diagnose cancer, guide treatment, predict patient survival. Harvard Gazette.
- Oksuz, K., Cam, B. C., Kalkan, S., & Akbas, E. (2019, August 31). Imbalance Problems in Object Detection: A review. arXiv.org. <https://arxiv.org/abs/1909.00169>
- Ultralytics. (2024a). YOLOv11 model documentation. <https://docs.ultralytics.com/models/yolo11/>
- Ultralytics. (2024b). YOLOv8 model documentation. <https://docs.ultralytics.com/models/yolov8/>
- Ultralytics. (2024c). Ultralytics GitHub repository. <https://github.com/ultralytics/ultralytics>
- Terven, J., Córdova-Esparza, D. M., & Romero-González, J. A. (2023). A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. Machine Learning and Knowledge Extraction, 5(4), 1680-1716.
- Tykocki, T., & Eltayeb, M. (2018). Ten-year survival in glioblastoma. A systematic review. Journal of Clinical Neuroscience, 54, 7-13.