Segmentation and 3D Reconstruction of Fetal Ventricles

Project report submitted to the Amrita Vishwa Vidyapeetham in partial fulfilment of the requirement for the Degree of

BACHELOR of TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Akash Harikumar AM.EN.U4CSE20204
Alan Koshy Luke AM.EN.U4CSE20205
Amal Vinod AM.EN.U4CSE20206
Aravind S AM.EN.U4CSE20211

AMRITA SCHOOL OF COMPUTING

AMRITA VISHWA VIDYAPEETHAM
(Estd. U/S 3 of the UGC Act 1956)
AMRITAPURI CAMPUS

KOLLAM -690525

MAY 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM
(Estd. U/S 3 of the UGC Act 1956)

Amritapuri Campus

Kollam -690525

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"Segmentation and 3D Reconstruction of Fetal Ventricles"** submitted by **Akash Harikumar(AM.EN.U4CSE20204), Alan Koshy Luke(AM.EN.U4CSE20205), Amal Vinod(AM.EN.U4CSE20206) and Aravind S(AM.EN.U4CSE20211))**, in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Computer Science and Engineering from Amrita Vishwa Vidyapeetham, is a bonafide record of the work carried out by them under my guidance and supervision at Amrita School of Computing, Amritapuri during Semester 8 of the academic year 2023-2024.

Dr. Sandhya Harikumar
Project Guide

Dr.Namitha K
Project Coordinator

Dr. Swaminathan J
Chairperson
Dept. of Computer Science & Engineering

Reviewer

Place : Amritapuri
Date : 15 May 2024

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

(Estd. U/S 3 of the UGC Act 1956)

Amritapuri Campus

Kollam -690525

DECLARATION

We, **Akash Harikumar(AM.EN.U4CSE20204), Alan Koshy Luke(AM.EN.U4CSE20205), Amal Vinod(AM.EN.U4CSE20206) and Aravind S(AM.EN.U4CSE20211))** hereby declare that this project entitled **"Segmentation and 3D Reconstruction of Fetal Ventricles"** is a record of the original work done by us under the guidance of **Dr. Sandhya Harikumar**, Dept. of Computer Science and Engineering, Amrita Vishwa Vidyapeetham, that this work has not formed the basis for any degree/diploma/associationship/fellowship or similar awards to any candidate in any university to the best of our knowledge.

Place : Amritapuri

Date : 17 May 2024

Signature of the student

Signature of the Project Guide

Acknowledgements

The successful culmination of this project was made possible through the invaluable support, guidance, and contributions of numerous individuals, to whom we extend our deepest gratitude. Foremost among them are Dr. Sandhya Harikumar, Dr. Simi Surendran, Dr. Suhas Udaykumaran and Dr. Shilpa Radhakrishnan whose expertise, encouragement, and insightful feedback were instrumental throughout the project's duration, significantly shaping our ideas and refining our approach. We express our sincere gratitude to AMRITA SCHOOL OF COMPUTING and AMRITA INSTITUTE OF MEDICAL SCIENCE, KOCHI for providing the essential resources and fostering a conducive environment that facilitated the undertaking of this endeavor. This institutional support played a vital role in the project's successful execution. In closing, we extend our appreciation to everyone who supported and contributed to this project in various capacities. The collective efforts and contributions of each individual involved have been indispensable to the realization of this endeavor.

Abstract

Ventriculomegaly is a condition marked by enlarged ventricles in the brain. The need for better visualization techniques is highlighted by how difficult it is to understand the data, particularly for the untrained eye. Previous approaches have focused on advanced image processing to enhance 2D image visualization. These approaches, however, might fall short of offering thorough insights into ventriculomegaly and its underlying causes. This project proposes an approach that utilizes MRI scans to create segmented three-dimensional (3D) models. Preliminary findings indicate that these 3D models significantly improve diagnostic accuracy, providing detailed insights into ventriculomegaly and its underlying causes. Furthermore, they facilitate longitudinal tracking of disease progression, aiding in treatment planning and evaluation. Utilising a multi-model approach to help segment the regions of interest within the MRI scan and methods that combine the regions of interest to develop a 3D structure of the ventricle for better diagnosis, despite challenges in image segmentation and model generation, such as ensuring accuracy and efficiency, this approach holds promise for enhancing ventriculomegaly diagnosis and management. Overall, the integration of 3D modeling into clinical practice represents a significant advancement in improving patient outcomes and enhancing care quality for individuals with ventriculomegaly.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ventriculomegaly, a condition defined by an abnormal enlargement of the fluid-filled cavities within the brain known as ventricles, presents a significant challenge for diagnosis and treatment. Interpreting the complex data gleaned from traditional 2D brain scans can be difficult, especially for those without extensive medical training. These 2D images often lack the necessary detail to reveal subtle variations in brain anatomy that can be crucial for diagnosing ventriculomegaly. Furthermore, current methods often fall short in providing a comprehensive understanding of the underlying causes of enlarged ventricles and the impact they have on surrounding brain tissue. This limited understanding hinders the development of optimal treatment strategies.

In light of these limitations, researchers are exploring the potential of 3D modeling as a transformative tool for improving ventriculomegaly diagnosis and management. 3D modeling technology allows for the creation of detailed, three-dimensional representations of the brain's ventricles. These models offer a far more intuitive and realistic view compared to traditional 2D scans. By providing a more comprehensive picture, 3D models can enable a clearer analysis of the size, shape, and any potential abnormalities present within the ventricular system.

This enhanced visualization has the potential to significantly improve diagnostic accuracy, allowing healthcare professionals to make more informed decisions about treatment options.

The benefits of 3D modeling extend beyond initial diagnosis. These models can be instrumental in tracking the progression of ventriculomegaly.While 3D modeling holds immense promise for revolutionizing ventriculomegaly diagnosis and management, there are some challenges that need to be addressed. One hurdle lies in the process of image segmentation, which involves separating the ventricles from the surrounding brain tissue in the scans used to create the models.The segmentation process is essential for accurate diagnosis and understanding of fetal brain development. During the developmental stages of the ventricles, clinicians rely on detailed imaging to understand growth patterns and potential abnormalities. By utilizing our segmentation model, clinicians gain a well-defined view of the ventricles, as the output is a binary image highlighting only the regions of interest, i.e., the ventricles. This aids in the identification of abnormalities in ventricular development, enhancing diagnostic accuracy . In the context of our project, segmentation plays a critical role in reconstructing the fetal brain ventricles. Furthermore, segmentation serves as the initial step for further stages of reconstruction, whether it be single-plane or 3D visualisation. By providing clinicians with clear and isolated views of the ventricles, the segmentation approach also helps non-medical professionals to easily identify regions of interest. Ensuring the efficiency and accuracy of the segmentation model generation is a challenge that can be addressed through advanced image processing techniques which would pave the way to a precise 3D reconstruction. Additionally, wider adoption of 3D modeling in clinical settings requires further development in software tools specifically designed for this purpose. Healthcare professionals will also need to undergo training to effectively utilize these tools and interpret the resulting 3D models. Finally, standardized

protocols need to be established to integrate 3D models seamlessly into existing clinical workflows.

Despite these challenges, the potential benefits of 3D modeling in improving patient outcomes for those with ventriculomegaly are undeniable. By providing a more comprehensive understanding of this condition, 3D modeling has the potential to revolutionize the way ventriculomegaly is diagnosed, treated, and monitored. As the technology continues to develop and these challenges are addressed, we can expect to see 3D modeling become a standard tool in the clinical management of ventriculomegaly, leading to improved quality of care for patients.

Chapter 2

Problem Definition

The problem definition outlines the challenges posed in identifying ventriculomegaly, the limitations of current approaches, and proposes a solution-oriented approach using 3D modelling.

Ventriculomegaly, characterized by enlarged ventricles in the brain, poses significant challenges in accurate diagnosis and effective treatment planning. Traditional 2D imaging techniques often lack the necessary detail to provide comprehensive insights into the condition, making it difficult for healthcare professionals to interpret and analyze the data, particularly for those without specialized medical training. Current approaches focusing on advanced image processing for 2D visualization may not sufficiently address the complexities of ventriculomegaly and its underlying causes.

The primary issue lies in the inadequacy of existing methods to offer thorough insights into ventriculomegaly and its progression. Traditional 2D imaging techniques are limited in their ability to provide detailed anatomical information and fail to capture subtle variations crucial for diagnosis and treatment planning. Consequently, there is a pressing need for innovative approaches that can enhance

diagnostic accuracy and facilitate tracking of disease progression.

This project aims to address these challenges by proposing a approach utilizing MRI scans to generate three-dimensional (3D) models of the brain's ventricular system. Preliminary findings suggest that these 3D models have the potential to significantly improve diagnostic accuracy and provide detailed insights into ventriculomegaly and its underlying causes. By employing a multi-modal approach for image segmentation and model generation, this project seeks to overcome challenges such as accuracy and efficiency in 3D model development.

Key objectives of this project include:

1. Enhancing diagnostic accuracy and understanding of ventriculomegaly through 3D modeling.

2. Facilitating lengthwise tracking of disease progression for improved treatment planning and evaluation.

3. Addressing challenges in image segmentation and model generation to ensure accuracy and efficiency.

4. Integrating 3D modeling into clinical practice to enhance patient outcomes and care quality.

By using advancements in iimage processing techniques, this project endeavors to revolutionize the diagnosis and management of ventriculomegaly, ultimately improving patient care and outcomes.

Chapter 3

Related Work

Table 3.1: Literature Review

Paper	**Title/Year**	**Problem Addressed**	**Contributions**	**Limitations**	**Open Problems**
1	U-Net and Its Variants for Medical Image Segmentation: A Review of Theory and Applications 2021	The application of U-Net and its variants for medical image segmentation, focusing on challenges in medical imaging such as separating touching objects of the same class and the scarcity of properly labeled images, and how U-Net addresses these challenges.	The use of skip connections. Reviewing the theory and applications of U-Net and its variants for medical image segmentation. Comparison with other segmentation models.	Not covering all possible U-net variations due to the large body of ongoing research in medical image segmentation. less emphasis on alternative segmentation techniques. Challenges related to computational power, scarcity of annotated data for training, and the "black box" nature of deep learning models.	Conduct in-depth analyses of specific U-Net variants to learn more about it and identify opportunities for improvement. The use of explainable AI techniques to make U-Net more transparent and interpretable for medical professionals.

Continued on next page

Table 3.1 – continued from previous page

Paper	Title/Year	Problem Addressed	Contributions	Limitations	Open Problems
2	Image Stitching Techniques Applied to Plane or 3-D Models 2023	The challenges faced in image stitching techniques that include handling fast motion and object changes, where existing methods struggle with real-time dynamics and are limited in dealing with fast motion and object changes in scenes. Another challenge is low-texture overlapping regions, where algorithms face difficulties due to scarce feature points in low-texture areas, leading to potential mismatches. The inability to convert from one image format to another. Improving feature point distribution in such regions is a significant challenge.	Explains the four steps involved in image stitching like Prepossessing, Registration, Alignment and Blending. Discusses the challenges faced in image stitching techniques like: Handling fast motion and object changes,Dealing with low-texture overlapping region,Representing the geometry of panoramic model,Balancing stitching performance and computing resources.	Limited focus on practical application scenarios with complex datasets Emphasis on standard datasets, potentially not addressing real-world challenges Insufficient discussion on innovative strategies for handling image stitching in complex scenes	Further studies on the conversion from different image formats and more detailed research about the technical details involved in different stages

Continued on next page

Table 3.1 – continued from previous page

Paper	Title/Year	Problem Addressed	Contributions	Limitations	Open Problems
3	MONAIFBS: Monai-based fetal brain MRI deep learning segmentation(2021)	Improve fetal brain segmentation for Super Resolution Reconstruction in MRI scans.They proposed a new tool called MONAIfbs, based on the MONAI framework, which utilized a well-trained single-step UNet model (dynUNet) to achieve accurate performance without the need for a 2-step approach. The model was trained with data from two different centers, making it more robust to domain shift. By using this approach, the authors were able to reduce the need for manual corrections in automated SRR pipelines, providing an improvement in fetal brain segmentation techniques.	Improving fetal brain segmentation for Super Resolution Reconstruction in MRI scans by proposing a new tool called MONAIfbs, which utilizes the MONAI framework for fetal brain segmentation. The authors hypothesize that a well-trained single-step UNet model can achieve accurate performance, eliminating the need for a 2-step approach. The proposed dynUNet model showed higher performance and reduced the number of outliers, leading to an improvement in state-of-the-art fetal brain segmentation techniques and reducing the need for manual corrections in automated Super Resolution Reconstruction pipelines.	The reliance on 2D stacks for training and validation potentially limits the ability to fully capture the complexity of the fetal brain in a 3D context because 2D slices may not fully represent the spatial relationships and structures present in a 3D volume. This limitation could result in a loss of information regarding the overall shape, size, and orientation of the fetal brain, which are crucial for accurate segmentation and reconstruction in 3D.	The refining of the MONAIfbs tool and simplifying its usage, include further optimization of the network architecture and hyperparameters, as well as enhancing the user interface for easier and more intuitive operation. Additionally, the authors plan to incorporate feedback from users and continue to improve the tool based on real-world usage and feedback.

Continued on next page

Table 3.1 – continued from previous page

Paper	Title/Year	Problem Addressed	Contributions	Limitations	Open Problems
4	Automatic Ventriculomegaly Detection in Fetal Brain MRI: A Step-by-Step Deep Learning Model for Novel 2D-3D Linear Measurements 2023	The potential of artificial intelligence (AI) in automatically estimating ventriculomegaly with high accuracy. The study focused on developing a deep learning model to measure the size of lateral ventricles in fetal brain MRIs, showcasing the feasibility and accuracy of AI-based ventricle measurement. The AI model demonstrated the capability to improve diagnostic capabilities and streamline the evaluation of fetal brain abnormalities in clinical practice.	Focused Investigation on Ventriculomegaly Detection and the feasibility and accuracy of AI-based ventricle measurement in fetal brain MRIs, offering a novel approach in this domain. The combination of automated segmentation and accurate ventricle measurement demonstrated the potential of AI to improve diagnostic capabilities and streamline the evaluation of fetal brain abnormalities in clinical practice.	The AI model relies on 3D reconstruction for the highest prediction accuracy and highest image quality, so it cannot be used if the original images do not have enough quality for 3D reconstruction. The limitations of the study related to hardware dependency include the need for high-quality original images for 3D reconstruction to achieve the highest level of prediction accuracy.	Improving the speed and accuracy of 3D reconstruction, exploring advanced AI models for gestational age prediction and brain pathology classification using MRI, developing automated frameworks for fetal brain segmentation, and investigating the use of CNN architectures like U-Net for medical image segmentation in fetal MRI. Additionally, research could focus on enhancing the quality of reconstructed images through techniques such as super-resolution reconstruction and exploring new segmentation strategies and models for improved results.

Continued on next page

Table 3.1 – continued from previous page

Paper	Title/Year	Problem Addressed	Contributions	Limitations	Open Problems
5	Comprehensive Multimodal Segmentation in Medical Imaging: Combining YOLOv8 with SAM and HQ-SAM Models,2023	Improving medical image segmentation models to achieve higher segmentation accuracy, enabling precise identification and delineation of anatomical structures and abnormalities in diverse medical imaging datasets.	A comprehensive evaluation of three different models (SAM, HQ-SAM, and YOLOv8) for medical image segmentation across multiple datasets. It also emphasizes the importance of selecting the appropriate model based on the specific requirements of the medical imaging task and dataset characteristics. And the potential for further improvement in medical image segmentation models by enhancing existing models, exploring new architectures and attention mechanisms, and investigating model ensembles.	The evaluation being limited to three specific models, potentially excluding other state-of-the-art models. Additionally, the datasets used may not fully represent the diversity of medical imaging challenges, and performance on other datasets might differ.	Future work in the field of medical image segmentation could focus on enhancing existing models like the SAM model, exploring new architectures and attention mechanisms, and investigating model ensembles or combining multiple models for even higher segmentation performance. Additionally, incorporating more diverse datasets and considering the transferability of models to different medical imaging modalities should be considered in future studies.

Chapter 4

Requirements

The design of this project contains both hardware and software. The specifications are listed below.

4.1 Hardware

GPU Minimum requirement: 2GB of memory

RAM Minimum requirement: 12GB of memory

Hard Disk Space requirement: 3GB

Connectivity Internet connectivity

4.2 Software

4.2.1 Python Packages

- SimpleITK
- Vedo

- segment_anything
- Streamlit
- NumPy
- cv2
- matplotlib.pyplot
- os
- PIL (Python Imaging Library) Pillow
- shuttle
- pathlib
- IPython.display
- json
- glob
- yaml
- shutil
- matplotlib.patches
- locale
- nibabel

4.2.2 Web Browser Capability

Web browser compatibility is required. Since the application runs on streamlit, javascript support is also required within the browser.

Chapter 5

Proposed System

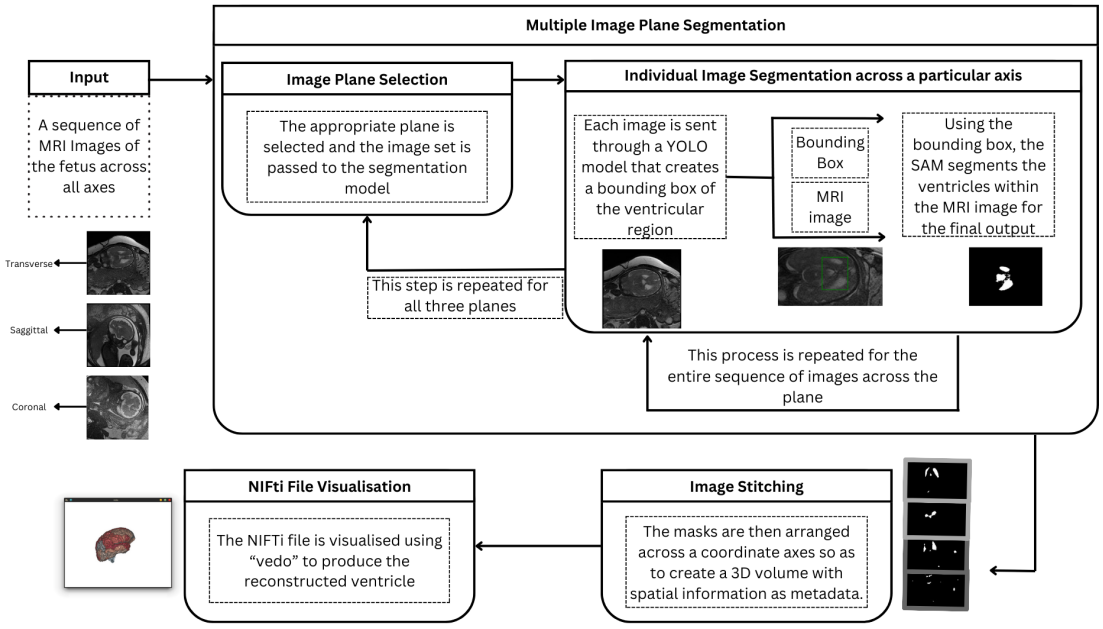

Figure 5.1: Proposed System Methodology

The project proposes the utilization of segmentation and reconstruction algorithms to reconstruct a specific region of the fetal brain i.e the ventricles from a series of MRI images. The pipeline/architecture shown above goes into detail on how the final application is supposed to function.

5.0.1 Input

Since the project works primarily with JPEG images, the input to the system would be a series of MR images, each within in its own folder according to the axis along which the scans were taken or all combined within one folder. During each prompt made by the system, the user would be required to upload the set of images (as shown in fig. 5.4) along a particular axis. This process would occur once for the entire life cycle of a reconstruction. Following the insertion of the appropriate data, all the images are resized to a 800x800 (width x height) resolution for a faster segmentation procedure.

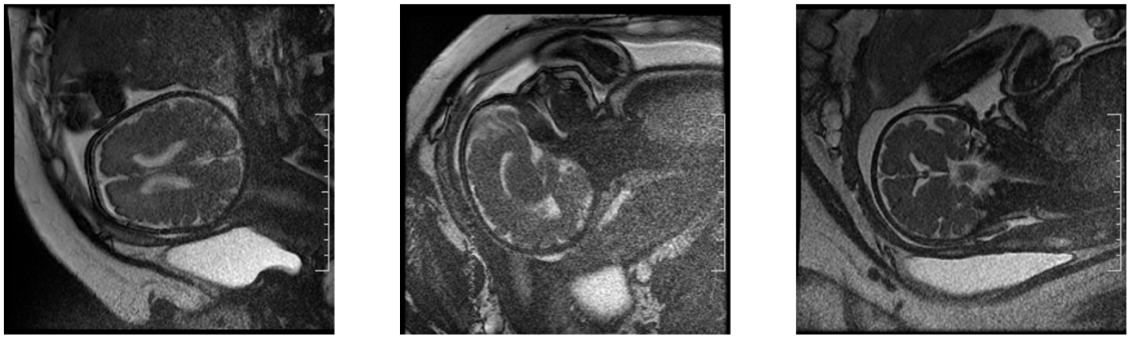

Figure 5.2: A Set of Transverse, Sagittal and Coronal MRI Images

5.0.2 Multiple Plane Image Segmentation

Each of the resized images is passed through the appropriate segmentation model where it identifies the ventricles and stores them in individual folders based on different planes (transverse/axial, sagittal, and coronal). To delve deeper into the segmentation process, it utilizes a multi-model approach to identify the regions of interest. The initial set of images are passed through a pre-trained YOLOv8 model to perform object detection and mark bounding boxes across the ventricular region. It's worth noting that in most cases, there might be more than one bounding box. In cases where there is only one bounding box, the bounding box accompanied with the corresponding image is passed into the SAM (Segment Anything

Model) for instance segmentation. SAM works by first encoding the image into a high-dimensional vector representation and the prompt into a separate vector representation. These two representations are then combined and passed to a mask decoder, which outputs a mask for the object specified by the prompt. The image encoder, a vision transformer (ViT-H) model, has been pre-trained on a massive dataset of images. The prompt encoder, converts the input prompt into a vector representation. The mask decoder, a lightweight transformer model, predicts the object mask from the image and prompt embeddings. For cases where more than one bounding box is detected, each bounding box is passed alongside the image to SAM for segmentation. Later, all the segmented masks are combined into a single mask and stored as the final output mask for that image. This process is repeated for all the images within a set.

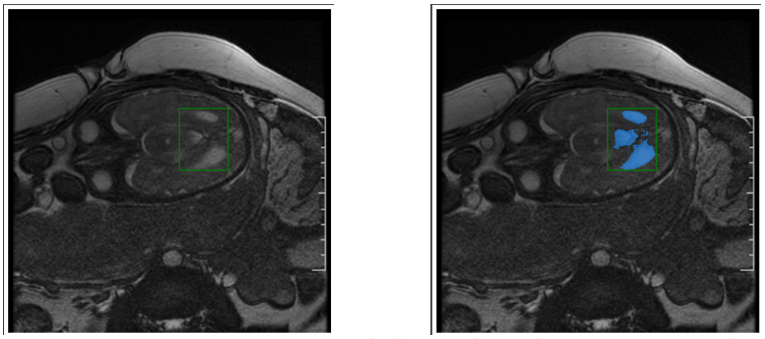

Figure 5.3: MRI image with bounding box on ventricles

5.0.3 Image Stitching

Once all the ventricles have been segmented and masks have been generated, all of the masks across the multiple planes are combined together to form a volumetric data (stored as a .nii file). Since it is not possible to generate a completely new nii file from simple png/jpg images, the project proposes to utilise a pre-existing nii file as a template and to replace the images inside the file with the generated segmentation masks. This idea is based on the assumption that in most cases,

during MRI scans, the spatial data between each slice would remain the same and it would be the images that vary from patient to patient.

5.0.4 Visualisation

Once the volumetric data has been created, it is visualised through a library known as "vedo" which has provides various functions similar to the VTK Renderer function(discussed in the previous phase) through which an nii file can be interpreted and interacted with by the end-user.

Figure 5.4: Individual Plane Reconstruction (Transverse and Coronal) alongside Reconstruction of a Proper NII file

5.0.5 Algorithms

Algorithm 1 Data Conversion Pipeline

Require: Folder paths for input and output data

Convert JPEG to PNG:
for each file in input folder **do**
    if file ends with ".jpg" **then**
        Convert to PNG format
        Remove original JPG file
    end if
end for

Algorithm 1 Data Conversion Pipeline (continued)

Rename Mask Files to Match Source Image:
for each file in folder **do**
    if file starts with "mask_" and has a number **then**
        Rename file
    end if
end for
Resize Images:
for each image in input folder **do**
    resize image to target resolution
    Save resized image
end for

Generate COCO JSON:
process mask images
create annotations
Save annotations to JSON file

Convert to YOLO Format:
Load annotations from JSON
Create YOLO format:
for each annotation **do**
    calculate object center coordinates and width/height
    convert bounding box coordinates to YOLO format
    append YOLO-formatted annotation to output file
end for
Save YOLO format

Create YAML Configuration:
Extract categories and counts
Create YAML data:
define classes with their names and IDs
organize train, validation, and test data paths
specify the number of classes and paths to data
Write YAML data to file

Algorithm 2 YOLOv8 Object Detection Algorithm

Input: Image to be analyzed
Preprocess the input image:
- Follow the steps in algorithm 1
Load the YOLOv8 model architecture:
- YOLOv8 consists of a backbone network (e.g., Darknet) for feature extraction
- Additional layers for object detection and classification
Forward Pass:
- Pass the preprocessed image through the YOLOv8 model
- Extract features using the backbone network
- Perform object detection and classification using detection heads
Post-processing:
- Apply non-maximum suppression to remove duplicate detections
- Filter out detections below a certain confidence threshold
Output:
- Return the bounding boxes, class labels, and confidence scores of detected objects in the image

Algorithm 3 Segment Anything Model (SAM) Operation with Bounding Box Prompt

Input:
- Bounding box coordinates.
- MRI Image

Initialization:
- Load the pre-trained SAM model.

Processing:
- Extract the region within the bounding box from the image.
- Use the image encoder to extract features from the region.
- Pass the features through the segmentation network to generate masks.

Segmentation:
- Generate segmentation masks based on the region within the bounding box.

Output:
- Return the segmented masks corresponding to the region with reference to the bounding box.

Chapter 6

Results and Analysis

In the developmental phase of the study, a comprehensive dataset comprising 275 Magnetic Resonance Imaging (MRI) scans focusing specifically on the fetal brain has been acquired from Amrita Institute of Medical Sciences and Research Center in Kerala, India. These high-resolution images were captured along the transverse axis, coronal axis, and sagittal axis providing a detailed cross-sectional view of the developing neural structures, particularly the ventricles.

During the initial stages of development, a U-Net model was constructed for segmentation in the transverse plane. However, to enhance performance, we transitioned to experimenting with the Segment Anything Model (SAM) and finally settled with a multi-model approach utilizing the YOLOv8 model along with SAM (as shown in 6.1).

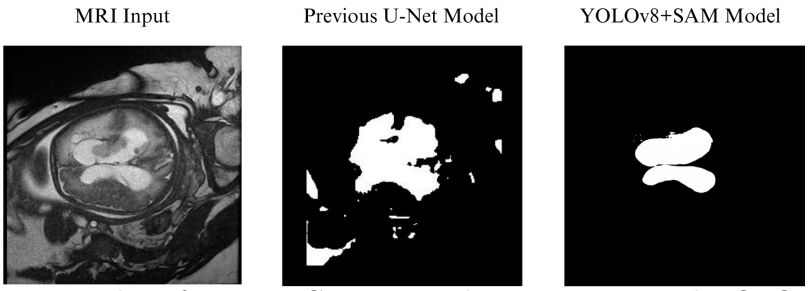

Figure 6.1: Visual Performance Comparison between UNet and YOLOv8+SAM Models for Transverse Plane Segmentation

To train and validate our segmentation model, we utilised :

- 89 transverse plane MRI images

- 76 sagittal plane MRI images

- 88 coronal plane MRI images

The performance of each model was evaluated using various metrics, including the confusion matrix, to assess its accuracy and robustness. Additionally, we analyzed the training process using the analysis graph, which provides insights into the training and validation performance. This graph includes multiple plots such as train/box_loss, train_seg_loss, val_box_loss and val_seg_loss as shown in figures 6.2, 6.4, 6.6 for each of the three planes respectively.

Furthermore, we illustrate the output of the YOLO model for both training and validation images in figures 6.3, 6.5 and 6.7 respectively for transverse, sagittal and coronal planes.

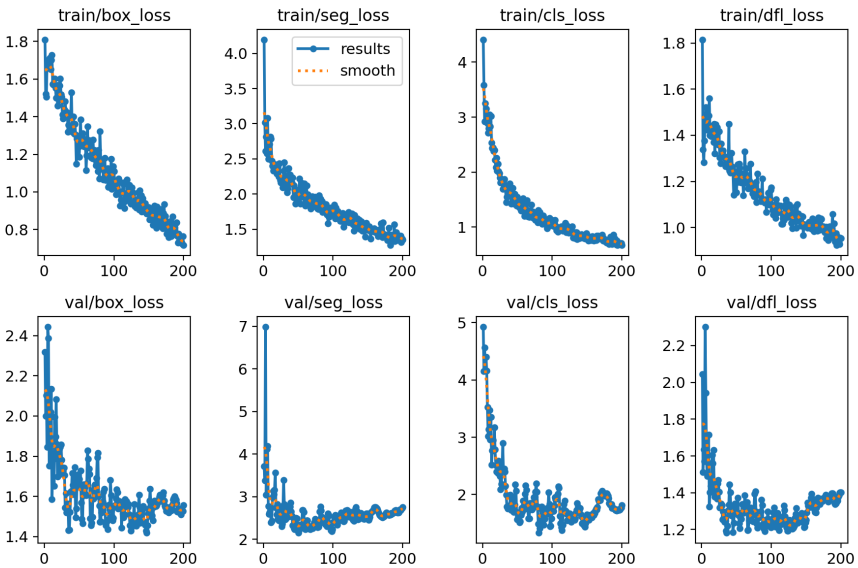

Figure 6.2: Training and Validation Performance

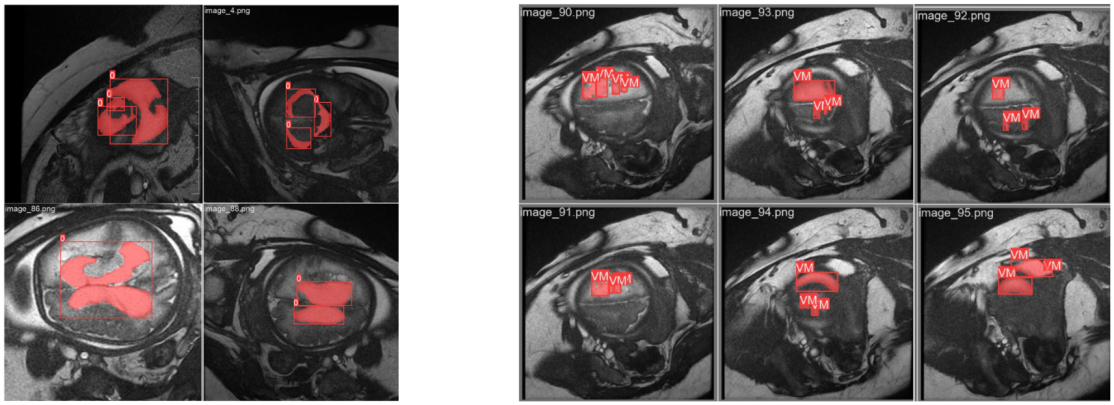

Figure 6.3: Output of YOLO Model for Training and Validation Images in the Transverse Plane

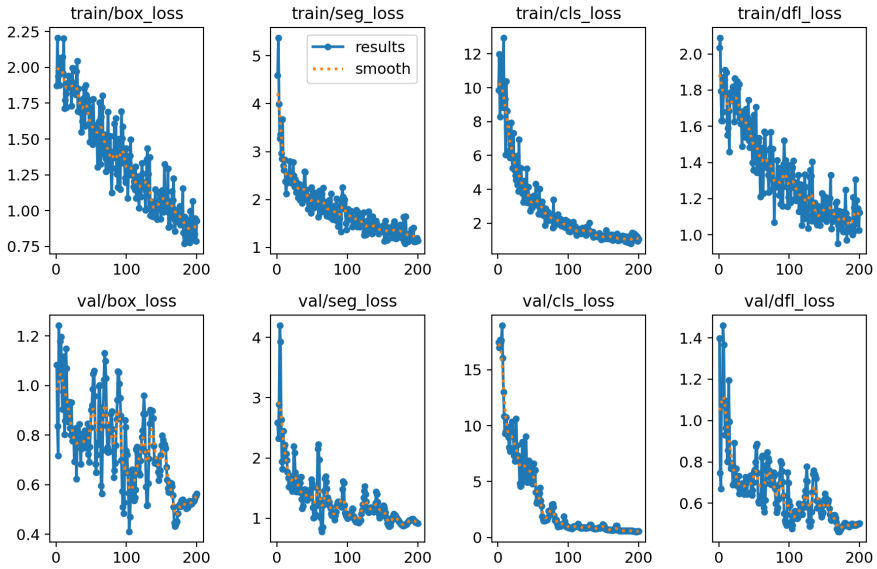

Figure 6.4: Training and Validation Performance for sagittal Plane Segmentation

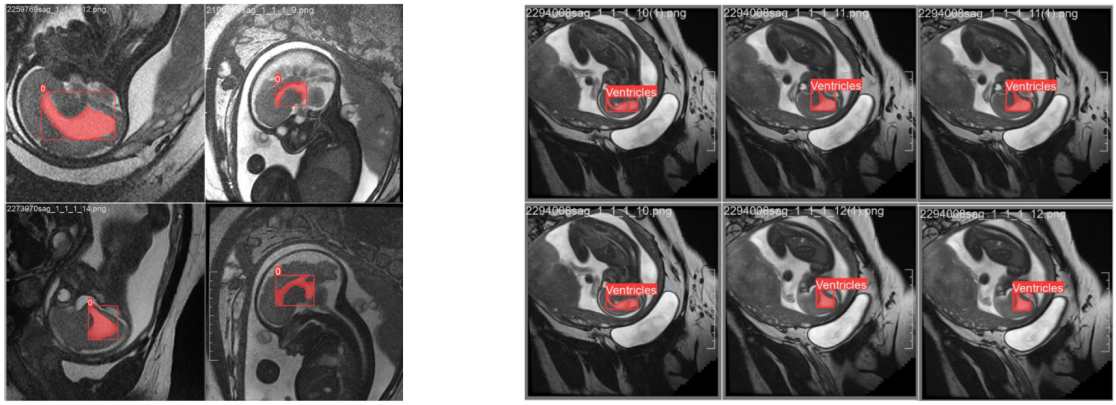

Figure 6.5: Output of YOLO Model for Training and Validation Images in the sagittal Plane

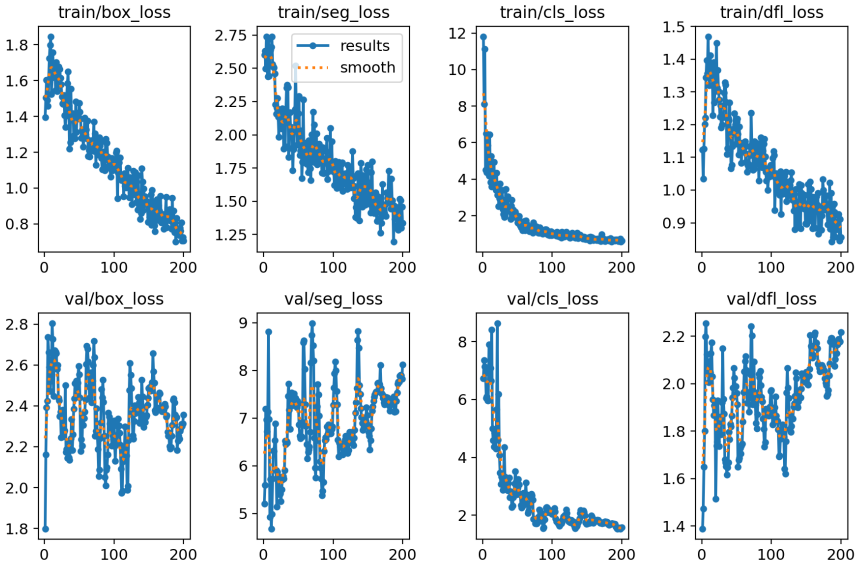

Figure 6.6: Training and Validation Performance for Coronal Plane Segmentation

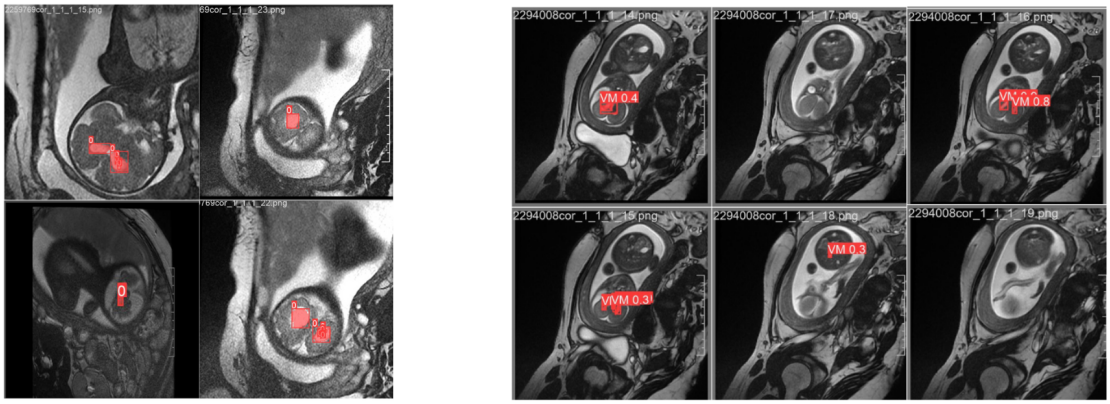

Figure 6.7: Output of YOLO Model for Training and Validation Images in the Coronal Plane

6.0.1 Analysis

Analysis of our results indicate the success of incorporating the Segment Anything Model (SAM) alongside the YOLOv8 model (as shown in fig 6.8) for segmenting ventricles from an MRI image. However it is to be noted that each image within the MRI set requires some time interval to be segmented properly. To denote a standard, running the program on Google's free tier T4 GPU took a rough estimate

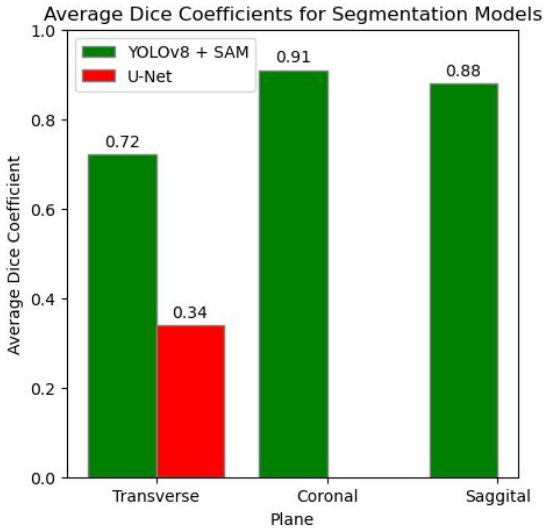

Figure 6.8: Dice-Coefficient

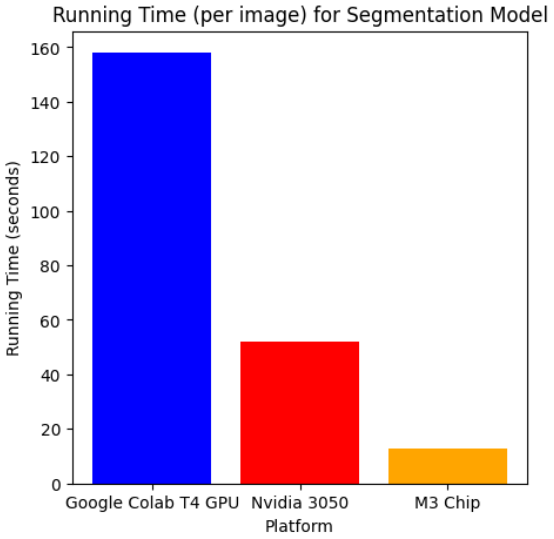

Figure 6.9: Performance Time

of 2 minutes and 30 seconds for each image. Performance can be improved by using better GPU or CPU as shown in fig 6.9.

Even though YOLOv8 offers instance segmentation, it is often limited to the region within the bounding box that it predicts, as a result certain regions of the ventricles are often cut off. SAM offers to extend segmentation to regions outside the bounding box that it assumes to be part of the region within the box, thereby eliminating the occurrence of the previous issue.

Overall, our segmentation and reconstruction pipeline demonstrate promising performance across all three planes, with substantial decrease in noise as compared to the U-Net and Dense U-Net model for segmentation.

6.0.2 Image Stitching: The Conversion of JPEG Images to NIfTI Format

The conversion of JPEG images to NIfTI format involved several methods and analyses:

- **Stacking of Images:**

  - The input images (coronal, sagittal, and transverse) were initially stacked on top of each other to form volumetric data using SimpleITK.
  - However, upon opening the resultant NIfTI file with 3D Slicer, it was observed that the images were not converted properly, and interaction with the images was not possible.
  - We discovered that JPEG images could not be directly converted to NIfTI format due to the absence of manually entered spatial data.
  - This highlights the challenge of converting 2D JPEG images, which lack spatial information, into a 3D volumetric NIfTI format.
  - However this stacking method could be ultilized to create a volumetric data of a single plane and display it. But the information it conveys is limited to view of the particular plane and on further movement to other planes, it cannot accurately convey the required information.

- **Usage of Pre-existing NIfTI File:**
  - As a solution, we attempted to use a pre-existing NIfTI file and replace the images within it which involved utilizing the spatial data from the pre-existing NIfTI file for our images.
  - However, during the image replacement process, we encountered the error "Mismatch in shape between coronal and NIfTI plane."
  - This suggests that the spatial information of the pre-existing NIfTI file did not match the JPEG images, leading to compatibility issues.

- **Resolving Mismatch Errors:**
  - To address this error, we investigated potential causes and found a mismatch in resolution between the NIfTI and JPEG images.

  - To rectify this, we adjusted the resolution of our input JPEG images to match that of the NIfTI file.
  - Despite this adjustment, the error persisted. We then attempted to match the number of slices in each coronal, sagittal, and transverse JPEG image with those in the NIfTI file by duplicating the images.
  - These efforts show the complexity of aligning the spatial properties of JPEG images with those of the NIfTI format.

- **Further Investigation:**
  - We examined the shape of the NIfTI plane before and after passing it to the function and found that it remained constant.
  - Similarly, the shapes of our coronal, sagittal, and transverse images matched those of the NIfTI file.
  - Further research revealed its not possible to manipulate the spatial data, prompting ongoing investigations into methods for modifying spatial data and converting JPEG images to NIfTI format effectively.
  - This highlights the need for novel approaches to handle spatial information and optimize the conversion process.

This analysis highlights the challenges encountered in converting JPEG images to NIfTI format and underscores the need for continued research in this area.

Chapter 7

Conclusion

Ventriculomegaly, characterized by enlarged brain ventricles, poses challenges for accurate diagnosis and effective treatment planning using traditional 2D imaging. This project explored generating 3D models from MRI scans to enhance understanding and management of this condition.

The proposed 3D modeling approach showed promising results in preliminary findings. These models provide intuitive visualizations of brain anatomy, potentially improving diagnostic accuracy over 2D scans alone. Additionally, they facilitate tracking disease progression to aid treatment planning and evaluation over time.

Attempts to convert 2D JPEG images to 3D NIfTI format for interoperability were unsuccessful, as spatial information encoded in NIfTI cannot be fully replicated from 2D stacks. Robust handling of diverse imaging data remains an ongoing research area crucial for clinical translation of 3D modeling pipelines.

Other challenges including accurate and efficient image segmentation and model generation processes have been solved to a great degree. However, runtime efficiency on smaller hardware still remains a challenge. Advancing imaging technology and computational methods continue addressing these hurdles. Integrating 3D

modeling into clinical workflows and training professionals in model interpretation are vital for broader adoption.

Overall, this project demonstrated 3D modeling's potential to revolutionize ventriculomegaly diagnosis and management. Providing comprehensive condition insights, 3D models may improve patient outcomes and care quality. As technology evolves, integrating 3D modeling into standard clinical practice holds promise for deeper understanding and more effective treatment of neurological conditions like ventriculomegaly.

References

[1] Labrosse, Kathrin B., Johanna Buechel, Huelya Guelmez, Annkathrin Butenschoen, Heidrun Schoenberger, Eva Visca, Andreas Schoetzau, and Gwendolin Manegold-Brauer. "Presentation of ventriculomegaly at 11–14 weeks of gestation: An analysis of longitudinal data." Prenatal Diagnosis 43, no. 7 (2023): 910-918.

[2] Vahedifard, Farzan, H. Asher Ai, Mark P. Supanich, Kranthi K. Marathu, Xuchu Liu, Mehmet Kocak, Shehbaz M. Ansari et al. "Automatic ventriculomegaly detection in fetal brain MRI: A step-by-step deep learning model for novel 2D-3D linear measurements." Diagnostics 13, no. 14 (2023): 2355.

[3] Hesse, Linde S., Moska Aliasi, Felipe Moser, Monique C. Haak, Weidi Xie, Mark Jenkinson, Ana IL Namburete, and INTERGROWTH-21st Consortium. "Subcortical segmentation of the fetal brain in 3D ultrasound using deep learning." NeuroImage 254 (2022): 119117.

[4] Singh, Ayush, Seyed Sadegh Mohseni Salehi, and Ali Gholipour. "Deep predictive motion tracking in magnetic resonance imaging: application to fetal imaging." IEEE transactions on medical imaging 39, no. 11 (2020): 3523-3534.

[5] Siddique, Nahian, Sidike Paheding, Colin P. Elkin, and Vijay Devabhaktuni. "U-net and its variants for medical image segmentation: A review of theory

and applications." Ieee Access 9 (2021): 82031-82057.

[6] Fu, Mengyin, Hao Liang, Chunhui Zhu, Zhipeng Dong, Rundong Sun, Yufeng Yue, and Yi Yang. "Image Stitching Techniques Applied to Plane or 3D Models: A Review." IEEE Sensors Journal (2023).

[7] Ranzini, Marta, Lucas Fidon, Sébastien Ourselin, Marc Modat, and Tom Vercauteren. "MONAIfbs: MONAI-based fetal brain MRI deep learning segmentation." arXiv preprint arXiv:2103.13314 (2021).

[8] Benkarim, Oualid M., et al. "Toward the automatic quantification of in utero brain development in 3D structural MRI: A review." Human brain mapping 38.5 (2017): 2772-2787.

[9] Aleem, Sidra, Fangyijie Wang, Mayug Maniparambil, Eric Arazo, Julia Dietlmeier, Kathleen Curran, Noel E. O'Connor, and Suzanne Little. "Test-Time Adaptation with SaLIP: A Cascade of SAM and CLIP for Zero shot Medical Image Segmentation." arXiv preprint arXiv:2404.06362 (2024).

[10] Reddy, Digvijay, V. Bhavana, and H. K. Krishnappa. "Brain tumor detection using image segmentation techniques." In 2018 International conference on communication and signal processing (ICCSP), pp. 0018-0022. IEEE, 2018.

[11] Tyagi, Prachee, Tripty Singh, Ravi Nayar, and Shiv Kumar. "Performance comparison and analysis of medical image segmentation techniques." In 2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), pp. 1-6. IEEE, 2018.

Appendix A

Source code

A.1 Dataset Details

The dataset used throughout this project was gathered from the radiology department of Amrita Institute of Medical Science, Kochi. It consisted of fetal brain MRI images all segregated into different folders based on patient_id and plane of scan. The dataset consisted of:

- 221 transverse plane images
- 281 coronal plane images
- 243 saggittal plane images

Out of these images, we utilized 275 MRI images for training and validation of our multi-model pipeline. Further on, the generated segmented masks were utilised for the creation of volumetric data as well.

A.2 Source Code

A.2.1 Preparation of raw data for training

Ensure that the images and the binary masks are arranged in separate folders according to the hierarchy as shown below in Fig A.1:

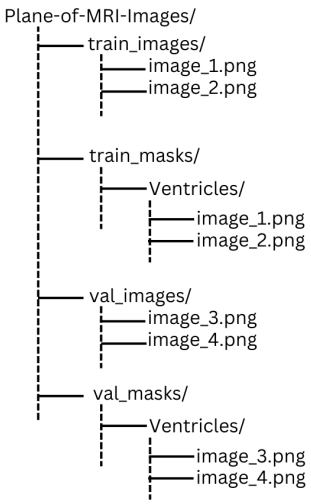

Figure A.1: Expected File Hierarchy

Also ensure that the images and masks have the same name and the same resolution. It is important to note that higher the resolution of the image, the more training time the detection model will take. So it is recommended to keep the resolution at an ideal 800x800 pixels(width x height).

```
import glob
import json
import os
import cv2

# Label IDs of the dataset representing different categories
category_ids = {
```

```
    "Ventricles": 1
}

MASK_EXT = 'png'
ORIGINAL_EXT = 'png'
image_id = 0
annotation_id = 0

def images_annotations_info(maskpath):
    """
    Process the binary masks and generate images and annotations information.

    :param maskpath: Path to the directory containing binary masks
    :return: Tuple containing images info, annotations info, and annotation count
    """
    global image_id, annotation_id
    annotations = []
    images = []

    # Iterate through categories and corresponding masks
    print(category_ids.keys())
    for category in category_ids.keys():
        print(maskpath,category,f'*.{MASK_EXT}')
        for mask_image in glob.glob(os.path.join(maskpath, category, f'*.{MASK_EXT}')):
            original_file_name = f'{os.path.basename(mask_image).split(".")[0]}.{ORIGINAL_EXT}'
            mask_image_open = cv2.imread(mask_image)

            # Get image dimensions
            height, width, _ = mask_image_open.shape
```

```
37
38          # Create or find existing image annotation
39          if original_file_name not in map(lambda img: img['
    file_name'], images):
40              image = {
41                  "id": image_id + 1,
42                  "width": width,
43                  "height": height,
44                  "file_name": original_file_name,
45              }
46              images.append(image)
47              image_id += 1
48          else:
49              image = [element for element in images if element[
    'file_name'] == original_file_name][0]
50
51          # Find contours in the mask image
52          gray = cv2.cvtColor(mask_image_open, cv2.
    COLOR_BGR2GRAY)
53          _, thresh = cv2.threshold(gray, 0, 255, cv2.
    THRESH_BINARY + cv2.THRESH_OTSU)
54          contours = cv2.findContours(thresh, cv2.RETR_TREE, cv2
    .CHAIN_APPROX_SIMPLE)[0]
55
56          # Create annotation for each contour
57          for contour in contours:
58              bbox = cv2.boundingRect(contour)
59              area = cv2.contourArea(contour)
60              segmentation = contour.flatten().tolist()
61
62              annotation = {
63                  "iscrowd": 0,
64                  "id": annotation_id,
```

```
                    "image_id": image['id'],
                    "category_id": category_ids[category],
                    "bbox": bbox,
                    "area": area,
                    "segmentation": [segmentation],
                }

                # Add annotation if area is greater than zero
                if area > 0:
                    annotations.append(annotation)
                    annotation_id += 1

    return images, annotations, annotation_id


def process_masks(mask_path, dest_json):
    global image_id, annotation_id
    image_id = 0
    annotation_id = 0

    # Initialize the COCO JSON format with categories
    coco_format = {
        "info": {},
        "licenses": [],
        "images": [],
        "categories": [{"id": value, "name": key, "supercategory":
    key} for key, value in category_ids.items()],
        "annotations": [],
    }

    # Create images and annotations sections
    coco_format["images"], coco_format["annotations"],
   annotation_cnt = images_annotations_info(mask_path)
```

```
96
97      # Save the COCO JSON to a file
98      with open(dest_json, "w") as outfile:
99          json.dump(coco_format, outfile, sort_keys=True, indent=4)
100
101     print("Created %d annotations for images in folder: %s" % (
        annotation_cnt, mask_path))
102
103 if __name__ == "__main__":
104     train_mask_path = "absolute_path_to_training_masks"
105     train_json_path = "absolute_path_to_training_images/train.json
        "
106     process_masks(train_mask_path, train_json_path)
107
108     val_mask_path = "absolute_path_to_validation_masks"
109     val_json_path = "absolute_path_to_valdidation_images/val.json"
110     process_masks(val_mask_path, val_json_path)
```

Listing A.1: Converting Binary Images to COCO JSON

```
1  import json
2  import os
3  import shutil
4  import yaml
5
6  # Function to convert images to YOLO format
7  def convert_to_yolo(input_images_path, input_json_path,
       output_images_path, output_labels_path):
8      # Open JSON file containing image annotations
9      f = open(input_json_path)
10     data = json.load(f)
11     f.close()
12
13     # Create directories for output images and labels
```

```
    os.makedirs(output_images_path, exist_ok=True)
    os.makedirs(output_labels_path, exist_ok=True)

    # List to store filenames
    file_names = []
    for filename in os.listdir(input_images_path):
        if filename.endswith(".png"):
            source = os.path.join(input_images_path, filename)
            destination = os.path.join(output_images_path,
filename)
            shutil.copy(source, destination)
            file_names.append(filename)

    # Function to get image annotations
    def get_img_ann(image_id):
        return [ann for ann in data['annotations'] if ann['
image_id'] == image_id]

    # Function to get image data
    def get_img(filename):
        return next((img for img in data['images'] if img['
file_name'] == filename), None)

    # Iterate through filenames and process each image
    for filename in file_names:
        img = get_img(filename)
        img_id = img['id']
        img_w = img['width']
        img_h = img['height']
        img_ann = get_img_ann(img_id)

        # Write normalized polygon data to a text file
        if img_ann:
```

```
44              with open(os.path.join(output_labels_path, f"{os.path.
    splitext(filename)[0]}.txt"), "a") as file_object:
45                  for ann in img_ann:
46                      current_category = ann['category_id'] - 1
47                      polygon = ann['segmentation'][0]
48                      normalized_polygon = [format(coord / img_w if
    i % 2 == 0 else coord / img_h, '.6f') for i, coord in enumerate
    (polygon)]
49                      file_object.write(f"{current_category} " + " "
    .join(normalized_polygon) + "\n")
50
51 # Function to create a YAML file for the dataset
52 def create_yaml(input_json_path, output_yaml_path, train_path,
    val_path, test_path=None):
53     with open(input_json_path) as f:
54         data = json.load(f)
55
56     # Extract the category names
57     names = [category['name'] for category in data['categories']]
58
59     # Number of classes
60     nc = len(names)
61
62     # Create a dictionary with the required content
63     yaml_data = {
64         'names': names,
65         'nc': nc,
66         'test': test_path if test_path else '',
67         'train': train_path,
68         'val': val_path
69     }
70
71     # Write the dictionary to a YAML file
```

```
72      with open(output_yaml_path, 'w') as file:
73          yaml.dump(yaml_data, file, default_flow_style=False)
74
75
76  if __name__ == "__main__":
77      base_input_path = "absolute_path_to_plane_of_MRI"
78      base_output_path = "absolute_path_to_plane_of_MRI/yolo_dataset
    /"
79
80      #Processing validation dataset (if needed) --> Skipping this
    for now
81      convert_to_yolo(
82          input_images_path=os.path.join(base_input_path, "
    resized_valid_images"),
83          input_json_path=os.path.join(base_input_path, "
    resized_valid_images/val.json"),
84          output_images_path=os.path.join(base_output_path, "valid/
    images"),
85          output_labels_path=os.path.join(base_output_path, "valid/
    labels")
86      )
87
88      # Processing training dataset
89      convert_to_yolo(
90          input_images_path=os.path.join(base_input_path, "
    resized_train_images"),
91          input_json_path=os.path.join(base_input_path, "
    resized_train_images/train.json"),
92          output_images_path=os.path.join(base_output_path, "train/
    images"),
93          output_labels_path=os.path.join(base_output_path, "train/
    labels")
94      )
```

```
95
96      # Creating the YAML configuration file
97      create_yaml(
98          input_json_path=os.path.join(base_input_path, "
    resized_train_images/train.json"),
99          output_yaml_path=os.path.join(base_output_path, "data.yaml
    "),
100         train_path="VM/train/images", #Change these when actually
    training
101         val_path="VM/valid/images",   #Change these when actually
    training
102         test_path='../test/images'  # or None if not applicable
103     )
```

Listing A.2: Converting COCO JSON to YOLOv8 compatible dataset

A.2.2 Training ventricle detection model

```
1  from ultralytics import YOLO
2  from matplotlib import pyplot as plt
3  from PIL import Image
4
5  #Instance
6  model = YOLO('yolov8n-seg.yaml')  # build a new model from YAML
7  model = YOLO('yolov8n-seg.pt')  # Transfer the weights from a
       pretrained model (recommended for training)
8
9  # Define the file path
10 file_path = "absolute_path_to_data.yaml"
11
12 # define number of classes based on YAML
13 import yaml
14 with open("absolute_path_to_data.yaml", 'r') as stream:
```

```
    num_classes = str(yaml.safe_load(stream)['nc'])

#Define a project --> Destination directory for all results
project = "path_to_store_results/YOLOv8/MRI_plane_/yolo_dataset/
    results"
#Define subdirectory for this specific training
name = "location_to_store_results_within_project_folder"

# Train the model
results = model.train(data='absolute_path_to_data.yaml',
                      project=project,
                      name=name,
                      epochs=200,
                      patience=0, # setting patience=0 to disable
    early stopping.
                      batch=4,
                      imgsz=800)
```

Listing A.3: Training Ventricle Detection Model

After training the model, the model will be saved in the result folder of the project as two separate versions (namely best.pt and last.pt for the best model and latest model after training), it is recommended to prefer the best version of the model over the latest one.

A.2.3 Multi-Model Pipeline

The SAM VIT-H checkpoint should be downloaded using the shell command:

```
!wget https://dl.fbaipublicfiles.com/segment_anything/
    sam_vit_h_4b8939.pth
```

Listing A.4: Shell Command to download SAM VIT-H Checkpoint

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from ultralytics import YOLO
from segment_anything import sam_model_registry,
    SamAutomaticMaskGenerator, SamPredictor

yolo_model = YOLO('absolute_path_to_YOLOv8_model.pt')
results = yolo_model.predict(source='absolute_path_to_input_image.
    png', conf=0.20)

sam_checkpoint = "absolute_path_to_sam_vit_h_4b8939.pth"
model_type = "vit_h"
sam_model = sam_model_registry[model_type](checkpoint=
    sam_checkpoint)
predictor = SamPredictor(sam_model)

segmented_images = []

for result in results:
    boxes = result.boxes
    bbox = boxes.xyxy.tolist()

for i in range(len(bbox)):
    bbox = boxes.xyxy.tolist()[i]
    image = cv2.cvtColor(cv2.imread('absolute_path_to_input_image.
    png'), cv2.COLOR_BGR2RGB)
    predictor.set_image(image)
    input_box = np.array(bbox)

    masks, _, _ = predictor.predict(
        point_coords=None,
        point_labels=None,
```

```
        box=input_box[None, :],
        multimask_output=False,
    )

    segmentation_mask = masks[0]
    binary_mask = np.where(segmentation_mask > 0.5, 1, 0)

    black_background = np.ones_like(image) * 0
    mask_color = np.array([255, 255, 255])


    new_image = black_background * (1 - binary_mask[..., np.
    newaxis]) + mask_color * binary_mask[..., np.newaxis]
    segmented_images.append(new_image)

overlayed_image = np.zeros_like(image)

for new_image in segmented_images:
    overlayed_image = overlayed_image * (1 - new_image) +
    new_image


plt.subplot(1, 2, 1)
plt.imshow(image)
plt.axis('off')
plt.title('MRI')

plt.subplot(1, 2, 2)
plt.imshow(overlayed_image.astype(np.uint8))
plt.axis('off')
plt.title('Predicted Mask')

```

```
plt.show()
```

Listing A.5: YOLOv8+SAM Multi-Model Pipeline

A.3 Marching Cubes

```
# Marching Cubes function
def marching_cubes_3d(image_stack, threshold=0.25, voxel_spacing=(0.5, 0.5, 0.5)):
    binary_images = image_stack > threshold
    try:
        verts, faces, _, _ = measure.marching_cubes(binary_images, level=0, spacing=voxel_spacing)
        return verts, faces
    except RuntimeError as e:
        print(f"Error: {e}")
        return None, None

# Smoothing function
def smooth_surface(verts, sigma=1, axis=0):
    if verts is not None:
        smoothed_verts = np.copy(verts)
        smoothed_verts[:, axis] = gaussian_filter1d(verts[:, axis], sigma=sigma)
        return smoothed_verts
    return None
```

Listing A.6: Marching Cubes and Smoothening function

A.4 Visualisation

```
def plot_3d_surface_vtk(verts, faces, threshold):
    # Create a VTK PolyData object
    poly_data = vtk.vtkPolyData()

    # Create VTK points
    points = vtk.vtkPoints()
    points.SetNumberOfPoints(len(verts))
    for i, vert in enumerate(verts):
        points.SetPoint(i, vert[0], vert[1], vert[2])
    poly_data.SetPoints(points)

    # Create VTK cells (triangles)
    polys = vtk.vtkCellArray()
    for face in faces:
        polys.InsertNextCell(3, face)  # Assuming faces are
   triangles
    poly_data.SetPolys(polys)

    # Create VTK actor
    mapper = vtk.vtkPolyDataMapper()
    mapper.SetInputData(poly_data)

    actor = vtk.vtkActor()
    actor.SetMapper(mapper)

    # Create VTK renderer and render window
    renderer = vtk.vtkRenderer()
    render_window = vtk.vtkRenderWindow()
    render_window.AddRenderer(renderer)

    # Create VTK render window interactor
    render_window_interactor = vtk.vtkRenderWindowInteractor()
    render_window_interactor.SetRenderWindow(render_window)
```

```
33
34      # Add actor to the renderer
35      renderer.AddActor(actor)
36
37      # Set up the camera
38      renderer.GetActiveCamera().Azimuth(30)
39      renderer.GetActiveCamera().Elevation(30)
40
41      # Reset the camera to show the entire scene
42      renderer.ResetCamera()
43
44      # Set background color
45      renderer.SetBackground(1.0, 1.0, 1.0) # (1,1,1) denotes
    maximum intensity for all RGB colors
46
47      # Render the scene
48      render_window.Render()
49
50      # Start the render window interactor
51      render_window_interactor.Start()
```

Listing A.7: Visualisation through VTK