*A project report on*

**DEEP LEARNING FOR LIVER SEGMENTATION**

*Submitted in partial fulfillment for the award of the degree of*

**BACHELORS OF TECHNOLOGY (B.Tech.)**

***By***

**AMARNATH CHIGURUPATI (21BCE8893)**

**MADHURI SIRASANAGANDLA (21BCE8217)**

**ANKIT KOMMALAPATI (21BCE8228)**

****

**SCOPE- School of Computer Science and Engineering**

**May,2025**

**DECLARATION**

I hereby declare that the thesis entitled “DEEP LEARNING FOR LIVER SEGEMENTATION”

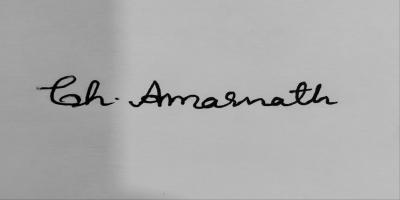
submitted by me, for the award of the degree of Bachelors of Technology (B.Tech.), VIT

is a record of bonafide work carried out by me under the supervision of Dr. Siddique Ibrahim Peer Mohammed.

I further declare that the work reported in this thesis has not been submitted

and will not be submitted, either in part or in full, for the award of any other degree

or diploma in this institute or any other institute or university.

Place: Amaravati 

Date: 06-05-2025 Signature of the Candidate

**CERTIFICATE**

This is to certify that the Senior Design Project titled “DEEP LEARNING FOR LIVER SEGMENTATION” that is being submitted by

Amarnath Chigurupati (21BCE8893) is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. Siddique Ibrahim Peer Mohammed

Guide

**The thesis is satisfactory / unsatisfactory**

**Internal Examiner External Examiner**

**Approved by**

**PROGRAM CHAIR DEAN**

**ABSTRACT**

Today, Liver tumors and liver cancers are one of the major causes of liver-related deaths. Detection in early stages and improving the accuracy of the tumor's location are important for effective treatment planning and enhancing patient outcomes. Manual segmentation for finding liver and the abnormality regions using Computed Tomography (CT) scans is time-consuming, more complex, and has a high chance of human-biased errors, which might cause inaccurate results and also which will eventually be the reason for negative outcome for the patient.

Improved technology in aspects like deep learning has given results that can be used in real time, particularly while automating medical image segmentation tasks and reducing human-biased errors.

Our project develops a deep learning model that helps in segmenting liver from **Computed Tomography (CT)** scans and also helps in finding the abnormalities, if any. For this, we adopted a U-net-based Convolutional Neural Network architecture along with Resnet 34 and trained the model using the **Liver Tumor Segmentation Benchmark (LiTS)** dataset and achieved high, promising accuracy, which can be used for identifying liver and abnormalities in real-world scenarios.

To make our model usable, accessible, and user-friendly, we used a Python package **PyQT5**-based application. This application makes the user give a CT scan of liver slices in JPEG format, and then our model comes into action and completes the liver segmentation and abnormality. The segmented image and the abnormality found will be kept side by side to understand the region of abnormality.

Our model's performance is evaluated using various standard metrics like Dice coefficient and Intersection over Union (IoU). This developed model could assist radiologists by reducing the complexity and dependency on highly complex equipment, and helps in minimizing errors and improving the accuracy of liver diagnosis, and helps in getting good outcomes.

**ACKNOWLEDGEMENT**

It is my pleasure to express with deep sense of gratitude Dr. Siddique Ibrahim Peer Mohammed, Associate Professor, SCOPE, VIT-AP, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Deep Learning.

I would like to express my gratitude to Chancellor- G Vishwanathan Sir, Vice President- Sankar Viswanathan Sir and Sekhar Viswanathan Sir, Vice Chancellor- SV Kota Reddy Sir, and Dean- Madhusudhan Rao Sir, SCOPE, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Program Chair- Saroj Kumar Panigrahy Sir, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati Amarnath Chigurupati

Date: 06-05-2025 Name of the student

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Title** | **Page No.** |
|  | Abstract | 4 |
| 1 | Introduction | 8 |
| 2 | Objective of the Project  2.1 Problem Statement  2.2 Motivation of the project | 11  14  17 |
| 3 | Literature Survey | 19 |
| 4 | Hardware and Software requirements | 23 |
| 5 | Proposed Methodology  5.1 Overview  5.2 Dataset description  5.3 Preprocessing techniques  5.4 Model Architecture  5.5 Model Training  5.6 Graphic User Interface | 25 |
| 6 | Project Flow/ Framework of the Proposed System  6.1 Flow of project  6.2 UML diagrams | 31 |
| 7 | Results and Discussions | 38 |
| 8 | Conclusion | 44 |
| 9 | Future works | 45 |
| 10 | References | 48 |
|  | Appendix | 50 |
|  | Source code with screenshots | 53 |

# LIST OF FIGURES

|  |  |  |  |
| --- | --- | --- | --- |
| S. No | Section | Figure description | Page number |
| 1 | 3.1 | Literature survey | 22 |
| 2 | 5.4 | Model Architecture | 26 |
| 3 | 6.2 | Use case diagram | 34 |
| 4 | 6.2.2 | Sequence diagram | 35 |
| 5 | 6.2.3 | Class diagram | 36 |
| 6 | 6.2.4 | Activity diagram | 37 |
| 7 | 7.1 | LiTS dataset | 38 |
| 8 | 7.2 | Trend analysis | 39 |
| 9 | 7.3 | Evaluation metrics | 40 |
| 10 | 7.4.1 | GUI | 41 |
| 11 | 7.4.2 | Load user given image | 41 |
| 12 | 7.4.3 | Status of prediction | 42 |
| 13 | 7.4.4 | Contact information | 42 |
| 14 | 7.4.5 | Output | 45 |

# INTRODUCTION

**1.1 Overview**

Medical imaging had changed medical healthcare tremendously and helped doctors to diagnose quickly and provide an early solution for patient and helps in saving many lives. Out of several medical segmentation scans, computed tomography (CT) had gained significant importance especially for diagnosis of liver-based diseases which provides a detailed image which benefits for better analysis.

However, analyzing liver-based CT scans is not an easy task as liver and tumors might be in various sizes, various shapes and appearances which makes manual identification a difficult task. This process is time consuming and prone to bias error and also might causes differences in doctor opinions which is the reason fast and accurate diagnosis required.

With rapid development of technology, Artificial Intelligence and deep learning have gradually gaining importance in the healthcare industry. Convolution Neural Networks which is a part of deep learning algorithm was providing excellent outcomes in many tasks like image segmentation.

For this we had build a model implementing several deep learning architectures that will automate the segmentation of liver and its abnormality region. Our major goal is to diagnose faster, more easily, accurately, and performs more efficiently than manual segmentation which helps for research in real-time purposes.

**1.2 Importance of Liver Segmentation**

Liver plays a major role in our body such helping in body metabolism, detoxification and producing proteins which are necessary for our body. Due to various reasons liver diseases like tumors, hepatocellular carcinoma (HCC) is gradually increasing significantly which was becoming a potential risk and also death rate caused by liver-based diseases were increasing, so early diagnosis is required to save lives.

Through early and accurate diagnosis, we can identify the diagnosis required and proper treatment like surgeries, transplant and required therapies are done.

By automating liver segmentation, abnormalities can be found in less time.

Through this, we could help doctors in several ways:

• By providing organ size and positions, doctors can prepare for surgery accordingly.

• The doctor will categorize the tumor and provide personalized treatment for the patient.

• A doctor can provide personalized treatment depending on the overall patient’s health condition.

• Doctors can perform medical research and come up with new therapies and can also help doctors in identifying and understanding diseases in quicker and better way.

**1.3 Role of Artificial Intelligence in Medical Imaging**

Artificial Intelligence has gaining significant importance in several medical healthcare industries. As AI help in diagnosing faster, accurate and more efficient there is no chance or less chance of human errors which helps in making whole diagnosis and personalized treatment faster and accurate.

Deep Learning which is a sub union of AI is very efficient and powerful because it automates segmentation by finding the patterns and can also take large sets of images and find feature and can get very accurate which can sometimes do better than human technicians.

In our project, Our AI model detect liver and abnormality quicker and accurate which not only saves doctors time but gives doctors to increase the life of the patients by saving them by providing reliable or promising results.

**1.4 Project Scope**

Our main goal is to create and validate an AI based model which can automate and perform segmentation of liver and its abnormal region from CT scans which provides an efficient and quicker solution.

Our project includes:

• Data Collection from reliable datasets like Liver Tumor Segmentation Benchmark (LiTS) and preprocessing was done on the dataset.

• Choosing and training the model with suitable deep learning algorithm which provides an efficient solution.

• Providing a user-friendly interface application so model can make new predictions.

• We evaluate our model with standard metrics like dice coefficient and IoU.

• Making the model reliable for real-time through a user-friendly interface.

Our aim is to create an AI based model which is flexible and can be integrated to any part of body and making sure that the model is accurate and user friendly to use in hospitals and research centers.

**1.5 Challenges**

Even though our deep learning AI model is efficient and advanced while segmenting liver and abnormalities from CT scans, there are some challenges, like:

• Tumors that were present in liver are of different size, different shape and present in different appearance.

• If the CT scan image consists of more noise and artifact then possibility to confuse the model will higher which will eventually lead to improper results.

• If liver tissues and organs which were present nearby have similar intensity values, differentiating them becomes difficult.

• Availability of annotated medical data is limited so training the model and performing for real-time tasks was a challenge.

• Variation in imaging protocols across machines and hospitals.

To overcome we need proper model design which should be accurate, preprocessing should be done smartly and proper fine tuning is required.

**1.6 Structure of Report**

The report is organized as follows:

• Chapter 1 explains the ideology, project background and its goals, outcomes, and challenges faced.

• Chapter 2 explains the objectives of our project

• Chapter 3 explains the problem that we are trying to solve.

• Chapter 4 talks about the motivation behind our project.

• Chapter 5 researches past work done and reviews existing work

• Chapter 6 describes how the project was done, discusses about training done, and talks about the user-friendly interface.

• Chapter 7 describes the project and talks about future improvements that can be made for our project

## 2.1 PROBLEM STATEMENT

Medical imaging had become most important part of decision making, especially while giving treatment life threatening diseases like cancer and HCC. CT (Computed Tomography) scans are one of the most widely used for examining liver which provides clear cut details of internal organs. But accurate interpretation and analysis was depended mainly on efforts provided by skilled radiologists.

Liver segmentation is one of the most important tasks for any diagnosis, whether finding a tumor or cancer, or any HCC. Manual segmentation is time-consuming and requires skilled professionals or people who have better clinical experience. Moreover, there is high probability of human error and many inconsistencies, which lead to misinterpretation, which can be hazardous to the patients. The growing number of patients proportionally leads to an increase in several scan volumes, which is becoming infeasible and getting difficult for accurate diagnosis and personalized treatment planning.

**Existing Limitations**

Despite advancements in technology, many limitations remain while performing liver segmentation workflows such as:

• Radiologist should examine and each slice of segmented liver should be carefully examined which takes more time.

• Different radiologists might interpret the scan differently which can lead to inconsistent results.

• In early stages tumor will be small and also sometimes has low contrast intensities which can lead to misinterpretations.

• In many places there was shortage of radiologists who are qualified who were suitable enough for diagnosis.

Commercial solutions for automated segmentation are very expensive which makes them not usable and also demand high end computation and infrastructure or integrated directly medical proprietary software systems. Also using such kinds of tools can’t be adaptable to local imaging conditions.

**Technological challenges**

From a technical perspective, liver and abnormality segmentation is difficult for the following reasons:

• Tumors are of different sizes, shapes, and appearances.

• Liver tissues may have adjacent intensities, which might cause difficulty in identifying the boundaries.

• Abnormal regions occupy a tiny portion of the liver when compared to regions of the liver, distinguishing from the background, which affects training.

• Availability of datasets is limited due to various concerns like privacy, complexity, and cost.

**Problem Definition:**

Considering clinical and technical challenges, the problem statement is defined as:

“To design a deep learning-based model which segments liver and abnormal region accurately and efficiently by taking CT scan images as input and integrate the model in real-time user-friendly GUI application for real time and practical use for diagnosis professionals”.

Our model satisfies:

• Reduce the dependency on manual segmentation.

• Provide promising accuracies that can be used for practical purposes.

• Run efficiently without any specialized hardware.

• Should offer a user-friendly interface with no requirement for programming.

**Project scope and boundaries:**

The system is designed for performing 2D segmentation of CT slices accurately but project has several boundaries:

• 3D reconstruction and volume rendering

• Can’t be directly integrated to hospital PACS system (future scope).

• Since only segmentation mask was provided radiologist can only predict but can generate full detailed report.

Our projects set a strong foundation for medical support by predicting liver diseases efficiently and displays how AI can be address specific and define medical problems clearly.

## 2.2 MOTIVATION FOR THE PROJECT

Liver diseases are becoming a global concern, increasing rapidly and taking many lives, and becoming a huge threat. Diseases like Hepatocellular carcinoma (HCC), tumors, and cancer-based diseases are treated in advanced stages due to delayed diagnosis and detection. So, early detection and diagnosis are required for improving the survival rates. Our motivation for the project is primarily to reduce the burden using AI and a technology that can assist researchers without replacing them.

Most commonly, computed tomography (CT) is used for the diagnosis of liver-based diseases. Sometimes, while interpreting CT scans, identifying the tumor and the boundaries of the liver region from its background becomes time-intensive and requires expertise in diagnosing. Due to an increase in liver disease-related patients and limited access to expertise, diagnosing is becoming a tough task.

**Power and potential of AI in the fields of Medical Imaging**

In today’s world, rapid advancement of technology is happening with the help of AI, and it is being used in various medical fields. Deep Learning majorly uses convolutional neural networks, showing promising results in solving complex image-related problems like detection, identification, and segmentation.

When those techniques are integrated into a model and applied to medical images, which can automate and provide results in a shorter amount of time helps in reducing the burden on medical researchers and diagnostic expertise.

• AI will find the region of the liver and the abnormality with reliable accuracy and precision

• Identification of tumors can be done without any bias or fatigue.

• Segmentation can be done in a shorter amount of time, making it implementing practical for assistance

This process enabled our team to take part in the medical field and make the model and user-friendly interface operable in real-time and helping health care professionals.

**Practical limitations in existing models**

There are several studies done widely using AI, but only a few models are reliable for hospitals or researchers for real-time usage. Most models are:

• Requires more technical knowledge

• Not flexible, which might cause an issue during integration

• Due to various concerns like privacy, there is limited availability of datasets that are not generalized

This observation led us to deliver a model that can perform segmentation of liver and its abnormal regions efficiently, and doesn’t require any technical knowledge.

**Interdisciplinary Learning and Educational Motivation**

This project enabled us to implement theoretical knowledge in real-world aspects. We integrate:

• Concepts of Machine Learning and Deep Learning (U-Net, ResNet-34, and FastAI library)

• A few concepts of medical imaging, like understanding CT scan intensities

• Developed a user-friendly interface with the help of the PyQt5 Python package

This project motivated us to work and come up with innovations by combining of healthcare sector with Artificial Intelligence and making it reliable and ethical.

**Responsibility and Scope**

In many parts of the world, majorly rural areas or hospitals that have fewer or no facilities, and where good expertise is scarce. Delays in diagnosis for better accuracy in tumor prediction require more time to get used to. Building a cost-efficient model, an offline GUI interface that can assist in liver segmentation and the tumor region.The social responsibility -applying our engineering skills, which contributes to a better society- is a major motivation behind doing the project.

Our model helps in:

• Reducing diagnosis time

• Improve outcomes with better treatment and early prediction

# LITERATURE SURVEY

# Segmentation of the liver and its abnormal regions from CT scans is widely used for research on computer-based diagnosis and segmentation automation. Segmentation plays a key role in monitoring and proper early detection and helps doctors create personalized treatment plans. Several methods, like traditional image processing and segmentation techniques, are not suitable for current-day scenarios because of the increase in population suffering from liver disease and improper identification due to bias or any other human-based errors. We propose an AI model that automates the segmentation and helps to prevent the limitations caused by traditional imaging methods. We provide a detailed literature survey explaining the evolution of liver segmentation and the development of CNN, making it suitable for medical imaging, and different strategies for preprocessing and other methodologies.

# Traditional methods

# Before the integration of AI into the medical field, there were several approaches for segmentation, like:

# • Thresholding: Organs are identified by pixel intensity at different levels. This thresholding fails when parts have similar intensity values or when there is any overlap it hard to distinguish between neighboring organs.

# • Growing region and active contours: This method is used to expand the image by expanding seed points, but it is sensitive to noise and the placement of the CT scan.

# • Atlas-based segmentation: It is an anatomical model that is used with data from the patient. These methods require complex computation.

# Traditional methods have less advantage because of time-consuming, limited adaptability, limited accuracy, and require high-demanding computation.

# Deep Learning for Medical Image Segmentation

# With the advancement of technology and the implementation of AI in medical fields, image analysis has seen various advancements, which help the required AI model to enable its feature representation directly from the data.

# 1. Convolution Neural Networks:

# CNN laid the foundation for image analysis. Major reasons for its success are:

# • Several local receptive fields can capture spatial hierarchies

# • In CNN, parameters can be shared, which helps in reducing the complexity

# • In CNN, multiple layers help in feature abstraction

# While performing segmentation tasks, CNN makes each pixel into several categories, making for distinguish organ boundaries in CT scan images.

# 2. U-Net architecture:

# U-Net is used most widely for medical imaging and has gained huge importance for its accuracy and precision. It was introduced by Ronneberger et al. in 2015. U-net consists of an encoder and a decoder with a connection between layers. It has many advantages, like:

# • Efficient for training small datasets

# • Spatial information can be preserved and has higher flexibility than CNN

# • It could work with any type of dataset, which can be very small or robust.

# There are several variants in U-Net architecture that are introduced based on the requirements:

# • Attention U-Net: Adds mechanisms to focus mainly on the research region, like the tumor region.

# • Residual U-Net: Several gradient blocks are used for better gradient flow

# • 3D U-Net: It operated mainly on volumetric data for a better understanding of inter-slice context.

# • U-Net++: Skip connections are used in a nested way for better extraction of features

# Challenges faced while segmenting the liver from CT scan images

# Abnormality segmentation has several segmentation challenges:

# • Abnormality can be of various sizes, shapes, and appearances

# • Some abnormalities are contrasted poorly and can’t be seen properly in a CT scan

# • Imbalanced classes, which make training difficult.

# Researchers identify and solve problems by:

# • Instead of cross entropy, Dice loss or Focal loss are used

# • Data is augmented with flips, rotations, and shifting intensities for better identification of abnormality

# • Researchers use multi-task learning where both liver and abnormality are predicted

# Preprocessing strategies

# Preprocessing is one of the most important tasks to be done before training. It plays one of the key roles in getting good outcomes. For preprocessing, there are some techniques used, such as:

# Intensity windowing: CT scans have a wide range of pixel intensity values from -1000(air) to 3000+ for bone. For liver segmentation, for better visibility window should be of width 150 and breadth 30.

# Histogram Equalization: We use histogram equalization and contrast-limited adaptive (CLAHE), which helps in improving contrast and enhancing the details in the image.

# Channel Stacking: Researchers create pseudo-RGB images:

# • Window view

# • Histogram equalized version

# • Normalized mapping based on intensities

# Chanel stacking helps to improve a model pretrained on datasets like ImageNet.

# Segmentation datasets:

# Liver Tumor Segmentation Challenge (LiTS):

# • A publicly available dataset that consists of more than 100 CT scans.

# • It was annotated by people who are experts in radiology

# • It has segmentation masks of liver and tumors.

# • For literature review LiTS dataset is used frequently

# Evaluation metrics:

# Dice Similarity Coefficient (DSC):

# It helps bridge between gap between prediction and ground truth to give better results. DSC > 0.9 is considered for real-time performance and can be tested for practical usage.

# Intersection over Union (IoU):

# It is the ratio of the predicted region to the actual region. It gives better metrics than Dice

# Hausdorff Distance:

# It is used to measure the spatial difference between the predicted and actual boundary, which helps in identifying anatomical errors.

# Real Time Deployment:

# Many researchers focus on training models but fail to tell how well it is usable. For real-world usability:

# • Making the GUI easy to use

# • PyQt5, Tkinter are integrated with an AI model and make the GUI into usable application

# • For getting results .jpg/jpeg format was given as input, and then the prediction is performed, and the output was given highlighting the liver and its abnormal region.

# Related work

# 

# 

Fig 3.1 Literature survey

# The above shows the efficiency of deep learning models, which are used for segmentation and explain the importance of architectural changes, using loss functions, and using various techniques like augmentation for better accuracy.

## HARDWARE AND SOFTWARE REQUIREMENTS

## Hardware:

1. **Computer or Server:**

* A computer with 8 GB RAM and a multi-core processor is sufficient for running the GUI-based prediction tool.
* For managing PyQt-based GUI operations, we need a multi-core CPU, which helps in loading the model and handling the image preprocessing in an efficient way.

1. **GPU for Model Training:**

* For the model training phase, an NVIDIA GPU is highly recommended. The training was performed on a T4x2 GPU setup on Kaggle, which significantly accelerated the training process.
* GPUs like NVIDIA T4, RTX 2060, or GTX 1080 provide excellent speedups during deep learning model training, which is especially beneficial for liver segmentation tasks involving 2D CT scan slices.

1. **RAM and Storage:**

* 8 GB of RAM is at least required, with 16 GB or more is generally required for loading the data and processing the batches.
* Around 20–30 GB of free disk space is advised to store the liver segmentation datasets, pre-processed images, trained models, and prediction outputs.

1. **Operating System:**

* The application is compatible with Windows 10 or more and Linux 20 or more. Windows was primarily used for the GUI (working.py), leveraging libraries like PyQt5.

## Software:

1. **Python Programming Language:**
   * Pythonprogramming language was used in our project.
   * It allows for seamless integration of multiple frameworks and tools.
   * Implementation of the segmentation pipeline was carried out in Python (compatible with versions 3.7 through 3.9). Python is an ideal choice for deep learning, medical imaging, and GUI application development.

## Deep Learning Libraries:

* + FastAI was used to train and load the liver segmentation models.
  + FastAI, Torch, Torchvision -These libraries provided easy-to-use abstractions for building, training, and deploying the deep learning model.
  + They offer high-level APIs for model development, facilitating the implementation of complex architectures.

## GUI Development:

* + The PyQt5 library was used to design the standalone graphical user interface (GUI) application for running liver segmentation predictions locally.
  + It handled file uploads, model loading, prediction, and result display within a user-friendly environment.

## Image Processing and Visualization:

* + Libraries such as OpenCV, PIL (Pillow), Matplotlib, and NumPy were essential for reading medical images, preprocessing them into RGB/Grayscale formats, and visualizing the predicted segmentation masks.
  + Matplotlib is used for data visualization, providing insights into data distribution, trends, and model performance.
  + These tools aid in debugging, analyzing model behavior, and communicating results.

1. **Medical Imaging Libraries:**

* The project utilized SimpleITK and Nibabel library for handling the medical imaging formats like NIfTI files during the preprocessing of these images, although the final GUI expects standard .jpg images as inputs for prediction.

## Operating System:

* + Linux or Windowsoperating systems provide the necessary environment for deploying machine learning applications.

1. **Development Tools:**

* **Jupyter Notebook** (for initial model experimentation and training visualization)
  + **Git/GitHub** (for version control and collaboration)

# PROPOSED METHODOLOGY

**5.1 Overview:**

The proposed methodology has several steps, like dataset preparation, preprocessing, training, and integration with a GUI application. The LiTS dataset has CT scan images in grayscale, these images are converted into 3 channel RGB images. This preprocessing includes steps like histogram scaling, normalization, intensity cropping and windowing. These are done so that the components of the CT scan are highlighted clearly, with the main focus on the liver and tumors, and resulting in the perfect output segmentation mask.

The segmentation mode was done with the U-Net architecture implemented through the FastAI framework. Predefined ResNet 34 weights, which are pretrained on ImageNet, act like an encoder that helps in obtaining the required features from the required input image. U-Net consists of both encoder and decoder, however with combining the ResNet-34 encoder we generate the required segmentation mask.

The architecture that we used for training is a combination of U-Net and ResNet-34. SaveModelCallback is used to save the model metrics after each epoch and also based on the validation scores and performance the best model is saved.

**5.2 Dataset description:**

The dataset that is used in our project is Liver Tumor Segmentation (LiTS).

We used this dataset as an input dataset for training, testing, and validating the model. A collection of 131 CT images is provided, with each image annotated to indicate liver and tumor regions through segmentation masks. It has corresponding labeled masks for liver and tumor regions. Each scan consists of various number of axial slices and this helps in understanding the difference between the liver region and tumor areas and the rest is the background. These were generated by certified radiologists, making them more accurate and reliable in real-time. The dataset was divided into a training, testing, and validation ratio of 80:10:10, which helps in maintaining balance between tumor and liver volume across all samples. Since the model architecture is compatible with 2D slices, preprocessing is done on the dataset 3D images to make them fit into the model.

**5.3 Preprocessing techniques:**

The CT scan images exhibited large dimensions and varying intensity levels.

The below preprocessing steps are followed to normalize them:

* Intensity clipping: Applied windowed technique of width: 150 and level: 30 so that the focus is on the intensity values of the liver.
* Normalization: Normalization involves adjusting the pixel values so that the minimum is 0 and the maximum is 1, resulting in a range of [0, 1].
* Multi-Channel conversion: The images of histogram equalized and windowed images are combined into 3 channels to make it as an RGB input for the model.

Techniques such as windowing and histogram equalization were applied to the original nii images to help isolate the liver and tumors using the provided masks in the dataset. These nii files were then transformed into RGB windowed JPG files, which were subsequently used to train the model, enhancing its performance.

**5.4 Model Architecture**

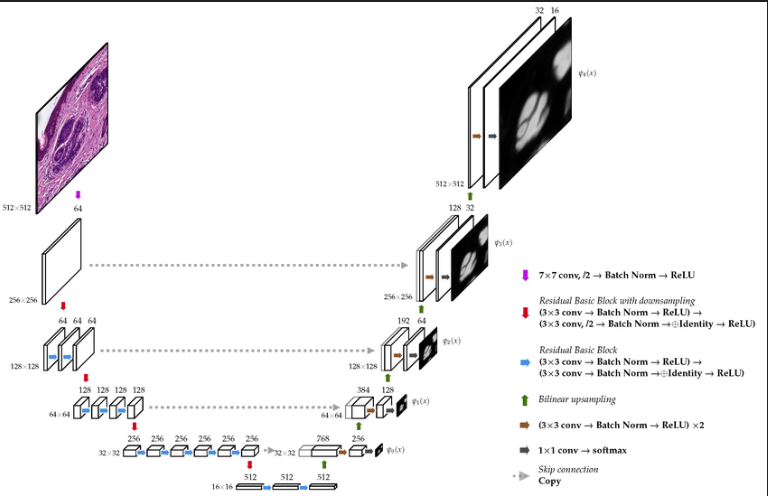


Fig. 5.4 Model Architecture

A ResNet-enhanced U-Net was selected for the segmentation pipeline, combining the strengths of both architectures for better performance. This particular ResNet-34 model has been pretrained using the ImageNet dataset.

The model’s architecture is organized into these primary sections:

* Encoder
* ResNet-34 strikes a balance between depth and efficiency, making it apt for deployment on CPUs.
* The encoder down samples the path by using the convolutional layers and max-pooling.
* Decoder
* It is essential for maintaining the edge details.
* It up samples the feature maps and uses skip connections to integrate the low-level encoder features.
* Activation
* ReLU activations which are then followed by the softmax output for the multi class segmentation (background, liver, tumor)
* Loss Function
* To address the class imbalances, Cross Entropy is combined with Dice Loss.
* Output Layer
* A convolutional layer with a 1×1 configuration translates the features into a probability distribution across three classes.

The model architecture is primarily based on FastAI, with fine-tuning performed using the SaveModelCallback feature to enable automatic checkpointing during training.

**5.4.1 Advantages of U-Net + ResNet Encoder:**

U-Net is ideal for segmentation tasks. Its advantages are: It has encoder-decoder structure where the encoder extracts the high-level features through down sampling while the decoder reconstructs the segmentation task. This makes sure that the model learns both liver and also local details of the tumor boundary. Even with small datasets, U-Net performs well through the use of transfer learning and augmented data. U-Net’s skip connections allow the model to retain the low-level spatial features so that the object boundaries are preserved. It performs well with imbalanced class which is very often in medical images where the region of interest like tumors may occupy a very little portion of the image.

ResNet-34 uses residual connections and enables models to learn deeper representations without losing the information that is present in very deep layers. By using the pretrained ResNet-34 model we are leveraging the transfer learning which reduces the amount of training data that is required and makes it suitable for the datasets that have limited samples of images. ResNet-34 also maintains a balance between depth and efficiency, this ensures that inference can be performed even with limited computational resources like CPUs.

The combination of U-Net with ResNet-34 encoder is designed to handle the complexities of medical images ensuring high accuracy when segmenting critical regions like the liver and tumors. This hybrid architecture benefits from the precise localization ability of the U-Net and also the extraction capacity of the ResNet.

This setup can also be scaled to work with other types of medical imaging such as CT scans, MRI, or ultrasound, making the system versatile for various diagnostic applications. It is also compatible with modern techniques like test-time augmentation and ensembling which can further boost the performance of the segmentation.

This combination of U-Net’s encoder decoder structure with the ResNet’s feature extraction results in high quality segmentation output with precise liver and tumor regions. Once trained, this system can perform inference in real-time, offering fast and reliable predictions for medical professionals to use in the clinical environments. By using the combination of U-Net and ResNet encoder, we can ensure the model’s performance is measured and improved using metrics like Dice Score, Foreground Accuracy and IoU (Intersection over Union). It allows easy integration of this system into the clinical decision support systems and can also be fine tuned when the new samples of data become available.

**5.5 Model Training**

**5.5.1 Training details**

* Framework:

This model was developed using the FastAI library, which enables efficient deep learning workflows.

* Model Architecture:

A U-Net Segmentation model was created with a ResNet-34 encoder. This improves convergence speed and generalization.

* Learning rate:

A cyclic learning rate schedule was used with a maximum learning rate of 1e-3, so that it escapes sharp minima and converges better.

Hardware:

Training is done on Kaggle’s GPU environment which provides ample computational power for high resolution image data.

* Batch size:

A small batch of 8 slices per batch was chosen and the model was trained for 8 epochs ensuring sufficient training and also monitoring for potential overfitting.

* Model checkpointing:

An automated saving mechanism was used to store the best-performing model as sdp\_best\_seg\_model.pkl, determined by the highest validation Dice score. This ensures that best model is saved so that performance is stored even if later epochs degrade performance.

* Custom metrics:

Two metrics, foreground accuracy and custom\_foreground\_accuracy, were designed to assess prediction accuracy by focusing exclusively on the foreground class, excluding background regions. Train\_losses.npy, test\_losses.npy, train\_dice\_score.npy, test\_dice\_score.npy automatically saves the metrics arrays at the end of the training for later analysis.

* Custom dice score:

The Dice coefficient was employed to evaluate how closely the predicted segmentation matched the actual ground truth, measuring the extent of overlap. It was implemented to handle the 2D and 3D data shapes by appropriate tensor reshaping and one-hot encoding.

* Fine-Tuning:

The model was fine tuned for 8 more epochs using weight decay regularization so that overfitting is prevented.

**5.5.2 Advantages of this setup:**

Efficient training: Combination of cyclic learning rates and Adam optimizer enables fast and stable training.

Metric centric approach: It ensures that model is evaluated meaningfully for segmentation on both accuracy and dice scores.

Robust: It avoids training interruptions by custom callbacks.

Reliable: Checkpoints ensure best model is available even if fine tuning degrades performance.

**5.6 Graphical User Interface (GUI)**

To improve accessibility, a desktop interface was created using PyQt5, allowing users to interact with the trained deep learning model in real time.

The graphical user interface (GUI) is designed as a single-window application offering several interactive features:

* Users can browse and upload CT scan images in .jpg format.
* The interface enables execution of segmentation using a pretrained deep learning model.
* It provides visualization outputs, including:
  + - The original CT scan in RGB,
    - A grayscale anatomical reference,
    - The predicted segmentation overlay.

To ensure smooth visualization, plots are embedded directly using Matplotlib. Model inference is handled in a separate thread via a custom QThread, which helps maintain the interface's responsiveness even during heavy processing.

A "Go Back" button is available to reset the interface, making it easy for users to test multiple images without relaunching the application.

For user support, a toggle option reveals contact details including name, phone number, and LinkedIn profile—allowing users to seek assistance if the GUI encounters any issues.

The entire application is lightweight and runs efficiently on standard local systems without requiring GPU support, making it an accessible tool for academic and offline environments.

# Project Flow / Framework of the Proposed System

# 6.1 Flow of Proposed System

1. Data flow:

* Input:

The system takes 2D slices from the CT Scan image in JPG format as an input.

* Preprocessing:

For building a suitable model, the input image undergoes normalization, intensity clipping and multi-channel conversion to makes outcomes very promising.

* Model inference:

This processed image is passed to the trained U-Net with ResNet-34 encoder for segmentation which generates the mask for liver and tumor regions.

* Output:

The system outputs the segmentation mask in the form of a 3-channel image which is displayed on the GUI.

1. Training and Testing flow:

* Training Phase: The model is trained on a set of preprocessed CT scan images using the U-Net with ResNet-34 encoder. By validation, we can make sure that the liver and its abnormal region have been segmented more accurately and efficiently.
* Validation Phase: The model's performance is assessed using metrics such as Dice Coefficient and accuracy, helping verify its ability to accurately segment liver and tumor regions.
* Testing Phase: Our trained model is tested on unseen data or input data to evaluate its generalization ability.

1. Integration Framework:

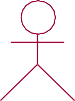
* Backend: The trained model and its weights are stored in a serialized format in a .pkl file. This model is loaded and is served by the backend to make the real-time predictions.
* Frontend: The GUI acts as the frontend, this allows the users to interact with the system to upload their CT scan images and visualize segmentation results.
* Asynchronous execution: The model inference is done using custom q thread in the background so that the GUI is not frozen and ensures smooth user interaction.

**6.2 UML DIAGRAMS**

Identifying Actors

In UML, an actor refers to any entity (usually a user or an external system) that interacts with the system without controlling its internal behaviour. Actors contribute inputs and receive outputs but do not influence the logic of use cases.

Visual Element:



Actor

An actor can be:

* A provider of inputs or recipient of outputs.
* An external party that initiates or triggers system functionalities but cannot modify how those functionalities are internally handled.

Common Criteria for Identifying Actors:

To understand who the actors are or what the actors are, consider:

* Who directly operates or uses the system?
* Who is in charge of managing or maintaining it?
* Are there any external devices or software integrated with the system?
* Are there other applications or services that must communicate with it?

Sample Questions to Recognize Actors:

* Who must ensure the system performs correctly?
* Are there supporting roles like admin or technical support involved?
* Are any external tools or platforms linked to the system?
* What actions will be taken through the graphical user interface, and by whom?

The actors identified in this system are:

1. End User / Medical Practitioner / Radiologist
2. System Administrator

Defining Use Cases

A use case outlines a specific task or interaction an actor can carry out using the system.

Diagram Representation:

A black and white image of a oval  AI-generated content may be incorrect.

## Why Use Cases Are Important:

## They help document system requirements.

## They serve as a basis for validating and testing system behaviour.

## How to Identify Use Cases:

## Start by listing tasks each actor should be able to perform.

## Each task becomes a candidate use case that leads to a defined outcome.

## Use familiar and domain-specific terms for naming and describing use cases to avoid confusion.

## Focus on clarity and relevance from the actor’s perspective.

## Use case diagram:

It visually outlines the system's functionality by representing user goals as use cases and showing how those use cases relate to each other. Its primary function is to indicate which features or services of the system are utilized by each actor, effectively capturing user roles and expectations.

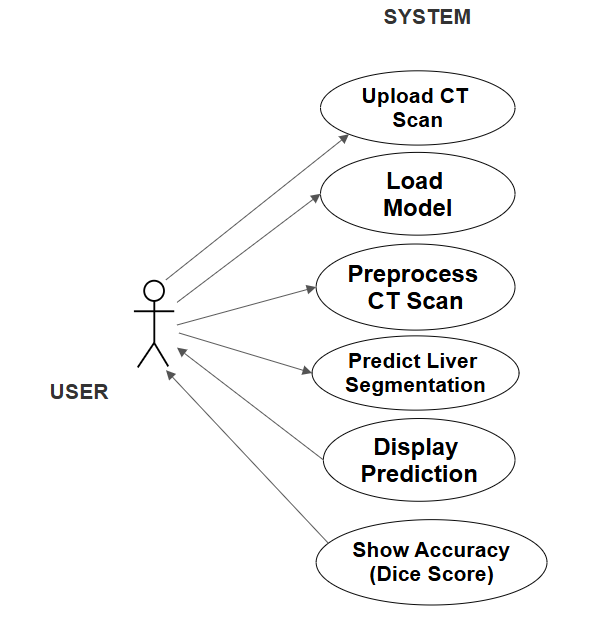


Fig 6.2 Use Case Diagram

## The figure illustrates a single actor, labeled as "User," engages with various system functionalities through a simple graphical user interface. The user can upload CT scan images, initiate the loading of the pre-trained segmentation model, and trigger preprocessing steps for the input images. Following this, the user can perform liver segmentation predictions. The system then allows the user to visualize the predicted results and view the segmentation accuracy, typically represented using the Dice Score. This diagram clearly outlines the sequence of user-initiated operations that support the end-to-end liver segmentation workflow.

## SEQUENCE DIAGRAMS:

In UML, a sequence diagram is an interaction model that displays how different components or objects in a system communicate over time. It captures the sequence of messages exchanged between them to complete a process.

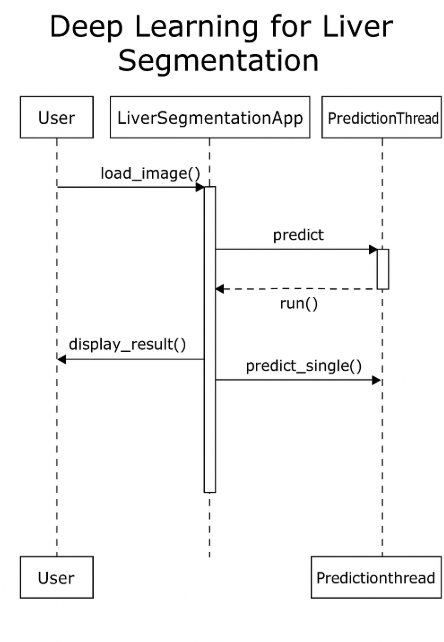


Figure 6.2.2 Sequence diagram

The sequence diagram shows the flow where the user initiates image loading (load\_image()), followed by prediction (predict()), which triggers a separate thread (run()) to perform background computation. The segmentation is performed via predict\_single(), and finally, the result is displayed to the user with display\_result().

## CLASS DIAGRAM:

It defines the classes within the system, detailing their attributes, behaviors (methods), and how they are interconnected. It is commonly used to map out the blueprint of an application’s structure.

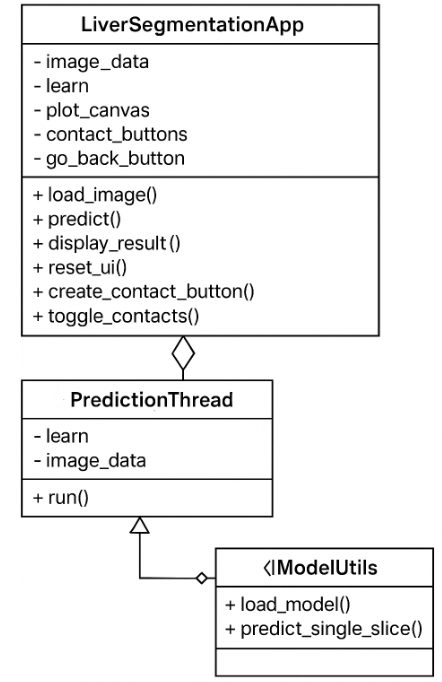


Figure 6.2.3 Class Diagram

The diagram illustrates that the main class LiverSegmentationApp manages the user interface and core functions like loading images, making predictions, and displaying results. It interacts with PredictionThread, which handles background prediction tasks using the model and image data. The PredictionThread class, in turn, depends on the IModelUtils interface, which provides methods to load the trained model and predict a single CT slice. This design separates UI, prediction logic, and model utilities, improving modularity and clarity.

## ACTIVITY DIAGRAM:

These diagrams can include conditions, loops, and parallel processes. In UML, they are typically used to describe how specific processes or components behave over time in a step-by-step manner.

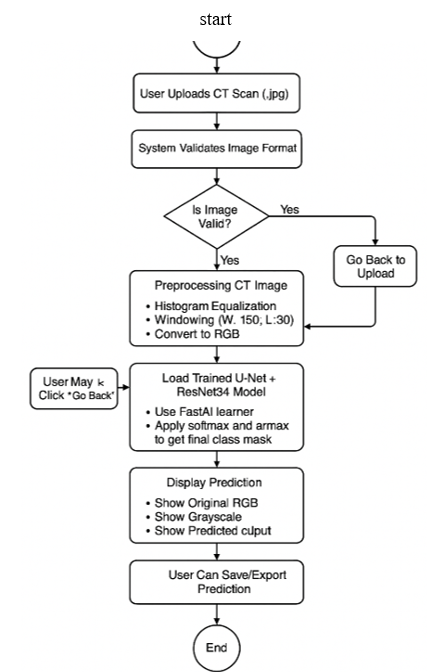


Figure 6.2.4 Activity Diagram

The diagram starts when the user uploads a CT scan image in .jpg format. The system first checks whether the uploaded image is valid. If the image format is incorrect, the user is prompted to re-upload the image. For valid images, preprocessing is performed, which includes histogram equalization, windowing (W:150, L:30), and conversion to RGB. The prediction is displayed in multiple formats—original RGB, grayscale, and the segmented output.

# Results and Discussions

**7.1 Overview of Model Performance:**

The deep learning model that was designed for liver and tumor segmentation shows strong performance across various metrics. It is made of U-Net architecture with ResNet-34 encoder, trained on the LiTS dataset comprising of 130 real CT scans.

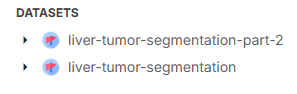


Fig 7.1 Liver Tumor Segmentation (LiTS) dataset

After preprocessing the grayscale CT slices into RGB channels using histogram scaling and windowing, the model achieved exceptional segmentation quality with clear boundaries and minimal noise. The final .pkl model was used for inference and is integrated into a GUI for clinical applications.

**7.2 Trend Analysis**

The training was conducted over eight epochs, and a consistent improvement was observed in both loss values and accuracy metrics, indicating stable and effective model learning.

1. Loss Curves

Training Loss steadily declined from 0.0053 (Epoch 0) to 0.0020 (Epoch 7), showcasing efficient convergence without signs of overfitting. Validation Loss followed a similar downward trend, decreasing from 0.0057 to 0.0020 over the epochs, confirming that generalization improved as training progressed. The best validation loss was recorded at Epoch 7, making the model's performance suitable for real-time usage.

1. Accuracy Metrics

The foreground accuracy increased from 94.44% at the start to 97.32%, with a notable jump between Epochs 2 and 3 (94.83% → 97.41%). This indicates that the model became better at distinguishing foreground elements as training progressed. Custom foreground accuracy started at an already high 99.79%, and consistently improved to reach 99.90%. The small incremental gains here suggest that the model was fine-tuning its segmentation on nuanced differences in the foreground region.

1. Dice Scores

The Train Dice improved from 95.42% to 98.50%, while the Validation Dice improved from 96.23% to 98.18%, confirming robust performance across both seen and unseen data. These high Dice coefficients reflect excellent overlap between predicted and ground truth masks, which is critical for segmentation tasks.

1. Inference Quality Metrics

Post-training evaluations revealed outstanding model performance: Accuracy, Precision, Recall, and F1-Score were all recorded at 99.89%, indicating near-perfect classification capability. Hausdorff Distance (95th percentile) was 1.00, showing minimal boundary deviations, which is vital for precise segmentation. The Best Dice Score achieved during validation was 98.20%, validating the model's top-tier segmentation quality.

1. Time Efficiency

The training time per epoch remained stable, around 2 minutes 40 seconds, demonstrating consistent computational efficiency throughout training.

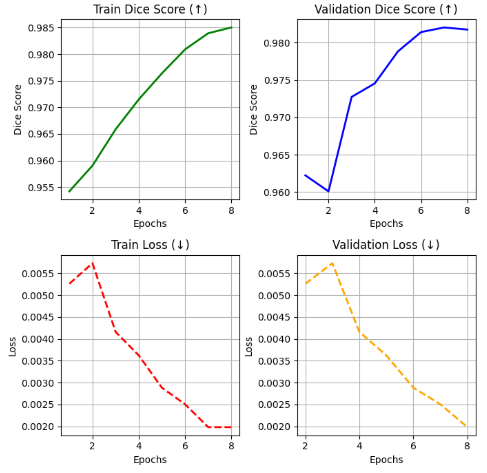


Fig 7.2 Trend Analysis

**7.3 Performance Evaluation:**

The performance of our automated segmentation model was evaluated using several metrics:

1. Dice Coefficient:  
   The Dice coefficient is used as a metric to quantify the similarity between the actual ground truth image and the predicted segmentation mask.
   * Best model achieved a training Dice of 98.50%
   * Validation Dice of 98.20%
2. Precision and Recall:
3. Both scored 98.89%, which provides a promising balance between false positives and false negatives, making our model usable for real time.
4. F1 Score:
   * Achieved up to 98.89%, confirming robust segmentation performance
5. Hausdorff Distance (95%):
   * Achieved a value of 1.00, signifying high boundary accuracy
6. Inference Time:
   * Around 2 seconds per image on CPU
   * Suitable for real-time use

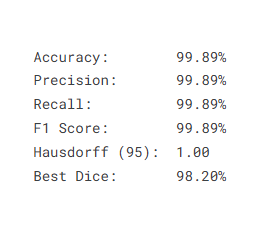


Fig 7.3 Evaluation metrics

The model was evaluated in real-time settings and consistently maintained performance across diverse image inputs, validating its robustness.

**7.4 GUI Functionality and User Experience**

The GUI application allows the user to visualize their uploaded CT scan image alongside the predicted segmentation mask . This visualization aids interpretability and enables quick validation of the model output . Visual inspection of the generated masks, which closely align with ground truth, confirms that liver and tumor segmentation is both smooth and complete.

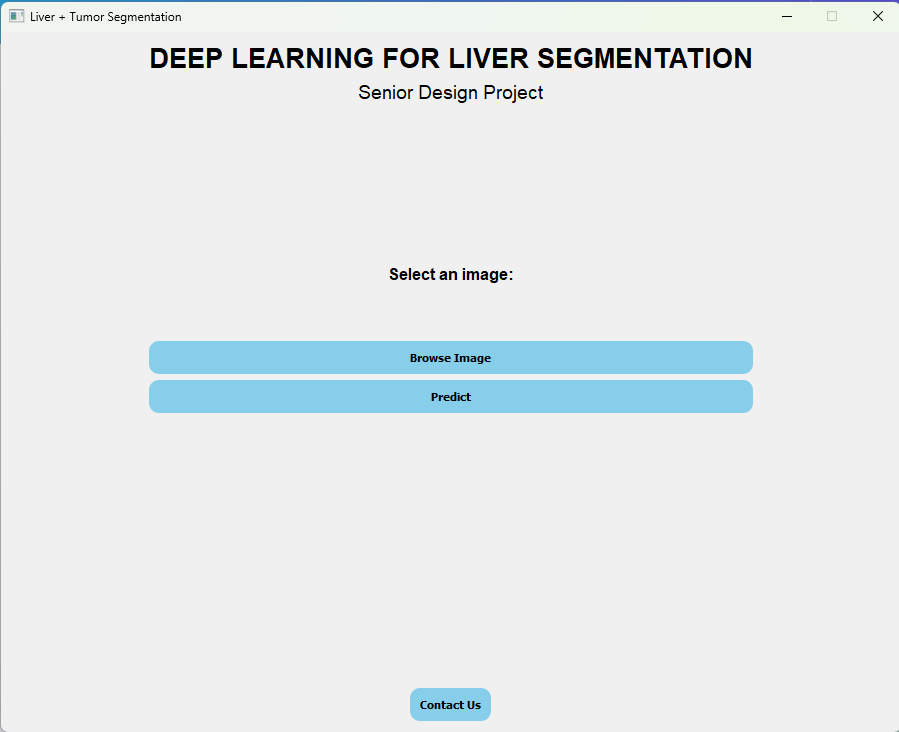


Fig 7.4.1 GUI made using PyQt5

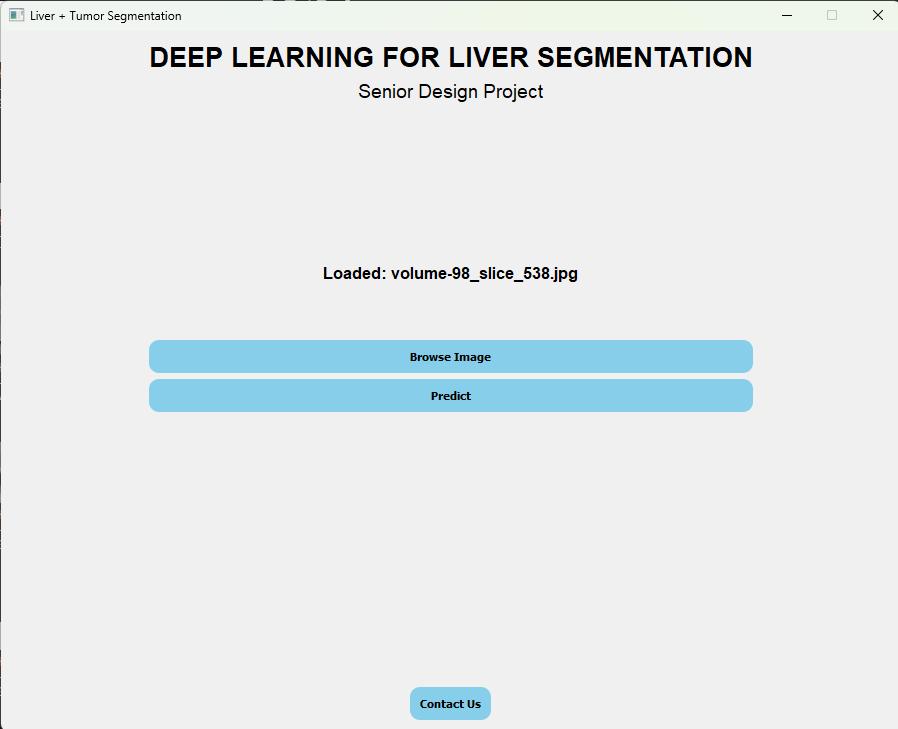


Fig 7.4.2 Loaded the image given by user

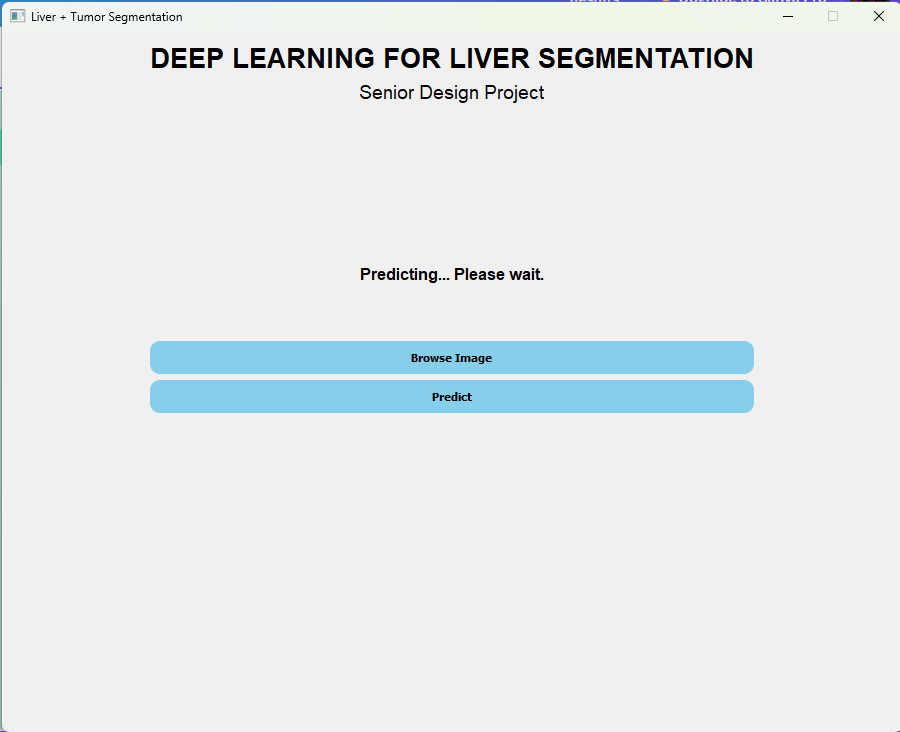


Fig 7.4.3 Provides the status of prediction upto visualizing the liver and its abnormality



Fig 7.4.4 Contact Information

**SEGEMENTATION OUTPUT OF OUR MODEL**

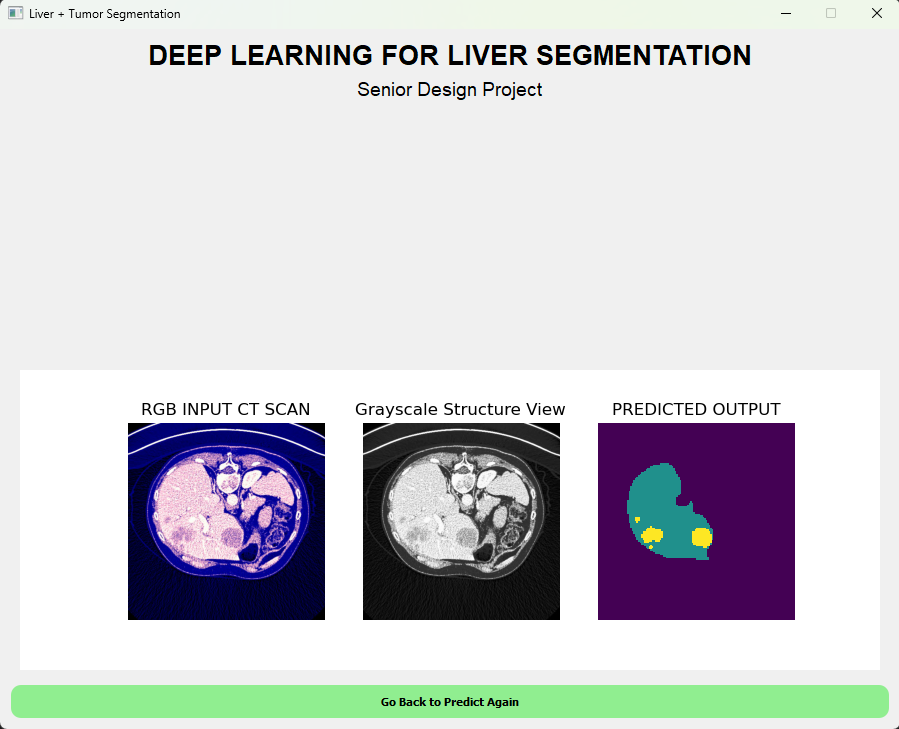


Fig 7.4.5 Output

The interface displays the original RGB-converted CT scan alongside the predicted segmentation mask, allowing for a clear visual comparison. In the predicted output:

* The regions of liver highlighted with colours like green, providing a promising results and give clear outline of the regions like liver and its abnormality.
* Tumor regions are distinctly marked in yellow, enabling clinicians to easily locate and assess tumor size and position.

**CONCLUSION**

Our project, "Deep Learning for Liver Segmentation," presents an efficient, accurate, and reliable solution suitable for real-world clinical and research applications. Leveraging deep learning algorithms, we successfully developed a model capable of segmenting the liver and detecting abnormalities with high precision.Throughout the development, we faced significant challenges, primarily related to accurately segmenting liver regions and handling variability in CT scan data. To overcome these, we trained our model on the Liver Tumor Segmentation Benchmark (LiTS) dataset, which contains over 131 real-world CT scans intended for research use.

# Our final model, based on a U-Net architecture integrated with ResNet-34 as the encoder, achieved impressive results — attaining a Dice coefficient of 98.20% and an overall accuracy of 99.89%. These metrics demonstrate the model’s robustness and potential for clinical assistance.

# To enhance usability, we developed a PyQt5-based desktop application that allows users to upload CT scan slices, perform real-time liver segmentation, and visually compare the original and segmented outputs side-by-side. Our GUI ensures accessibility without requiring specialized medical expertise, thereby promoting broader adoption.

# We also addressed challenges such as CT scan variability, class imbalance, and noise through careful preprocessing, regularization, and model fine-tuning. Our continuous integration of design, training, testing, and evaluation helps us to make our model more flexible and resolve many issues which makes it special to manual liver segmentation.

# By automating the segmentation process, our model significantly reduces the workload on radiologists, minimizes human error, ensures consistency across cases, and speeds up diagnosis — all of which are critical for time-sensitive applications like oncology.

# Moreover, our solution is cost-effective and can be readily integrated into existing healthcare infrastructures. Overall, our project achieves its objectives and demonstrates the transformative potential of AI in the healthcare sector.

# Future Works

**8.1 Advantages of Our Model**

* High Accuracy: Achieved a Dice coefficient of 98.20% and an overall accuracy of 99.89%.
* Efficient Architecture: Integration of ResNet-34 with U-Net resulted in better feature extraction and faster convergence.
* User-Friendly Interface: PyQt5 desktop application allows real-time segmentation without requiring technical expertise.
* Scalable Design: Can be adapted or integrated into existing healthcare systems easily.
* Reduces Radiologist Workload: Automation reduces human errors and saves diagnosis time.
* Cost-Effective: The model and application are lightweight and affordable for small or large healthcare institutions.

**8.2 Disadvantages of General Other Current Models**

* Lower Accuracy on Complex Cases: Many models fail when tumors are small, diffused, or have irregular boundaries.
* Heavy Computational Requirements: Some architectures (e.g., deeper U-Nets, DenseNets) require extensive GPU resources.
* Overfitting on Small Datasets: Models trained on limited data perform poorly when generalized to new CT scans.
* Poor Handling of Class Imbalance: Many segmentation models struggle when normal liver tissue dominates and abnormalities are rare.
* Lack of Real-Time Capability: Some models are too slow for practical clinical usage.

**8.3 Disadvantages of Our Model**

* Limited Dataset Exposure: Trained primarily on the LiTS dataset; might underperform on unseen data from different populations or hospitals.
* Dependence on CT Scan Quality: Severe noise or low-quality images can degrade performance.
* Limited Abnormality Coverage: Focused mainly on liver tumors; may not generalize well to other liver conditions (like cysts or infections).
* Static Learning: Once trained, the model doesn't adapt or fine-tune itself automatically when new data arrives.
* The current GUI supports only RGB .jpg image inputs, which requires prior manual preprocessing of CT scan slices.

**8.4 Potential Future Improvements to Our Model**

* We aim to extend support for medical imaging formats such as DICOM (.dcm) and NIfTI (.nii, .nii.gz), allowing direct integration with clinical imaging workflows and improving ease of use for medical professionals.
* Multi-Institutional Training: Train on diverse datasets from various hospitals globally to improve generalization.
* Incorporate Attention Mechanisms: Adding attention layers (e.g., Attention U-Net) can enhance segmentation focus on important regions.
* Semi-Supervised Learning: Use unannotated CT scans with semi-supervised methods to expand training without full labels.
* Noise Robustness: Implement advanced pre-processing pipelines (e.g., denoising autoencoders) to handle poor-quality CT scans.
* Explainable AI (XAI): Integrate interpretability techniques like Grad-CAM or saliency maps to show which parts of the image influence the segmentation.
* Continuous Learning: Introduce mechanisms for online learning or model updating as new cases are encountered.
* Mobile and Web Deployment: Optimize the model for lightweight deployment on mobile devices or cloud-based web apps.

**8.5 Real-World Applications and Scalability**

* Clinical Diagnostic Assistance: Integrate the model into PACS (Picture Archiving and Communication Systems) in hospitals.
* Telemedicine: Enable remote diagnosis, especially useful in rural or resource-constrained settings.
* Surgical Planning: Assist surgeons in pre-operative planning by providing clear liver segmentation.
* Research and Drug Trials: Help researchers monitor liver condition progression or response to therapies.
* Large-Scale Healthcare Systems: Deploy at scale in national healthcare systems to streamline liver disease diagnosis.
* Cross-Hospital Collaboration: Standardize liver segmentation outputs across different hospitals to aid multicentre research.

**8.6 Cross-Domain Applications**

* Other Organ Segmentation: Extend the model framework to segment kidneys, lungs, pancreas, etc.
* Cancer Detection in Other Modalities: Adapt the model for MRI or PET scan liver images or for tumours in other organs.
* Emergency Diagnostics: Apply in rapid response cases like trauma, where liver damage needs to be assessed quickly.
* Veterinary Medicine: Making liver segmentation suitable for animals and making predictions accordingly.

**8.7 Ethical Considerations**

* Data Privacy: Ensure CT scans used respect patient confidentiality and follow HIPAA (or similar) guidelines.
* Bias Mitigation: Prevent biases where the model may perform better for certain demographics.
* Clinical Validation: Properly validate the model through clinical trials before large-scale deployment.
* Explainability: Address ethical issues of black-box predictions by making the model's decision process more interpretable.

**Summary of Future Work**

In the future, we aim to broaden the model’s applicability, enhance robustness across diverse datasets, and address current limitations like explainability and adaptability. We foresee a strong impact in clinical diagnostics, research assistance, and real-time healthcare applications. Ensuring ethical deployment and patient-centered development will be central to advancing our solution into real-world medical environments

# References

1. Nico Klinger, *U-Net: A Comprehensive Guide to Its Architecture and Applications*, April 2024.
2. Patrick Bilic, Patrick Christ, Hongwei Bran Li, Eugene Vorontsov, Avi Ben-Cohen, Georgios Kaissis, Adi Szeskin, Colin Jacobs, Gabriel Efrain Humpire Mamani, Gabriel Chartrand, Fabian Lohöfer, Julian Walter Holch, Wieland Sommer, Felix Hofmann, Alexandre Hostettler, Naama Lev-Cohain, Michal Drozdzal, Michal Marianne Amitai, Refael Vivanti, Jacob Sosna, Ivan Ezhov, Anjany Sekuboyina, Fernando Navarro, Florian Kofler, Johannes C. Paetzold, Suprosanna Shit, Xiaobin Hu, Jana Lipková, Markus Rempfler, Marie Piraud, Jan Kirschke, Benedikt Wiestler, Zhiheng Zhang, Christian Hülsemeyer, Marcel Beetz, Florian Ettlinger, Michela Antonelli, Woong Bae, Míriam Bellver, Lei Bi, Hao Chen, Grzegorz Chlebus, Erik B. Dam, Qi Dou, Chi-Wing Fu, Bogdan Georgescu, Xavier Giró-i-Nieto, Felix Gruen, Xu Han, Pheng-Ann Heng, Jürgen Hesser, Jan Hendrik Moltz, Christian Igel, Fabian Isensee, Paul Jäger, Fucang Jia, Krishna Chaitanya Kaluva, Mahendra Khened, Ildoo Kim, Jae-Hun Kim, Sungwoong Kim, Simon Kohl, Tomasz Konopczynski, Avinash Kori, Ganapathy Krishnamurthi, Fan Li, Hongchao Li, Junbo Li, Xiaomeng Li, John Lowengrub, Jun Ma, Klaus Maier-Hein, Kevis-Kokitsi Maninis, Hans Meine, Dorit Merhof, Akshay Pai, Mathias Perslev, Jens Petersen, Jordi Pont-Tuset, Jin Qi, Xiaojuan Qi, Oliver Rippel, Karsten Roth, Ignacio Sarasua, Andrea Schenk, Zengming Shen, Jordi Torres, Christian Wachinger, Chunliang Wang, Leon Weninger, Jianrong Wu, Daguang Xu, Xiaoping Yang, Simon Chun-Ho Yu, Yading Yuan, Miao Yue, Liping Zhang, Jorge Cardoso, Spyridon Bakas, Rickmer Braren, Volker Heinemann, Christopher Pal, An Tang, Samuel Kadoury, Luc Soler, Bram van Ginneken, Hayit Greenspan, Leo Joskowicz, Bjoern Menze, *The Liver Tumor Segmentation Benchmark (LiTS)*, February 2023.
3. Omar Ibrahim Alirr, *Deep Learning and Level Set Approach for Liver and Tumor Segmentation from CT Scans*, August 2020.
4. Priscilla Benedetti, Mauro Femminella, Gianluca Real, *Mixed-Sized Biomedical Image Segmentation Based on U-Net Architectures*, December 2022.
5. Jacob Solawetz, *Use ResNet34 for Image Classification*, September 2020.
6. Lu Tinglan, Qin Jun, Qin Guihe, Shi Weili, Zhang Wentao, *Liver Segmentation Network Based on Detail Enhancement and Multi-Scale Feature Fusion*, January 2025.
7. Zhendong Songa, Huiming Wua, Wei Chena, Adam Slowikb, *Improving Automatic Segmentation of Liver Tumor Images Using a Deep Learning Model*, April 2024.
8. Nghi Huynh, *Understanding Evaluation Metrics in Medical Image Segmentation*, March 2023.
9. Dominik Müller, Iñaki Soto-Rey, Frank Kramer, *Towards a Guideline for Evaluation Metrics in Medical Image Segmentation*, June 2022.
10. Hassan Sajjad, *GUI Marvels: Decrypting Brain Tumor Segmentation using Python, PyQt5, and Image Processing*, January 2024.
11. Ang Li-Lian, *Image Classification with ResNet (PyTorch)*, October 2022.
12. Michal Drozdzal, Gabriel Chartrand, Eugene Vorontsov, Mahsa Shakeri, Lisa Di Jorio, An Tang, Adriana Romero, Yoshua Bengio, Chris Pal, Samuel Kadoury, *Learning Normalized Inputs for Iterative Estimation in Medical Image Segmentation*, November 2017.
13. Risheng Wang, Tao Lei, Ruixia Cui, Bingtao Zhang, Hongying Meng, Asoke K. Nandi, *Medical Image Segmentation Using Deep Learning: A Survey*, January 2022.
14. Hameedur Rahman, Tanvir Fatima Naik Bukht, Azhar Imran, Junaid Tariq, Shanshan Tu, Abdulkareeem Alzahrani, *A Deep Learning Approach for Liver and Tumor Segmentation in CT Images Using ResUNet*, August 2022.
15. Michal Drozdzal, Gabriel Chartrand, Eugene Vorontsov, Mahsa Shakeri, Lisa Di Jorio, An Tang, Adriana Romero, Yoshua Bengio, Chris Pal, Samuel Kadoury, *Learning Normalized Inputs for Iterative Estimation in Medical Image Segmentation*, Medical Image Analysis, Volume 44, February 2018, Pages 1-13.
16. B. Lakshmipriya, Biju Pottakkat, G. Ramkumar, *Deep Learning Techniques in Liver Tumour Diagnosis Using CT and MR Imaging - A Systematic Review*, May 2023.
17. Mubasher H. Malik, Hamid Ghous, Tahir Rashid, Bibi Maryum, Zhang Hao, Qasim Umer, *Feature Extraction-Based Liver Tumor Classification Using Machine Learning and Deep Learning Methods of Computed Tomography Images*, April 2024.
18. Bharath K, *U-Net Architecture for Image Segmentation*, October 2024.
19. Yongqi Yuan, Yong Cheng, *Medical Image Segmentation with UNet-Based Multi-Scale Context Fusion*, October 2024.
20. Ahmed AL Qurri, Mohamed Almekkawy, *Improved UNet with Attention for Medical Image Segmentation*, October 2023.

# Appendix

## Appendix A: Dataset Details

* + Dataset Used: LiTS (Liver Tumor Segmentation)
  + Image Type: 3D CT Scan volumes
  + Format: NIfTI (.nii) format
  + Total Samples: 130 CT scans
  + Ground Truth: Segmentation masks of liver and tumor
  + Annotations: Liver and tumor regions labeled by experts

## Appendix B: Hardware and Software requirements

## Hardware:

## Processor: Intel Core i7-11800H CPU @ 2.30GHz

## RAM: 16GB DDR4

## GPU: NVIDIA GeForce RTX 3060 (6GB VRAM) *(optional for training; inference was CPU-based)*

## Storage: 512GB SSD

## Software:

## Operating System: Windows 10 (64-bit)

## Programming Language: Python 3.9

## Libraries and Frameworks Used:

## FastAI: 2.7.12

## PyTorch: 1.13.1

## NumPy: 1.23.5

## Pillow (PIL): 9.2.0

## Matplotlib: 3.6.2

## Nibabel: 4.0.2

## PyQt5: 5.15.9

## scikit-learn: 1.1.3

## pathlib: Built-in for path handling

## • Tools and Environment:

## Jupyter Notebook for model development and experimentation

## PyQt5 for GUI application development

**Appendix C: Model Architecture**

The deep learning model employed for liver and tumor segmentation is based on the U-Net architecture with a ResNet-34 encoder backbone. Below is an outline of the model architecture: Base Architecture: U-Net: A fully convolutional encoder-decoder network designed for semantic segmentation.

Encoder Backbone: Pre-trained ResNet-34

* Function: Extracts hierarchical features from input images using residual connections and down sampling.
* Composition:
  + Initial convolution + batch normalization + ReLU
  + Multiple residual blocks with increasing feature maps
  + Down sampling via max-pooling and strided convolutions

Decoder:

* Function: Reconstructs segmentation mask using up sampling and feature concatenation (skip connections) from corresponding encoder layers
* Composition:
  + Up sampling via transposed convolutions
  + Skip connections with encoder outputs
  + Convolution + batch normalization + ReLU in each up-sampling step

Final Layers:

* 1×1 Convolution Layer: This helps in reducing feature maps to the number of output classes (background, liver, tumor)
* Activation Function: Softmax (for multi-class segmentation)

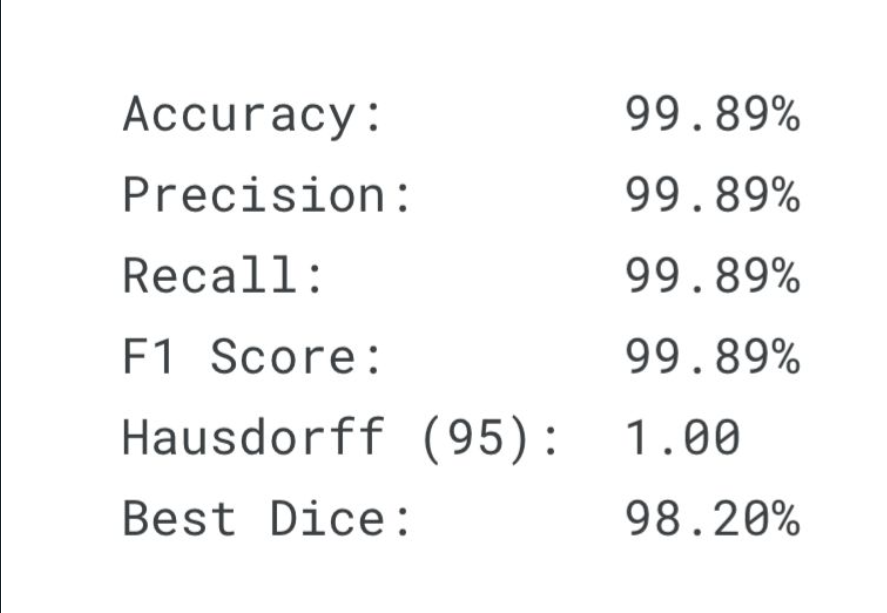
Input Image Dimensions:

* RGB images of size [C×H×W] = [3×256×256], obtained via preprocessing from grayscale CT slices

Output:

* Segmentation mask of shape [1×256×256], where each pixel is assigned a class (0 = background, 1 = liver, 2 = tumor)

**Appendix D: Performance Metrics**

****

**Appendix E: Limitations of the Study**

* Trained only on LiTS dataset; may not generalize to other CT datasets
* Uses 2D slices; lacks 3D spatial context for accurate tumor volume.

* Performance may drop on low-contrast or small tumor regions.
* GUI accepts only .jpg RGB images; DICOM/NIfTI not yet supported.

## Appendix F: Future Research Directions

* Train on diverse datasets for better generalization.
* Extend to 3D segmentation using volumetric models.
* Improve detection of small and low-contrast tumors.
* Add support for DICOM and NIfTI formats in the GUI.

# Source Code

# DEEP LEARNING FOR LIVER SEGMENTATION[¶](https://www.kaggle.com/code/madhuri42/sdp-liver-segmentation#DEEP-LEARNING-FOR-LIVER-SEGMENTATION)

# GPU CHECK[¶](https://www.kaggle.com/code/madhuri42/sdp-liver-segmentation#GPU-CHECK)

# In [1]:

# import torch

# *# Set device to GPU if available, else fallback to CPU*

# device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# print("Using device:", device)

# print("GPU Name:", torch.cuda.get\_device\_name(0) if torch.cuda.is\_available() else "No GPU")

# IMPORTS

# In [2]:

# import numpy as np

# import pandas as pd

# import os

# import matplotlib.pyplot as plt

# import glob

# import nibabel as nib

# import cv2

# import imageio

# from tqdm.notebook import tqdm

# from ipywidgets import \*

# from PIL import Image

# In [3]:

# !python --version

# Python 3.11.11

# In [4]:

# *!pip install torch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1 --index-url https://download.pytorch.org/whl/cpu*

# *!pip install fastai==2.7.12*

# !pip install medpy

# *!pip install scikit-learn*

# In [5]:

# *!pip uninstall torch torchvision torchaudio -y*

# *!pip cache purge*

# *!pip install torch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2 --index-url https://download.pytorch.org/whl/cpu*

# In [6]:

# import torch

# import torchvision

# print(torch.\_\_version\_\_) *# Should be 2.0.1*

# print(torchvision.\_\_version\_\_) *# Should be 0.15.2*

# 2.5.1+cu124

# 0.20.1+cu124

# In [7]:

# from sklearn.metrics import f1\_score, accuracy\_score, recall\_score

# from medpy.metric.binary import hd95, hd, assd

# import torch

# import os

# from fastai.vision.augment import aug\_transforms

# from fastai.vision.learner import cnn\_learner

# from fastai.basics import \*

# from fastai.vision.all import \*

# from fastai.data.transforms import \*

# import warnings

# warnings.filterwarnings("ignore")

# from fastai.vision.all import \*

# from fastai.callback.core import Callback

# from fastai.callback.tracker import SaveModelCallback

# from fastai.learner import Recorder

# In [8]:

# *# Create a meta file for nii files processing*

# file\_list = []

# for dirname, \_, filenames in os.walk('../input/liver-tumor-segmentation'):

# for filename in filenames:

# *# print(os.path.join(dirname, filename))*

# file\_list.append((dirname,filename))

# for dirname, \_, filenames in os.walk('../input/liver-tumor-segmentation-part-2'):

# for filename in filenames:

# file\_list.append((dirname,filename))

# df\_files = pd.DataFrame(file\_list, columns =['dirname', 'filename'])

# df\_files.sort\_values(by=['filename'], ascending=True)

|  |
| --- |

# 

# In [9]:

# *#Map CT scan and label*

# df\_files["mask\_dirname"] = "" ; df\_files["mask\_filename"] = ""

# for i in range(131):

# ct = f"volume-{i}.nii"

# mask = f"segmentation-{i}.nii"

# df\_files.loc[df\_files['filename'] == ct, 'mask\_filename'] = mask

# df\_files.loc[df\_files['filename'] == ct, 'mask\_dirname'] = "../input/liver-tumor-segmentation/segmentations"

# *# drop segment rows*

# df\_files = df\_files[df\_files.mask\_filename != ''].sort\_values(by=['filename']).reset\_index(drop=True)

# print(len(df\_files))

# df\_files

# 131

# 

# def read\_nii(filepath):

# *'''*

# *Reads .nii file and returns pixel array*

# *'''*

# ct\_scan = nib.load(filepath)

# array = ct\_scan.get\_fdata()

# array = np.rot90(np.array(array))

# return(array)

# In [11]:

# *# Read sample*

# sample = 0

# sample\_ct = read\_nii(df\_files.loc[sample,'dirname']+"/"+df\_files.loc[sample,'filename'])

# sample\_mask = read\_nii(df\_files.loc[sample,'mask\_dirname']+"/"+df\_files.loc[sample,'mask\_filename'])

# sample\_ct.shape, sample\_mask.shape

# Out[11]:

# ((512, 512, 75), (512, 512, 75))

# In [12]:

# linkcode

# print(np.amin(sample\_ct), np.amax(sample\_ct))

# print(np.amin(sample\_mask), np.amax(sample\_mask))

# -3024.0 1410.0

# 0.0 2.0

# In [13]:

# *# Preprocess the nii file*

# dicom\_windows = types.SimpleNamespace(

# brain=(80,40),

# subdural=(254,100),

# stroke=(8,32),

# brain\_bone=(2800,600),

# brain\_soft=(375,40),

# lungs=(1500,-600),

# mediastinum=(350,50),

# abdomen\_soft=(400,50),

# liver=(150,30),

# spine\_soft=(250,50),

# spine\_bone=(1800,400),

# custom = (200,60)

# )

# @patch

# def windowed(self:Tensor, w, l):

# px = self.clone()

# px\_min = l - w//2

# px\_max = l + w//2

# px[px<px\_min] = px\_min

# px[px>px\_max] = px\_max

# return (px-px\_min) / (px\_max-px\_min)

# plt.imshow(tensor(sample\_ct[...,50].astype(np.float32)).windowed(\*dicom\_windows.liver), cmap=plt.cm.bone);

# 

# def plot\_sample(array\_list, color\_map = 'nipy\_spectral'):

# fig = plt.figure(figsize=(18,15))

# plt.subplot(1,4,1)

# plt.imshow(array\_list[0], cmap='bone')

# plt.title('Original Image')

# plt.subplot(1,4,2)

# plt.imshow(tensor(array\_list[0].astype(np.float32)).windowed(\*dicom\_windows.liver), cmap='bone');

# plt.title('Windowed Image')

# plt.subplot(1,4,3)

# plt.imshow(array\_list[1], alpha=0.5, cmap=color\_map)

# plt.title('Mask')

# plt.subplot(1,4,4)

# plt.imshow(array\_list[0], cmap='bone')

# plt.imshow(array\_list[1], alpha=0.5, cmap=color\_map)

# plt.title('Liver & Mask')

# plt.show()

# In [15]:

# sample=50

# sample\_slice = tensor(sample\_ct[...,sample].astype(np.float32))

# plot\_sample([sample\_ct[...,sample], sample\_mask[...,sample]])

# 

# sample=56

# sample\_slice = tensor(sample\_ct[...,sample].astype(np.float32))

# plot\_sample([sample\_ct[...,sample], sample\_mask[...,sample]])

# 

# # Check the mask values

# mask = Image.fromarray(sample\_mask[...,sample].astype('uint8'), mode="L")

# unique, counts = np.unique(mask, return\_counts=True)

# print( np.array((unique, counts)).T)

# 

# #Preprocessing functions

# class TensorCTScan(TensorImageBW): \_show\_args = {'cmap':'bone'}

# @patch

# def freqhist\_bins(self:Tensor, n\_bins=100):

# "A function to split the range of pixel values into groups, such that each group has around the same number of pixels"

# imsd = self.view(-1).sort()[0]

# t = torch.cat([tensor([0.001]),

# torch.arange(n\_bins).float()/n\_bins+(1/2/n\_bins),

# tensor([0.999])])

# t = (len(imsd)\*t).long()

# return imsd[t].unique(

# @patch

# def hist\_scaled(self:Tensor, brks=None):

# "Scales a tensor using `freqhist\_bins` to values between 0 and 1"

# if self.device.type=='cuda': return self.hist\_scaled\_pt(brks)

# if brks is None: brks = self.freqhist\_bins()

# ys = np.linspace(0., 1., len(brks))

# x = self.numpy().flatten()

# x = np.interp(x, brks.numpy(), ys)

# return tensor(x).reshape(self.shape).clamp(0.,1.)

# @patch

# def to\_nchan(x:Tensor, wins, bins=None):

# res = [x.windowed(\*win) for win in wins]

# if not isinstance(bins,int) or bins!=0: res.append(x.hist\_scaled(bins).clamp(0,1))

# dim = [0,1][x.dim()==3]

# return TensorCTScan(torch.stack(res, dim=dim))

# @patch

# def save\_jpg(x:(Tensor), path, wins, bins=None, quality=90):

# fn = Path(path).with\_suffix('.jpg')

# x = (x.to\_nchan(wins, bins)\*255).byte()

# im = Image.fromarray(x.permute(1,2,0).numpy(), mode=['RGB','CMYK'][x.shape[0]==4])

# im.save(fn, quality=quality)

# \_,axs=subplots(1,1)

# sample\_slice.save\_jpg('test.jpg', [dicom\_windows.liver,dicom\_windows.custom])

# show\_image(Image.open('test.jpg'), ax=axs[0])

# 

# GENERATE\_JPG\_FILES = True # warning: generation takes ~ 1h

# if (GENERATE\_JPG\_FILES) :

# path = Path(".")

# os.makedirs('train\_images',exist\_ok=True)

# os.makedirs('train\_masks',exist\_ok=True)

# for ii in tqdm(range(0,len(df\_files),3)): # take 1/3 nii files for training

# curr\_ct = read\_nii(df\_files.loc[ii,'dirname']+"/"+df\_files.loc[ii,'filename'])

# curr\_mask = read\_nii(df\_files.loc[ii,'mask\_dirname']+"/"+df\_files.loc[ii,'mask\_filename'])

# curr\_file\_name = str(df\_files.loc[ii,'filename']).split('.')[0]

# curr\_dim = curr\_ct.shape[2] # 512, 512, curr\_dim

# for curr\_slice in range(0,curr\_dim,2): # export every 2nd slice for training

# data = tensor(curr\_ct[...,curr\_slice].astype(np.float32))

# mask = Image.fromarray(curr\_mask[...,curr\_slice].astype('uint8'), mode="L")

# data.save\_jpg(f"train\_images/{curr\_file\_name}\_slice\_{curr\_slice}.jpg", [dicom\_windows.liver,dicom\_windows.custom])

# mask.save(f"train\_masks/{curr\_file\_name}\_slice\_{curr\_slice}\_mask.png")

# else:

# 

# path = Path("../input/liver-segmentation-with-fastai-v2")

# 

# MODEL TRAINING¶

# bs = 16

# im\_size = 128

# codes = np.array(["background","liver","tumor"])

# def get\_x(fname:path): return fname

# def label\_func(x): return path/'train\_masks'/f'{x.stem}\_mask.png'

# tfms = [IntToFloatTensor(),Normalize()]

# db = DataBlock(blocks=(ImageBlock(),MaskBlock(codes)), #codes = {"Backround": 0,"Liver": 1,"Tumor": 2}

# batch\_tfms=tfms,

# splitter=RandomSplitter(),

# item\_tfms=[Resize(im\_size)],

# get\_items=get\_image\_files,

# get\_y=label\_func

# )

# ds = db.datasets(source=path/'train\_images')

# for idx in range(5):

# imgs = [ds[idx][0], ds[idx][1]]

# fig, axs = plt.subplots(1, 2, figsize=(8, 4))

# for i, ax in enumerate(axs.flatten()):

# ax.axis('off')

# ax.imshow(imgs[i])

# plt.suptitle(f"Patient {idx}")

# plt.show()

# 

# 

# 

# unique, counts = np.unique(array(ds[idx][1]), return\_counts=True)

# print( np.array((unique, counts)).T)

# 

# dls = db.dataloaders(path/'train\_images',bs = bs) #, num\_workers=0

# dls.show\_batch(max\_n=16, nrows=4, ncols=4, figsize=(15, 15))

# 

def foreground\_acc(inp, targ, bkg\_idx=0, axis=1):

"Computes non-background accuracy for multiclass segmentation"

targ = targ.squeeze(1)

mask = targ != bkg\_idx

return (inp.argmax(dim=axis)[mask] == targ[mask]).float().mean()

def cust\_foreground\_acc(inp, targ):

"Custom version that includes background in the metric"

return foreground\_acc(inp=inp, targ=targ, bkg\_idx=3, axis=1) *# 3 is a dummy value*

In [27]:

linkcode

from fastai.callback.core import Callback

from fastai.callback.tracker import TrackerCallback

from fastai.callback.progress import ProgressCallback

from fastai.learner import Recorder

from fastai.metrics import Dice

In [28]:

import torch

import torch.nn.functional as F

import numpy as np

import warnings

from fastai.callback.core import Callback

*# Globally silence warnings and FastAI callback errors*

warnings.filterwarnings("ignore")

torch.set\_printoptions(sci\_mode=False)

*# DICE SCORE COMPUTATION FUNCTION*

def compute\_dice\_score(preds, targets, smooth=1e-5, axis=1):

preds = torch.argmax(preds, dim=axis)

num\_classes = int(torch.max(targets)) + 1

*# One-hot encode predictions and targets*

preds = F.one\_hot(preds, num\_classes=num\_classes)

targets = F.one\_hot(targets.long(), num\_classes=num\_classes)

*# Reshape to [B, C, D, H, W] or [B, C, H, W]*

if preds.ndim == 5: *# [B, D, H, W, C]*

preds = preds.permute(0, 4, 1, 2, 3).float()

targets = targets.permute(0, 4, 1, 2, 3).float()

dims = (2, 3, 4)

elif preds.ndim == 4: *# [B, H, W, C]*

preds = preds.permute(0, 3, 1, 2).float()

targets = targets.permute(0, 3, 1, 2).float()

dims = (2, 3)

else:

return torch.tensor(0.0)

intersection = (preds \* targets).sum(dim=dims)

union = preds.sum(dim=dims) + targets.sum(dim=dims)

dice = (2. \* intersection + smooth) / (union + smooth)

return dice.mean(dim=1).mean().item()

*# CALLBACK TO TRACK DICE AND LOSS PER EPOCH (fully silent)*

class ExtendedMetrics(Callback):

def before\_fit(self):

self.train\_losses, self.val\_losses = [], []

self.train\_dice, self.val\_dice = [], []

self.train\_preds, self.train\_targets = [], []

self.preds, self.targets = [], []

self.epoch = 1

def after\_batch(self):

try:

if self.learn.pred is None:

return

pred = self.learn.pred.detach().cpu()

target = None

if hasattr(self.learn, "y") and self.learn.y is not None:

target = self.learn.y.detach().cpu()

elif self.yb is not None and isinstance(self.yb, (list, tuple)) and self.yb[0] is not None:

target = self.yb[0].detach().cpu()

if target is None:

return

if self.learn.training:

self.train\_preds.append(pred)

self.train\_targets.append(target)

else:

self.preds.append(pred)

self.targets.append(target)

except Exception:

pass *# Fully silent error handling*

def after\_epoch(self):

train\_loss = self.\_to\_float(getattr(self, 'smooth\_loss', None))

val\_loss = None

if hasattr(self.learn, 'recorder') and hasattr(self.learn.recorder, 'values') and self.learn.recorder.values:

for v in reversed(self.learn.recorder.values):

if v and v[0] is not None:

val\_loss = self.\_to\_float(v[0])

break

else:

val\_loss = np.nan

train\_dice = None

if self.train\_preds and self.train\_targets:

try:

tp = torch.cat(self.train\_preds)

tt = torch.cat(self.train\_targets)

train\_dice = compute\_dice\_score(tp, tt)

except:

train\_dice = np.nan

val\_dice = None

if self.preds and self.targets:

try:

vp = torch.cat(self.preds)

vt = torch.cat(self.targets)

val\_dice = compute\_dice\_score(vp, vt)

except:

val\_dice = np.nan

self.train\_losses.append(self.\_to\_safe(train\_loss))

self.val\_losses.append(self.\_to\_safe(val\_loss))

self.train\_dice.append(self.\_to\_safe(train\_dice))

self.val\_dice.append(self.\_to\_safe(val\_dice))

self.train\_preds.clear()

self.train\_targets.clear()

self.preds.clear()

self.targets.clear()

print(f"[Epoch {self.epoch}] "

f"Train Loss: {self.\_fmt(train\_loss)} | "

f"Val Loss: {self.\_fmt(val\_loss)} | "

f"Train Dice: {self.\_fmt(train\_dice)} | "

f"Val Dice: {self.\_fmt(val\_dice)}")

self.epoch += 1

def after\_fit(self):

np.save('train\_losses.npy', np.array(self.train\_losses))

np.save('test\_losses.npy', np.array(self.val\_losses))

np.save('train\_dice\_score.npy', np.array(self.train\_dice))

np.save('test\_dice\_score.npy', np.array(self.val\_dice))

def \_to\_float(self, x):

try:

if isinstance(x, torch.Tensor): x = x.detach().cpu().item()

return float(x)

except:

return np.nan

def \_to\_safe(self, x):

return float(x) if x is not None and not np.isnan(x) and x <= 1.0 else np.nan

def \_fmt(self, x):

return f"{float(x):.4f}" if x is not None and not np.isnan(x) else "nan"

In [29]:

learn = unet\_learner(

dls,

resnet34,

loss\_func=CrossEntropyLossFlat(axis=1),

metrics=[foreground\_acc, cust\_foreground\_acc],

cbs=[ExtendedMetrics()]

).to\_fp32()

learn.model = learn.model.to(device)

from fastai.callback.tracker import EarlyStoppingCallback, SaveModelCallback

learn.fine\_tune(

8,

wd=0.1,

cbs=[ SaveModelCallback ]

)

# 

# import numpy as np

# print("Train Losses:", np.load('train\_losses.npy'))

# print("Val Losses:", np.load('test\_losses.npy'))

# print("Train Dice:", np.load('train\_dice\_score.npy'))

# print("Val Dice:", np.load('test\_dice\_score.npy'))

# 

# # Load metric arrays

# train\_losses = np.load('train\_losses.npy')

# val\_losses = np.load('test\_losses.npy')

# train\_dice = np.load('train\_dice\_score.npy')

# val\_dice = np.load('test\_dice\_score.npy'

# # Epochs for x-axis

# epochs = np.arange(1, len(train\_losses) + 1)

# # Plot in 2x2 grid

# plt.figure(figsize=(7, 7))

# # 1. Train Dice

# plt.subplot(2, 2, 1)

# plt.plot(epochs, train\_dice, color='green', linewidth=2)

# plt.title('Train Dice Score (↑)')

# plt.xlabel('Epochs')

# plt.ylabel('Dice Score')

# plt.grid(True)

# # 2. Val Dice

# plt.subplot(2, 2, 2)

# plt.plot(epochs, val\_dice, color='blue', linewidth=2)

# plt.title('Validation Dice Score (↑)')

# plt.xlabel('Epochs')

# plt.ylabel('Dice Score')

# plt.grid(True)

# # 3. Train Loss

# plt.subplot(2, 2, 3)

# plt.plot(epochs, train\_losses, color='red', linestyle='--', linewidth=2)

# plt.title('Train Loss (↓)')

# plt.xlabel('Epochs')

# plt.ylabel('Loss')

# plt.grid(True)

# # 4. Val Loss

# plt.subplot(2, 2, 4)

# plt.plot(epochs, val\_losses, color='orange', linestyle='--', linewidth=2)

# plt.title('Validation Loss (↓)')

# plt.xlabel('Epochs')

# plt.ylabel('Loss')

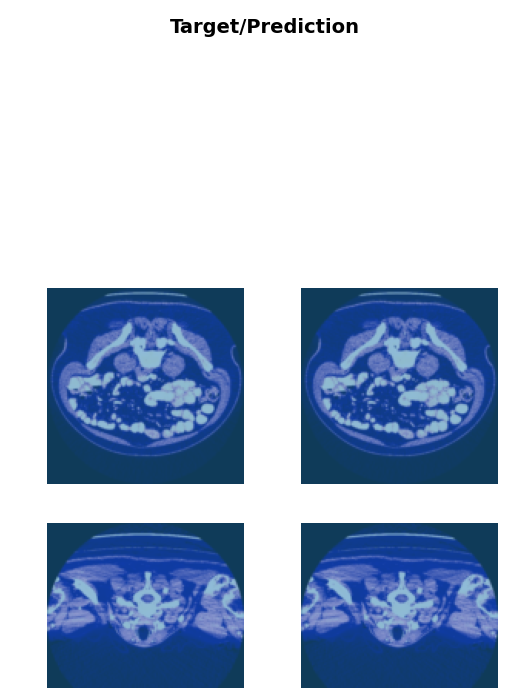
# plt.grid(True)

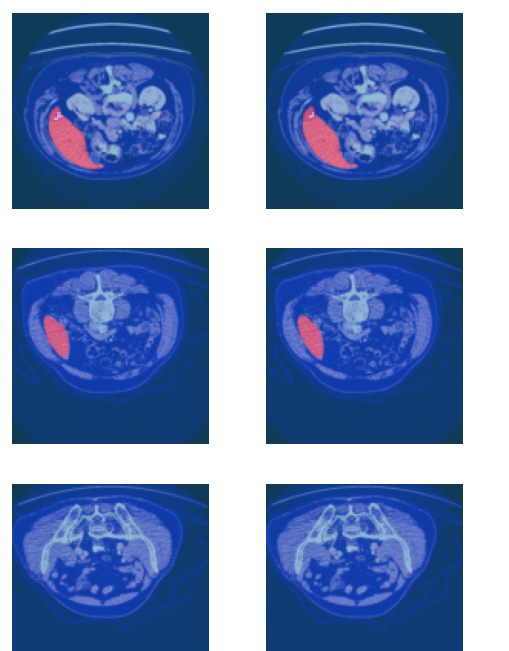
# plt.tight\_layout()

# plt.show()

# 

learn.show\_results()





*Save the model*

learn.export('sdp\_best\_seg\_model.pkl')

torch.save(learn.model.state\_dict(), 'sdp\_seg\_model\_state\_dict.pth')

In [36]:

learn.export(path/f'Liver\_segmentation1')

TESTING MODEL

In [37]:

*# Load saved model*

if (GENERATE\_JPG\_FILES) :

tfms = [Resize(im\_size), IntToFloatTensor(),Normalize()]

learn0 = load\_learner(path/f'Liver\_segmentation1',cpu=False )

learn0.dls.transform = tfms

In [38]:

linkcode

def nii\_tfm(fn,wins):

test\_nii = read\_nii(fn)

curr\_dim = test\_nii.shape[2] *# 512, 512, curr\_dim*

slices = []

for curr\_slice in range(curr\_dim):

data = tensor(test\_nii[...,curr\_slice].astype(np.float32))

data = (data.to\_nchan(wins)\*255).byte()

slices.append(TensorImage(data))

return slices

In [39]:

tst = 20

test\_nii = read\_nii(df\_files.loc[tst,'dirname']+"/"+df\_files.loc[tst,'filename'])

test\_mask = read\_nii(df\_files.loc[tst,'mask\_dirname']+"/"+df\_files.loc[tst,'mask\_filename'])

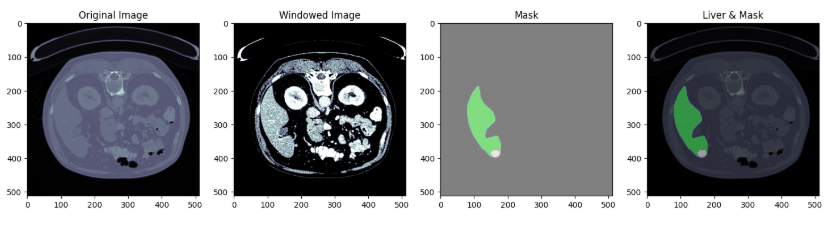
print(test\_nii.shape)

test\_slice\_idx = 500

sample\_slice = tensor(test\_nii[...,test\_slice\_idx].astype(np.float32))

plot\_sample([test\_nii[...,test\_slice\_idx], test\_mask[...,test\_slice\_idx]])

(512, 512, 908)



# *Prepare a nii test file for prediction*

test\_files = nii\_tfm(df\_files.loc[tst,'dirname']+"/"+df\_files.loc[tst,'filename'],[dicom\_windows.liver, dicom\_windows.custom])

print("Number of test slices: ",len(test\_files))

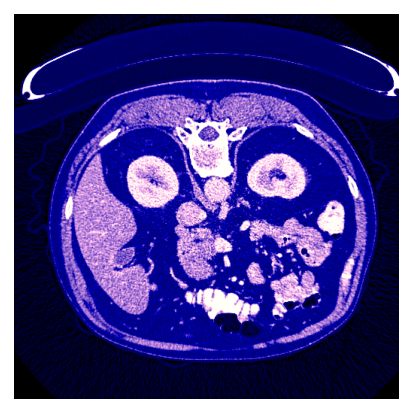
Number of test slices: 908

In [41]:

linkcode

*# Check an input for a test file*

show\_image(test\_files[test\_slice\_idx])



PREDICTION[¶](https://www.kaggle.com/code/madhuri42/sdp-liver-segmentation#PREDICTION)

In [43]:

linkcode

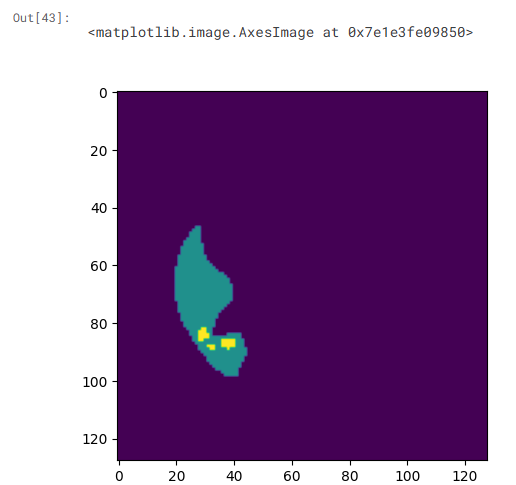
*# Get predictions for a Test file*

test\_dl = learn0.dls.test\_dl(test\_files)

preds, y = learn0.get\_preds(dl=test\_dl)

predicted\_mask = np.argmax(preds, axis=1)

plt.imshow(predicted\_mask[test\_slice\_idx])



Metrics[¶](https://www.kaggle.com/code/madhuri42/sdp-liver-segmentation#Metrics)

In [45]:

linkcode

from sklearn.metrics import accuracy\_score, f1\_score, recall\_score, precision\_score

from medpy.metric.binary import hd95

import numpy as np

import contextlib, sys, os

*# Suppress verbose FastAI outputs*

@contextlib.contextmanager

def suppress\_output():

with open(os.devnull, 'w') as fnull:

old\_stdout = sys.stdout

old\_stderr = sys.stderr

sys.stdout = fnull

sys.stderr = fnull

try:

yield

finally:

sys.stdout = old\_stdout

sys.stderr = old\_stderr

*# Get model predictions silently*

with suppress\_output():

preds, targs = learn.get\_preds(dl=dls.valid)

preds\_bin = (preds.argmax(dim=1)).int()

targs\_bin = targs.squeeze(1).int()

*# Flatten for classification metrics*

y\_pred\_flat = preds\_bin.numpy().flatten()

y\_true\_flat = targs\_bin.numpy().flatten()

*# Compute classification metrics*

accuracy = accuracy\_score(y\_true\_flat, y\_pred\_flat) \* 100

precision = precision\_score(y\_true\_flat, y\_pred\_flat, average='weighted') \* 100

recall = recall\_score(y\_true\_flat, y\_pred\_flat, average='weighted') \* 100

f1 = f1\_score(y\_true\_flat, y\_pred\_flat, average='weighted') \* 100

*# Hausdorff (95%) on the first valid sample only*

hausdorff\_95 = float('nan')

for i in range(len(preds\_bin)):

p = preds\_bin[i].numpy()

t = targs\_bin[i].numpy()

if np.any(p) and np.any(t):

hausdorff\_95 = hd95(p, t)

break

*# Final Clean Output*

print(f"Accuracy: {accuracy:.2f}%")

print(f"Precision: {precision:.2f}%")

print(f"Recall: {recall:.2f}%")

print(f"F1 Score: {f1:.2f}%")

if not np.isnan(hausdorff\_95):

print(f"Hausdorff (95): {hausdorff\_95:.2f}")

else:

print(f"Hausdorff (95): Skipped (no valid prediction-mask pair)")

print(f"Best Dice: {max(val\_dice) \* 100:.2f}%")



GUI CODE:

import sys

import torch

import numpy as np

import matplotlib.pyplot as plt

from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QLabel, QFileDialog, QHBoxLayout

from PyQt5.QtCore import Qt, QThread, pyqtSignal, QTimer, QUrl

from PyQt5.QtGui import QFont, QDesktopServices

from fastai.vision.all import \*

from fastai.learner import load\_learner

from PIL import Image

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from matplotlib.figure import Figure

import pathlib

from pathlib import Path

import warnings

import nibabel as nib

# ---------------------- MODEL LOADING ----------------------

def label\_func(x): return x

def cust\_foreground\_acc(inp, targ): return 1.0

def compute\_dice\_score(preds, targets, smooth=1e-5, axis=1): return 1.0

def get\_x(x): return x

def \_to\_float(x): return 0.0

def \_to\_safe(x): return 0.0

def \_fmt(x): return "0.0000"

class ExtendedMetrics:

    def \_\_init\_\_(self, \*args, \*\*kwargs):

        pass

    def \_\_call\_\_(self, event\_name):

        pass

    def before\_fit(self):

        pass

    def after\_batch(self):

        pass

    def after\_epoch(self):

        pass

    def after\_fit(self):

        pass

def windowed(img, w, l):

    px\_min = l - w // 2

    px\_max = l + w // 2

    img = np.clip(img, px\_min, px\_max)

    return (img - px\_min) / (px\_max - px\_min)

def freqhist\_bins(img, n\_bins=100):

    sorted\_vals = np.sort(img.flatten())

    t = np.concatenate([

        [0.001],

        np.arange(n\_bins) / n\_bins + 1 / (2 \* n\_bins),

        [0.999]

    ])

    t = (len(sorted\_vals) \* t).astype(int)

    return np.unique(sorted\_vals[np.clip(t, 0, len(sorted\_vals)-1)])

def hist\_scaled(img, brks=None):

    if brks is None:

        brks = freqhist\_bins(img)

    ys = np.linspace(0., 1., len(brks))

    flat = img.flatten()

    interp = np.interp(flat, brks, ys)

    return interp.reshape(img.shape).clip(0., 1.)

def to\_nchan(img):

    img = img.astype(np.float32)

    win1 = windowed(img, 150, 30)

    win2 = windowed(img, 200, 60)

    hist = hist\_scaled(img)

    stacked = np.stack([win1, win2, hist], axis=-1)

    return (stacked \* 255).astype(np.uint8)

def load\_model():

    MODEL\_PATH = r"C:\\Users\\ramya\\Downloads\\sdp\_best\_seg\_model.pkl"

    with warnings.catch\_warnings():

        warnings.simplefilter("ignore")

        learn = load\_learner(MODEL\_PATH, cpu=True)

    learn.model.eval()

    return learn

def predict\_single\_slice(learn, slice\_rgb):

    img = PILImage.create(slice\_rgb)

    test\_dl = learn.dls.test\_dl([img])

    preds, \_ = learn.get\_preds(dl=test\_dl)

    pred\_mask = torch.argmax(preds, dim=1)[0].numpy()

    return pred\_mask

class PredictionThread(QThread):

    finished = pyqtSignal(np.ndarray)

    def \_\_init\_\_(self, learn, image\_data):

        super().\_\_init\_\_()

        self.learn = learn

        self.image\_data = image\_data

    def run(self):

        pred = predict\_single\_slice(self.learn, self.image\_data)

        self.finished.emit(pred)

# ---------------------- GUI CLASS ----------------------

class LiverSegmentationApp(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Liver + Tumor Segmentation")

        if isinstance(pathlib.Path(), pathlib.WindowsPath):

            pathlib.PosixPath = pathlib.WindowsPath

        self.resize(900, 700)

        self.setWindowFlags(Qt.Window | Qt.WindowMinimizeButtonHint | Qt.WindowCloseButtonHint)

        layout = QVBoxLayout()

        layout.setAlignment(Qt.AlignTop | Qt.AlignHCenter)

        title = QLabel("DEEP LEARNING FOR LIVER SEGMENTATION")

        title.setFont(QFont('Arial', 20, QFont.Bold))

        title.setAlignment(Qt.AlignCenter)

        layout.addWidget(title)

        subtitle = QLabel("Senior Design Project")

        subtitle.setFont(QFont('Arial', 14))

        subtitle.setAlignment(Qt.AlignCenter)

        layout.addWidget(subtitle)

        spacer = QLabel("")

        spacer.setFixedHeight(150)

        layout.addWidget(spacer)

        self.label = QLabel("Select an image:")

        self.label.setFont(QFont('Arial', 12, QFont.Bold))

        self.label.setAlignment(Qt.AlignCenter)

        layout.addWidget(self.label)

        spacer2 = QLabel("")

        spacer2.setFixedHeight(45)

        layout.addWidget(spacer2)

        button\_style = """

            QPushButton {

                background-color: #87CEEB;

                color: black;

                font-weight: bold;

                padding: 10px;

                border-radius: 10px;

            }

            QPushButton:pressed {

                background-color: #00BFFF;

            }

        """

        self.select\_button = QPushButton("Browse Image")

        self.select\_button.setStyleSheet(button\_style)

        self.select\_button.clicked.connect(self.load\_image)

        layout.addWidget(self.select\_button)

        self.predict\_button = QPushButton("Predict")

        self.predict\_button.setStyleSheet(button\_style)

        self.predict\_button.clicked.connect(self.predict)

        layout.addWidget(self.predict\_button)

                # Bottom container for contact info

        self.bottom\_container = QVBoxLayout()

        self.bottom\_container.setAlignment(Qt.AlignCenter)

        self.contact\_button = QPushButton("Contact Us")

        self.contact\_button.setStyleSheet(button\_style)

        self.contact\_button.clicked.connect(self.toggle\_contacts)

        self.bottom\_container.addWidget(self.contact\_button)

        self.contact\_buttons = []

        # Example usage

        self.create\_contact\_button("Madhuri Sirasanagandla", "7416759083", "https://www.linkedin.com/in/madhuri-sirasanagandla-988957232/", self.bottom\_container)

        self.create\_contact\_button("Amarnath Chigurupati", "7995016856", "https://www.linkedin.com/in/amarnath-chigurupati-3a1b5b238/", self.bottom\_container)

        self.create\_contact\_button("Ankit Kommalapati", "9885081606", "https://www.linkedin.com/in/ankitkommalapati/", self.bottom\_container)

        for btn\_group in self.contact\_buttons:

            for widget in btn\_group:

                widget.hide()

        # Add stretch and then the bottom contact container to main layout

        layout.addStretch(1)

        layout.addLayout(self.bottom\_container)

        self.setLayout(layout)

        self.image\_data = None

        self.learn = load\_model()

        self.plot\_canvas = None

        self.axes = None

        self.go\_back\_button = None

    def create\_contact\_button(self, name, phone, linkedin, layout):

        vbox = QVBoxLayout()

        vbox.setAlignment(Qt.AlignCenter)

        contact\_row = QWidget()

        row\_layout = QHBoxLayout(contact\_row)

        row\_layout.setAlignment(Qt.AlignCenter)

        name\_label = QLabel(f'<a href="{linkedin}">{name.strip()}</a>')

        name\_label.setFont(QFont('Arial', 11))

        name\_label.setTextInteractionFlags(Qt.TextBrowserInteraction)

        name\_label.setOpenExternalLinks(True)

        name\_label.setStyleSheet("color: black;")

        phone\_label = QLabel(phone.strip())

        phone\_label.setFont(QFont('Arial', 11))

        phone\_label.setStyleSheet("color: black;")

        link\_icon = QLabel("🔗")

        link\_icon.setFont(QFont('Arial', 11))

        # Add widgets centered in horizontal layout

        row\_layout.addWidget(name\_label)

        row\_layout.addSpacing(15)

        row\_layout.addWidget(phone\_label)

        row\_layout.addSpacing(10)

        row\_layout.addWidget(link\_icon)

        vbox.addWidget(contact\_row)

        layout.addLayout(vbox)

        self.contact\_buttons.append((name\_label, phone\_label, link\_icon))

    def toggle\_contacts(self):

        for btn\_group in self.contact\_buttons:

            for widget in btn\_group:

                widget.setVisible(not widget.isVisible())

    def load\_image(self):

            file\_path, \_ = QFileDialog.getOpenFileName(self, "Open Image File", "", "JPEG Files (\*.jpg \*.jpeg)")

            if not file\_path:

                return

            image = Image.open(file\_path)

            rgb\_array = np.array(image)

            if rgb\_array.ndim != 3 or rgb\_array.shape[2] != 3:

                self.label.setText("Invalid RGB image. Please use .jpg ")

                return

            self.image\_data = rgb\_array

            self.label.setText(f"Loaded: {file\_path.split('/')[-1]}")

    def predict(self):

        if self.image\_data is None:

            self.label.setText("Please load an image first!")

            return

        self.label.setText("Predicting... Please wait.")

        self.thread = PredictionThread(self.learn, self.image\_data)

        self.thread.finished.connect(self.display\_result)

        self.thread.start()

        # Hide contact section

        self.contact\_button.hide()

        for btn\_group in self.contact\_buttons:

            for widget in btn\_group:

                widget.hide()

    def display\_result(self, prediction\_mask):

        self.label.setText("Prediction complete.")

        self.select\_button.hide()

        self.predict\_button.hide()

        QTimer.singleShot(1000, lambda: self.label.hide())

        if self.plot\_canvas is None:

            self.plot\_canvas = FigureCanvas(Figure(figsize=(9, 3)))

            self.axes = self.plot\_canvas.figure.subplots(1, 3)

            self.plot\_container = QWidget()

            h\_layout = QVBoxLayout()

            h\_layout.setAlignment(Qt.AlignCenter)

            h\_layout.addWidget(self.plot\_canvas)

            self.plot\_container.setLayout(h\_layout)

            self.layout().addWidget(self.plot\_container)

        else:

            for ax in self.axes:

                ax.clear()

            self.plot\_canvas.show()

            self.plot\_container.show()

        self.axes[0].imshow(self.image\_data)

        self.axes[0].set\_title("RGB INPUT CT SCAN")

        self.axes[0].axis('off')

        gray\_img = np.mean(self.image\_data, axis=2).astype(np.uint8)

        self.axes[1].imshow(gray\_img, cmap='gray')

        self.axes[1].set\_title("Grayscale Structure View")

        self.axes[1].axis('off')

        self.axes[2].imshow(prediction\_mask, cmap='viridis', interpolation='nearest')

        self.axes[2].set\_title("PREDICTED OUTPUT")

        self.axes[2].axis('off')

        self.plot\_canvas.draw()

        if not self.go\_back\_button:

            self.go\_back\_button = QPushButton("Go Back to Predict Again")

            self.go\_back\_button.setStyleSheet("""

                QPushButton {

                    background-color: #90EE90;

                    color: black;

                    font-weight: bold;

                    padding: 10px;

                    border-radius: 10px;

                }

                QPushButton:pressed {

                    background-color: #32CD32;

                }

            """)

            self.go\_back\_button.clicked.connect(self.reset\_ui)

            self.layout().addWidget(self.go\_back\_button)

        else:

            self.go\_back\_button.show()

    def reset\_ui(self):

        self.label.setText("Select an image:")

        self.image\_data = None

        if self.plot\_canvas:

            self.plot\_canvas.hide()

        if hasattr(self, 'plot\_container'):

            self.plot\_container.hide()

        self.select\_button.show()

        self.predict\_button.show()

        self.label.show()

        if self.go\_back\_button:

            self.go\_back\_button.hide()

        # Show contact section again

        self.contact\_button.show()

        for btn\_group in self.contact\_buttons:

            for widget in btn\_group:

                widget.hide()

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    window = LiverSegmentationApp()

    window.show()

    sys.exit(app.exec\_())